UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN
CARACAS - VENEZUELA

COLONOSCOPÍA VIRTUAL USANDO TÉCNICAS DE OPTIMIZACIÓN DE RAY CASTING

Trabajo Especial de Grado presentado ante la ilustre
Universidad Central de Venezuela
por el bachiller:

Christian Rodríguez Badiola

Tutores:
Prof. Navarro, Héctor
Prof. Carmona, Rhadamés

Caracas, 6 de julio de 2010

Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

Computación Gráfica

Acta del Veredicto

Quienes suscriben, Miembros del Jurado designados por el consejo de la Escuela de Computación, para examinar el Trabajo Especial de Grado, presentado por el bachiller Christian Rodríguez Badiola con la C.I.: 15.664.829 con el título “Colonoscopía Virtual usando técnicas de optimización de Ray Casting”, a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído, como fue, dicho trabajo por cada uno de los miembros del Jurado, se fijó el día 6 de Julio de 2010, a las 4:00 p.m., para que su autor lo defienda en forma pública, en el Centro de Computación Gráfica, mediante la exposición oral de su contenido, y luego de la cual respondió satisfactoriamente a las preguntas que le fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el Jurado decidió aprobarlo.

En fe de lo cual se levanta la presente Acta, en Caracas a los 6 días del mes de Julio del año dos mil diez, dejándose también constancia de que actuaron como Coordinador del Jurado el Profesor Tutor Héctor Navarro.

_____

Prof. Héctor Navarro

(Tutor)

_____	_____
Prof. Haydemar Nuñez	Prof. Ernesto Coto
(Jurado Principal)	(Jurado Principal)

Resumen

La colonoscopía virtual es una forma no invasiva de realizar un examen médico del colon. Para esto se toman los datos de la zona mediante rayos-x o tomografía computarizada para que estos sean desplegados usando *volume rendering*. En este trabajo se utilizó el *Ray Casting* concurrente y acelerado con técnicas como terminación temprana de rayo, salto de espacios vacíos, coherencia de píxeles y aceleración de rotaciones. Además se uso la estructura de datos *brick* para poder desplegar volúmenes de gran tamaño. También se implementó una iluminación con tablas discretizadas para usar menor cantidad de memoria física y acelerar el despliegue con la misma.

Fue realizado en forma de una librería dinámica en C++ (DLL) para que pueda ser utilizada en un sistema mayor y así pueda ser ampliada.

Abstract

Virtual colonoscopy is a non-invasive medical examination of the colon. For this purpose, the data is taken by an x-ray or CT scans so that they can be used by a volume rendering program. In this paper we implemented a concurrent Ray Casting, using acceleration techniques such as early ray termination, empty space skipping, pixel coherence and accelerated rotations. In addition, the data structure been used is the brick, in order to deploy larger volumes. A especial lighting was used, it consisted in the use of look-up tables that require less physical memory and accelerate the render process.

The system was implemented as a dynamic library in C ++ (DLL) so that it can be used in a larger system.

Dedicatoria

Este trabajo se lo dedico a Dios y a todos los que creyeron en mí durante los años de mi carrera: mis familiares y mis amigos, con una mención especial a mis padres y a Michelle.

También a mis tutores: al prof. Héctor Navarro y al prof. Rhadamés Carmona por ayudarme con este trabajo.

Agradecimientos

En primer lugar a Dios por darme las oportunidades y la fuerza.

A mi madre y a mi padre por estar siempre ahí y ayudarme en todo lo que han podido, gracias por todo.

A Michelle, por estar siempre a mi lado ayudándome en todo y dándome fuerzas, muchas gracias.

A mis amigos, y a mis compañeros de trabajo, por aguantarme todo este tiempo y ayudarme en lo que pudiesen.

A todos los profesores de la universidad que me dieron clases o que conozco y en especial a mis tutores Héctor Navarro y Rhadamés Carmona por brindarme su conocimiento.

A Todos por su paciencia conmigo.

Índice

Introducción

El cáncer de colon es la tercera forma más común de cáncer y la segunda causa más importante de mortalidad asociada al cáncer en el hemisferio occidental; éste causa aproximadamente 639.000 muertes para el año 2004 a nivel mundial [55], solo en Venezuela es el causante de 1.192 muertes para ese mismo año [2].

Los tumores se forman a partir de la unión de células que tienen un comportamiento anormal; éstas se dividen y reproducen sin control. Existen dos tipos de tumores: los benignos, que no son cancerosos y por ende no se extienden a otras partes del cuerpo lo que permite que puedan ser removidos sin presentar mucho riesgo para el paciente; y los tumores malignos, que son cancerosos y pueden dañar a otros tejidos. Existen dos mecanismos de propagación; uno de ellos, la metástasis, es la capacidad que tienen las células del cáncer de penetrar en los vasos sanguíneos y linfáticos, para a través de la circulación sanguínea llegar a otros tejidos y crecer en ellos; el otro mecanismo consiste en penetrar directamente los tejidos vecinos lo que es conocido como invasión.

El intestino grueso está formado por el ciego, el colon y el recto (ver Fig. 1). El colon es un tubo muscular de aproximadamente un metro y medio que se encarga de absorber agua y nutrientes minerales de los alimentos y sirve como área de almacenamiento de las heces.

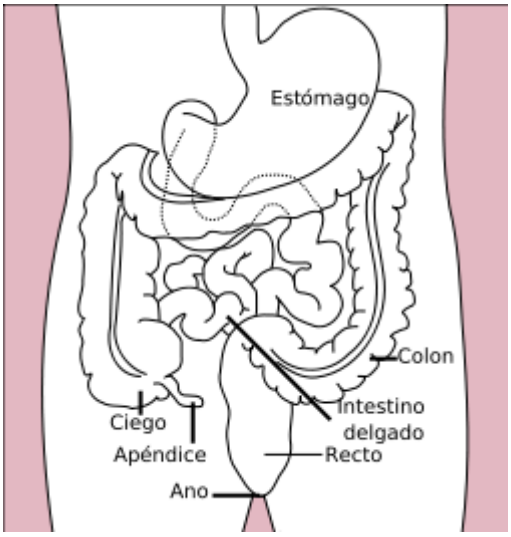

Figura 1: Sistema digestivo inferior mostrando la ubicación del colon.

En el colon, los tumores benignos son conocidos como pólipos (ver Fig. 2), mientras que a los tumores malignos se les llama cáncer de colon o cáncer colorectal.

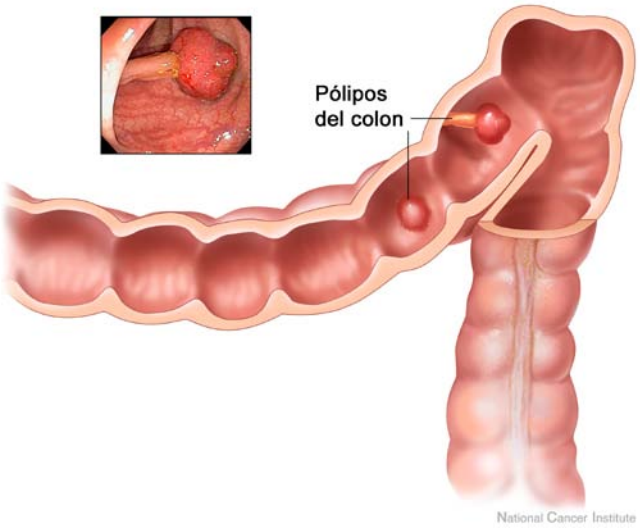

Figura 2: Pólipos en el colon, crecimientos normales de tejido que pueden ser cancerosos.

Para su tratamiento temprano se recomienda la realización de exploraciones en búsqueda de pólipos que puedan formar cáncer. Esto se puede realizar de muchas formas. Entre ellas tenemos las no-invasivas, en donde se obtienen imágenes a través de Rayos-X, RM (Resonancia Magnética) y TC (Tomografía Computarizada), entre otras; se les dice así ya que en el proceso no se remueve tejido del paciente ni se realiza ninguna incisión en el mismo. También se encuentran los mecanismos mínimamente-invasivos, entre los que tenemos la endoscopia (Sigmoidoscopía y la Colonoscopía), el Enema de Bario (ver Enema de Bario); se les llama de esta forma debido a que se realizan a través de una cavidad del cuerpo o a través de una abertura en vez de ser realizada con cirugía abierta.

Para visualizar las imágenes obtenidas a través de los métodos no-invasivos la técnica más utilizada es la visualización de volúmenes *(Volume rendering)*, esta es una técnica que permite la visualización y manipulación de conjuntos de datos en tres dimensiones. Estos datos se obtienen a través de escáneres RM y TC. Esto nos permite extraer información importante a partir de los datos sin utilizar representaciones intermedias. Entre los problemas

que trae esta técnica se encuentran el extenso uso de la memoria y la cantidad de procesamiento para desplegar una imagen final.

En el presente trabajo se buscó una manera de minimizar estos aspectos para permitir que el *volume rendering* sea utilizado en computadoras sin recursos avanzados, como una tarjeta de aceleración gráfica, que cuenten con una memoria principal amplia o procesadores de última generación. Además de aplicar la mencionada técnica, se desean aplicar otras series de algoritmos que ayuden o puedan mejorar la percepción del usuario y así facilitar la identificación de los pólipos del colon, uno de los objetivos de este trabajo. Para ello se emplearan *zonas de importancia* (regiones de interés).

Para lograrlo, se revisaron los algoritmos que actualmente se utilizan para el *volume rendering*, entre los que tenemos:

- Procesador (*acelerado por software*)
  - Ray Casting (Ver 1.6.1)
  - Shear-Warp (Ver 1.6.2)

- GPU (*acelerado por hardware*)
  - Planos alineados al objeto (Axis aligned slices) (Ver 1.6.3)
  - Planos alineados al viewport (Ver 1.6.4)

Además de implementar las técnicas de aceleración *bricking* (ver 1.8.2), *terminación temprana del rayo* (ver 1.8.3), *salto de espacios vacíos* (ver 1.8.4) y *coherencia del espacio de píxeles* (ver 1.8.5.1); también se usaron técnicas para aprovechar los procesadores de varios núcleos (ver 1.9) para mejorar el rendimiento.

Los datos de visualización se obtienen a través de un escaneo TC del abdomen del paciente después de que se limpia y se llena de aire o CO_2 el colon. El paciente debe contener el aire por un corto período y no moverse mientras el escáner toma los cortes de alta

resolución del abdomen. Luego se procede a extraer la zona donde se encuentra el colon de la data obtenida.

Al realizar la colonoscopía de esta manera, si se encontrase una anomalía, el médico podría examinar a la misma en mucho más detalle ya que puede revisarla desde todos los ángulos y puede ver qué hay debajo del tejido para así poder distinguir mejor entre un pólipo maligno de uno benigno [46].

Este trabajo se divide en cuatro capítulos de la siguiente forma:

- Capítulo 1: Contiene el marco teórico que es la base de toda la implementación. Aquí trataremos que son los pólipos, en que consisten las colonoscopías y las nociones del despliegue de volúmenes, así como la teoría de las técnicas de aceleración a usar.

- Capítulo 2: Aquí realizamos el planteamiento del problema y se definen los objetivos.

- Capítulo 3: En este capítulo explicamos la implementación del sistema, que técnicas de aceleración y rendering se usaron.

- Capítulo 4: En este se muestran los resultados de las pruebas de rendimiento, de calidad de imagen obtenida y de memoria.

Por último tenemos las conclusiones y los trabajos futuros.

Capítulo 1: Marco Teórico

La colonoscopía virtual fue introducida por Vining et al. [49] como una alternativa confiable y menos invasiva a la colonoscopía tradicional. Esta representación virtual del colon puede realizarse mediante el uso del *volume rendering* lo que permite una fácil visualización del área interna del colon y así poder buscar a través de el la aparición de posibles pólipos.

Existen varios tipos de pólipos y éstos se clasifican en [55]:

- Según la histología:
  - Pólipo neoplásico epitelial o pólipos adenomatosos o adenomas: Adenoma tubular, adenoma tubulovelloso y adenoma velloso.
  - Pólipo neoplásico no epitelial: Leiomiomas, lipomas, neurofibromas y hemangiomas.
  - Pólipo hamartomatoso: Pólipo juvenil, pólipo de Peutz-Jeghers.
  - Pólipo inflamatorio: Pólipo linfoide benigno. No suelen ser malignos.
  - Pólipo hiperplásico: No suelen ser malignos, pero recientemente se ha descubierto que pudieran ser precancerosos si crecen en el lado derecho o colon ascendente.

- Según la forma de crecimiento:
  - Pólipo pediculado: Tiene un tallo de implantación de unos 1,5 cm e implica menos malignidad porque la degeneración cancerosa tarda en llegar más tarde a la base de sujeción.
  - Pólipo sésil: Tiene una base de implantación amplia (sin tallo) de unos 2 cm e implica mayor malignidad porque la degeneración cancerosa llega antes a la base.

1.1 Adquisición de imágenes médicas del colon

Para realizar la exploración del colon en búsqueda de pólipos que puedan degenerarse en cáncer, se emplean dos grupos de técnicas, las mínimamente-invasivas con las que se

obtienen imágenes del colon a través de cámaras que se introducen por el recto o inyectando un líquido a través del mismo y tomando rayos-X. Por otro lado, se encuentran las técnicas no-invasivas. Éstas sólo toman imágenes del área del colon para luego ser estudiadas. Estas imágenes obtenidas como datos para el *volume rendering* pueden ser usadas para realizar una representación virtual del colon que pueda ser navegada por un médico y así simular una colonoscopía real.

1.1.1 Enema de Bario

Éste es un examen mínimamente-invasivo, consiste en introducir bario en el colon del paciente mediante una sonda, para luego con una máquina de rayos-X observar los cambios en el contraste del líquido (ver Fig. 3).

Este procedimiento se realiza de la siguiente forma:

- La persona se acuesta en la mesa de rayos-X y se le toma una radiografía preliminar. Luego, se le solicita que se acueste de lado y el médico le inserta suavemente una sonda bien lubricada (enema) en el recto. La sonda va conectada a una bolsa que contiene el bario, el cual fluye dentro del colon.
- Se puede inflar un pequeño globo ubicado en la punta de la sonda del enema para ayudar a que el bario permanezca en el interior. El médico vigila el flujo del bario en una pantalla.

Existen dos tipos de enemas de bario:

- Enema de bario de un solo contraste, que utiliza bario para resaltar el intestino grueso.
- Enema de bario de doble contraste que utiliza el bario, pero que también introduce aire dentro del colon para expandirlo.

Se le pide a la persona que adopte diferentes posiciones y la mesa se inclina un poco para obtener diferentes ángulos. En ciertos momentos en los que se están tomando las radiografías, el paciente debe contener la respiración y permanecer quieto durante unos cuantos segundos para que las imágenes no salgan borrosas.

Una vez que se han tomado las radiografías, se retira la sonda de enema y a la persona se le entrega un recipiente o se le ayuda a llegar hasta el baño para que evacue del intestino la mayor cantidad posible de bario. Después de salir del baño, se pueden tomar 1 ó 2 radiografías más.

En un colon sano, el bario debe llenar al mismo uniformemente y éste no debe presentar ninguna deformidad mostrando un contorno normal y debe lucir libremente abierto. Este es un examen exploratorio por lo que no se puede realizar la extirpación del pólipo durante el mismo.

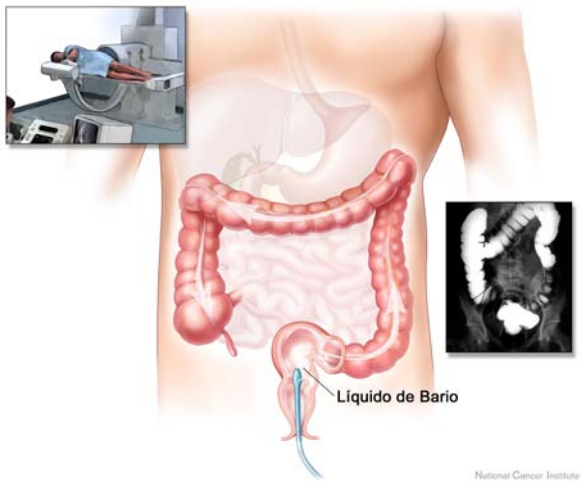

Figura 3: Enema de bario. Se muestra el procedimiento de inserción del enema que llenará el colon con el bario. A la izquierda se muestra una máquina de rayos-X tomando imágenes que permitirán ver el contraste del bario a través del colon. A la derecha se muestra una imagen de rayos-X.

1.1.2 Sigmoidoscopía

Éste es un examen mínimamente-invasivo en donde se usa un endoscopio. En este caso se le llama sigmoidoscopio, pues revisa el colon sigmoide (ver Fig. 4) que se encuentra en la parte final del colon, antes del recto.

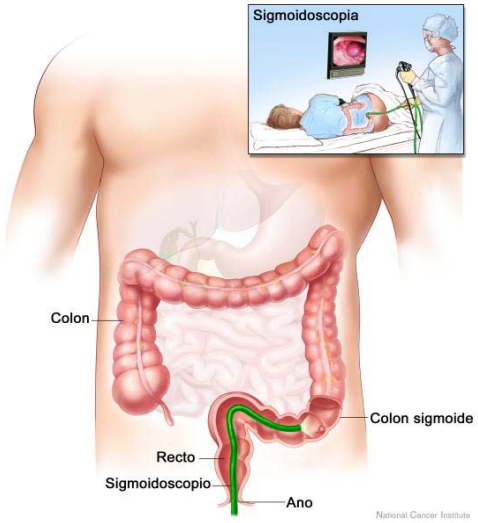

Figura 4: Sigmoidoscopía, mostrando el área del colon llamada sigmoide. A la derecha se observa un médico realizando el procedimiento en donde se inserta el endoscopio (sigmoidoscopio) para obtener las imágenes que se muestran en el monitor.

Existen dos tipos de sigmoidoscopía, la flexible y la rígida. Éstas se diferencian por el tipo de endoscopio empleado, siendo la flexible la más utilizada hoy en día. Se pueden realizar extracciones de tejidos a través del sigmoidoscopio ya que éste posee un conducto que permite al médico realizar esta operación. Se pueden extraer pólipos mediante esta técnica.

1.1.3 Colonoscopía

La colonoscopía es una exploración médica que permite observar el interior del intestino grueso o colon desde el recto hasta el intestino delgado para así poder detectar inflamaciones, úlceras, sangrado o crecimientos anormales (pólipos) los cuales son tumores. Estos tumores pueden ser benignos, pero ciertos tipos de pólipos (adenomatosos) pueden pasar a ser malignos [10]. Durante esta exploración, el médico puede también administrar medicamentos o calor a las paredes del colon así como también extirpar los pólipos.

El proceso se realiza de la siguiente forma:

- La persona se acuesta sobre el costado izquierdo con las rodillas flexionadas hacia el tórax. Luego de administrar un sedante y un analgésico, se inserta el colonoscopio a través del ano y se avanza suavemente hasta la parte más baja del intestino delgado.
- Se inserta aire a través del colonoscopio para tener una mejor visualización y se puede utilizar la succión para retirar secreciones.
- Durante el retiro del instrumento se pueden obtener imágenes asi que se debe examinar con cuidado al momento de retirarlo. Se pueden tomar muestras de tejido con pinzas pequeñas para biopsia que se introducen a través del colonoscopio. Asimismo, los pólipos se pueden extirpar con un asa metálica para su electro-cauterización y se pueden tomar fotografías.

Este tipo de examen conlleva ciertos riesgos a la salud entre los que tenemos:

- Dolor o inflamación de la parte baja del abdomen si se bombeó aire en el intestino durante el procedimiento (esto durará hasta que el cuerpo haga salir el aire adicional poco después del procedimiento).
- Daño al colon (perforación) causado por el colonoscopio y posible infección, que podría tener que ser reparada con cirugía.
- Sangrado dentro del colon, que también puede requerir cirugía.

Estas complicaciones son raras, menos de una por cada 1.000 exploraciones, pero pueden requerir un tratamiento urgente e incluso una intervención quirúrgica. Estos riesgos son algo más altos cuando la colonoscopía se utiliza con aplicación terapéutica, por ejemplo extirpación de pólipos.

La diferencia entre la colonoscopía y la sigmoidoscopía es que en la última sólo se explora el recto y la parte final del colon mientras que en la colonoscopía se examina todo el colon.

1.1.4 Colonografía de TC

La tomografía computarizada, antes conocida como tomografía axial computarizada (TAC), es una técnica para obtener imágenes médicas donde se utiliza la tomografía y la geometría digital para generar una representación tridimensional del cuerpo a partir de imágenes bidimensionales de rayos-X. Generalmente estas imágenes se toman del plano transversal o axial de la persona (ver Fig.5), pero en los tomógrafos más modernos también pueden ser tomadas de forma sagital o coronal.

Para obtener una imagen del colon, primero se le realiza al paciente una limpieza de intestino o se le administra un líquido que permita cambiar el contraste del material fecal con respecto al de los pólipos. Luego se infla el colon con aire o con CO_2 buscando obtener una mejor imagen del mismo y poder detectar más fácilmente los pólipos. El paciente debe contener la respiración por unos segundos para evitar que la imagen tenga errores producto de sus movimientos.

Normalmente, para obtener una mejor imagen y reducir los errores en las mismas se realizan dos barridos con el tomógrafo, uno en posición supina (con la espalda en la tabla del tomógrafo) y otro en posición tendiente (contraria a la anterior), para así poder distinguir mejor entre lo que son residuos fecales y las paredes del colon. Pero por su parte unir estos datos es muy difícil ya que es muy probable que hacer que las imágenes coincidan es un trabajo tedioso ya que cualquier movimiento altera la forma en la que el colon es registrado.

Estos datos se representan en cortes o *slices* que luego seran unidos para realizar su visualización tridimensional. Generalmente, estos cortes son concebidos como un volumen digital, que sirve de entrada a un visualizador de volúmenes, que permite la navegación dentro

del colon. A este proceso se le conoce como Colonoscopía virtual (ver sección1.2 Colonoscopía Virtual).

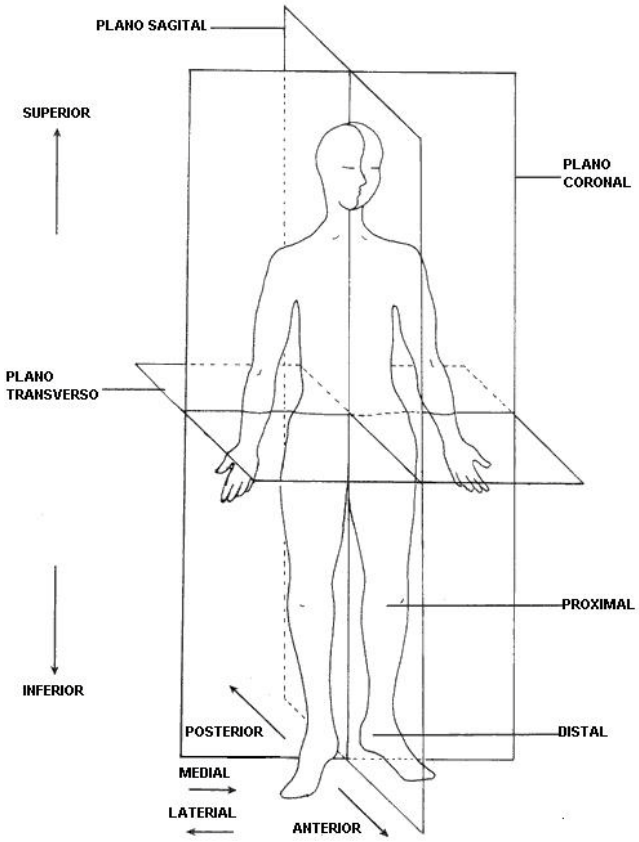

Figura 5: Distintos planos de corte del cuerpo. El plano que normalmente se usa en las tomografías es el plano transverso o axial.

1.1.5 Colonografía de RM

La Resonancia Magnética (RM) es una técnica de captura de imágenes médicas no-invasiva muy similar a la TC. Es capaz de producir imágenes en donde se diferencian más los tejidos que no son iguales pero es mucho más lenta que la TC lo que puede generar artefactos en las imágenes debido al movimiento del paciente. Para obtener las imágenes se puede inflar el colon con aire o CO_2 o también se puede usar un enema para acentuar el contraste entre las áreas. Existen dos tipos de enemas para realizar este procedimiento, enemas brillantes y enemas oscuros (ver Fig. 6).

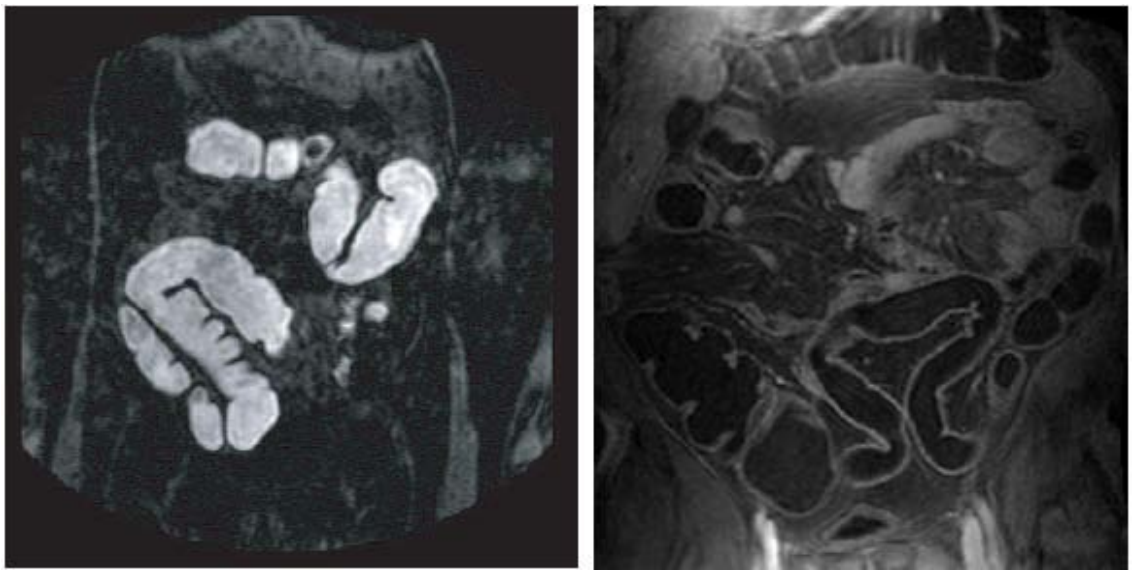

Figura 6: Resonancia magnética usando enemas. A la izquierda usando el enema brillante y a la derecha el enema oscuro.

Al igual que la Colonografía de TC, estos datos se pueden visualizar con la técnica de *volume rendering* para realizar la Colonoscopía virtual.

1.2 Colonoscopía Virtual

Esta es una técnica que genera una representación tridimensional de las paredes del colon para que puedan ser navegadas en búsqueda de pólipos u otras lesiones. Lo que se busca con la Colonoscopía Virtual (CV) es simular una Colonoscopía óptica mediante el uso de los datos del volumen del colon. Estos datos pueden ser obtenidos mediante escáneres MRI (*Magnetic Resonance Imaging*) o TC.

La Tomografía Computarizada se realiza de la siguiente forma:

- Se coloca al paciente generalmente acostado boca arriba o posiblemente de costado o boca abajo. Es posible que se utilicen correas y cojines para ayudarlo a que mantenga una posición correcta y a que permanezca inmóvil durante el examen.

- Se le coloca un pequeño tubo flexible en el recto, hasta dos pulgadas, para poder bombear aire suavemente en el colon por medio de una pera de hule. A veces se utiliza una bomba electrónica para introducir gas de CO_2 dentro del colon. El propósito del gas es distender el colon un poco para eliminar cualquier doblez o pliegue que pudiera ocultar alguno de los pólipos de la vista del médico.
- Luego, se mueve al paciente a través el dispositivo de exploración. Se le solicita al paciente contener la respiración durante aproximadamente 15 segundos, antes de darse vuelta y quedarse boca arriba para que la mesa vuelva a pasar por el dispositivo de exploración. En algunas instituciones la secuencia de posiciones puede ser la contraria: boca arriba primero y boca abajo después. Una vez finalizada la exploración, se retira el tubo.

Cuando ya se han obtenido todas las imágenes del colon, se puede unir toda esta información en un volumen y desplegarlo para navegar a través de él.

Ventajas y limitaciones

La Colonoscopía Virtual es un procedimiento mínimamente invasivo que no requiere anestesia y que permite la obtención de imágenes con buen detalle anatómico. Sin embargo, esta es únicamente un procedimiento diagnóstico, si se encuentran pólipos clínicamente significativos, habrá que eliminarlos mediante la Colonoscopía convencional.

Por otra parte, es más difícil la evaluación de lesiones planas y no es posible valorar alteraciones en la coloración de la mucosa del colon. Se tolera bien y no se necesitan anestésicos ni sedación, lo que elimina el período de recuperación. El paciente podrá reanudar sus actividades normales después del examen.

En un 5% de los pacientes, la Colonografía de TC muestra alteraciones abdominales fuera del colon, que de otro modo no se detectan, puesto que la Colonoscopía convencional

examina de manera visual la zona intestinal. En algunos casos estos hallazgos pueden ser de relevancia clínica.

1.3 Representación de los datos

Para representar los datos se utiliza una matriz tridimensional (ver Fig. 7) compuesta por valores escalares llamados *voxels*. Cada *voxel* contiene el valor de la muestra para esa zona.

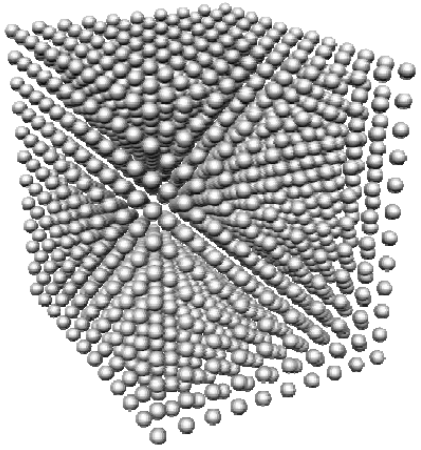

Figura 7: Representación 3D de un volumen digital compuesto de muestras.

Sin embargo, muchas veces los valores representados en estas matrices no están igualmente espaciados entre ellos. Un volumen MRI, por ejemplo, está compuesto por cierta cantidad de imágenes que en su conjunto forman un volumen (ver Fig. 8), en el cual la distancia entre imágenes puede ser distinta a la distancia entre píxeles.

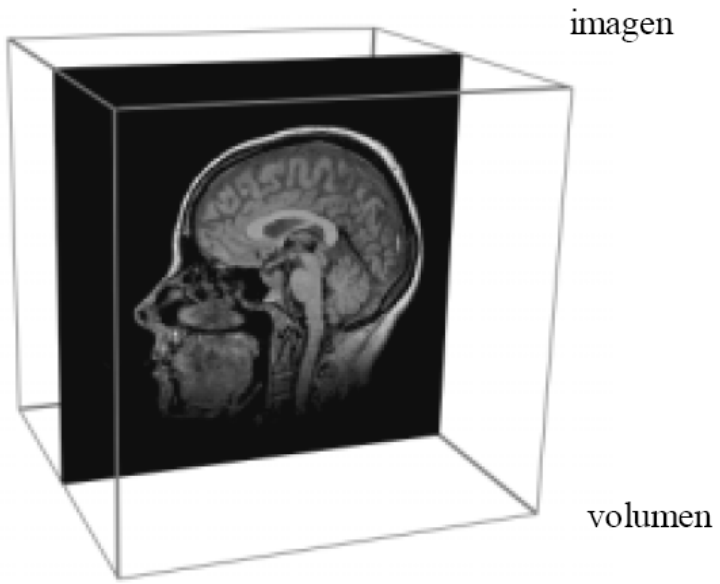

Figura 8: Formación de un volumen MRI a partir de imágenes.

Nuestro sistema visualizador tiene soporte para volúmenes no regulares. Para lograr esto se crean *voxels* vacíos en los ejes no regulares.

1.4 Proyección del volumen

El objetivo del VR es obtener una imagen bidimensional que corresponda a la proyección del volumen sobre un plano denominado *plano de imagen*. Esta proyección puede ser ortogáfica o perspectiva (ver Fig. 9). Para una proyección perspectiva, se lanzan rayos desde el punto de vista hacia el plano de imagen de manera que éstos pasen a través de cada píxel. Estos rayos atraviesan el volumen, lo que genera un color para cada píxel del plano de imagen. Para una proyección ortográfica, los rayos son paralelos y lanzados en forma perpendicular al plano de imagen.

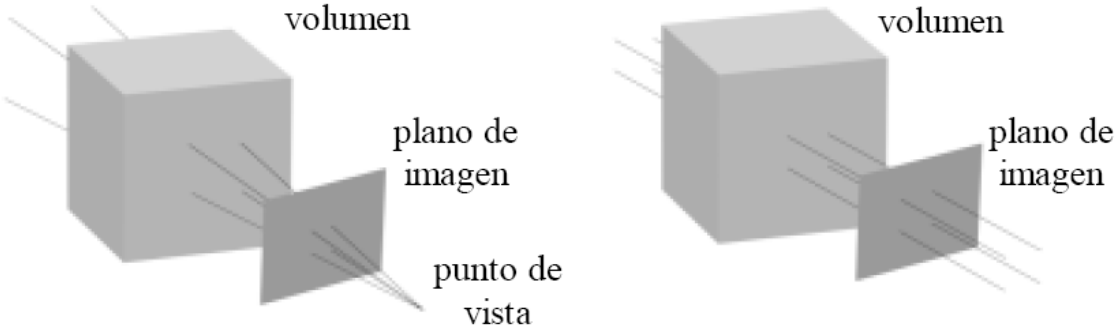

Figura 9: Proyección perspectiva (izquierda) y ortográfica (derecha).

1.5 Ecuación de *Volume Rendering* y su aproximación

La proyección del volumen se realiza a través de un modelamiento de la interacción entre la luz y las partículas en el volumen. La ecuación para VR se puede obtener mediante el modelo óptico de absorción y emisión de luz [36] considerando a cada *voxel* en un volumen de datos como una posible fuente de luz contribuyendo a la intensidad final de cada píxel del plano de imagen y como un atenuador para reducir las contribuciones de otros *voxels* detrás de éste a lo largo de una línea de proyección (rayo) desde el ojo.

Se asigna a cada *voxel* una intensidad como fuente de luz I y un coeficiente de atenuación característica σ. La intensidad I_F de cada píxel en el plano de imagen puede ser calculada como una integral de línea a lo largo del rayo pasando a través del volumen de datos.

$$I_F = \int_0^{\infty} I(t) e^{-\int_0^t \sigma(p)dp} dt \qquad \text{[Ec. 1]}$$

En esta ecuación, t es la coordenada a través del rayo, se coloca en 0 para el plano de imagen y positivo en la dirección opuesta al punto de vista. Esta integral puede ser calculada como una aproximación discreta mediante sumas de Riemann:

$$I_F \approx \sum_{i=0}^{N} I_i * e^{-\sum_{j=0}^{i-1} \sigma_j \Delta x} \Delta x$$

$$I_F \approx \sum_{i=0}^{N} I_i \Delta x \prod_{j=0}^{i-1} e^{-\sigma_j \Delta x} \qquad \text{[Ec. 2]}$$

donde I_i representa la intensidad para una muestra, y $e^{-\sigma \Delta x}$ representa la transparencia T_i de una muestra (comprendida entre 0 y 1). La opacidad α (valor opuesto a la transparencia) queda de la siguiente forma:

$$\alpha_i = 1 - T_i = 1 - e^{-\sigma_p \Delta x} \qquad \text{[Ec. 3]}$$

Ahora, si el paso de integración Δx es unitario, se tiene:

$$I_F = \sum_{i=0}^{N} I_i \prod_{j=0}^{i-1} (1 - \alpha_j) \qquad \text{[Ec. 4]}$$

Esta aproximación es la base para las ecuaciones de composición de imágenes como las definidas por Porter & Duff [42], donde el operador *over* (sobrepuesto) es entendido como:

$$I_F = I_0 + I_1(1 - \alpha_0) + I_2(1 - \alpha_0)(1 - \alpha_1) + \ldots$$

$$I_F = I_0 over I_1 over I_2 over \ldots \qquad \text{[Ec. 5]}$$

Así, cuando el rayo atraviesa el volumen en pasos discretos desde el frente hacia atrás, es decir, desde el punto de vista hacia el volumen (*Front to Back*), el valor de la integral se construye iterativamente a partir de las siguientes ecuaciones:

$$I_{out} = I_{in} + (1 - \alpha_{in})\alpha_i$$

$$\alpha_{out} = \alpha_{in} + (1 - \alpha_{in})\alpha_i \qquad \text{[Ec. 6]}$$

Estas ecuaciones definen la etapa de composición de *Volume Rendering* directo (ver sección 1.7.4 Composición). Una expresión similar se puede dar para una composición desde el volumen hacia el punto de vista (*Back to Front*). Nótese que el valor de *I* no es lo mismo que el color. Lo más común es la interpretación de *I* como color asociado *C* [3], esto es:

$$I = \overline{C} = C * \alpha \qquad \text{[Ec. 7]}$$

1.6 Algoritmos de visualización

Los algoritmos de visualización se pueden clasificar en dos grandes grupos, algoritmos de orden imagen y algoritmos de orden objeto. El algoritmo más conocido, *Ray Casting*, presentado por Levoy [32], es un algoritmo de orden imagen. Se obtiene el color de cada píxel en el plano de imagen lanzando rayos desde el punto de vista hacia el volumen cuando es una proyección perspectiva (*Backward Projection*), y en perpendicular al plano de imagen cuando es ortográfica.

Este rayo es discretizado y se toman muestras del volumen en cada uno de los puntos obtenidos para componerlos en el plano de imagen. Algunas ventajas de este tipo de algoritmos son la alta calidad de las imágenes generadas y la fácil implementación de terminación temprana del rayo (ver sección 1.8 Técnicas de Aceleración).

El algoritmo más representativo de los de orden objeto es el denominado *Splatting*, el cual proyecta el volumen hacia el plano de imagen a partir de cada dato del volumen (*Forward Projection*). Entre las ventajas de este algoritmo se encuentran la paralelización natural del mismo, el direccionamiento simple de los datos debido a que se recorre el volumen en el orden en que los datos están almacenados, y la fácil implementación de aceleraciones por regiones espaciales. La desventaja es la complejidad del filtro de proyección.

Existen algoritmos que pueden ser optimizados utilizando las ventajas de los algoritmos de orden objeto e imagen. Por ejemplo, la factorización *Shear-Warp* (SW), la cual funciona trasladando (*shear*) *slices* del volumen en el eje coordenado más paralelo al vector de

visualización[1] (eje principal) de forma que los rayos en la proyección actual queden perpendiculares al eje principal, facilitando su proyección en una imagen intermedia, la cual posteriormente es transformada en la imagen final (*warp*) (ver sección 1.6.2 *Shear-warp*).

1.6.1 *Ray Casting*

El objetivo básico de la técnica de *Ray Casting* es permitir el mejor uso de los datos contenidos en forma tridimensional sin imponer ninguna estructura geométrica a la misma. Esta técnica resuelve una de las limitaciones de las técnicas de extracción de superficie, la cual es que éstas no toman en cuenta la opacidad/transparencia de los objetos a los que se les está haciendo el *rendering*, como lo serían: fluidos, cristales, etc., lo cual no es un problema para el *Ray Casting*.

El modelo más clásico es el de *Blinn/Kajiya* [3][25], en el cual tenemos un volumen con densidad *D(x, y, z)* que es atravesado por el rayo *R* (ver Fig. 10).

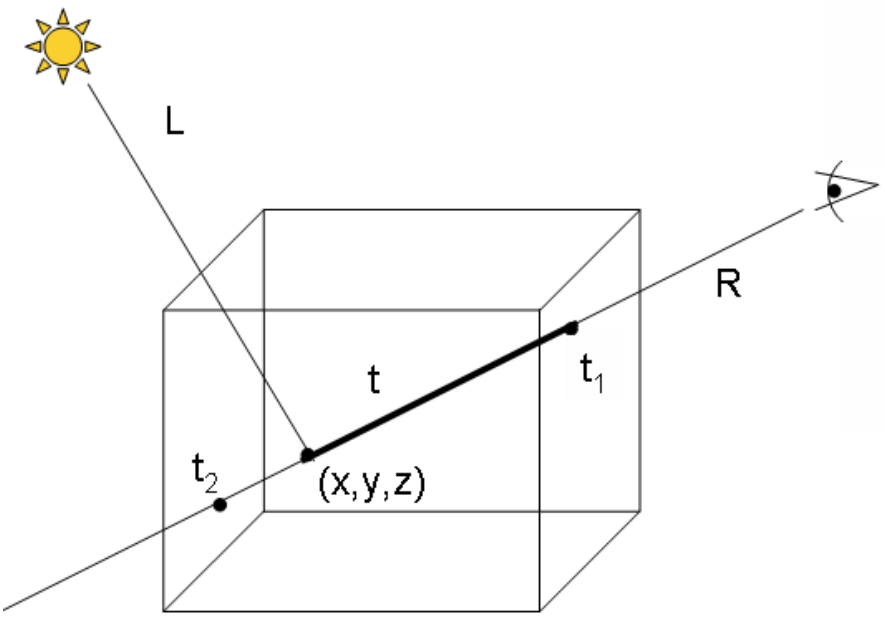

Figura 10: El rayo R es muestreado en el punto (x,y,z) el cual tiene una iluminación dada por L y una opacidad acumulada desde t_1.

[1] Es el vector que determina la posición del punto de vista para una proyección perspectiva. Para una proyección ortográfica, es el vector que determina la dirección perpendicular a los *slices*.

Este algoritmo, utilizando la proyección perspectiva, consiste en lanzar un rayo por cada píxel de la imagen desde el centro de proyección, para aplicar la [Ec. 4] en forma directa (Ver Fig. 11). El rayo atraviesa el volumen en un orden arbitrario y no en el orden en el que está almacenado en memoria. En general, el rayo no pasa por las muestras del volumen, por lo que hay que utilizar un filtro de remuestreo. Comúnmente se emplea un filtro trilineal entre las 8 muestras más cercanas a la muestra a reconstruir. Sin embargo, se pueden utilizar filtros que involucren una mayor cantidad de muestras.

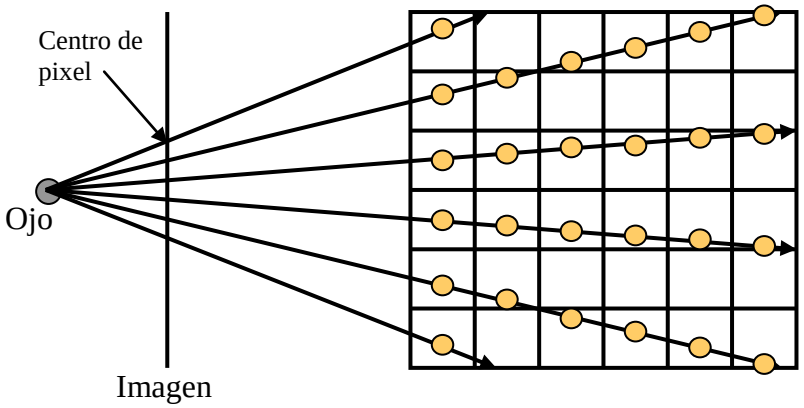

Figura 11: El *Ray Casting*. Por cada centro de píxel de la imagen se lanza un rayo. Aquellos rayos que atraviesan el volumen acumulan un color y opacidad. El paso entre muestras es constante. Un filtro debe ser aplicado para obtener las muestras requeridas.

Si se usa la proyección ortográfica, entonces este algoritmo consiste en trazar un rayo por píxel de la imagen perpendicular a la misma (ver Fig. 9).

Ventajas:

- Se pueden aplicar muchas optimizaciones como lo son la terminación temprana de rayo que consiste en dejar de tomar muestras una vez se haya pasado un umbral definido por el usuario. Con esto se puede reducir considerablemente el cómputo y, más aún, cuando el volumen es poco transparente [30]. Entre las otras técnicas tenemos el salto de espacios vacíos (ver sección 1.8.4 Salto de espacios vacíos)

- Se puede controlar la calidad de la imagen resultado haciendo que la distancia entre las muestras sea mayor o menor según sea el caso requerido.
- La imagen se recorre en el mismo orden en que los píxels están almacenados en memoria principal (algoritmo *image order* u orden imagen) [30].

Desventajas:

- No se accede al volumen en el orden en que está almacenado, ya que los rayos de visualización pueden atravesar el volumen en cualquier dirección. Como resultado, los algoritmos de *Ray Casting* ocupan mucho tiempo en cálculos de posición de las muestras (falta de localidad espacial), y los fallos de página limitan el rendimiento del mismo.
- Es muy lento si el volumen es muy vacío (posee muchos *voxels* sin data) ya que debe atravesarlos completamente accediendo a mucha memoria innecesariamente. Esta desventaja puede ser solucionada con los algoritmos de optimización propuestos en este trabajo.

1.6.2 *Shear-warp*

Éste es un método de visualización en el que se usan cuatro sistemas de coordenadas: coordenadas del volumen, coordenadas de objeto, coordenadas de objeto desplazadas (también llamadas coordenadas de imagen intermedia o *sheared object space*) y las coordenadas de imagen (ver Fig. 12).

Las coordenadas del volumen son, como su nombre lo indica, las coordenadas de los datos del volumen, representados por *voxels*. El eje se encuentra en una esquina del volumen. La unidad tiene el tamaño de un *voxel*. Para pasar a las coordenadas de objeto se permutan los ejes tal que el eje de visualización principal sea el tercer eje. Para pasar a las coordenadas de objeto desplazadas se debe buscar que todos los rayos de visualización sean paralelos al tercer eje de coordenadas. En ese espacio de coordenadas los cortes son tanto alineados al objeto como alineados al *viewport* (región del plano de visualización que será mostrada por el dispositivo de salida). Esto permite la generación de una imagen intermedia, proyectando los

cortes desplazados sobre un plano, accediendo la data volumétrica corte por corte desde el más cercano al más lejano. Luego, se aplica un *warping* a la imagen desplazada para generar la imagen final (ver Fig. 13).

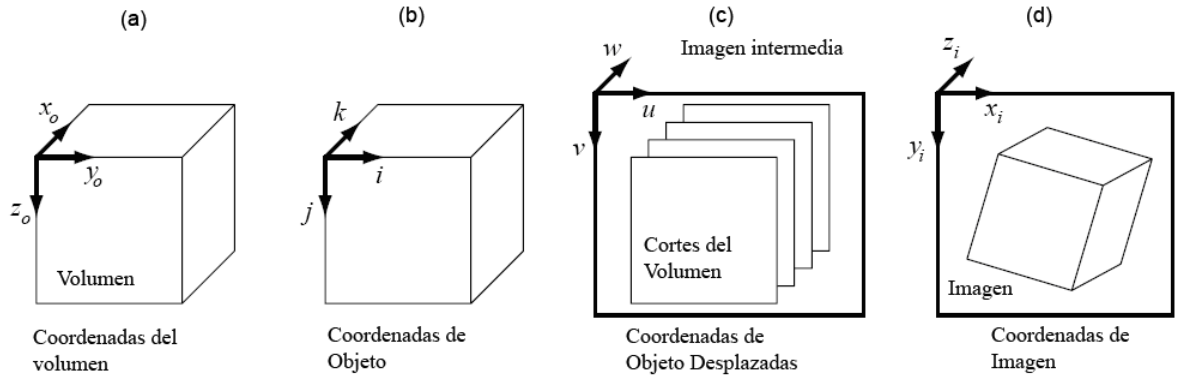

Figura 12: Sistemas de coordenadas usados en la factorización Shear-warp. (a) Es el espacio en donde se encuentran las coordenadas del volumen. (b) Es el espacio objeto, donde se representara el volumen. (c) Es la imagen intermedia compuesta por los cortes del volumen (antes de que se aplique el warp). (d) Imagen final una vez que se le han aplicado las matrices a los cortes.

La versión con proyección paralela de este algoritmo tiene sus orígenes en dos trabajos [26][8]. Posteriormente, la versión perspectiva fue propuesta por Lacroate et al. [30]. Todas las implementaciones iniciales se basaban en software, tanto para la generación de la imagen intermedia como para el *warping*. La versión estéreo fue implementada [24] y, posteriormente, el tiempo de respuesta fue mejorado al realizar el *warping* vía aplicación de texturas 2D [9][45].

Esta técnica se puede usar tanto con software como con hardware. En ambas técnicas se utilizan matrices para lograr las dos transformaciones.

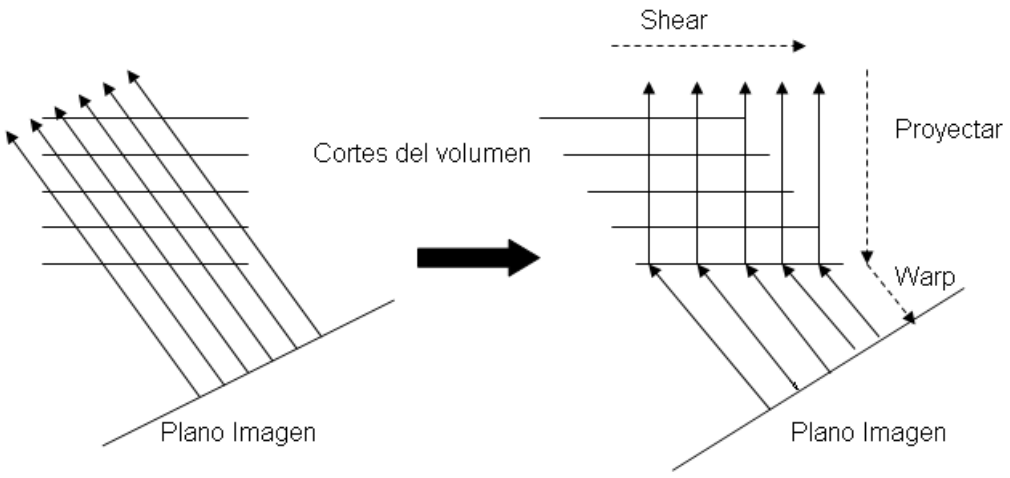

Figura 13: Visualización volumétrica usando la factorización *Shear-warp.*

Matemáticamente, la factorización *Shear-warp* consiste en expresar la matriz de visualización M_{view} como un producto de dos matrices $M_{warp}*M_{shearT}$. La matriz M_{shearT} es usada para desplazar los cortes y generar la imagen intermedia, y M_{warp} es la matriz de *warping*, que corrige la deformación 2D en la imagen intermedia.

$$M'_{view} = M_{warp} * M_{shearT} \Rightarrow M_{warp} = M'_{view} * M^{-1}_{shearT}$$

1.6.3 Planos alineados al objeto (PAO)

Esta técnica es muy sencilla ya que consiste en aplicar texturas a planos alineados a los ejes del volumen. Para lograr esto se deben tener tres copias del volumen en memoria, una por cada eje (Ver Fig. 14). Dependiendo de la dirección del vector de visualización, se escoge qué copia se debe mostrar dado su perpendicularidad con el vector. Este cálculo se realiza comúnmente en coordenadas de ojo o en coordenadas de objeto.

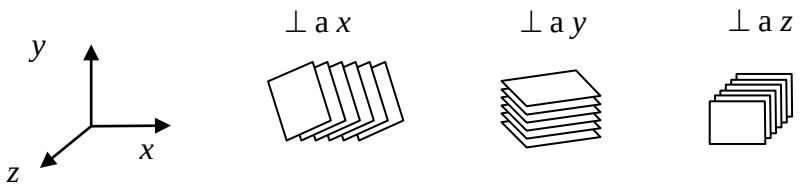

Figura 14: Almacenamiento de tres copias del volumen, una por cada eje principal. Cada copia contiene una serie de cortes paralelos 2D del volumen. Estos cortes son tratados como texturas bidimensionales.

En coordenadas de ojo, la dirección de visualización es v_{ojo}=[0,0,–1,0] para cualquier punto del objeto, cuando la proyección es paralela. Los ejes del volumen e_x[1,0,0,0], e_y[0,1,0,0] y e_z[0,0,1,0] son llevados a coordenadas de ojo mediante la transformación del modelo y vista $vm=V*M$.

$$e_{i\_ojo} = vm * e_i \text{ , } i=x,y,z$$

Luego debemos buscar el vector e_{i_ojo} que forme el menor ángulo con la dirección de visualización. Para esto debemos encontrar el vector que satisfaga

$$\underset{i}{Min}\left\{ \text{ángulo entre } e_{i\_ojo} \text{ y } v_{ojo} \right\} = \underset{i}{Max}\left\{ \text{coseno del ángulo entre } e_{i\_ojo} \text{ y } v_{ojo} \right\}$$

Dado que ambos vectores, e_{i_ojo} y v_{ojo} están normalizados, el coseno del ángulo entre los dos vectores equivale al producto escalar entre ellos:

$$\text{Eje principal} = \underset{i}{Max}\left\{ \left| e_{i\_ojo}.v_{ojo} \right|, i = 1,2,3 \right\} = \underset{i}{Max}\left\{ \left| -e_{i\_ojo}, z \right|, i = 1,2,3 \right\}$$

Para el caso de proyección perspectiva, la dirección de visualización en coordenadas de ojo puede escribirse como v_{ojo}=[$C_{ojo.x}$, $C_{ojo.y}$, $C_{ojo.z}$, 1] – [0, 0, 0, 1], es decir, el vector que parte del ojo y llega al centro del objeto en coordenadas de ojo. En este caso, el cálculo del eje principal queda expresado como

$$\text{Eje principal} = \underset{i}{Max}\left\{ \left| e_{i\_ojo}.v_{ojo} \right|, i = 1,2,3 \right\}$$

Si las dimensiones del volumen son potencias de dos, cada polígono es texturizado con un corte del volumen, utilizando las coordenadas de textura 0 y 1 (ver Fig. 15). Cuando la textura no es potencia de dos, las coordenadas hay que determinarlas según la ubicación del corte dentro de la textura potencia de dos inmediatamente superiores. Por ejemplo, si la dimensión x del volumen tiene 490 *voxels*, podemos crear una textura de 512 píxeles. Si ubicamos el volumen en los primeros 490 píxeles de la textura, las coordenadas serían 0 y 490/512.

El número de polígonos a texturizar está limitado en principio a las dimensiones del volumen. Sin embargo, implementaciones actuales interpolan cortes durante el despliegue vía *register combiners* (combinadoras de registros) para aproximar la interpolación trilineal [13][16], y aumentar la cardinalidad de la geometría intermedia.

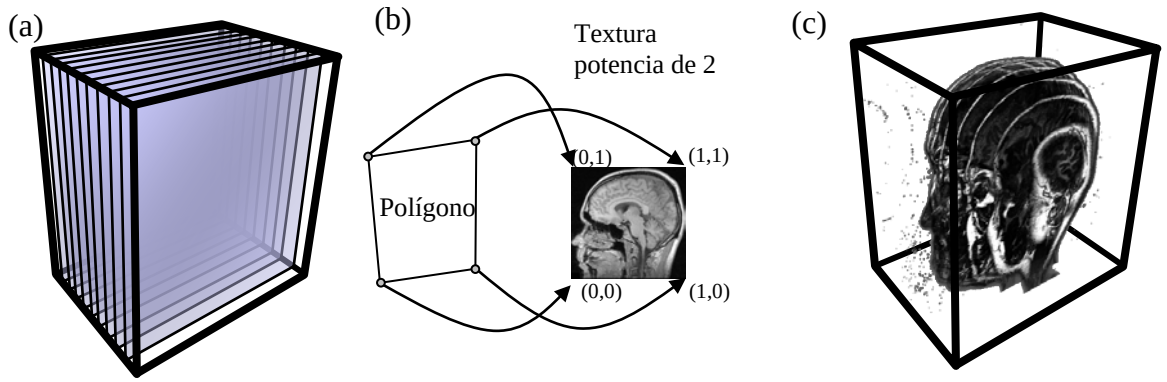

Figura 15: Despliegue de polígonos texturizados alineados al objeto. Si el número de cortes es pequeño, visualmente se pueden notar las separaciones entre cortes, lo cual crea confusión. (a) Muestra los polígonos alineados rotados por la matriz *vm*. (b) Muestra cómo son asignadas las coordenadas de textura a un polígono. (c) Muestra el resultado de componer los polígonos texturizados de atrás hacia adelante. Note que se pueden distinguir los cortes del volumen por insuficiencia en el número de cortes.

Ventajas:

- Fácil de implementar.
- Aplicación de texturas 2D soportada por la mayoría de tarjetas gráficas.
- Excelente tiempo de respuesta, si la copia utilizada en un momento puede ser almacenada completamente en memoria de textura.

- Si el volumen es muy grande, y la copia activa no cabe totalmente en memoria de textura, se pueden cargar los cortes a medida que se realiza el despliegue. El despliegue se podrá así efectuar y el API OpenGL [39] realiza la paginación en forma transparente.

Desventajas:

- Artefactos notorios en ángulos cercanos a 45 grados para cualquiera de los ejes.
- Utiliza interpolación bilineal, que reduce la calidad del despliegue.
- Si las tres copias del volumen no caben en memoria de textura, ocurrirá un fallo de página importante al momento de cambiar de copia.
- Al rotar los polígonos cambia la distancia entre las muestras. Para obtener coherencia en los despliegues hay que corregir las opacidades de la muestras, ya que a mayor distancia recorrida por el rayo entre dos muestras, mayor es la absorción o atenuación. Por lo tanto, la opacidad de cada muestra debe aumentar (ver Fig. 16). Sin embargo, realizar este proceso por cada muestra cuando la proyección es perspectiva, reduce la posibilidad de un despliegue en tiempo real.

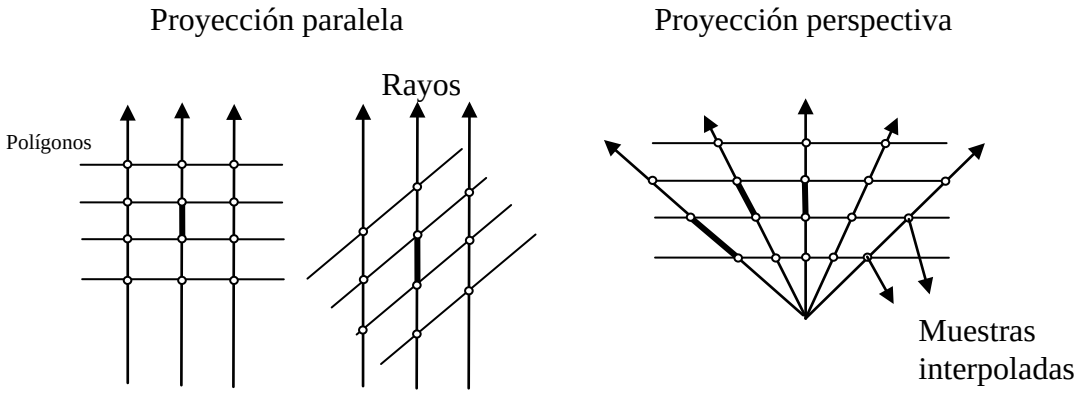

Figura 16: Distancia entre muestras. Al texturizar polígonos alienados a los ejes del objeto, las distancia entre muestras cambia por cada despliegue (proyección paralela) o por cada rayo (proyección perspectiva).

1.6.4 Planos alineados al *viewport* (PAV)

Esta técnica posee dos grandes ventajas con respecto a los PAO; la primera es que sólo requiere una copia del volumen en almacenamiento ya que utiliza una textura 3D que es aplicada a un conjunto de polígonos que están alineados al *viewport* y equivalentemente con el plano imagen (ver Fig. 17). La otra ventaja radica en que al aplicar una textura 3D se puede hacer uso de un filtro trilineal para mejorar la calidad de la imagen resultado y reducir considerablemente los artefactos que se puedan generar.

El primero en mencionar el uso de las texturas 3D para acelerar la visualización fue Akeley en 1993 [1]. Posteriormente la idea fue utilizada y mejorada en trabajos del año siguiente [7] [21] [54].

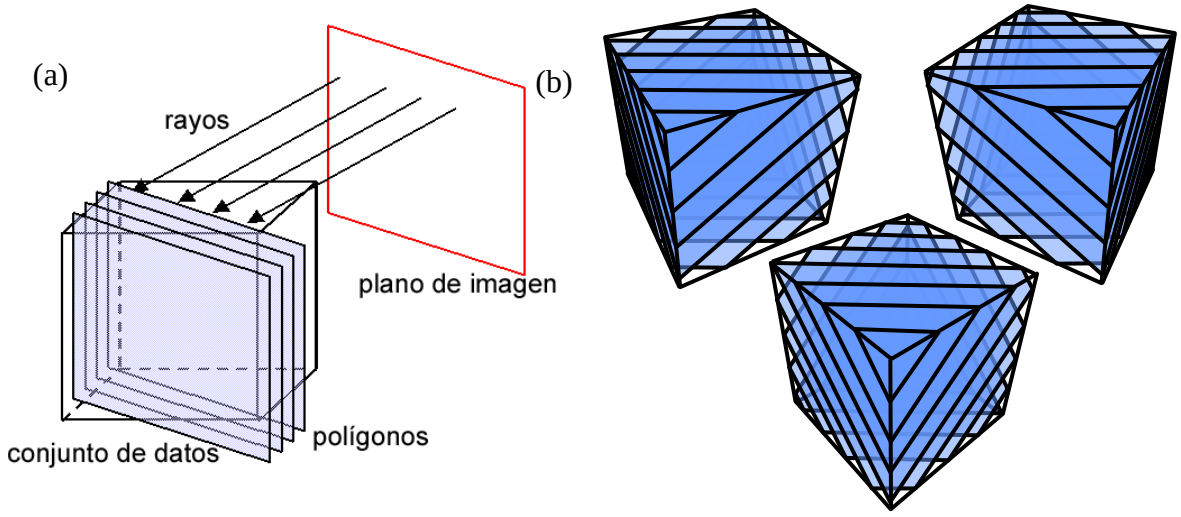

Figura 17: Técnica de planos alineados al *viewport*. Los polígonos no se rotan; ellos están dispuestos siempre paralelos al plano imagen. Se rota el volumen, es decir, las coordenadas de textura de los polígonos, utilizando la matriz de textura provista por el hardware. (a) Muestra la disposición de los polígonos paralelos al plano imagen. (b) Muestra estos polígonos ajustados (proceso denominado *clipping*) a la textura 3D, para distintos ángulos de rotación.

La implementación es bastante sencilla. Se definen polígonos lo suficientemente grandes para que cubran el volumen en cualquier rotación para luego hacerle *clipping* a la geometría, el cual puede ser implementado por software [41] o por hardware. El *clipping*

consiste en cortar los planos alineados con seis planos de corte (*clipping planes*) que se corresponden a las seis caras del cubo. Estos planos son especificados en coordenadas de objeto y transformados con la matriz *model view* para llevarlos a coordenadas de ojo en donde realizaremos el *clipping.*

Los planos se pueden desplegar *front-to-back* o *back-to-front*: Éstos se diferencian en la ecuación que calcula el color y la opacidad en donde tenemos que para *back-to-front* la ecuación es,

$$C_{dst} = (1-\alpha_{src})C_{dst} + \alpha_{src}C_{src}$$

mientras que para *front-to-back* la ecuación se transforma en la siguiente

$$C_{dst} = C_{dst} + (1-\alpha_{dst})\alpha_{src}C_{src}$$

$$\alpha_{dst} = \alpha_{dst} + (1-\alpha_{dst})\alpha_{src}$$

Un punto que se debe tener en cuenta es la cantidad de planos a texturizar. Si se desea crear un conjunto de polígonos cuya distancia entre los cortes sea una unidad (misma distancia que hay entre los *voxels* del volumen), el número máximo de planos *MAXP* para cubrir la diagonal de la caja delimitadora del volumen es:

$$MAXP = \left| \sqrt{width^2 + height^2 + depth^2} \right|$$

Debido a que en esta técnica se usan texturas planas a la misma distancia una de la otra y en paralelo, al usar el método de proyección perspectiva podemos observar que la distancia entre la muestras de dos texturas consecutivas es diferente a medida que nos alejamos del centro como se puede apreciar en la Fig. 19 lo cual crea artefactos en la visualización final ya que estas distancias no se tomaron en cuenta.

Aparte de usar planos como geometría para la proyección, se pueden usar también conchas esféricas; con esta geometría se eliminan los artefactos generados durante la proyección perspectiva (ver Fig. 18).

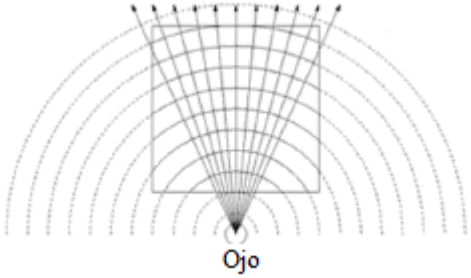

Figura 18: Técnica de planos alineados al viewport usando conchas esférica como geometría para desplegar el volumen. Los clipping planes siguen siendo los mismos que con los polígonos usados en la proyección paralela.

Ventajas:

- Una sola copia del volumen es almacenada en memoria de textura.
- Si la proyección es paralela, la distancia entre muestras interpoladas es siempre la misma, independientemente del ángulo de rotación (ver Fig. 19).
- Utiliza interpolación trilineal, por lo que reduce artefactos.

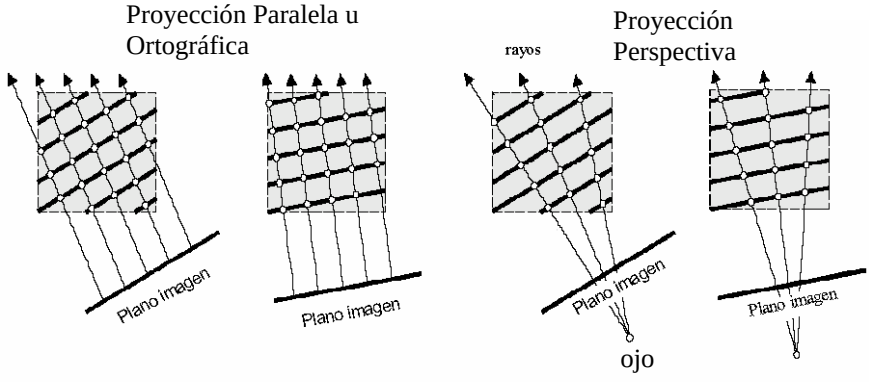

Figura 19: Distancia entre muestras utilizando polígonos alineados al *viewport*. La distancia entre muestras es constante con proyección paralela, independientemente del ángulo de rotación, y es variable por cada rayo con proyección perspectiva.

Desventajas:

- La interpolación trilineal es mucho más lenta que la interpolación bilineal, por lo que el tiempo de respuesta se degrada considerablemente.
- En proyección perspectiva, la distancia entre muestras varía rayo por rayo, lo cual acarrea artefactos (ver Fig. 19). Para reducir el problema, hay que ajustar las opacidades de todas las muestras. Esto reduce la posibilidad de un despliegue en tiempo real.
- Si el volumen es muy grande, y la textura no cabe en memoria de textura, el despliegue del volumen no puede efectuarse, a menos que se utilice una técnica de aceleración como *bricking* [29].
- La implementación es ligeramente más difícil que utilizando planos alineados al objeto: la utilización de matriz de textura y *clipping* deben ser considerados.

1.6.5 *Splatting*

Esta técnica, de orden objeto a diferencia de todas las anteriores, sacrifica calidad por velocidad, debido a esto no es apta para visualizaciones médicas que requieran precisión en la imagen final. Consiste en "lanzar" los *voxels* del volumen al plano imagen en orden *back-to-front* o *front-to-back* (ver Fig. 20).

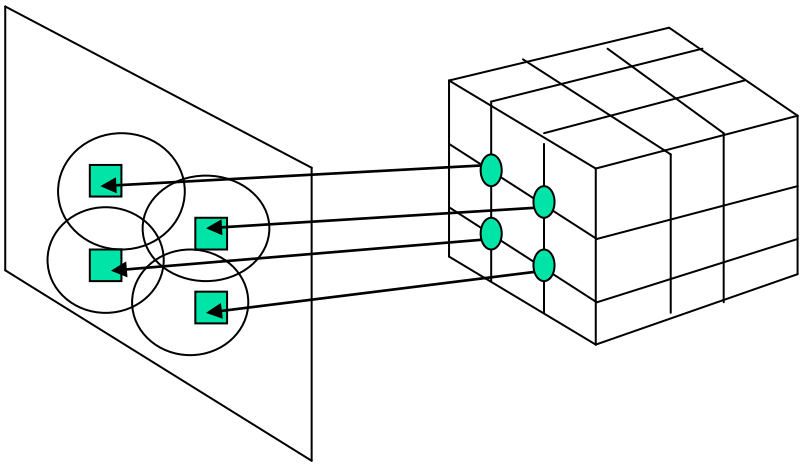

Figura 20: Splatting de los *voxels* al plano imagen.

Estos “splats” se renderizan como discos cuyas propiedades (color y transparencia) varían de forma usando filtros de convolución que se encargan de indicarle al plano imagen cuánto peso tiene ese *voxel* en la imagen final. Entre los filtros encontramos el más clásico que es el Gaussiano, aunque también se puede usar un filtro plano que aporte lo mismo a todos los píxeles que afecte.

1.7 Etapas del *Rendering*

En la proyección de un volumen mediante VR las posibles etapas que se aplican a los datos son preprocesar, segmentar, interpolar, clasificar, y componer los datos. Adicionalmente es posible aplicar iluminación *Phong*, efectos de sombreado, mapeo de texturas y otros. Al proceso de aplicar de manera secuencial algunas o todas estas etapas, se le denomina *pipeline* de VR.

1.7.1 Pre-procesamiento

Se realiza para preparar el volumen, por ejemplo: eliminando el ruido o generando un volumen de distinto tamaño que el original aplicando filtros de interpolación de alto orden. Para la eliminación del ruido generalmente se realiza un filtrado Gaussiano o se eliminan los valores menores y mayores a ciertos umbrales.

A veces la segmentación del volumen se realiza en una etapa de pre-procesamiento segmentando *voxel* a *voxel* para luego almacenar los resultados y renderizar el volumen ya segmentado. Otras veces se incorpora la segmentación al sistema visualizador; en tal caso es necesario incorporarla al *pipeline* de VR.

1.7.2 Interpolación

La ecuación de VR asigna una intensidad como fuente de luz y un coeficiente de atenuación a cada *voxel* a lo largo de cada línea de proyección, pero los *voxels* no se encuentran justo en las líneas de proyección por lo que es necesario interpolar los valores de

los mismos en los puntos de las líneas de proyección que van a ser utilizados en la composición (ver Fig. 10).

Es posible utilizar distintos filtros de interpolación para obtener los valores de los puntos de muestreo, por ejemplo: vecino más cercano, lineal, *splines* [40], *sinc* (filtrado pasa bajos) [58], etc., y la elección del filtro utilizado puede realizarse en forma estática o adaptativa. El filtro de interpolación más utilizado es el filtro de interpolación lineal, que interpola las 8 muestras conocidas más cercanas a la muestra requerida.

1.7.3 Clasificación

La clasificación es el proceso mediante el cual se asigna un parámetro como opacidad, color u otro a los datos. La función de transferencia (FT) define qué color y qué opacidad tendrá la muestra, y esto puede ser en función de la intensidad de los datos, de la magnitud del gradiente local, o de cualquier parámetro que se utilice.

Existen dos tipos básicos de clasificación, los cuales son, pre-clasificación y post-clasificación. Como sus nombres lo indican, estas dos técnicas se diferencian en el orden en que son aplicadas. La interpolación de muestras es necesaria para la construcción de *voxels* inexistentes en el volumen a partir de los *voxels* existentes [13].

Pre-clasificación ocurre cuando se aplica la función de transferencia a cada *voxel* antes de que éstos sean interpolados, es decir, primero se calculan los colores y transparencias de los *voxels* y luego son interpolados. Mientras que la post-clasificación ocurre cuando se aplica la función de transferencia a cada *voxel* luego de que los valores escalares de éstos han sido interpolados (ver Fig. 21).

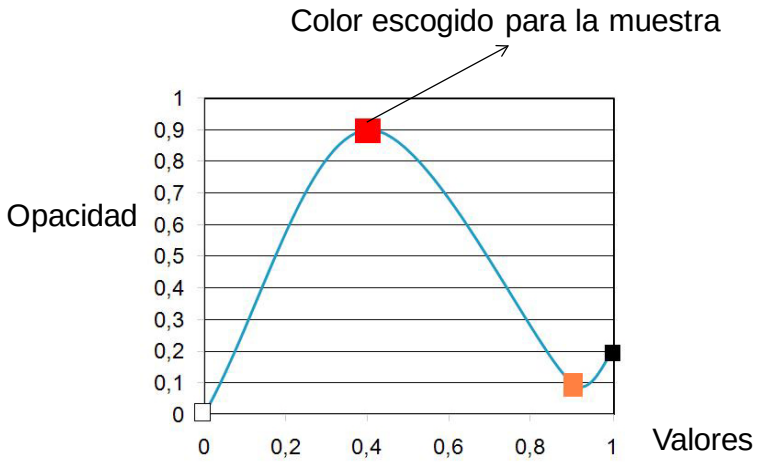

Figura 21: Función de transferencia con valores de color y opacidad para las muestras.

1.7.4 Composición

La composición mezcla el aporte de cada punto de muestreo a la imagen final, mediante el modelo de absorción y emisión de la luz. Intuitivamente esto se ve como la cantidad de luz que logra pasar a través de cada muestra, dependiendo de su opacidad. La composición es una ecuación acumulativa, y relaciona la intensidad que llega a una muestra I_{in} con la intensidad ponderada al dejar la muestra I_{out}, dependiendo de la opacidad actual α_i y la intensidad actual I_i como se muestra en [Ec. 6].

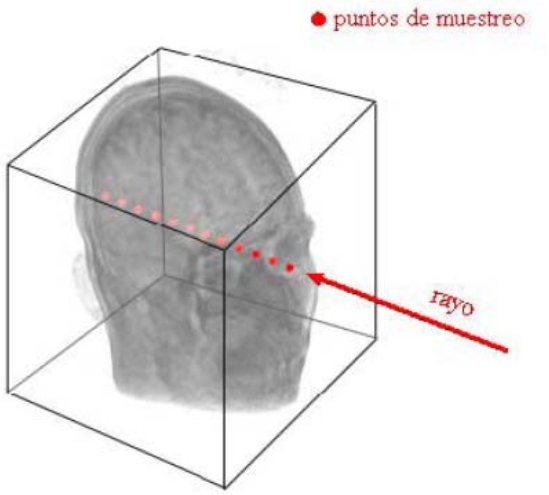

Figura 22: Remuestreo de los datos a través de las líneas de proyección.

La composición mezcla el color y la opacidad obtenida a través del rayo, normalmente a una frecuencia constante (ver Fig. 22).

Si se generan imágenes en escala de grises, el color tiene sólo un componente y las ecuaciones anteriores se aplican a un canal, en cambio si se generan imágenes en colores, las ecuaciones anteriores deben aplicarse para cada canal.

1.8 Técnicas de aceleración

Para tener visualización interactiva se requiere de una frecuencia de *rendering* de 5 a 10 *FPS* (*Frames Per Second* o cuadros por segundo), más aún, en tiempo real se requieren aproximadamente 30 *FPS*. Los algoritmos para VR a fuerza bruta vía software requieren un enorme poder computacional. Por ejemplo, un volumen de N^3 *voxels* generando una imagen de N^2 y tomando N muestras a través del rayo mediante un algoritmo de *Ray Casting* a fuerza bruta requiere del orden de $O(N^3)$ operaciones.

Si $N = 256$ y considerando interpolación trilineal (7 multiplicaciones), composición (3 multiplicaciones) y color (3 canales), se requieren al menos $256^3 * (7 + 3*3) = 268.435.456$ operaciones de punto flotante, para un solo cuadro. Para obtener 10 *FPS*, se requiere de un sistema que tenga una capacidad de procesamiento de al menos 2.6 Giga FLOPS.

Sin embargo, estos requerimientos computacionales pueden ser disminuidos mediante técnicas de aceleración. Algunas de éstas son: estructuras espaciales, terminación temprana del rayo (TTR), *Ray Casting* de 2 fases, etc.

1.8.1 Estructuras Espaciales

Típicamente para un cierto tipo de visualización de un volumen parte de los datos son irrelevantes. Las estructuras espaciales de datos codifican la coherencia espacial de modo que las regiones que contienen datos irrelevantes no son procesadas. Con esto se obtiene un

importante ahorro de recursos computacionales. Entre éstas se encuentran los árboles *Binary space partitioning* o Partición Binaria del Espacio (BSP) [18], *Octrees* [27], etc.

El uso de estructuras jerárquicas como los árboles BSP y los *Octrees* puede beneficiar mucho también a la compresión de los datos (que ocupen menos memoria) ya que estos pueden saber qué áreas requieren más detalle y cuáles no, y cargar en memoria sólo las necesarias.

1.8.2 *Bricking*

Bricking es una técnica que normalmente se usa cuando el volumen sobrepasa la capacidad de la memoria física. Consiste en dividir el volumen en bloques o ladrillos (*bricks*) [53], en lo posible de igual tamaño, en donde cada *brick* no excede la capacidad de memoria y puede ser desplegado individualmente. Así, es posible desplegar volúmenes muy grandes.

Esta técnica permite explotar conceptos como la proximidad de los datos y podría ayudar a eliminar secciones de datos que no contienen información que aporte algo al *rendering*. Esto ayuda a que técnicas como *Ray Casting* reduzcan la cantidad de cálculos y accesos a memoria.

Para utilizar *bricking*, hay que tomar consideraciones especiales en las fronteras de los *bricks*, para lograr una interpolación correcta entre éstos. Cuando escogemos una muestra, tomamos también a sus vecinos más cercanos, esto hace que cada vez que seleccionemos un *voxel* contenido en el *brick*, procedemos a buscar a sus 4 vecinos inmediatos. Debido a esto, cuando buscamos los *voxels* borde del *brick* deberíamos buscar en el *brick* contiguo a éste para así poder realizar la interpolación entre muestras correcta. Para evitar esto se conserva en el *brick* un *voxel* más de redundancia hacia cada dirección. Esto se puede observar en una dimensión para facilitar su comprensión (ver Fig. 23).

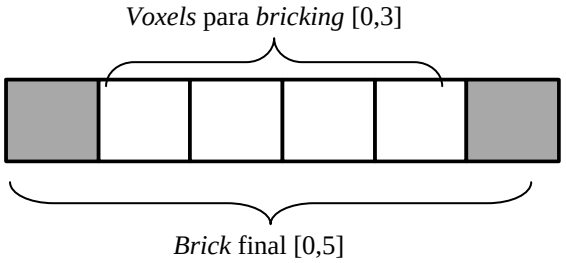

Figura 23: *Brick* usando *voxels* de holgura para poder interpolar las muestras de los bordes sin tener que consultar en más de un *brick*.

Los *voxels* de holgura pueden verse como una envoltura de la caja de *voxels* originales en tres dimensiones (ver Fig. 24).

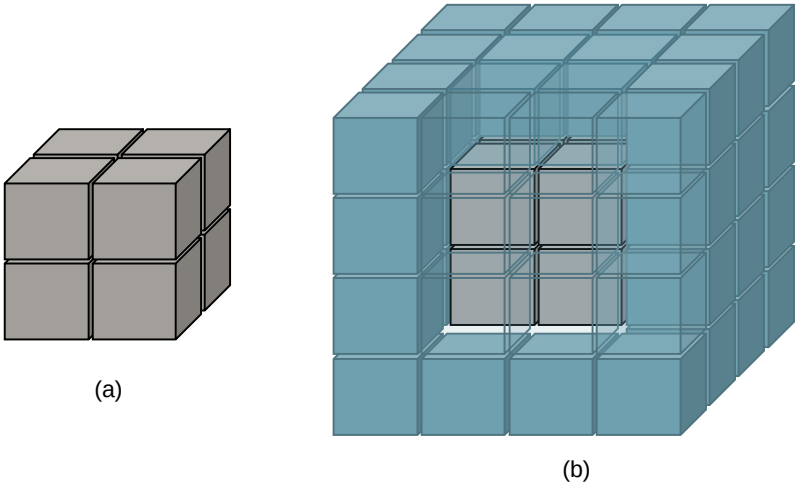

Figura 24: En (a) tenemos los *voxels* asignados al *brick*. En (b) tenemos ese *brick* con los *voxels* de holgura.

Así podemos dividir el volumen en una serie de *bricks* que pueden entrar y salir de la memoria principal fácilmente. La única consideración es que debemos mantener los *voxels* de redundancia para no cambiar de *brick* y así aprovechar al máximo la proximidad de los datos dentro de un *brick*.

1.8.3 Terminación Temprana del Rayo (TTR)

La aceleración de un algoritmo mediante TTR hace que no se procesen los datos que son opacados por datos que se encuentran delante de éstos en la proyección actual. Así, el procesamiento de cada rayo que está siendo proyectado desde el frente hacia atrás es detenido cuando la opacidad acumulada de éste sobrepasa un cierto umbral ya que lo que se encuentra después no aportará nada a la imagen resultado (ver Fig. 25).

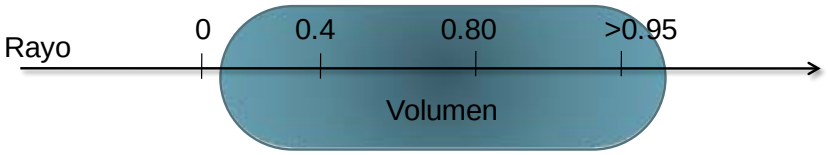

Figura 25: El rayo a medida que atraviesa el volumen va acumulando opacidades hasta llegar a un punto en el que estos valores no aportaran nada a la imagen final.

Para hacer esto, se pueden usar también estructuras de datos que permitan predecir la opacidad del rayo hasta ese punto. Meagher [37] propuso un algoritmo en 1982 que remplazaba la búsqueda en un *octree* por una búsqueda en un *quadtree* 2D, lo que aceleraba el proceso de búsqueda para la terminación temprana de rayo. También, cuando se traza un rayo, la suma del segmento (opacidad) se puede estimar como lo hizo Greener en 1993 [19]. Pero estas técnicas pueden ser muy costosas para el procesador.

1.8.4 Salto de espacios vacíos (*Empty Space Skipping*)

En muchos casos el sistema de *rendering* de volúmenes tendrá que identificar regiones del volumen que no contengan ningún material visible (regiones donde los *voxels* posean una opacidad 0 ó muy cercana a éste). Con esta información se pueden elaborar algoritmos para "saltar" estas zonas vacías y acelerar el proceso de *rendering*.

Para realizar este salto de espacios vacíos se proponen varias técnicas. Sharghi et al. [47] realizó una aproximación del interior del colon usando esferas y esta información era usada en su *Ray Casting* para atravesar el volumen de forma eficiente. You et al. [59] utilizó el *Ray Casting* asistido por polígono introducido por Ávila et al. [61], y utilizó un algoritmo de

surface rendering para encontrar la superficie interna del colon; se calculaba la intersección de cada punto del rayo con el polígono para así poder realizar el salto de espacios vacíos y comenzar así desde la superficie del colon a calcular las opacidades. En este trabajo hablaremos de otros métodos para realizar el salto de espacios vacíos que toman en cuenta la distancia del rayo al volumen. Zuiderveld et al. [60] fue el primero en proponer el uso de salto por distancia para acelerar el *Ray Casting* y Wan et al. [51] aplicó esto a la colonoscopía virtual. Entre estas técnicas tenemos:

Acceleration Corridor: este método utiliza la información que da un rayo del *Ray Casting*, como lo es la distancia al próximo píxel no transparente, para acelerar o mejorar la velocidad de todos los rayos que atraviesan esa misma zona ya que éstos tendrán que saltar una porción vacía similar a la que el primer rayo hizo. De esta manera, todos los rayos del mismo grupo podrán comenzar a tomar muestras desde un *voxel* no vació. Esta técnica se emplea actualmente en colonoscopías virtuales para acelerar la velocidad de *rendering* mientras se está moviendo la cámara a través del colon. A cada grupo de 8 *voxels* cercanos que forman un cubo se les denomina una celda. Si todos los *voxels* del cubo son vacíos a esta celda se le llama una *celda vacía*; si alguno de los *voxels* no es vacío, ésta se denomina *celda no vacía*. Esta información es obtenida en una etapa de pre-procesamineto y se guarda en una matriz para acelerar el proceso de determinar si una celda es vacía o no.

Lo que se busca con esta técnica es lanzar un rayo que sirva de guía para tomar la distancia al próximo *voxel* opaco o que aporte algo al *rendering* para luego usar esta distancia para acelerar a los rayos vecinos a éste haciendo que comiencen a tomar muestras desde esta distancia.

El rayo atraviesa el volumen preguntando si se encuentra en una *celda no vacía*; si es así, se toma la distancia y se calculan los rayos que pasarán por el mismo corredor que este rayo, o sea, sus vecinos, para que estos nuevos rayos usen la distancia de salto para llegar a una *celda no vacía* sin atravesar todo el volumen.

Mehran Sharghi e Ian W. Ricketts [48] realizan esto de la siguiente forma: la distancia hacia la primera *celda no vacía* se obtiene lanzando un rayo del grupo. Esta distancia es usada para todos los otros rayos que pertenecen al grupo y éstos comienzan su *Ray Casting* en la *celda no vacía* que tocó el primer rayo. Ellos sostienen que no es eficiente determinar el grupo exacto de rayos que atraviesa el corredor así que usan una aproximación, el grupo de rayos es obtenido mediante la proyección de la *celda no vacía* intersectada por el primer rayo al plano imagen. Los rayos que forman parte del grupo son aquellos píxeles que son cubiertos por esta proyección (ver Fig. 26 y 27).

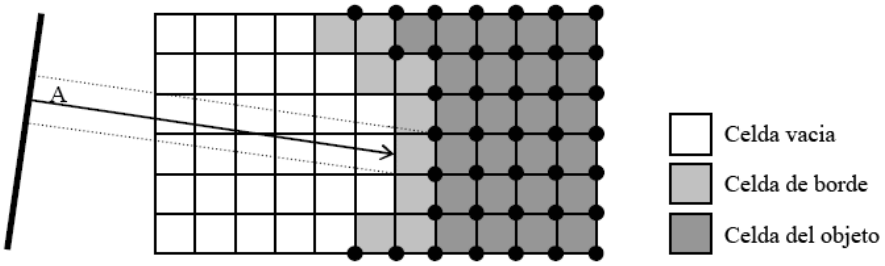

Figura 26: Acceleration Corridor especificado por el rayo A.

El método de Sharghi y Ricketts [48] sólo contempla el *Ray Casting* con proyección paralela pero esta técnica puede ser llevada a proyección buscando la forma de escoger el grupo de rayos que pasa por las mismas celdas. Ellos también hacen uso de un *buffer* de distancia en donde almacenan la información que obtienen los primeros rayos de cada grupo indicándole a los demás rayos a qué distancia está la próxima *celda no vacía*; si no se ha calculado, se lanza un rayo completo para calcularla. Este *buffer* hace uso de la matriz que contiene la información de si una celda es vacía o no que ya fue pre-calculada para acelerar este proceso.

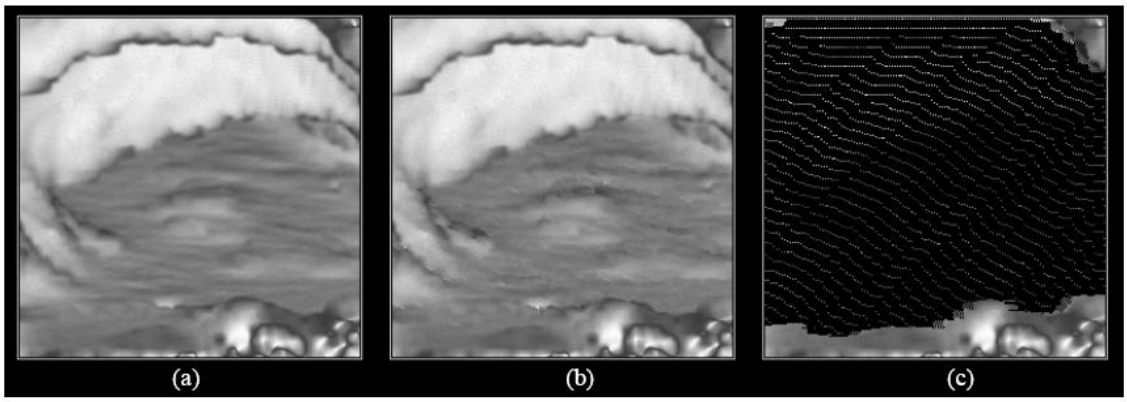

Figura 27: Render del colon. (a) *Ray casting* normal. (b) *Ray Casting* usando el método de Mehran Sharghi y Ian W. Ricketts [48]. (c) Igual que el (b) pero eliminando los píxeles que se benefician con el método.

1.8.5 Coherencia

La coherencia es una técnica de aceleración en donde se explota la similitud que existe entre dos elementos, bien sean píxeles o rayos cercanos como imágenes completas. Lo que se busca es acelerar el proceso de *render* usando esta premisa como base.

1.8.5.1 Coherencia del espacio de píxeles

Se asume que entre dos píxeles con similar tonalidad sólo existen píxeles que tienen un color similar y éstos pueden ser calculados mediante la interpolación de estos píxeles. Una aproximación a esta técnica realizada por Levoy [33] consiste en generar un arreglo lanzando rayos uniformes pero esparcidos en la data del volumen, creando la imagen final con la interpolación de los píxeles obtenidos de este trazado de rayos. Las próximas imágenes son generadas descartando los píxeles interpolados y lanzando más rayos en regiones que posean más complejidad.

1.8.5.2 Coherencia del espacio de objeto

Esta técnica busca evitar tomar muestras en regiones 3D que tienen valores uniformes o iguales. Laur y Hanrahan [31] presentaron una técnica basada en una representación del volumen *pyramidal* para acelerar la técnica del *splatting*, que ajusta un *octree* en la pirámide dada la precisión deseada. El *octree* luego es dibujado usando un set de splats, cada uno del

tamaño correspondiente al nodo del *octree*. Esto permite que se pueda mejorar la calidad de la imagen dependiendo de lo que se desee en el momento, si calidad de imagen o velocidad. Danskin y Hanrahan [11] propusieron un método que detecta eficientemente regiones con poca variación usando una pirámide de volúmen que almacena el mínimo y el máximo valor de sus *voxels* y la distancia entre estas variaciones. Con esta aproximación también se pueden encapsular los espacios vacíos para eliminarlos del proceso de render. Existen otras técnicas como la propuesta por Lorang [35] en la que se indexa de forma eficiente el volumen para acelerar el algoritmo de *ray casting*.

1.8.5.3 Coherencia entre rayos

Esta técnica se basa en que en la proyección paralela todos los rayos poseen la misma pendiente. Fue propuesto por Yagel y Kaufman [56] y consiste en lanzar rayos en paralelo repitiendo una secuencia de pasos especificada por un rayo discreto usándolo como *template* para el resto. Debido a que en la proyección paralela todos los rayos tienen la misma pendiente, si todos los rayos tienen la misma forma entonces no hay que realizar un algoritmo de línea como el de Bresenham para cada uno de ellos. Los rayos sólo se diferencian en el punto inicial pero luego se comportan de igual forma que el original; así que un plano paralelo a una de las caras del volumen garantiza cubrir el mismo.

1.8.5.4 Coherencia cuadro a cuadro (*frame to frame*)

Esta técnica explota la coherencia que hay entre dos imágenes del *render* sucesivas. Para la proyección ortográfica Gudmundsson y Randen [22] presentan un algoritmo que de forma incremental reproyecta los píxeles calculados en una vista a la siguiente y sólo calcula las áreas donde nuevos elementos pueden aparecer. Una mejora a este algoritmo fue presentada por Yagel y Shi [57]; ellos no sólo guardaban el color del píxel sino que también guardaban las coordenadas del primer *voxel* no transparente. Estos valores eran reproyectados y los rayos comenzaban en la posición que le correspondía según las coordenadas reproyectadas. De esta forma, la coherencia cuadro a cuadro puede ser usada para realizar salto de espacios vacíos. Brady et al. [6][5] presentó un método para acelerar el *render* de proyección perspectiva. Éste consistía en dos fases: en la primera se lanzaban rayos cortos y se

componía el color para estos segmentos y éstos eran divididos en niveles dependiendo de la distancia a la que se encontraban del punto de vista; en la segunda fase, los niveles eran compuestos para generar la imagen final. El algoritmo se basa en reutilizar la primera fase para transiciones y rotaciones pequeñas de la cámara.

1.9 *Threading* para *Ray Casting*

Recientemente se han venido desarrollando muchas aplicaciones para *volume rendering* que utilizan algoritmos en paralelo pero la mayoría se basan en hardware especializado que es, por lo tanto, costoso. En este trabajo se buscará aplicar algoritmos paralelos sin el uso de ningún hardware especial sino más bien explotando la capacidad de los procesadores actuales para manejar hilos (*Threading)* y más con los nuevos procesadores que poseen más de un *core* (placa base) que permiten así el uso de hilos que corren en cada uno de éstos.

Los algoritmos de *volume rendering* con hilos se pueden clasificar en dos tipos:

- *Image-partition* (partición de imagen): consiste en dividir la imagen equitativamente entre todos los procesadores o hilos que se vayan a usar. Cada procesador o hilo se encargará de trazar los rayos para su porción de imagen. La carga de trabajo de cada procesador es equivalente a la cantidad de rayos que tenga que trazar. Si se quiere asegurar que la carga entre los procesadores/hilos esté balanceada, se pueden migrar tareas a los procesadores/hilos que tuvieron que hacer rayos menos costosos y por lo tanto más fáciles de producir. Como cada procesador/hilo es responsable de completar una región de la imagen entonces no es necesario pasar por un proceso de composición de la imagen final.
- *Volume-partition* (partición del volumen): este método realiza una reconstrucción y un remuestreado de la data del volumen. Los procesadores/hilos todavía manejan el procesamiento de una parte del plano imagen pero ahora sólo para las regiones que están relacionadas con la data volumétrica que se ve afectada por la transformación de

la vista. La data del plano imagen se puede transmitir a los otros procesadores/hilos para ayudar a la composición final de la imagen.

En este trabajo se utilizará la técnica de *Image-partition* en la que la imagen será dividida en dos o en cuatro partes, según las especificaciones del procesador, para que cada hilo se encargue de una zona del *rendering*. Con esta técnica no es necesario el uso de semáforos o alguna otra técnica de sincronización, ya que la data del volumen, a la que los procesadores/hilos van a acceder, sólo será accedida para lectura haciendo innecesaria una sincronización.

1.10 *Importance Driven Volume Rendering*

La técnica que mencionaremos en este trabajo fue presentada por Ivan Viola et al. [50]. Los otros métodos como el de Martin Haidacher [23] son muy similares a este variando un poco el *pipeline* y añadiendo mejoras como iluminación Phong y resaltado de siluetas.

Esta técnica tiene como objetivo resaltar o permitir que el usuario vea de una forma más clara aquellas estructuras o zonas que se consideren de mayor importancia. Para el trabajo que se desea realizar, que es la detección de pólipos en el colon que puedan ser riesgosos, la estructura o zona a resaltar son los pólipos como tal, ya que de esta forma el médico podrá detectar más fácilmente aquellos pólipos que puedan presentar un riesgo para el paciente.

Para lograr esto, el volumen aparte de contar con información como opacidad, color, etc., debe contener la información de la importancia de cada estructura que lo compone; ésta vendrá dada por un entero positivo al cual llamaremos *object importance*. Aquellas estructuras que posean un valor bajo serán "renderizadas" con una menor opacidad (serán más transparentes), pero esto sólo ocurre en caso que esté obstruyendo a una estructura de mayor valor. Si ese no fuera el caso, la estructura tendría su opacidad normal.

El cambio entre un objeto que se está "renderizando" con menor importancia porque obstruye la visión de un objeto de mayor importancia, a su estado normal se realiza paso a

paso, para que no se noten cambios bruscos en la animación. A esto se le llama *level of sparseness* (nivel de esparcimiento) e indica qué tan denso o transparente es un objeto en un determinado momento. Este proceso se realiza durante el *importance composing* que consiste en "renderizar" las zonas con mayor nivel de *object importance* con el menor *level of sparseness*, o sea, lo más densas posible, y las demás "renderizarlas" con el mayor *level of sparseness*, para que sean lo más transparentes posibles y así no afecten la visibilidad de las zonas con mayor importancia.

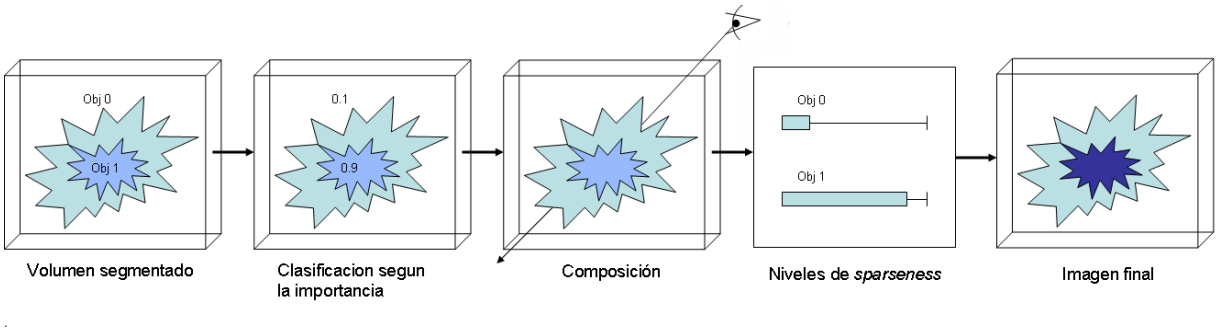

Figura 28: Pipeline del *Importance Driven Volume Rendering*

Existen dos métodos simples para realizar la etapa de *importance composing* y estos son:

Proyección de máxima importancia o *Maximum Importance Projection* (MIP) [50]: es muy simple y rápido. La composición consiste en seleccionar el valor más alto de *object importance* que se encuentre en el rayo. Las demás estructuras que fueron atravesadas por el rayo son "renderizadas" con el más alto *level of sparseness* haciendo así que las estructuras que son "renderizadas" sean completamente densas o transparentes.

Composición de importancia promediada o *Average Important Compositing* [50]: esta composición toma en cuenta a todos los objetos que atraviesa un rayo. La influencia de cada objeto es independiente de la cantidad de rayos que lo atraviesan. Un objeto O tiene un valor de importancia I_o. El rayo r intersecta n_r objetos. El nivel de esparcimiento S_O de un objeto O en el rayo r es igual a la fracción de su propia importancia y la suma de la importancia de todos los objetos interceptados por el rayo.

$$S_o = \frac{I_o}{\sum_{i=1}^{n_r} I_i}$$

Esta técnica no remueve completamente los objetos que afectan la visibilidad de aquellos objetos con mayor importancia pero sí permite visualizarlos mejor.

1.11 *Multiresolution Volume Rendering*

Realizar un *render* de un volumen con multi-resolución significa que el volumen será desplegado con diferentes resoluciones dependiendo del punto de vista o punto de enfoque. Esto se logra realizando pasos de *render* más largos en zonas que están alejadas del punto de enfoque y más finos a medida que se acercan a este punto (ver Fig. 29). A este proceso se le llama nivel de detalle o LOD por sus siglas en inglés nivel de detalle (*level of detail*).

Para saber qué nivel de detalle requiere un área en particular, se pueden usar estructuras jerárquicas sobre el volumen que indiquen las zonas más importantes.

Estos algoritmos se usan para acelerar la velocidad de despliegue del volumen, ya que evitan muchos pasos innecesarios o que no aportan datos de interés para el usuario que se encuentra concentrado en un cierto punto del volumen. Esto se aprecia más en volúmenes muy grandes ya que el *render* de los mismos es muy lento.

Se han propuesto diferentes algoritmos para desplegar un volumen con diferentes resoluciones; muchos de ellos son acelerados por el hardware gráfico. Entre las técnicas para lograr este nivel de detalle se encuentran los métodos jerárquicos [28][52][12] que en esencia crean múltiples resoluciones del volumen y dado un criterio se escoge la resolución que será desplegada. La efectividad de estos algoritmos viene dada por su habilidad de simplificar el despliegue de zonas sin importancia o que no son de interés para el usuario para así disminuir

el consumo de memoria y procesador sin afectar significativamente la calidad de la imagen final.

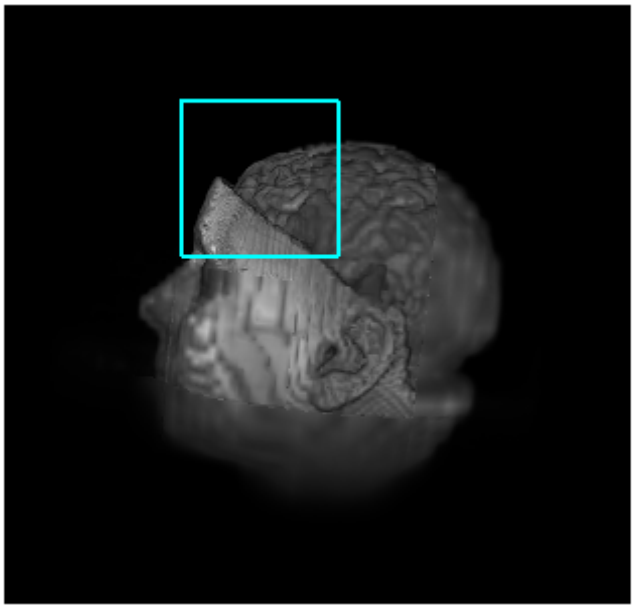

Figura 29: volumen con multi-resolución, la caja indica el punto de enfoque. A medida que se aleja la vista de ella se puede observar cómo se pierde detalle en el volumen

Los métodos existentes pueden ser clasificados en tres categorías: heurísticas estáticas, optimización global y una técnica llamada *interframe feedback*. Los algoritmos basados en heurísticas estáticas [43] determinan los niveles de detalle (LODs) usando criterios fijos como la distancia del objeto, el ángulo de visión y lo que cubre el volumen en la pantalla. Los algoritmos de optimización global [15][17] poseen heurísticas que indican qué tanto benefician a la visualización los cambios, y modelos de *render* que indican qué tan bueno es el mismo. El objetivo de optimizar es maximizar el beneficio de la visualización mientras se baja el costo de *render*. Y, por último, se encuentran los algoritmos que controlan el *feedback* [34][44], éstos ajustan los LODs dependiendo de la diferencia entre el tiempo del *render* deseado y el del actual en el *frame* anterior.

1.12 Iluminación

La iluminación en *volume rendering* tiene como objetivo ayudar a visualizar mejor ciertas características. Existen muchos modelos de iluminación entre los que se encuentra el de Phong [14]. Este modelo propone que la iluminación de un objeto en el espacio, dada una fuente de luz particular, esté determinada por la suma de las contribuciones de las componentes ambientales, difusos y especulares sobre dicho objeto. Haciendo la salvedad de que el volumen sobre el que se va a trabajar tiene una característica particular la cual es que la cámara se encuentra dentro del mismo, esto hace que la luz provenga del mismo punto de la cámara para que así pueda iluminar correctamente las paredes internas del colon.

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

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

Iluminación especular: representa el brillo producido por el reflejo directo de la luz sobre el ojo y es dependiente de la posición del observador. La contribución especular de la iluminación viene dada por: $I = I_s K_s (\vec{N} * \vec{H})^n$, donde el vector $\vec{H}$ es el vector cuya dirección está entre la dirección del vector de la luz y el vector hacia el observador. Este vector indica la dirección del máximo brillo, y su valor es el coseno del ángulo de los vectores que lo conforman. El valor *n* representa el exponente de reflexión especular. Si la superficie fuese un reflector ideal, este valor sería infinito.

Iluminación final: dados los términos de las contribuciones ambiental, difusa y especular del proceso de iluminación, se tiene entonces una fórmula que determina el valor final del color para cualquier elemento del volumen en estudio

$$I = I_a K_a + I_d K_d (\vec{N} * \vec{L}) + I_s K_s (\vec{N} * \vec{H})^n .$$

Para acelerar todos estos cálculos se han investigado muchas técnicas entre las cuales están el uso de *look-up tables* (tablas de consulta rápida) en las cuales se almacenan los valores precalculados de la iluminación final o el gradiente para un conjunto de datos. La calidad final de la imagen dependerá de cuántos valores fueron calculados en la tabla, pero esto trae una consecuencia: para mejorar la calidad de la imagen se tiene que usar mayor cantidad de memoria. Estas tablas pueden tener múltiples tablas enlazadas como lo presentó Lacroute [30] en donde se usaba una de estas tablas para los diferentes materiales de los que estaba compuesto el volumen, piel, huesos, etc. Y cada uno de estos tenía sus propios valores de iluminación. Para nuestro caso no se necesita algo tan complejo como esto ya que la cámara se mantendrá dentro de las paredes del colon donde el único material visible es el tejido del mismo. Algo que se puede tomar en cuenta al momento de usar esta optimización es precalcular las normales de los *voxels* y guardarlas en los mismos para mejorar el rendimiento, pudiendo codificarlas para que ocupen menos espacio.

Algunas técnicas existentes como la presentada en [38] usan esferas divididas uniformemente y con la misma cantidad de muestras discretas como posiciones haya en la tabla; para así llenar la misma con los valores de iluminación conseguidos en estos puntos (ver Fig. 30).

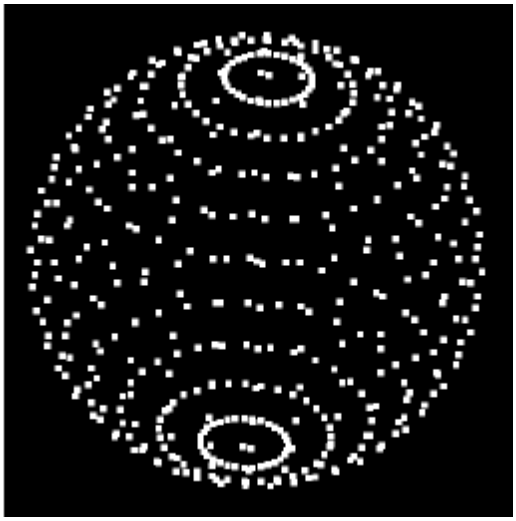

Figura 30: esfera discretizada para obtener los valores de iluminación con los que se llenara la *look-up table*

Luego que la tabla esté llena, se calcula el ángulo en el cual la luz se proyecta desde el *voxel* y se aproxima al valor más cercano de este en la *look-up table* para entonces asignarle ese valor de iluminación. Esto genera errores ya que pocas veces el ángulo conseguido se encuentra exactamente en la tabla. Para mejorar la calidad y disminuir los errores se debe aumentar la cantidad de subdivisiones de ésta, lo que hace que la separación entre los puntos de la esfera sea menor cubriendo así más ángulos.

Capítulo 2: Planteamiento del problema y objetivos

Planteamiento del problema:

Realizar una visualización del colon y navegar a través de el es un tema delicado ya que requiere de una muy buena calidad de la imagen final. El *volume rendering* nos puede dar la solución a esto pero es una técnica de visualización costosa tanto en tiempo como en memoria.

Objetivos:

Desarrollar un módulo de *volume rendering* con la técnica de *Ray Casting* por software, debido a que es una de las técnicas que otorga mejor calidad de la imagen final, haciendo uso de los métodos de aceleración de terminación temprana de rayo y salto de espacios vacíos ya que al estar haciendo el *render* de la parte interna del colon existe un vacío entre la cámara y las paredes del mismo, por lo que saltar este espacio y detener el rayo una vez que ya ha tocado las paredes del colon permiten obtener un mejor rendimiento. Además del uso de *importance driven volume rendering* para resaltar las zonas de interés que puedan ser pólipos, iluminación con *look-up tables* que permite mejorar la calidad de la imagen sin utilizar muchos recursos y *bricking* para poder hacer uso de datos volumétricos muy grandes.

Objetivos específicos:

Se implementará la visualización volumétrica con las siguientes características:

- *Ray Casting* basado en software.
- Terminación temprana del rayo.
- Salto de espacios vacios.
- *Importance driven volume rendering*.
- Iluminación con *look-up tables*.
- *Bricking.*

- Aceleración de las rotaciones.
- Coherencia de *pixels*.
- *Threshold render*.

La plataforma a ser utilizada será la siguiente.

Hardware:

- Computador personal basado en procesador Intel.

Software:

- Microsoft Windows.
- Microsoft Visual C++2005.

Metodología de programación

- Programación orientada a prototipos y orientada a objetos.

Capítulo 3: Implementación

Este capítulo presenta una descripción del visualizador de volúmenes usando técnicas de aceleración y manejo de importancia (*Importante Driven*), enfocado desde el punto de vista de la implementación. El objetivo de este capítulo es dar a conocer los componentes principales que conforman esta aplicación.

Para la visualización de los datos obtenidos a través de imágenes médicas como la tomografía computarizada o la resonancia magnética, mediante *volume rendering*, se utiliza el algoritmo *Ray Casting* tradicional con algunas modificaciones como lo son: a) ejecutar el algoritmo en hilos, lo que permite el despliegue del volumen en resoluciones mayores sin tanta pérdida en el rendimiento; b) *bricking*, usando una estructura de datos comprimida que explota la coherencia espacial de los datos; c) terminación temprana del rayo (*early ray terminación*) lo que acelera el proceso de *rendering* descartando datos que no aportan nada a la imagen resultado; d) también se utilizará la técnica de salto de espacios vacíos (*empty space skipping*); e) coherencia de píxeles, lo que permite realizar el *render* del colon en movimiento a mayor velocidad sacrificando un poco de definición, para acelerar la técnica de *Ray Casting* ya que en el volumen del colon existen muchas áreas que están vacías; f) Iluminación usando *look-up tables*; g) Aceleración de las rotaciones, haciendo un *render* de menor tamaño mientras se esté rotando o moviendo el volumen.

3.1 Aspectos generales de la implementación

Para el diseño y desarrollo de esta aplicación, se seleccionó como tecnología para la visualización del volumen, el renderizado por software por ser parte del requerimiento inicial, para ello se usaron las siguientes técnicas:

- *Volume Rendering* usando *Ray Casting:* Se escogió usar esta técnica por ser la que tiene mayor calidad de imagen entre las técnicas de software.
- *Bricking.*

- Algoritmos de aceleración: *terminación temprana de rayo*, *salto de espacios vacíos*, render intermedio para las rotaciones, *look-up tables* para la iluminación y *coherencia del espacio de píxeles.*
- *Importante Driven Volume Rendering* usando *Average Important Compositing*: Se escogió implementar ésta, debido a que produce resultados más relevantes ya que no elimina los datos que no son tan importantes, sino que acentúa los que tienen más importancia, difuminando un poco el resto.

Todo esto fue realizado en dos módulos principales. El *front-end* o la cara de la aplicación que fue realizada en el lenguaje de programación C# y que se encarga de recibir los datos de entrada y de desplegar el resultado del *Ray Casting*. Y el otro módulo que fue desarrollado como una librería dinámica (DLL) en C/C++. Este último es el que se encarga de toda la lógica de renderizado de la imagen final.

Los puntos de acceso a esta librería son los siguientes:

```
//f3 es un float[3]

//Permite inicializar el handler del volumen
RAYCASTERLIB_API void initVolumeHandler();

//Añadimos un volumen desde un archivo y con dimensiones ancho=w, alto=h,
//profundidad=s a la lista de volumenes a desplegar
RAYCASTERLIB_API int addVolume(char * fname, int s, int h, int w, bool
unicode);

/*Realizamos el Ray Casting sobre el volumen identificado por “handle” con
un ancho=w y alto =h de la imagen final “ptr” desde el punto de vista “eye”
con la direccion de visualizacion “visDir” un nivel maximo y minimo para los
voxeles levelA, levelB.
tf = funcion de transferencia
alpha = valores de opacidad de las muestras
angleX, angleY, angleZ nos indica la rotacion*/
RAYCASTERLIB_API void doRayCast(unsigned char * ptr, int w, int h, int
handle, float pd, f3 eye, f3 visDir, f3 up, unsigned short levelA, unsigned
short levelB, int * tf, float * alpha, int nf, float angleX, float angleY,
float angleZ);

//Rotamos la camara incrementalmente dado los angulos y devolvemos los
//vectores rotados al front-end
RAYCASTERLIB_API void rotateCamera(f3 eye, f3 up, f3 left, f3 vd, f3 currUp,
f3 currLeft, int handle, float angleX, float angleY, float angleZ);
```

```
//Rotamos la camara no incrementalmente sino lo que se pasa a traves de los
//angulos
RAYCASTERLIB_API void rotateCameraTo(f3 eye, f3 up, f3 left, f3 vd, f3
currUp, f3 currLeft, int handle, float angleX, float angleY, float angleZ);

//Obtenemos el histograma de un volumen dado su identificador y lo
//regresamos al front-end para renderizarlo
RAYCASTERLIB_API void getHistogram(int handle, int *histogram, int &
minValue, int & maxValue);

//Permite activar la coherencia de pixeles
RAYCASTERLIB_API void setPixelCoherence(bool enable);

//Permite activar la iluminacion
RAYCASTERLIB_API void setIlumination(bool enable);

//Permite activar el salto de espacios vacios
RAYCASTERLIB_API void setSpaceSkipping(bool enable);

//Permite activar el renderizado con valores de importancia e indica cual es
//el threshold seleccionado
RAYCASTERLIB_API void setImportanceDriven(bool enable, float
_importanceThreshold);

//Permite activar el renderizado que se detiene cuando consigue un voxel
//solido y se pasa la sensibilidad a este. Tambien se indica si se quiere
//mostrar las distancias a estos voxeles nada mas
RAYCASTERLIB_API void setThresholdRender(bool enable, float _threshold, bool
zOnly);

//Permite activar el renderizado de la luz especial que consiste en mezclar
//el render original con el iluminado en la proporcion que diga "_mix" e
//ignorar los gradientes que tengan un tamaño pequeño
RAYCASTERLIB_API void setSpecialLightRender(bool enable, float
_gradThreshold, float _mix);

//Permite decir cuantos hilos se van a ejecutar para renderizar
RAYCASTERLIB_API void setThreadAmount(int threads);

//Permite colocar los valores de opacidad de las muestras dado un volumen
RAYCASTERLIB_API void setAlphaValues(float *alphaVals, float alphaTol, int
handle);

//Permite colocar la distancia del plano a un volumen en particular
RAYCASTERLIB_API void setPlaneDistance(int handle, float distance);

//Inicializamos el ambiente de renderizado
RAYCASTERLIB_API void setEnviroment(int handle, int top, int bottom, int
right, int left, float ncp);

//Permite colocar el coeficiente de iluminacion
RAYCASTERLIB_API void setLightCoeficient(int handle, float lightCoeficient);
```

3.2 Pre-procesamiento de los datos

Para obtener una mejor calidad de imagen y acelerar el proceso de *render* se filtró el volumen y los gradientes de éste. Además se guardo el volumen en *bricks* para aprovechar sus ventajas.

3.2.1 Filtrado del volumen

El filtro empleado es sencillo, un simple filtro de tres dimensiones de tipo gaussiano, esto permite darle mayor peso al centro del mismo y a medida que se aleja se reduce el aporte.

```
// Calculamos el filtro
    int filterWidth = 3;
    for (n = 0; n <= filterWidth; n++) {
        sum = 0.0f;
        for (k = -filterWidth; k <= filterWidth; k++) {
            for (j = -filterWidth; j <= filterWidth; j++) {
                for (i = -filterWidth; i <= filterWidth; i++) {
                    sum += (filter[n][filterWidth + k]
                                                    [filterWidth + j]
                                                    [filterWidth + i] =
                                        (float)exp(-(SQR(i) + SQR(j) +
                                        SQR(k)) / SIGMA2));
                }
            }
        }
        for (k = -filterWidth; k <= filterWidth; k++) {
            for (j = -filterWidth; j <= filterWidth; j++) {
                for (i = -filterWidth; i <= filterWidth; i++) {
                    filter[n][filterWidth + k]
                                        [filterWidth + j]
                                        [filterWidth + i] /= sum;
                }
            }
        }
    }
```

Este proceso crea un archivo que contiene los *voxels* del volumen, previamente filtrados para poder cargar este archivo cuando vuelva a iniciar la aplicación, disminuyendo el tiempo de carga considerablemente.

3.2.2 Bricking

Se creó la estructura de datos "*brick*", ésta se define con los siguientes atributos:

- *data*: aquí se guardan los *voxels* del volumen asignados a este *brick.*
- *isEmpty*: indica si este *brick* está vacío o no aporta valores relevantes al *render* final (su valor máximo de opacidad es muy bajo).
- *index*: id único de *brick.*
- *minVal*, *maxVal*: indican la posición dentro del volumen máxima y mínima que abarca este *brick*.

Por conveniencia se crearán *bricks* cuadrados (mismas dimensiones en todos sus ejes), deben ser potencia de dos y se almacenará una posición más en todas las direcciones por redundancia, para evitar acceder al *brick* vecino si se necesitase interpolar los datos.

3.3 Volume Rendering: Ray Casting

Para poder hacer el *render* del volumen haciendo uso del *Ray Casting* se evaluó el uso de técnicas de aceleración como lo es el *render* de las caras frontales (*front faces*) y las caras posteriores (*back faces*) del *bounding box* o caja contenedora del volumen. Para esto se crea una caja con las coordenadas del volumen en sus esquinas y se hace un *render* para las caras frontales y uno para las caras posteriores guardando cada uno en una textura o imagen, esto nos permite saber cuál es el punto de entrada (*front face*) y cuál es el punto de salida del rayo (*back face*) sin tener que realizar los cálculos de intersección de línea contra caja y nos permite paralelizar más nuestro algoritmo, ya que solo tenemos que consultar en estas texturas para saber la dirección de nuestro rayo por *píxel*, (ver Fig. 31).

Dado que este despliegue es especial porque podemos entrar al volumen, es necesario implementar un plano de corte (*clipping plane*) para cortar la caja que generamos de los puntos de entrada y salida, y con los puntos dados de la intersección entre esta caja y el plano de corte, crear una nueva geometría que permita tener los nuevos puntos de entrada al volumen.

La distancia del plano *near* o cercano también es un factor importante para la creación de una imagen como esta. Esta distancia debe ser menor a lo acostumbrado para despliegue de objetos o volúmenes externos, se escogió el valor de 0,1 para que pudiésemos apreciar mejor el colon a renderizar.

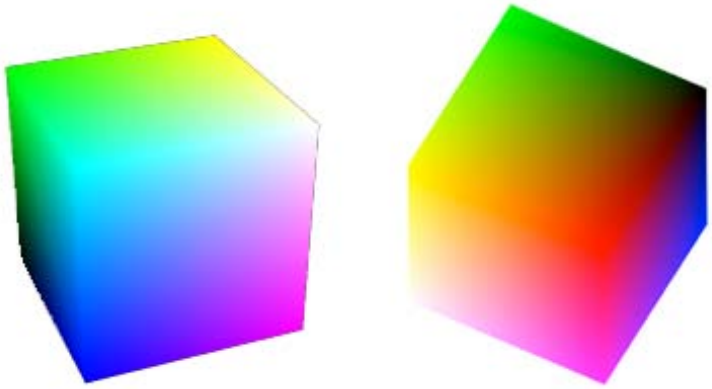

Figura 31: frontface y *backface* del rectángulo del volumen que indican el punto de entrada y salida del rayo.

En este sistema se trabajó con la proyección perspectiva ya que da una sensación más realista del objeto.

3.3.1 Despliegue del volumen

Una vez que tengamos los puntos de entrada y salida del volumen dado un *píxel*, podemos desplazarnos a través del volumen con la dirección dada por estos puntos e ir acumulando opacidad y color para obtener el resultado final por cada *píxel*.

3.3.2 Hilos (Threads)

Para acelerar el *render* final del volumen y permitir que esta imagen pudiese tener una mayor resolución sin una pérdida considerable en el rendimiento, se utilizó programación con *hilos* haciendo uso de la librería *Boost* [4]. Esta librería permite de manera muy sencilla establecer y ejecutar hilos en *C++*.

Estos *hilos* se usaron para dividir el *render* de la imagen final en franjas horizontales, haciendo un *sub-render* de cada una de ellas. La cantidad de hilos indica la cantidad de franjas en las que se dividirá este *render* (ver Fig. 32).

También fueron usados para generar los front-faces y back-faces del volumen, acelerando así el desempeño. En este caso, cada uno de estos se genera en un hilo; así que tener más de dos hilos no aumenta el rendimiento de la aplicación en este aspecto.

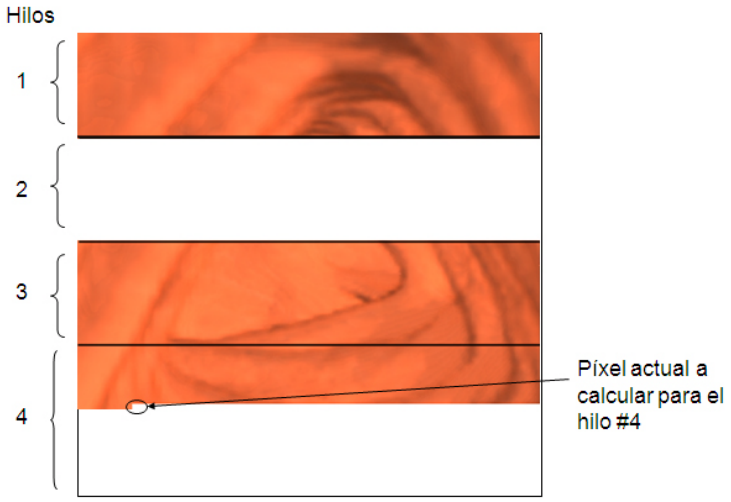

Figura 32: Imagen final generada en 4 hilos separados donde cada uno ejecuta su propio algoritmo de *Ray Casting.*

3.3.3 Terminación temprana de rayo (TTR)

Es muy sencilla de implementar, simplemente se revisa la transparencia acumulada a medida que avanzamos en el rayo y cuando ésta llegue a un valor dado (*threshold*) detenemos el rayo ya que éste no va a aportar ningún valor relevante después de pasar este punto.

3.3.4 Salto de espacios vacíos (Empty Space Skipping)

La implementación que se escogió para este algoritmo está vinculada tanto a los *bricks* como a los *voxels* individuales. Para el salto por *bricks* se establece un valor (*threshold* de

densidad) que indica cuando un *brick* no es lo suficientemente denso para ser tomado en cuenta y, cuando el rayo se encuentra dentro de uno de estos *bricks*, inmediatamente saltamos al próximo *brick* que se encuentre en la trayectoria del rayo. Mientras que para el salto por *voxels* individuales, tomamos la distancia del primer rayo y avanzamos directamente hasta esta posición para los rayos adyacentes a este (ver Fig. 33).

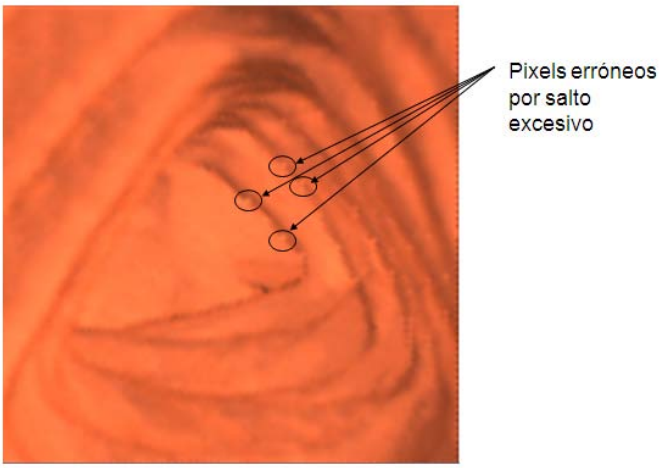

Figura 33: Errores inducidos por el salto de espacios vacíos, cuando la profundidad de los *voxels* cercanos es muy diferente como se ve en este caso entre el fondo y un pliegue del colon.

3.3.5 Aceleración de rotaciones

Al activar esta función, se crea un doble *render*. Mientras la cámara esté estática y no haya interacción con el usuario, se despliega un *render* completo del volumen. Cuando hay movimiento, o se está modificando algún parámetro del *render*, se cambia la resolución de la imagen final a una menor, haciendo que todo el *render* sea de menor resolución para luego expandir la imagen final por software (ver Fig. 34).

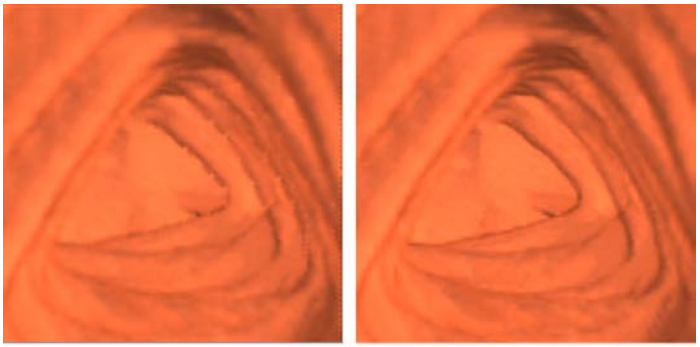

Figura 34: En (a) tenemos un *render* la mitad del tamaño, estirado por software. Mientras que en (b) tenemos el mismo *render* pero a tamaño completo.

3.3.6 Threshold render

Permite hacer un *render* especial convirtiendo a los *voxels* en transparentes o sólidos totalmente. Para ello se tiene una variable *threshold* configurable, que indica cuanta opacidad es necesaria para que el *voxel* sea renderizado. Una vez que el rayo encuentre un *voxel* cuya opacidad sea superior o igual al *threshold*, este recorre unos *voxels* más para obtener un promedio de color e interpola las opacidades para que la suma de estos *voxels* sea la de uno sólido haciendo que el *píxel* final se vuelva totalmente opaco. Esto permite diferenciar mucho mejor los bordes y evita que el rayo siga obteniendo muestras (ver Fig. 35).

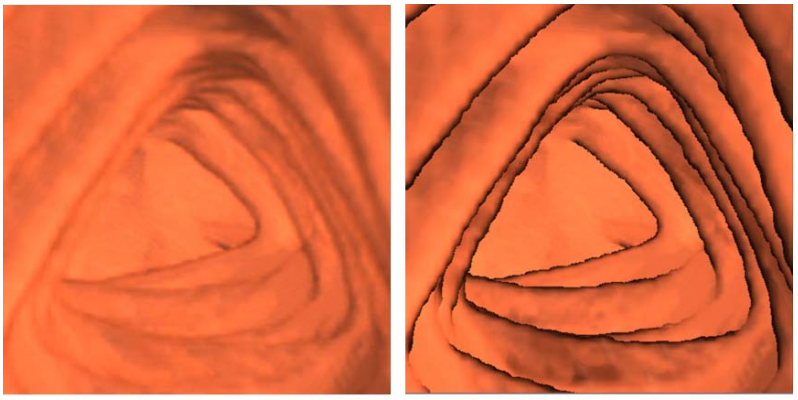

Figura 35: En (a) tenemos un *render* completo normal y en (b) tenemos un *render* utilizando un *threshold* de 0,15 lo que indica que cualquier *voxel* con alpha superior a esto es tomado en cuenta, los que son inferiores son descartados.

3.4 Iluminación

3.4.1 Look-up table para gradientes

Para la creación de esta tabla, se consideró crear un icosaedro y subdividirlo, donde los centros de cada uno de estos triángulos forman los gradientes (normales) discretos del volumen.

Esta tabla se calcula debido a los límites de memoria principal que existen. De esta forma reducimos la cantidad de memoria que ocupan los gradientes del volumen al usar esta tabla como valores discretos de los gradientes reales y asignando a cada *voxel* la posición en la tabla de donde se encuentra el valor del gradiente más cercano (en la *look-up table*), al calculado para el. Pero para realizar esto se buscó un equilibrio entre uso de memoria y calidad gráfica, por lo cual se llegó a la conclusión de usar tres subdivisiones del icosaedro, dando como resultado (4^3 x 20) = 1280 centros, (ver Fig. 36).

Aprovechando que tenemos un número reducido y discreto de gradientes, se puede pre-calcular la iluminación creando otro arreglo que contenga el producto punto entre la normal de la iluminación y el gradiente, así cuando se quiera aplicarle iluminación al volumen solo se tiene que buscar con un índice el valor para aplicárselo al color actual.

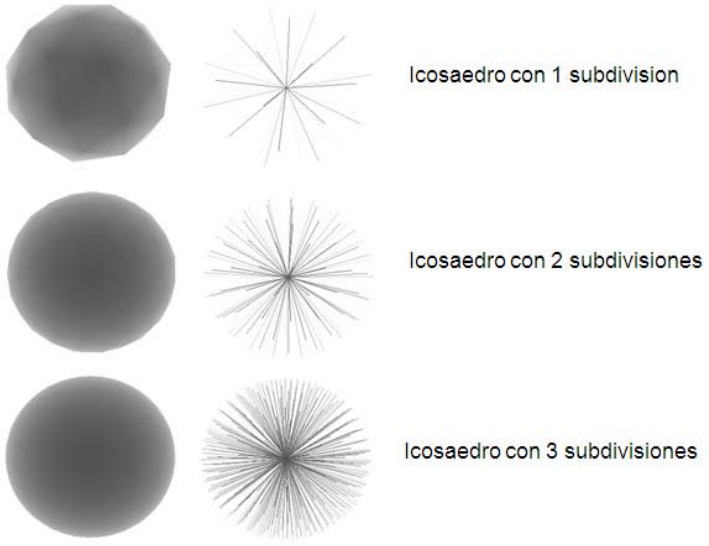

Figura 36: Subdivisiones de un icosaedro, si tomamos los centros de cada triangulo como vectores podemos obtener la discretización de las normales.

3.4.2 Iluminación especial

Simplemente consiste en mezclar bajo una proporción configurable por el usuario, el resultado del *render* con iluminación y el *render* para ese punto sin iluminación alguna, esto hace que las sombras que se generan por errores en los datos se suavicen y obtenemos así una imagen más limpia.

3.5 Coherencia del espacio de píxeles

Se escoge esta técnica ya que beneficia mucho más al *ray casting*, haciendo que la cantidad de rayos se reduzca a la mitad pero sin sacrificar calidad de imagen.

Esta coherencia consiste en lanzar rayos no consecutivos entre si para luego con un algoritmo simple, rellenar estos espacios vacíos con los *píxeles* vecinos que si fueron generados durante el *render*. En este caso estamos saltando un rayo por iteración el cual es casi imperceptible y por las propiedades de la imagen resultado al interpolar los valores resultados vecinos a este se obtiene un color muy cercano al que se obtendría si se trazase el rayo, (ver Fig. 37).

Figura 37: En la imagen final lanzamos rayos no consecutivos para luego interpolar los valores obtenidos y crear la imagen completa.

3.6 Importance Driven Volume Rendering

La técnica que se escoge para esta implementación consiste en tener una matriz del tamaño del volumen que contiene los valores de importancia previamente calculados de cada *voxel*. Teniendo esto lo que hacemos es premultiplicar la opacidad del *voxel* por este valor y así obtenemos al momento de trazar el rayo, una acumulación menor de estos *voxels* mientras que los que poseen mayor importancia son resaltados incrementando su opacidad para que sean más visibles. Esto nos permite resaltar las áreas importantes sin sacrificar mucho rendimiento (ver Fig. 38).

Esta técnica es conocida como *Average Important Compositing* y fue escogida ya que permite mantener una buena ubicación espacial del contexto en donde se encuentra, facilitando así encontrar y comprender las zonas más importantes que se desean resaltar.

Los valores de importancia que tomamos van desde 0 hasta 255 permitiendo así un mejor control de cómo visualizar las zonas con importancia.

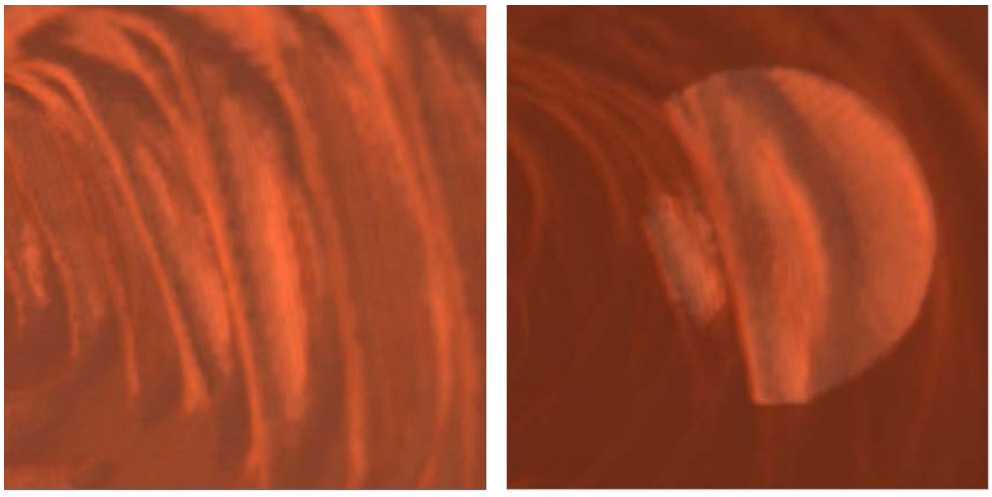

Figura 38: En (a) tenemos el *render* base, y en (b) tenemos una zona circular de mayor importancia, observamos que todos los *voxels* alrededor de la zona de importancia son más transparentes.

Capítulo 4: Pruebas

Las pruebas llevadas a cabo tienen como objetivo realizar un estudio comparativo del rendimiento obtenido con el uso de las técnicas de aceleración y el uso de hilos para el renderizado de la imagen final. Para esto, probamos como funciona este programa desactivando una a una las opciones de aceleración y como esto afecta al usar 1, 2, 4 u 8 hilos. Estas se conocen como pruebas de rendimiento, las cuales nos indicaron si se aceleró el renderizado del volumen o no, y fueron medidas en cuadros por segundo (*Frames per Second*) *FPS*. También se realizaron pruebas de calidad visual en donde se observó si hay mejoras en la calidad de la imagen aplicando diferentes técnicas para ello, se midieron comparando una imagen contra otra. Y por ultimo se realizó una prueba con la memoria usada antes y después de aplicar la iluminación discretizada.

Estas pruebas se realizaron sobre un volumen que tiene una resolución de 512x512x426. En éste creamos un camino de navegación a través de una sección del colon, por el cual nos desplazamos en ambos sentidos para obtener así una experiencia cercana a la de un usuario operando la aplicación. Al final de este camino se colocó una zona de importancia esférica para poder apreciar la técnica.

Entre otras configuraciones tenemos:

- Tamaño de la imagen resultado es de 512x512 *pixels.*
- Tamaño de la imagen reducida para la aceleración de las rotaciones y traslaciones es de 256x256 *pixels.*
- La distancia del plano de cercanía es de 0,1.
- La tolerancia de opacidades es de 0,05; donde 0 es totalmente transparente y 1 es totalmente opaco.
- Las dimensiones del volumen son 512x512x426 *voxels.*
- La tabla de gradientes es un icosaedro subdividido tres niveles tomando sus centros normalizados como entradas de este arreglo.

Pruebas de rendimiento:

Estas pruebas se realizaron de la siguiente forma:

- Desactivamos todas las técnicas de aceleración, para luego proceder a recorrer el colon.
- Activamos todas las técnicas de aceleración, para luego proceder a recorrer el colon.
- Se desactiva una a una las técnicas de aceleración, para luego proceder a recorrer el colon.
- Se hace un *Threshold render* y un *Importance Driven Volume Rendering*.

La primera prueba consiste en activar/desactivar todas las aceleraciones y medir el tiempo que tarda cada cuadro en hacer su *render*. Para esto se tomaron tres sistemas diferentes con diferentes configuraciones y rendimiento, (ver Fig. 39, 40, 41).

Configuración #1: PC Intel Core 2 Duo a 2.13Ghz con 2Gb de memoria RAM. 2 cores

Hilos	Aceleraciones activadas (FPS)	Aceleraciones desactivadas (FPS)
1	7,02 (Min 0,5 / Max 12,8)	0,348 (Min 0,2 / Max 0,79)
2	7,738 (Min 0,74 / Max 13,6)	0,623 (Min 0,3 / Max 1,3)
4	6,519 (Min 0,7 / Max 12,9)	0,635 (Min 0,3 / Max 1,5)
8	4,522 (Min 0,6 / Max 10)	0,624 (Min 0,3 / Max 1,4)

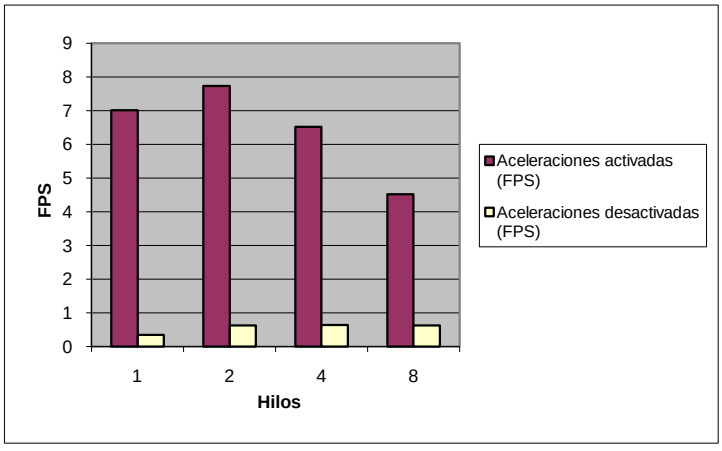

Figura 39: Prueba de rendimiento y comparación entre el *render* sin aceleraciones y el *render* con aceleraciones para la configuración #1.

Configuración #2: PC Intel Pentium4 M a 1.73 Ghz con 1Gb de memoria RAM. 1 core

Hilos	Aceleraciones activadas (FPS)	Aceleraciones desactivadas (FPS)
1	5,916(Min 0,3 / Max 9,1)	0,262(Min 0,2 / Max 0,6)
2	6,014(Min 0,3 / Max 9,1)	0,278(Min 0,2 / Max 0,6)
4	6,001(Min 0,3 / Max 9,1)	0,278(Min 0,2 / Max 0,6)
8	5,956(Min 0,3 / Max 9,1)	0,279(Min 0,2 / Max 0,6)

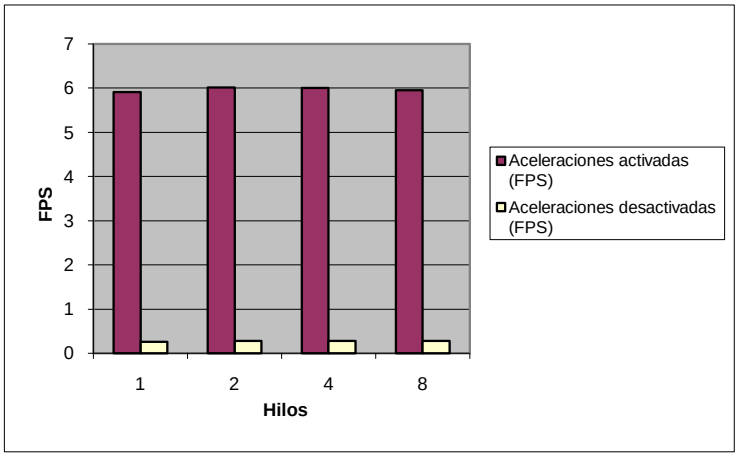

Figura 40: Prueba de rendimiento y comparación entre el *render* sin aceleraciones y el *render* con aceleraciones para la configuración #2.

Configuración #3: PC Intel Core 2 Quad a 2.83 Ghz con 4Gb de RAM. 4 cores

Hilos	Aceleraciones activadas (FPS)	Aceleraciones desactivadas (FPS)
1	8,92(Min 0,6 / Max 13,1)	0,404(Min 0,2 / Max 0,9)
2	11,456(Min 0,9 / Max 16,1)	0,7413(Min 0,4 / Max 1,7)
4	13,686(Min 1,3 / Max 22,3)	1,349(Min 0,6 / Max 3,1)
8	13,918(Min 1,3 / Max 22,3)	1,4302(Min 0,6 / Max 3,1)

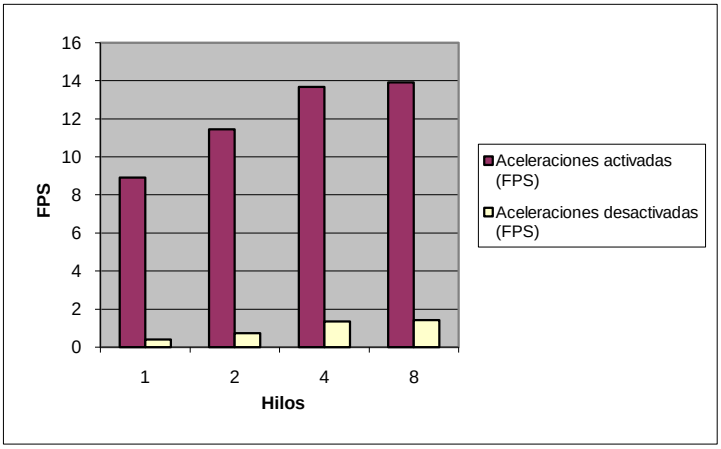

Figura 41: Prueba de rendimiento y comparación entre el *render* sin aceleraciones y el *render* con aceleraciones para la configuración #3.

Para la segunda prueba desactivamos la técnica de aceleración correspondiente y medimos su impacto en el tiempo de *render* para los tres sistemas, (ver Fig. 42, 43, 44).

Configuración #1: PC Intel Core 2 Duo a 2.13Ghz con 2Gb de memoria RAM. 2 cores

Hilos	Coherencia de *píxeles* (FPS)	Salto de espacios vacíos (FPS)	Aceleración de rotaciones (FPS)	Iluminación (FPS)	Iluminación especial (FPS)
1	6,119 (Min 0,3 / Max 10,8)	6,308 (Min 0,4 / Max 12,8)	0,939 (Min 0,7 / Max 1,8)	7,232 (Min 0,8 / Max 10,9)	8,719 (Min 0,7 / Max 10,8)
2	6,88 (Min 0,5 / Max 12,9)	6,9 (Min 0,9 / Max 13,8)	1,639 (Min 0,8 / Max 2,8)	7,771 (Min 0,5 / Max 14,8)	11,286 (Min 0,5 / Max 12,8)
4	5,92 (Min 0,4 / Max 11,2)	5,991 (Min 0,5 / Max 10,8)	1,597 (Min 0,5 / Max 2,5)	6,7 (Min 0,5 / Max 12,9)	13,391 (Min 0,5 / Max 11,5)
8	4,223 (Min 0,5 / Max 10,8)	4,239 (Min 0,5 / Max 10,1)	1,484 (Min 0,5 / Max 2,1)	4,651 (Min 0,4 / Max 11,8)	13,382 (Min 0,5 / Max 10,5)

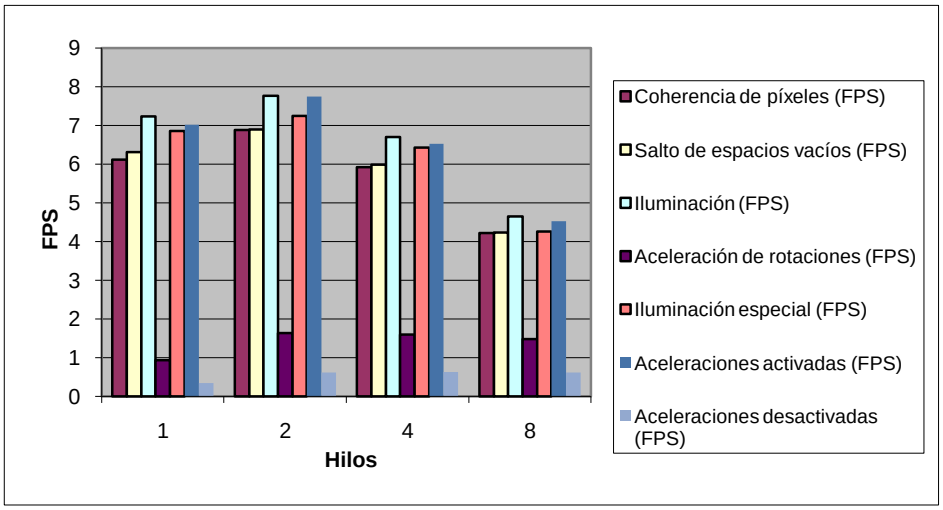

Figura 42: Prueba de rendimiento y comparación entre las diferentes técnicas de aceleración para la configuración #1.

Configuración #2: PC Intel Pentium4 M a 1.73 Ghz con 1Gb de memoria RAM. 1 core

Hilos	Coherencia de *píxeles* (FPS)	Salto de espacios vacíos (FPS)	Aceleración de rotaciones (FPS)	Iluminación (FPS)	Iluminación especial (FPS)
1	4,598(Min 0,3 / Max 7,1)	4,676(Min 0,3 / Max 9,1)	0,727(Min 0,3 / Max 1,3)	5,898(Min 0,3 / Max 9,2)	5,712(Min 0,2 / Max 8,6)
2	4,666(Min 0,3 / Max 7,1)	4,81(Min 0,3 / Max 9,1)	0,775(Min 0,3 / Max 1,3)	6,202(Min 0,3 / Max 9,2)	6,003(Min 0,2 / Max 8,6)
4	4,732(Min 0,3 / Max 7,1)	4,876(Min 0,3 / Max 9,1)	0,769(Min 0,3 / Max 1,3)	6,24(Min 0,3 / Max 9,2)	5,855(Min 0,2 / Max 8,6)
8	4,726(Min 0,3 / Max 7,1)	4,852(Min 0,3 / Max 9,1)	0,771(Min 0,3 / Max 1,3)	6,123(Min 0,3 / Max 9,2)	5,84(Min 0,2 / Max 8,6)

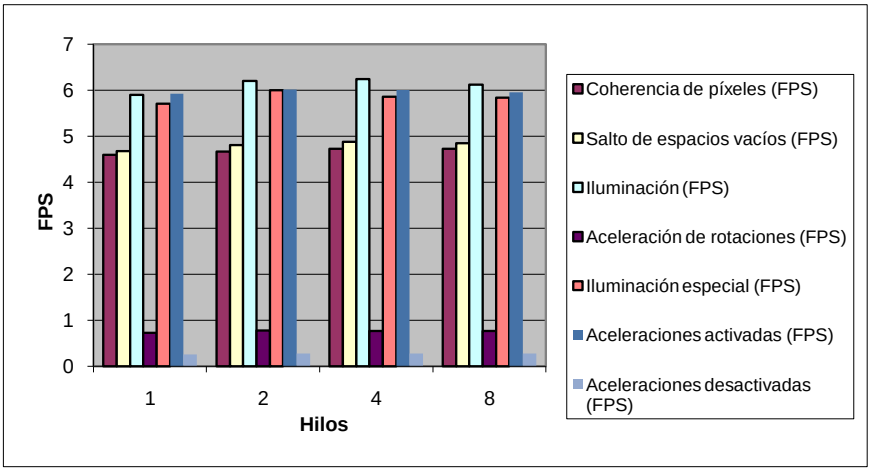

Figura 43: Prueba de rendimiento y comparación entre las diferentes técnicas de aceleración para la configuración #2.

Configuración #3: PC Intel Core 2 Quad a 2.83 Ghz con 4Gb de RAM. 4 cores

Hilos	Coherencia de *píxeles* (FPS)	Salto de espacios vacíos (FPS)	Aceleración de rotaciones (FPS)	Iluminación (FPS)	Iluminación especial (FPS)
1	6,902(Min 0,4 / Max 11,1)	7,157(Min 0,4 / Max 12,3)	1,124(Min 0,6 / Max 2,1)	9,182(Min 0,6 / Max 13,1)	8,719(Min 0,6 / Max 12,1)
2	9,693(Min 0,7 / Max 16,1)	9,757(Min 0,7 / Max 16,1)	1,993(Min 0,8 / Max 3,5)	11,491(Min 0,9 / Max 17)	11,286(Min 0,9 / Max 16,2)
4	12,298(Min 1 / Max 16,2)	12,274(Min 0,9 / Max 22,1)	3,368(Min 1,2 / Max 6,5)	13,791(Min 1,1 / Max 23,2)	13,391(Min 0,9 / Max 22,1)
8	12,339(Min 1 / Max 16,1)	12,473(Min 1,1 / Max 22,3)	3,501(Min 1,1 / Max 5,9)	13,995(Min 0,9 / Max 23,1)	13,382(Min 0,8 / Max 21,1)

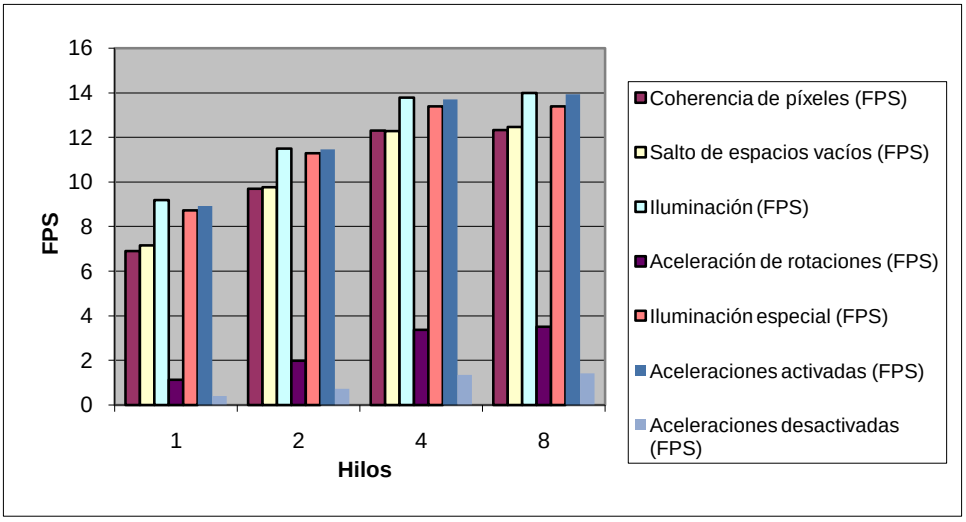

Figura 44: Prueba de rendimiento y comparación entre las diferentes técnicas de aceleración para la configuración #3.

Como era de esperarse, los valores de la configuración #2 son muy cercanos ya que este equipo sólo posee un procesador y no logramos ningún beneficio al usar hilos, únicamente tenemos una mejora dada por las aceleraciones.

Para la configuración #1 podemos ver que los valores permanecen muy cercanos excepto al remover la aceleración de rotaciones, donde se puede apreciar que al usar la cantidad exacta de hilos igual a la cantidad de núcleos del procesador los cuadros por segundo casi se duplican. Esto indica que la mejora sólo es perceptible en imágenes más pequeñas, debido a que esta técnica se enfoca en ese aspecto. Ella utiliza una resolución menor mientras se esta moviendo la cámara, y cuando se detiene realiza el *render* con la resolución completa.

Por su parte, en la configuración #3 si logramos apreciar una mejora general en todas las técnicas a medida que nos acercamos a la cantidad de núcleos del procesador, el cual en muchos casos casi llega a duplicar la cantidad de cuadros por segundo que observamos usando un solo hilo.

Para la tercera prueba realizamos los *render* especiales en las distintas configuraciones (ver Fig. 45, 46, 47).

Configuración #1: PC Intel Core 2 Duo a 2.13Ghz con 2Gb de memoria RAM. 2 cores

Hilos	*Threshold Render* (FPS)	*Importance Volume Rendering* (FPS)
1	6,677(Min 0,7 / Max 8,1)	2,395(Min 0,2 / Max 4,1)
2	7,268(Min 0,9 / Max 9,2)	3,38(Min 0,3 / Max 5,2)
4	6,492(Min 0,8/ Max 8,5)	3,242(Min 0,3 / Max 5,1)
8	4,324(Min 0,8/ Max 8,2)	2,823(Min 0,3 / Max 4,9)

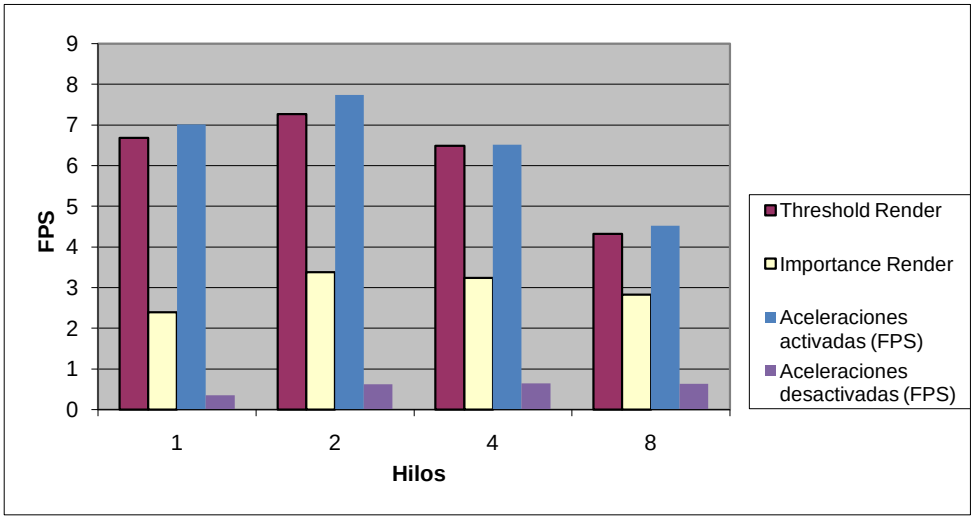

Figura 45: Prueba de rendimiento y comparación entre los *render* especiales para la configuración #1.

Configuración #2: PC Intel Pentium4 M a 1.73 Ghz con 1Gb de memoria RAM. 1 core

Hilos	*Threshold Render* (FPS)	*Importance Volume Rendering* (FPS)
1	5,609(Min 0,4 / Max 8)	1,854(Min 0,2 / Max 3,1)
2	6,013(Min 0,4 / Max 8)	1,904(Min 0,2 / Max 3,1)
4	5,751(Min 0,4 / Max 8)	1,904(Min 0,2 / Max 3,1)
8	5,778(Min 0,4 / Max 8)	1,893(Min 0,2 / Max 3,1)

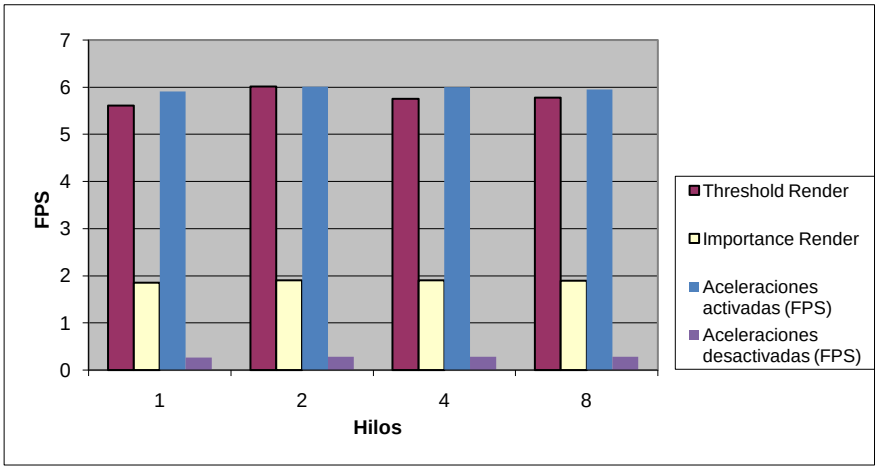

Figura 46: Prueba de rendimiento y comparación entre los *render* especiales para la configuración #2.

Configuración #3: PC Intel Core 2 Quad a 2.83 Ghz con 4Gb de RAM. 4 cores

Hilos	*Threshold Render* (FPS)	*Importance Volume Rendering* (FPS)
1	8,568(Min 0,6 / Max 12,1)	2,734(Min 0,2 / Max 4,5)
2	11,221(Min 0,9 / Max 16)	4,542(Min 0,3 / Max 7)
4	13,369(Min 1,2 / Max 21,1)	6,956(Min 0,5 / Max 9,3)
8	13,514(Min 1,3 / Max 21,2)	7,019(Min 0,5 / Max 9,1)

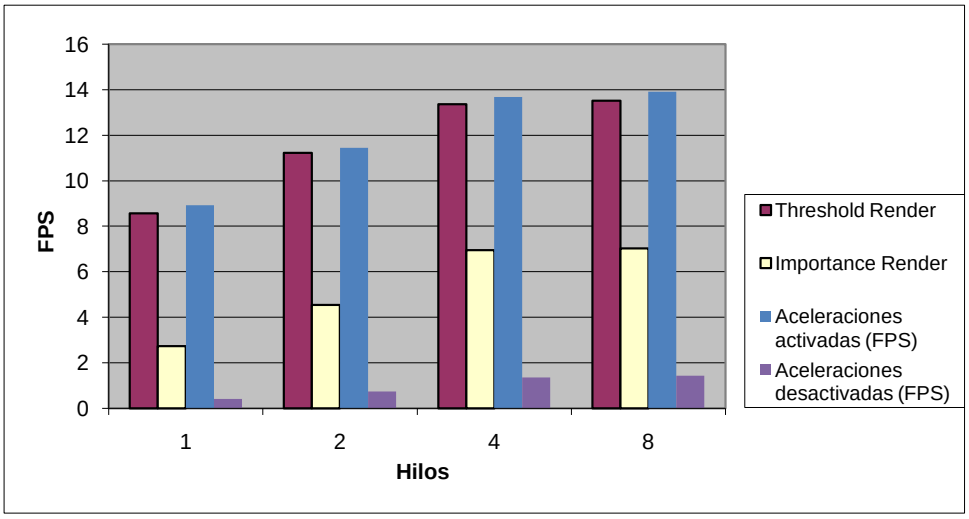

Figura 47: Prueba de rendimiento y comparación entre los *render* especiales para la configuración #3.

Para estas pruebas observamos resultados similares a los anteriores. Al usar el *Importante Driven Volume Rendering* notamos que se reduce la velocidad considerablemente, pero esto es debido a que en el volumen usado sólo poseía una zona esférica de importancia; mientras que todos los demás *voxels* eran tratados como si fuesen más transparentes de lo normal, por lo cual el rayo recorre más camino del que debería normalmente.

Pruebas de calidad de imagen:

Para realizar estas pruebas compararemos dos imágenes, una con el *render* sólo aplicándole iluminación y otra en las mismas condiciones pero con la técnica activada. A estas imágenes resultado les aplicaremos un aumento para poder ver más de cerca el detalle.

- Coherencia del espacio de píxeles*:* (ver Fig. 48)

(a) (b) (c)

Figura 48: En (a) tenemos el punto de referencia donde se está haciendo zoom de 7 veces la imagen original. En (b) está la imagen original solo con iluminación y en (c) está la imagen aumentada con la técnica de coherencia de *píxeles*.

Como podemos observar, no existe mucha diferencia entre estas imágenes, lo principal que notamos es que al aplicar la técnica los bordes se ven un poco suavizados debido a la interpolación de los *píxeles* usada.

- *Empty Space Skipping:* (ver Fig. 49)

(a) (b) (c)

Figura 49: En (a) tenemos el punto de referencia donde se esta haciendo zoom de 7 veces la imagen original. En (b) está la imagen original solo con iluminación y en (c) está la imagen aumentada con la técnica de *Empty Space Skipping*.

Aquí podemos apreciar una leve diferencia que radica en el error ocasionado generalmente en los bordes que separan a zonas de diferentes alturas, ya que con esta técnica podemos saltar *voxels* de más y atravesar ciertas regiones del volumen. Aunque este error suele ser muy pequeño y casi imperceptible en tamaño normal.

- Aceleración de las rotaciones: (ver Fig. 50)

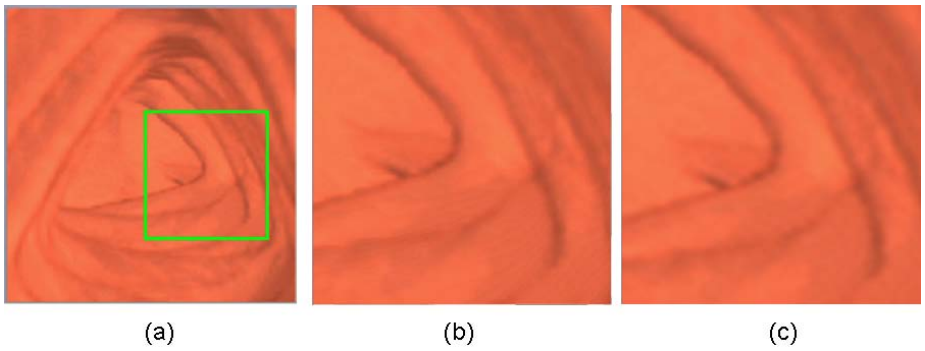

Figura 50: En (a) tenemos el punto de referencia donde se esta haciendo zoom de 3 veces la imagen original. En (b) está la imagen original solo con iluminación y en (c) está la imagen aumentada con la técnica de Aceleración de rotaciones.

Podemos observar que la imagen que ha sido tratada con esta técnica es más borrosa que la original, esto se debe a que el *render* de ésta se realiza a la mitad del tamaño original, esta imagen luego es estirada aplicando un filtro bilineal lo que ocasiona que la imagen final se vea más suave que la original.

- Iluminación especial: (ver Fig. 51)

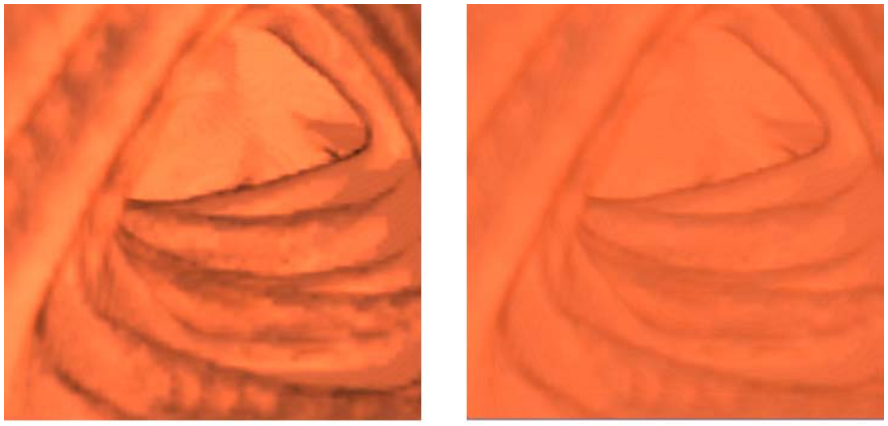

Figura 51: (a) representa al volumen con la iluminación tradicional y (b) muestra la iluminación especial que combina parte del volumen iluminado con el que se obtiene sin iluminar.

Como podemos observar, la imagen que solo está iluminada presenta artefactos como sombras que no deberían existir debido a lo sensible que es la función de gradientes, en donde es muy difícil diferenciar que *voxels* pertenecen al borde del colon y cuales no, esto hace que tomemos un valor para las normales erróneo. Al aplicar la iluminación especial vemos como la imagen mejora mucho y presenta menos artefactos, esto se logra al combinar el *render* iluminado con el *render* sin iluminación.

Pruebas de memoria:

Volumen:

Volumen almacenado en memoria: (Tamaño del volumen + *pixels* de relleno) x (tipo de dato (*unsigned short*)).

- (512x512x(426+86)) x (2) = 268.435.456 *bytes.*

Aproximado: 250 *megabytes*.

Usando *bricks* de 32x32x32 *voxels* con 1 línea de *voxels* de redundancia en cada dirección: (Tamaño de un *brick*) x (cantidad de *bricks* en el volumen) x (tipo de dato (*unsigned short*)).

- ((32+2)x(32+2)x(32+2)) x ((512/32)x(512/32)x(512/32)) x (2) = 39.304 x 4.096 x 2 = 321.978.368 *bytes*.

Aproximado: 300 *megabytes*

Gradientes:

Almacenar todos los gradientes en memoria: (Tamaño del volumen) x (vector [3] gradiente de flotantes).

- (512x512x426) x (3x4) = 1.340.080.128 *bytes*.

Aproximado: 1,3 *gigabytes*.

Usando las *look-up tables*: (Arreglo de índices) + (*look-up table* → vector [3] gradiente de flotantes).

- (512x512x426) + ((4^3x20) x (3x4)) = 111.673.344 + 15.360 = 111.688.704 *bytes*

Aproximado: 100 *megabytes*.

Capítulo 5: Conclusión

Se diseño el visualizador y las aceleraciones mejoraron el tiempo de respuesta. Además de haberse realizado en una librería dinámica que podrá ser usada en un sistema más grande posteriormente.

Muchas de estas mejoras se lograron sin afectar gravemente la imagen final siendo la que más deteriora la imagen la técnica de aceleración de las rotaciones, pero debido a que esta solo se usa durante el movimiento de la cámara, podemos reducir su uso para evitar pasar por alto información relevante.

El uso de los hilos ayuda a acelerar el despliegue de la imagen pero sólo se ve reflejado en imágenes de gran escala, lo que permite realizar *renders* de tamaños mayores a medida que tenemos más poder de procesamiento y más núcleos de procesador. Aunque su abuso también disminuye el rendimiento del sistema, por lo que se aprecia en las pruebas, se debe usar la misma cantidad de hilos que núcleos del procesador haya disponibles.

La iluminación es un punto muy importante en este tipo de *render* ya que es necesario que el usuario tenga una imagen lo más cercana a la realidad posible para que le sea más fácil la identificación de cualquier anomalía.

Otro punto relevante es la memoria física, ya que esta es la razón por la que nos vimos obligados a usar una iluminación discretizada debido a que los volúmenes médicos del colon suelen ser de un tamaño considerable y almacenar toda la información en memoria puede presentar problemas. Este es un problema común con todas las técnicas de *volume rendering* pero se acentúa más al usar datos médicos que requieren de más precisión y donde no podemos desechar datos que en otros casos podrían considerarse innecesarios.

Si bien se logró acelerar el *render* para hacerlo más cercano al *render* en tiempo real, algunas de estas mejoras deterioran la imagen final y debido a la naturaleza de este trabajo donde se requiere de mucha precisión visual para la buena detección de posibles tumores, es

necesario seguir mejorando estas técnicas teniendo en cuenta que no debemos abusar del uso de las mismas.

Trabajos Futuros

Entre las mejoras que se pueden realizar sobre el visualizador tenemos:

- Uso de mejores estructuras de datos que permitan acelerar la búsqueda de los datos.
- Mejorar la imagen final mejorando la iluminación actual. Una mejora podría consistir en la pre-búsqueda de los bordes del colon y la creación de una superficie dado estos valores, de ahí se podrían obtener unas normales más adecuadas y nuestro método de discretización sería más eficiente.
- Uso de técnicas de aceleración por hardware. Hoy en día es muy común conseguir computadoras que posean tarjetas gráficas que permitan la aceleración del *render* por hardware, y en caso de no ser así, los algoritmos usados en la aceleración por software son altamente eficientes.
- Mejor uso de los hilos para aprovechar cada procesador al máximo en todo momento.

Este sistema debería estar integrado a otro sistema que provea los datos necesarios para este como lo son la línea central del colon para la navegación automática y los valores de importancia para poder resaltar las zonas adecuadamente.

Otra mejora podría ser la inclusión de un gestor de memoria que nos permita cargar todos los valores del volumen con sus respectivos valores de iluminación para así no tener que discretizar los mismos.

Referencias

[1] Akeley, K. (1993). “Reality Engine Graphics”, *Computer Graphics*, T. 27, pp. 109-116.

[2] Anuario. Portal Ministerio del Poder Popular para la Salud: http://www.mpps.gob.ve/. pp. 225.

[3] Blinn, J. (1994). “Jim Blinn's Corner: Compositing, part I: Theory”. *IEEE Computer Graphics and Applications*, pp. 83-87.

[4] Boost C++ Libraries: http://www.boost.org/

[5] Brady, M. L.; Jung, K. K.; Nguyen, H. T. y Nguyen, T. (1998). “Interactive Volume Navigation”. *IEEE Transactions on Visualization and Computer Graphics*. pp. 243-256.

[6] Brady, M. L.; Jung, K.; Nguyen, H. T. y Nguyen, T. (1997). “Two-Phase Perspective Ray Casting For Interactive Volume Navigation”. *IEEE Visualization 97, Conference Proceedings*, pp. 183–189.

[7] Cabral, B.; Cam, N. y Foran, J. (1994). “Accelerated Volume Rendering And Tomographic Reconstruction Using Texture Mapping Hardware”. *In Proc. Symposium on Volume Visualization*, pp. 91-98.

[8] Cameron, G. G. y Undril, P. E. (1992). “Rendering Volumetric Medical Image Data On A SIMD-Arquitecture Computer”. *In Proc. Eurographics Workshop on Rendering. Briston, UK*. pp. 135-145.

[9] Carmona, Rhadamés (2000). “Shear-warp: una implementación eficiente”. *In Proc. XXVI Conferencia Latinoamericana de Informática*.

[10] Coto, Ernesto (2008). “Estrategias avanzadas de segmentación y visualización para colonoscopía virtual”. Tesis doctoral. Universidad Central de Venezuela.

[11] Danskin, J. y Hanrahan, P. (1992). “Fast Algorithms For Volume Ray Tracing”. *Workshop on Volume Visualization, Conference Proceedings, New York, ACM Press*. pp. 91–98.

[12] Ellsworth, D.; Chiang, L. y Shen, H. W. (2000). “Accelerating Time-Varying Hardware Volume Rendering Using TSP Trees And Color-Based Error Metrics”. *In Proc. 2000 Symposium on Volume Visualization. ACM SIGGRAPH*. pp 119-128.

[13] Engels, K.; Kraus, M. y Ertl, T. (2001). “High Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading”. *Siggraph/Eurographics Workshop on Graphics Hardware*. pp. 9-16

[14] Foley, J. D.; Van Dam, A.; Feiner, S.; Hughes, J. y Phillips, R. (1994). “Introduction To Computer Graphics”. *Addison-Wesley Publishing Company, first edition*.

[15] Funkhouser, T. y Sequin, C.. (1993). “Adaptive Display Algorithms For Interactive Frame Rate During Visualization Of Virtual Environment”. *In Proc. SIGGRAPH 93, ACM SIGGRAPH*. pp. 247–254.

[16] García, Orlando (2003). “Visualización de volúmenes con modelos de iluminación utilizando hardware gráfico”. *Tesis de pregrado. Universidad Central de Venezuela. Facultad de Ciencias*.

[17] Gobbetti, E. y Bouvier, E. (1999). “Time-Critical Multiresolution Scene Rendering”. *In Proc. 1999 Symposium on Volume Visualization, ACM SIGGRAPH*. pp. 123–130.

[18] Gordon, Dan y Shuhong, Chen, (s/f). “Front-To-Back Display Of BSP Trees”. *IEEE Computer Graphics and Applications, vol. 11, n°. 5*. pp. 79-85.

[19] Greene, N.; Kass, M. y Miller, G. (1993). “Hierarchical Z-Buffer Visibility”. *Computer Graphics, Siggraph*. pp. 231-238.

[20] Grzeszczuk, R.; Henn, C. y Yagel, R. (1998). “Advanced Geometric Techniques for Ray Casting Volumes”. *In Proc. Siggraph 98*.

[21] Guan, S. y R. Lipes. (1994). “Innovative Volume Rendering Using 3D Texture Mapping”. *In Proc. SPIE Medical Imaging-Image Capture, Formatting And Display, vol. 2104*. pp. 382-392.

[22] Gudmundsson, B. y Randen, M. (1990). “Incremental Generation Of Projections Of CT Volumes”. *The First Conference on Visualization in Biomedical Computing, Conference Proceedings, Atlanta*. pp. 27-34.

[23] Haidacher, Martín (2007). “Importance-Driven Rendering In Interventional Imaging”. *Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria*.

[24] He, Taosong y Kaufman, Arie. (1996). “Fast Stereo Volume Rendering”. *In Proc. IEEE Visualization Conference '96*. pp. 49-56.

[25] Kajiya, J. T. (1982). “Ray Tracing Parametric Patches”. *In Proc. SIGGRAPH: 9th Annual Conference on Computer Graphics and Interactive Techniques*. pp. 245–254.

[26] Klein, F. y Kübler, O. (1985). “A Prebuffer Algorithm For Instant Display Of Volume Data”. *In Proc. SPIE (Architectures and Algorithms for Digital Image Processing), vol. 596.* pp. 54-58.

[27] Knoll, A. (2006). “A Short Survey Of Octree Volume Rendering Techniques”. *In Proc. Primer IRTG Workshop.*

[28] LaMar, E.; Hamann, B. y Joy K. (1990). “Multiresolution Techniques For Interactive Texture-Based Volume Visualization”. *In Proc. Visualization ’99 IEEE Computer Society Press*. pp. 355 - 361.

[29] LaMar, Eric; Hamann, Bernd y Joy, Kenneth (1999). “Multiresolution Techniques For Interactive Texture-Based Volume Visualization”. *In Proc. IEEE Visualization '99.* pp. 355-362.

[30] Lacroute, Phillippe G. (1995). “Fast Volume Rendering Using Shear-Warp Factorization Of The Viewing Transformation”. *Reporte técnico CSL-TR-95-678. Universidad de Stanford, Computer Systems Laboratory*. pp. 451-458.

[31] Laur, D. y Hanrahan, P. (1991). “Hierarchical Splatting: A Progressive Refinement Algorithm For Volume Rendering”. *Computer Graphics*. pp. 285–288.

[32] Levoy, M. (1988). “Display Of Surfaces From Volume Data”. *IEEE Computer Graphics and Applications, vol. 8, no. 5*. pp. 29-37.

[33] Levoy, M. (1990). “Volume Rendering By Adaptive Refinement”. *The Visual Computer*. pp. 2-7.

[34] Li, X. y Shen, H. W. (2001). “Adaptive Volume Rendering\Using Fuzzy Logic Control”. *In Proc. Joint Eurographics-IEEE TCVG Symposium on Visualization. Springer-Verlag*.

[35] Lorang, T. (2001). “Organisation And Visualisation Of Medical Images In Radiotherapy”. *PhD thesis, Vienna University of Technology. Faculty of Technical Sciences. Institute of Medical Computer Sciences*.

[36] Max, N. (1995). “Optical Models For Direct Volume Rendering”. *IEEE Transactions on Visualization and Computer Graphics. vol. 1, no. 2*. pp. 99-108.

[37] Meagher, D. J. (1982). “Efficient Synthetic Image Generation Of Arbitrary 3D Objects”. *IEEE conference on Pattern Recognition and Image Procesing*. pp. 473-478.

[38] Meissner, M.; Kanus U. y Strasser, W. (1998). “Vizard II, A PCI-Card For Real-Time Volume Rendering”.

[39] OpenGL, http://www.opengl.org/

[40] Parejo, J. C.; Cordero, J. M. (2002). “Curvas y superficies para modelado geométrico”. *Editorial Ra-Ma*.

[41] Plate, J.; Tirsana, M.; Froehlich, B. y Carmona, R.(2002). “Octreemizer: A Hierarchical Approach For Interactive Roaming Through Very Large Volumes”. *In Proc. Joint Eurographics - IEEE TCVG Symposium on Visualization*. pp. 53-ff

[42] Porter, T. y Duff, T. (1984). “Compositing Digital Images”. *Computer Graphics, vol. 18, no. 3*. pp. 253-259.

[43] Reddy, M.; Watson, A. B.; Walker, N. y Hodges, F. L. (1997). “Managing Level Of Detail In Virtual Environments: A Perceptual Framework”. *In Presence: Teleoperators and Virtual Environments*. pp. 658–666.

[44] Rohlf, J. y Helman, J. (1994). “Iris Performer: A High Performance Multiprocessing Toolkit For Real-Time 3D Graphics”. *In Proc. SIGGRAPH 94, ACM SIGGRAPH*. pp. 381–395.

[45] Schulze, P.; Niemeier, L. y Lang, U. (2001). “The Perspective Shear-Warp Algorithm In A Virtual Environment”. *In Proc. IEEE Visualization 2001 Conference*. pp. 207-214.

[46] Shahidi, R. (1996). “Surface Rendering Versus Volume Rendering In Medical Imaging: Techniques And Applications”. *In Proc. Visualization*. pp. 439-440.

[47] Sharghi, M. Ricketts, I. W. (2001). “A Novel Method For Accelerating The Visualisation Process Used In Virtual Colonoscopy”. *IEEE Computer Society, 5th International Conference on Information Visualisation*. pp. 167-172.

[48] Sharghi, Mehran y W. Ricketts, Ian (2002). “Interactive Visualization Of A Virtual Colonoscopy By Combining Ray Casting With An Acceleration Corridor”. *6th Annual meeting on Medical Image Understanding and Analysis, UK*.

[49] Vining, D. J.; Gelfand, D. W.; Bechold, R. (1994). “Technical Feasibility Of Colon Imaging With Helical CT And Virtual Reality”, *In Proc. Annual Meeting of the American Roentgen Ray Society*. pp. 14.

[50] Viola, Iván; Kanitsar, Armin Y Gröller, Meister Eduard (2004). “Importance-Driven Volume Rendering”. *In Proc. IEEE Visualization 2004*. pp. 139-145.

[51] Wan, M.; Tang, Q.; Kaufman, A.; Liang, Z. et al. (1999). “Volume Rendering Based Interactive Navigation Within The Human Colon”. *In Proc. IEEE Visualization*. pp. 397-400.

[52] Weiler, M.; Westermann, R.; Hansen, C.; Zimmerman, K. y Ertl, T. (2000). “Level-Of-Detail Volume Rendering Via 3D Textures”. *In Proc. 2000 Symposium on Volume Visualization. ACM SIGGRAPH*. pp. 7–13.

[53] Westermann, R. y Ertl, T. (1998). “Efficiently Using Graphics Hardware In Volume Rendering Applications”. *In Proc. Siggraph. 32(4)*. pp.169-179.

[54] Wilson, O; Van Gelder, A. y Wilhems, J. (1994). “Direct Volume Rendering Via 3D Textures”. *Reporte Técnico UCSC-CRL-94-19, University of California*.

[55] World Health Organization, http://www.who.int/

[56] Yagel, R. y Kaufman, A. (1992). “Template-Based Volume Viewing”. *Computer Graphics Forum*. pp. 154-167.

[57] Yagel, R. y Shi, Z. (1993). “Accelerating Volume Animation By Space-Leaping”. *IEEE Visualization, Conference Proceedings, San José, CA, IEEE Computer Society Press*. pp. 62–69.

[58] Yaroslavsky, L. (2002). “Fast Signal Sinc-Sinterpolation And Its Applications In Signal And Image Processing”. *In Proc. SPIE, vol. 4667*.

[59] You, S.; Hong, L. y Junyaprasert, K. (1997). “Interactive Volume Rendering For Virtual Colonoscopy”, *In Proc. Visualisation*. pp. 433-436.

[60] Zuiderveld, K.; Koning, A. H. J. y Viergever, M. A. (1992). “Acceleration Of Ray Casting Using 3D Distance Transform”, *In Proc. Visualization in Biomedical Computing*. pp. 324-335.

[61] Ávila, R.; Sobierajaski, L. M. y Kaufman, A. (1992). “Towards A Comprehensive Volume Visualization System”, *In Proc. Visualization*. pp. 13-20.