Universidad Nacional Autónoma de México

Posgrado en Ciencias e Ingeniería de la Computación

"Segmentación Difusa como Generador de Funciones de Transferencia para Visualización Directa de Volúmenes"

T E S I S

que para obtener el grado de:

Maestra en Ciencias

(Computación)

presenta:

Cinthya Lizeth Ceja Mendoza

director de tesis:

Dr. Edgar Garduño Ángeles

Ciudad de México, México Junio, 2012

A toda mi familia.

Agradecimientos

Quiero agradecer a mis sinodales los Doctores Boris Escalante, María Elena Martínez, Luis Miguel de la Cruz por su aportación a este trabajo y en especial al Dr. Bruno Motta de Carvalho por haberme recibido en su universidad y haberme dado consejo sobre como resolver varios problemas del trabajo, su ayuda fue de gran importancia para el avance y conclusión de esta tesis.

Le agradezco a mi tutor, el Dr. Edgar Garduño, por haberme invitado y permitido trabajar en su proyecto. En realidad aprecio mucho sus consejos y el tiempo que dedicó a revisar mis avances y aclarar mis dudas, su apoyo y su paciencia permitieron que este trabajo llegara a buen termino.

Gracias al Posgrado en Ciencia e Ingeniería de la Computación de la U. N. A. M por haberme aceptado en su programa de maestría y dado la oportunidad de estudiar en sus instalaciones. Igualmente, gracias a todos los profesores con los que tome algún curso, por transmitirme sus conocimientos. A Diana, Lulú y Amalia por su dedicación, su buen trabajo y por mantenernos siempre al tanto de las cosas que tenemos que realizar como alumnos y darnos nuestros jalones de orejas cuando lo necesitamos. Así como al Departamento de Ciencias de la Computación del I. I. M. A. S, por haberme permitido hacer uso de sus instalaciones.

Agradezco al CONACyT por haberme otorgado la beca con CVU/Becario 331081/ 231743 para realizar mis estudios de maestría y este trabajo, así como por el apoyo en conjunto de esta institución con el Posgrado en Ciencias e Ingeniería de la Computación a través de una beca mixta para la realización de una estancia de investigación en la U. F. R. N. de Natal, Brasil, ya que esta estancia fue de gran importancia para el avance de mi tesis.

Gracias a mis papás, sin ustedes no habría llegado hasta aquí y a ustedes debo buena parte de la persona que soy, las palabras no bastan para agradecerles a ustedes y a mis tíos por todo su apoyo, comprensión y todo lo que me han dado. A mis hermanas, Ilse y Denise, por las horas de risa y enojos ocasionales, sé que siempre cuento con ustedes. A Héctor Alonso, por soportarme en mis ratos de estrés, motivarme y siempre darme ánimo. A todos mis compañeros y amigos del posgrado, en especial a César, Sergio, Verena, Jorge, Montse, Lisset, Pedro y Caro por su amistad, tantas pláticas y tanta ayuda.

Resumen

La visualización científica es un campo importante dentro de las ciencias de la computación puesto que permite la interacción, exploración y análisis de grandes cantidades y distintos tipos de datos. La visualización directa de volúmenes ha probado ser un método eficiente para la visualización de datos tridimensionales complejos (tal como lo son los datos obtenidos de la Tomografía Computarizada). A diferencia de la visualización por superficies, que usualmente genera una aproximación geométrica de la superficie de los datos, la visualización directa de volúmenes genera una imagen usando directamente los datos de volumen (*vóxeles*) por medio de una asignación de propiedades ópticas. Esta asignación es llevada a cabo a través de las *funciones de transferencia*, sin embargo, la tarea de diseñar una función apropiada es a menudo difícil y tardada.

El problema de diseño de funciones de transferencia puede considerarse similar al de la segmentación de imágenes, puesto que se trata de asignar propiedades ópticas a vóxeles que comparten ciertas características para, de esta forma, identificar y resaltar ciertos objetos.

En este trabajo presentamos una propuesta para el diseño de funciones de transferencia usando como base un algoritmo semi-automático de segmentación basado en principios de lógica difusa. Este método de segmentación no sólo decide si un vóxel pertenece o no a cierto objeto, sino que provee información sobre la calidad de su pertenencia asignándole un grado; de forma que un vóxel puede pertenecer a más de un objeto.

De esta manera, pretendemos reducir el tiempo usado para crear una función de transferencia inicial y de facilitar al usuario la posibilidad de modificarla.

Índice general

Agradecimientos III

Resumen V

1. Introducción 1

2. Visualización de Imágenes 3D 7

2.1. Discretización de imágenes 7

2.2. Visualización 9

2.2.1. Visualización por Superficies (*Surface Rendering*) 10

2.2.2. Visualización Directa de Volúmenes (*Volume Rendering*) . . . 13

2.2.2.1. Principio Básico 14

2.2.3. Composición 16

2.3. *Rendering Pipeline* 19

2.3.1. Iluminación 20

2.3.1.1. Modelo de Iluminación de Phong 21

2.3.2. Mejoras al principio básico de Visualización Directa de Volúmenes 22

2.3.2.1. Mapeo de Texturas 23

2.3.2.2. Funciones de Transferencia 24

3. Segmentación Difusa 29

3.1. Métodos de Segmentación 29

3.1.1. Técnicas basadas en Frontera 30

3.1.2. Técnicas basadas en Regiones 31

3.1.2.1. Transformada *Watershed* . . . 31
3.1.2.2. Conectividad Difusa . . . 33
3.2. Principios de la Segmentación Difusa . . . 33
3.2.1. Afinidad Difusa . . . 37
3.2.2. Medida de textura . . . 39
3.2.3. Algoritmo de segmentación difusa . . . 40
3.3. Diseño de Funciones de Transferencia . . . 42

4. Experimentos y Resultados **45**
4.1. Implementación . . . 45
4.1.1. *Voreen* . . . 47
4.1.1.1. *Shaders* . . . 48
4.1.1.2. Implementación de *raycasting* . . . 49
4.1.1.3. Modificaciones . . . 52
4.2. Resultados . . . 55
4.2.1. Muela . . . 56
4.2.2. Elipses . . . 59
4.2.3. Motor . . . 60
4.2.4. Langosta . . . 61
4.2.5. Rodilla . . . 61
4.2.6. Tórax . . . 64
4.2.7. Corazón . . . 66
4.2.8. Cabeza . . . 68
4.3. Validación . . . 71

5. Conclusiones **79**
5.1. Trabajo Futuro . . . 80

Bibliografía **83**

Índice de figuras

1.1. Interfaces para el diseño de funciones de transferencia de algunos programas para visualización directa de volúmenes. Arriba *Amira* ®, abajo *Simian.* 5

2.1. Datos volumétricos de un diente presentados por planos para algún valor fijo en los ejes (a) x (ancho), (b) y (altura) y (c) (d) z (profundidad) . 10

2.2. Imagen obtenida por medio del método *Marching Cubes* aplicado sobre los datos de una cabeza humana (tomada de [1]). 12

2.3. Imagen de un conjunto de datos de una cabeza humana, obtenida por medio del método estándar de visualización directa de volúmenes, implementado en el programa *Voreen* [2] 14

2.4. Integral de línea usada por el modelo de iluminación en *volume rendering*, donde I_i es la intensidad de luz de entrada, mientras que I_i' es la intensidad de salida. La intensidad de salida es calculada con base en las propiedades asignadas a cada vóxel de $v(\mathbf{k})$ que es atravesado por el rayo de luz. 16

2.5. Corte cilíndrico hipotético del conjunto de datos que sirve para el modelo de partículas en visualización directa de volúmenes. 18

2.6. Utilización de mapeo de texturas para generar imágenes por medio de visualización directa de volúmenes. 24

2.7. Visualización de datos volumétricos de un diente cuyos valores fueron asignados por distintas funciones de transferencia: (a) usando los valores de intensidad originales, (b) usando máxima intensidad a través de la trayectoria de integración del rayo, (c) usando suma a lo largo de la trayectoria de integración del rayo y (d) usando búsqueda manual de la función de transferencia (Imágenes obtenidas con *Amira*®). 26

3.1. Representación de una imagen de tamaño 3×3 como una gráfica, donde cada píxel corresponde a un nodo y las aristas representan un tipo de adyacencia entre los nodos (por "cara", en este caso). 35

3.2. El algoritmo de segmentación difusa realiza un mapeo de la gráfica que representa la imagen a una gráfica difusa, donde a cada nodo corresponde el valor $\sigma_0^{\mathbf{c}}$ y cada arista tiene un peso $\psi_{\mathbf{c},\mathbf{d}}$. 36

3.3. Corte del resultado de la aplicación del algoritmo de segmentación difusa sobre datos volumétricos del estudio de una cabeza por tomografía computarizada (CAT), el mismo conjunto utilizado en la Fig. (2.2). Pueden observarse cuatro clases: cerebro (rojo), cráneo (verde), piel (azul) y fondo (gris). Las tonalidades oscuras indican menor grado de pertenencia que las tonalidades claras. 42

4.1. Texturas usadas para calcular los puntos de entrada (a) y salida (b) de los rayos (imagen tomada de [3]). 52

4.2. Interfaz para diseño de funciones de transferencia de *Voreen*. 55

4.3. Visualización de un fragmento de volumen usando (a) sólo información de clase e (b) información de clase y conectividad. Si no se usan los valores de conectividad, no es posible distinguir variaciones internas dentro de cada clase. 56

4.4. Visualización del conjunto de datos por medio de (a) prueba y error [4], (b) espectro de contorno [5], (c) histogramas de volumen [6] y (d) selección de una galería de funciones de transferencia generada automáticamente [7] (Imágenes tomadas de [8]). 57

4.5. Distintas renderizaciones de la mismo conjunto de datos (muela) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra el conjunto con modificaciones ligeras a las funciones, (c) muestra un patrón similar al de la Fig. 4.4(d) y (d) incluye cálculo de gradientes para la iluminación. 58
4.6. Distintas visualizaciones del mismo conjunto de datos (esferas) y funciones de transferencia diferentes. La imagen (a) muestra las funciones de transferencia iniciales, la imagen (b) una modificación de esas mismas funciones y la imagen (c) oculta una de las clases y se realizó incluyendo el cálculo de gradientes para la iluminación. 59
4.7. Distintas visualizaciones del mismo conjunto de datos (motor) y funciones de transferencia diferentes. La imagen (a) muestra una modificación de las funciones de transferencia iniciales y la imagen (b) esas mismas funciones incluyendo el cálculo de gradientes para la iluminación. . . . 60
4.8. Distintas visualizaciones del mismo conjunto de datos (langosta) y funciones de transferencia diferentes. La imagen (a) muestra una modificación de las funciones de transferencia iniciales y la imagen (b) esas mismas funciones incluyendo el cálculo de gradientes para la iluminación. 61
4.9. Visualización del conjunto de datos por medio de (a) prueba y error [4], (b) histogramas de volumen [6] y (c) selección de una galería de funciones de transferencia generada automáticamente [7] (Imágenes tomadas de [8]). 62
4.10. Distintas renderizaciones del mismo conjunto de datos (rodilla) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra el conjunto con modificaciones ligeras a las funciones, (c) muestra un patrón similar a la Fig. 4.9(a) y (d) incluye cálculo de gradientes para la iluminación. 63

4.11. Distintas renderizaciones del mismo conjunto de datos (tórax) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra modificaciones para resaltar las costillas, (c) muestra modificaciones para resaltar pulmones y (d) muestra el patrón (b) incluyendo cálculo de gradientes para la iluminación. 65
4.12. Visualización del conjunto de datos por medio de (a) prueba y error [4], (b) espectro de contorno [5], (c) histogramas de volumen [6] y (d) selección de una galería de funciones de transferencia generada automáticamente [7] (Imágenes tomadas de [8]). 66
4.13. Distintas renderizaciones del mismo conjunto de datos (corazón) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra el conjunto tratando de imitar el patrón de la Fig. 4.12(a) y (c) incluye cálculo de gradientes para la iluminación. 67
4.14. Distintas renderizaciones de la mismo conjunto de datos (cabeza obtenida por tomografía computarizada) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra el conjunto con modificaciones para resaltar cráneo, (c) muestra modificaciones para resaltar cerebro y (d) muestra sólo el cráneo e incluye cálculo de gradientes para la iluminación. 69
4.15. Distintas renderizaciones de la mismo conjunto de datos (cabeza obtenido por resonancia magnética) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra el conjunto con modificaciones para resaltar el cerebro, (c) muestra el cerebro visto desde arriba y (d) usa otra función de transferencia e incluye cálculo de gradientes para la iluminación. 70
4.16. Visualización de los conjuntos de datos sintéticos utilizados para la validación del proyecto, (a) es el volumen generado por Xmipp y (b) el volumen *Shepp-Logan*. 71

4.17. Visualización de los conjuntos de datos sintéticos utilizados para la validación del proyecto, (a) y (b) muestran el volumen Xmipp con diferente orientación, ruido y clases desplegadas, mientras que (c) y (d) muestran el volumen *Shepp-Logan* con diferente orientación, ruido y clases desplegadas. 76

Índice de Tablas

4.1. Relación señal a ruido para cada uno de los casos y volúmenes. 72

4.2. Medidas obtenidas de la comparación entre las segmentaciones de los volúmenes con ruido del caso (i) y las segmentaciones sin ruido. 73

4.3. Medidas obtenidas de la comparación entre las segmentaciones de los volúmenes con ruido del caso (ii) y los objetos estándar. 74

4.4. Medidas obtenidas de la comparación entre las segmentaciones de los volúmenes con ruido del caso (iii) y los objetos estándar. 74

4.5. Medidas obtenidas de la comparación entre las segmentaciones de los volúmenes con ruido y los objetos estándar. 77

4.6. Medidas obtenidas de la comparación entre las segmentaciones de los volúmenes con ruido y los objetos estándar. 77

Capítulo 1

Introducción

La visión es uno de los sentidos de mayor importancia para el ser humano, por esta razón se han realizado grandes esfuerzos para traer la abstracción matemática y el modelado ante nuestros ojos por medio de los gráficos por computadora [9]. Adicionalmente, los gráficos proveen un medio natural para comunicarse con la computadora, debido a que las habilidades del ser humano para el reconocimiento de patrones en dos y tres dimensiones permiten percibir y procesar datos pictóricos de forma rápida y eficiente [10]. Diversas ciencias consideran importante observar fenómenos, objetos y datos para, entre otros fines, analizarlos o estudiarlos.

La visualización científica es un campo importante dentro de las ciencias de la computación que pretende realizar la representación gráfica de información de tal forma que sea posible entender mejor los datos, o bien, el fenómeno que los produce [9]. En general, la visualización es una herramienta que permite representar una gran cantidad de información en una imagen facilitando el entendimiento de los datos contenidos en ella. El origen de los datos puede ser tan diverso que incluye desde simulaciones hasta mediciones directas en dos o más dimensiones; sin embargo, en este documento se hará énfasis en los datos representados en 3D por campos escalares de la forma

$$f : \mathbb{R}^3 \to \mathbb{R}, \tag{1.1}$$

donde $\mathbb{R}$ es el conjunto de los números reales.

La visualización científica se convirtió en un campo importante desde finales de la

década de 1980, con la finalidad de interpretar grandes cantidades de datos sin resumirlos, resaltando tendencias y fenómenos en varios tipos de representaciones gráficas [10]. Los métodos de visualización de datos en 3D pueden englobarse en dos grupos principales: los métodos de visualización por superficies (VS o *surface rendering*) y métodos de visualización directa de volúmenes (VDV o *volume rendering*).

Los métodos de visualización por superficies asumen que en una función f se encuentran representados uno o mas objetos y qué para representarlos gráficamente es suficiente con desplegar las superficies que los encierran, puesto que se considera que los objetos son sólidos u opacos. Otro aspecto importante de estos métodos es que la función f tiene que ser preprocesada para identificar aquellas regiones de la función que representan a los objetos. Por otro lado, los métodos de visualización directa de volúmenes asumen que la función f ha sido discretizada en pequeños elementos de volumen denominados *vóxeles* (de los cuales ahondaremos más en el Capítulo 2) los cuales serán usados para generar la imagen final. Para crear una imagen usando métodos de visualización directa de volúmenes, primero debe asignarse la opacidad y el color a cada vóxel por separado. Posteriormente, todos los vóxeles se proyectan y se acumulan sobre un plano, lo que típicamente resulta en una imagen con regiones más opacas que otras; dicha opacidad depende del número de elementos o vóxeles proyectados y sus respectivos grados de translucidez.

A pesar de que los métodos de visualización por superficies son los métodos más utilizados, debido a que la mayoría del *hardware* y *software* para graficación están adaptados y enfocados a esta clase de visualización, la metodología de visualización directa de volúmenes tiene como ventaja que permite observar simultáneamente toda la función f (por ejemplo, varios órganos en un volumen del torso de un paciente obtenido por tomografía), también permite desplegar objetos que no podrían representarse mediante superficies, tal y como es el caso de partículas (por ejemplo, polvo). Además, permite mostrar las regiones de transición entre componentes que en otros métodos son aproximados como superficies. Sin embargo, aunque en teoría el reconocimiento de los componentes en la visualización directa de volúmenes se deja al observador, en la práctica dicho reconocimiento depende en gran medida de la forma

en que se asignan las opacidades y los colores a los vóxeles, por medio de la llamada *función de transferencia*. Por lo tanto, el diseño de funciones de transferencia es una tarea muy importante y en este trabajo nosotros hacemos una propuesta para su diseño.

Por otro lado, el campo de procesamiento de imágenes digitales tiene como objetivos principales mejorar la información pictórica para interpretación humana y procesar los datos de imagen para su almacenamiento, transmisión y representación a través de medios computacionales [11]. Un procedimiento muy común en las tareas de procesamiento de imágenes es la *segmentación*, la cual permite la cuantificación y visualización de los objetos de interés en una imagen por medio de su identificación o reconocimiento. Existen muchos métodos de segmentación [12, 13, 14, 15]; sin embargo, todo método de segmentación realiza la separación, aislamiento y/o clasificación de los objetos de interés dentro de una imagen. Por lo tanto, la interpretación y el análisis de los objetos de interés estarán determinados por la calidad del proceso de segmentación.

Las áreas de graficación computacional, visualización científica y procesamiento de imágenes se encuentran entrelazadas y es difícil diferenciar donde comienza una y termina otra; en este trabajo se hará uso de conceptos de estás tres áreas. Se puede considerar que para visualización directa de volúmenes, asignar las características de cada vóxel que pertenece a un objeto equivale a realizar una segmentación, puesto que se asignan características similares sólo a aquellos vóxeles que satisfacen cierta propiedad de pertenencia. Dentro de los métodos de segmentación podemos encontrar aquellos basados en lógica difusa, ésta se usa principalmente para trabajar con procesos complejos, donde no existe un modelo de solución sencillo, así como en procesos no-lineales. La lógica difusa también es útil cuando es necesario introducir el conocimiento empírico basado en conceptos imprecisos de un operador "experto". En la teoría de lógica difusa, los conjuntos no tienen límites definidos, de igual forma la pertenencia a dichos conjuntos no es binaria, si no que es determinada por una función de pertenencia.

Un algoritmo de segmentación que utiliza lógica difusa asigna valores en el rango

[0, 1] a cada elemento de una imagen (a diferencia de la clasificación binaria, donde solo se asignan los valores {0, 1}), donde dicho valor representa el grado de pertenencia de un elemento de la imagen a una clase u objeto (un valor de 0 indica que el elemento no pertenece, mientras que un valor de 1 indica que el elemento pertenece a la clase con toda certeza). En particular, es de interés usar el algoritmo de segmentación difusa implementado en [16, 17] para asignar las características para cada vóxel en la visualización directa de volúmenes. Este algoritmo ha demostrado ser eficaz ante presencia de ruido y diversas condiciones de iluminación. Por lo tanto, para este proyecto proponemos un tipo de funciones de transferencia para visualización directa de volúmenes que utiliza un método de segmentación basado en principios de lógica difusa.

Este documento se encuentra dividido de la siguiente manera: en el Capítulo 2 se desarrollarán los conceptos referentes a la visualización de campos escalares en 3D, el Capítulo 3 describe los conceptos básicos y la implementación del algoritmo de segmentación basado en lógica difusa. Posteriormente, el Capítulo 4 muestra la metodología, los experimentos y resultados obtenidos. Finalmente, en el Capítulo 5 presentamos las conclusiones del proyecto y ofrecemos propuestas para trabajo futuro.

Motivación y Objetivos Principales

En visualización directa de volúmenes es de gran importancia identificar los vóxeles que poseen características similares. El objetivo del proyecto es asignar propiedades visuales similares a los vóxeles que representan a cada objeto para su visualización por medio de visualización directa de volúmenes; la identificación de los vóxeles comunes a cada objeto se hace a través de un algoritmo de segmentación basado en principios de lógica difusa. El programa de segmentación difusa es semi-automático: requiere que el usuario elija puntos que pertenecen con toda certeza a cada objeto. De esta manera, se asignan propiedades visuales similares a los elementos que pertenecen a una misma clase para su posterior visualización.

Existen un gran número de programas, tanto comerciales como académicos, que

permiten la visualización de datos por medio de *volume rendering*; sin embargo, la selección de propiedades para la visualización es manual, o bien, implica que se tenga un buen conocimiento para el diseño de funciones de transferencia. Por ejemplo, en la Fig. 1.1 se pueden observar las interfaces para diseño de funciones de transferencia de los programas *Amira*® y *Simian*. En el caso de *Amira*® se pueden seleccionar funciones predefinidas o modificar alguna de ellas manualmente. Mientras que en el caso de *Simian* se pueden crear y modificar funciones de 2 o 3 dimensiones manualmente usando la información del volumen (por ejemplo, primera y segunda derivadas).

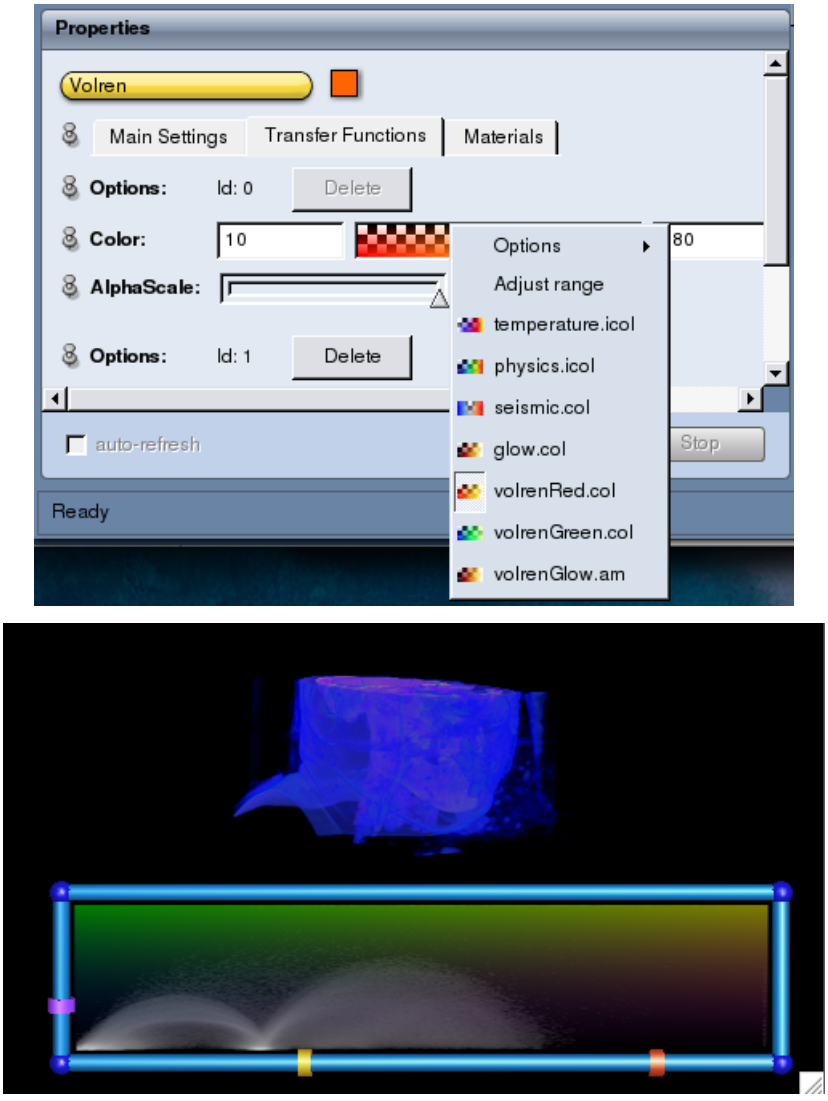


Figura 1.1: Interfaces para el diseño de funciones de transferencia de algunos programas para visualización directa de volúmenes. Arriba *Amira*®, abajo *Simian*.

Sin embargo, encontrar una función de transferencia que provea una mejor visualización por medio de la experimentación puede convertirse en una tarea tardada y tediosa [18]. Al utilizar un método semi-automático de segmentación se requiere una intervención mínima del usuario, lo que puede llegar a reducir el tiempo invertido en la tarea de encontrar una función de transferencia adecuada y al mismo tiempo se conserva la retroalimentación que puede mejorar el resultado.

Al generar la función de transferencia a partir de una segmentación, es posible asignar con facilidad las propiedades visuales por regiones para una correcta visualización de los objetos desde el momento de la clasificación de sus componentes, además se tiene la posibilidad de tratar aquellos elementos que posean un valor de pertenencia muy bajo como objetos separados. De esta forma la selección de la función de transferencia se vuelve una tarea un poco más intuitiva, puesto que al asignar propiedades objeto por objeto, se pueden diseñar funciones de transferencia *ad hoc* y a su vez, se puede experimentar con los parámetros del algoritmo de segmentación para obtener distintos resultados, según sea conveniente para los datos.

Capítulo 2

Visualización de Imágenes 3D

Los gráficos por computadora son usados extensivamente en muchas áreas del conocimiento, la lista de sus aplicaciones es enorme y sigue creciendo rápidamente conforme las computadoras con capacidades gráficas se vuelven más comunes [10]. La mayoría de los datos pueden ser representados por campos vectoriales, de la forma $f : \mathbb{R}^n \to \mathbb{R}^n$, o bien, por campos escalares, de la forma $f : \mathbb{R}^n \to \mathbb{R}$; sin embargo, en este trabajo nos enfocaremos a los campos escalares tridimensionales donde $n = 3$ como se definió en (1.1), puesto que ese es el tipo de datos producidos por diversos aparatos de imaginología. En este capítulo se plantearán los conceptos y técnicas de visualización más comunes para después concentrarnos en la visualización directa de volúmenes.

2.1. Discretización de imágenes

La información proveniente de un campo escalar es discretizada para poder ser procesada por un sistema de cómputo. Esta discretización típicamente es llevada a cabo al evaluar el campo escalar en ciertos puntos del espacio tridimensional [19]. Esto quiere decir que no se trabaja directamente con la función f de (1.1) sino con una aproximación que se obtiene, generalmente, evaluando f de manera uniforme en el espacio tridimensional.

En este trabajo representaremos un punto en el espacio n-dimensional de los reales, o $\mathbb{R}^n$, usando la n-tupla $\mathbf{x} = \{x_1,\ x_2,\ \ldots,\ x_n\}$, donde $x_i \in \mathbb{R}$; de forma similar,

utilizaremos la n-tupla $\mathbf{k} = \{k_1,\ k_2,\ \ldots,\ k_n\}$, donde $k_i \in \mathbb{Z}$, para representar un punto en el espacio n-dimensional de los enteros, o $\mathbb{Z}^n$. En particular, nos enfocaremos en los conjuntos $\mathbb{R}^3$ y $\mathbb{Z}^3$.

Para llevar a cabo la discretización de f sólo se usará un subconjunto $\Gamma_\Delta \subset \mathbb{Z}^3$ definido de la siguiente forma

$$\Gamma_\Delta = \{\Delta\mathbf{k} \mid A_i \leq k_i \leq B_i,\ \text{para } 1 \leq i \leq 3 \text{ con } A_i < B_i \text{ y } A_i,\ B_i \in \mathbb{Z}\}\,, \tag{2.1}$$

donde Δ es un número real positivo que se conoce comúnmente como intervalo de muestreo [20]. Es fácil ver que el vecindario de Voronoi de cada punto de la rejilla Γ_Δ es un vóxel (del inglés *volume element*) cúbico. Dado un punto $\mathbf{k} \in \Gamma_\Delta$ y un intervalo de muestreo Δ, un vóxel cúbico puede definirse como:

$$Voxel\,(\mathbf{k}) = \left\{\mathbf{x} \in \mathbb{R}^3 \mid \Delta\left(k_i - \frac{1}{2}\right) \leq x_i < \Delta\left(k_i + \frac{1}{2}\right),\ \text{para } 1 \leq i \leq 3\right\}. \tag{2.2}$$

La discretización de un campo escalar f que lleva a cabo una máquina de imaginología sobre los puntos de la rejilla definida en (2.1) usando un intervalo de muestreo Δ se puede definir como

$$p_\Delta(\mathbf{k}) = \frac{1}{\Delta^3}\int_{Voxel(\mathbf{k})} f(\mathbf{x})d\mathbf{x}. \tag{2.3}$$

Este proceso de digitalización produce una *imagen digital* en tres dimensiones (también llamado *volumen*) que se define como

$$v(\mathbf{k}) = \begin{cases} p_\Delta(\mathbf{k}), & \text{si } \mathbf{k} \in \Gamma_\Delta, \\ 0, & \text{en cualquier otro caso.} \end{cases} \tag{2.4}$$

Al subconjunto Γ_Δ de (2.1) usado para obtener v se le llama *escena*. Por convención, a cada uno de los límites de las dimensiones del dominio de v se le da un nombre. Típicamente, los intervalos válidos $[A_1, B_1]$, $[A_2, B_2]$ y $[A_3, B_3]$ son comúnmente llamados *ancho*, *altura* y *profundidad* de la imagen, respectivamente, y tienen una equivalencia

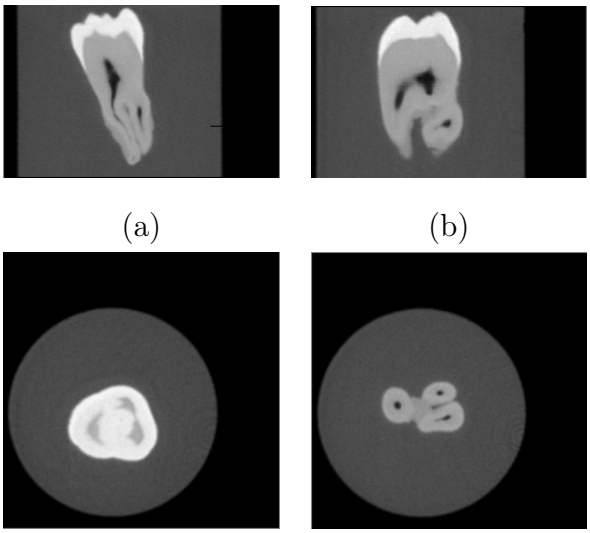
en un sistema coordenado xyz.

Las rejillas cúbicas en $\mathbb{R}^3$ están presentes en casi todo el procesamiento de imágenes, es por eso que son las que se usan más comúnmente. Sin embargo, existen otros tipos de rejillas que pueden muestrear el espacio de manera uniforme. Algunas de ellas, como la *face-centered cubic grid* o la *body-centered cubic grid* (cuyos vóxeles son dodecaedros y octaedros truncados, respectivamente [19]) son útiles en visualización directa de volúmenes y otros procedimientos de análisis y procesamiento de imágenes [19, 21]. En este proyecto asumiremos que se trabaja con vóxeles cúbicos, aunque nuestros resultados pueden ser generalizados a funciones discretizadas con otras rejillas. Los otros tipos de rejillas tienen como ventaja el hecho de que la visualización con estás rejillas es más "suave" (no se ven como si fueran construidas por bloques cúbicos), se simplifica la decisión sobre la adyacencia entre vóxeles y se mejora el muestreo del espacio [19], pero pueden suponer un incremento en la complejidad del manejo de datos o en la visualización de poliedros cuya superficie esta formada con polígonos diferentes (en otras palabras, polígonos con un numero diferente de aristas).

2.2. Visualización

En varias áreas del conocimiento se acostumbra visualizar las funciones discretizadas, como la definida por (2.4), usando la modalidad *plano por plano*; es decir, dada una imagen digital v, definida sobre una escena Γ_Δ, se fija el valor de una coordenada k_i, para $i \in \{1, 2, 3\}$ y se hacen variar k_j y k_l, para $j, l \neq i$, para generar una imagen digital en 2D tal y como se puede observar en la Fig. 2.1.

En principio, es difícil observar la estructura tridimensional del interior de una imagen digital al observar solo sus datos vía planos individuales (también llamados *rebanadas*), de ahí que surgiese el interés de observar los datos directamente en tres dimensiones [22]. Como mencionamos en el Capítulo 1, los métodos de visualización de datos en 3D pueden clasificarse en dos tipos: visualización por superficies (*surface rendering*) y visualización directa de volúmenes (*volume rendering*).



(a) (b)

(c) (d)

Figura 2.1: Datos volumétricos de un diente presentados por planos para algún valor fijo en los ejes (a) x (ancho), (b) y (altura) y (c) (d) z (profundidad) .

2.2.1. Visualización por Superficies (*Surface Rendering*)

Se puede considerar que los métodos de visualización por superficies más comunes buscan desplegar el siguiente conjunto de puntos

$$S_\tau = \{\mathbf{x} \mid f(\mathbf{x}) = \tau\}, \tag{2.5}$$

donde f es una función definida por (1.1), $\tau \in \mathbb{R}$ usualmente se conoce como umbral y S_τ es la superficie implícita o isosuperficie de f bajo el umbral τ. Estos métodos hacen la suposición de que existe un umbral τ para el cual el objeto de interés consiste de exactamente aquellos puntos en los que el valor de f es mayor que el umbral y que S_τ encierra el volumen. Por lo tanto, basta con desplegar la superficie S_τ, tal y como se definió en (2.5), para visualizar el objeto de interés. Sin embargo, es común trabajar con la discretización v, definida en (2.3), en lugar de la función continua f (aunque hay trabajos que usan una aproximación a f por medio de una combinación lineal de funciones base para obtener imágenes visualmente muy atractivas [23, 24]).

Las técnicas dominantes para desplegar superficies de objetos dentro de una ima-

gen 3D consisten en segmentar el volumen v para luego aplicar un "detector" de superficies al subconjunto de vóxeles resultantes y ajustar primitivas geométricas a la superficie detectada. Dicha aproximación es conocida como poligonización y permite desplegar estas primitivas usando algoritmos convencionales de graficación por computadora; las diferencias entre estas técnicas se encuentran en la elección de las primitivas y la escala a la que éstas se definen [25].

Se han desarrollado muchos métodos para definir una malla poligonal que aproxima al conjunto S_t de (2.5). Uno de los métodos más intuitivos y comunes para modelar objetos es utilizar sólidos geométricos de los cuales se poseen fórmulas analíticas, para obtener la triangulación de alguno de estos sólidos solo es necesario evaluar dicha fórmula analítica. Otro método propone efectuar la poligonización de una superficie implícita por medio de la subdivisión de paralelepípedos rectangulares [26]. La subdivisión se lleva a cabo de la siguiente forma, primero se encierra la imagen 3D dentro de un paralelepípedo rectangular. En cada vértice $\mathbf{k}_i$, para $1 \leq i \leq 8$, del paralelepípedo rectangular se prueba si $v(\mathbf{k}_i) \leq \tau$ o $v(\mathbf{k}_i) > \tau$. Si para todo i se tiene que $v(\mathbf{k}_i) \leq \tau$, o bien, $v(\mathbf{k}_i) > \tau$, entonces el paralelepípedo rectangular no es subdividido. Por el contrario, el paralelepípedo rectangular es subdividido en la arista cuyos vértices $\mathbf{k}_i$ y $\mathbf{k}_j$ poseen valores $v(\mathbf{k}_i) \leq \tau$ y $v(\mathbf{k}_j) > \tau$, respectivamente. Otro ejemplo común de esta metodología es el conocido *Marching Cubes* [27] la cual produce una malla de triángulos cuyos vértices aproximan al conjunto S_τ de (2.5) para una imagen digital v. Los vértices de esta malla son obtenidos por medio de la utilización de un paralelepípedo rectangular (o cubo) virtual que se forma usando los centros de ocho vóxeles 'adyacentes' como sus vértices. Este cubo se forma sobre todos los vóxeles de la imagen 3D. En cada vértice $\mathbf{k}_i$, para $1 \leq i \leq 8$, del cubo se prueba si $v(\mathbf{k}_i) \leq \tau$ o $v(\mathbf{k}_i) > \tau$. El vértice de un triángulo se agrega sobre la arista del cubo cuyos vértices $\mathbf{k}_i$ y $\mathbf{k}_j$ poseen valores $v(\mathbf{k}_i) \leq \tau$ y $v(\mathbf{k}_j) > \tau$, respectivamente. La creación de la malla se acelera al tomar en cuenta simetrías y rotaciones para los 256 casos posibles derivados de probar los vértices del cubo. Una vez que se ha creado la malla poligonal ésta se visualiza con técnicas comunes de graficación tales como operaciones de sombreado y mapeo de texturas [28]. Una imagen obtenida por medio

de este método puede observarse en la Fig. 2.2.

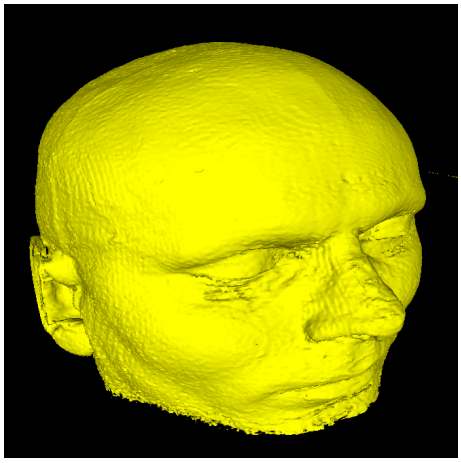


Figura 2.2: Imagen obtenida por medio del método *Marching Cubes* aplicado sobre los datos de una cabeza humana (tomada de [1]).

En general, los métodos de visualización por superficies son los más utilizados debido a que la mayoría del *hardware* y *software* para graficación están adaptados y enfocados a esta clase de visualización, una vez que se tiene una representación de la superficie se pueden generar imágenes rápidamente desde cualquier punto de vista. Sin embargo, la identificación preliminar del objeto de interés puede ser una tarea desafiante y, en ocasiones, puede producir resultados no deseados. Adicionalmente, hay objetos que son complicados de modelar usando polígonos (porque la representación obtenida no refleja la estructura real del objeto), tales como objetos suaves, naturales o nubes de partículas. Además la calidad de la visualización depende del modelo de iluminación usado por el algoritmo de graficación, de las características de la superficie y de la precisión de la aproximación por el algoritmo de poligonización (por ejemplo, dependerá del número de polígonos o del criterio para medir el error de aproximación).

2.2.2. Visualización Directa de Volúmenes (*Volume Rendering*)

A pesar de que la visualización por superficies es la metodología estándar en la actualidad, presenta varias desventajas. Al hacer la suposición de que el volumen original puede ser representado por un modelo que consiste de superficies delgadas suspendidas en un ambiente vacío y transparente (lo que constituye una forma indirecta de visualizar volúmenes [29]) y, aunque pueden representarse varios objetos y/o materiales (es decir, pueden distinguirse como objetos diferentes dentro de una misma escena), pueden perderse otras superficies y detalles sutiles entre materiales, así como variaciones locales de las propiedades volumétricas [22]. Estas dificultades surgen como resultado del efecto de volumen parcial, introducido no sólo por la digitalización sino también por la aproximación poligonal.

Tratar de extraer un modelo geométrico de los datos para poder utilizar métodos de graficación tradicionales puede ser un problema, especialmente cuando las superficies no están bien definidas o son complejas. Muchas técnicas de visualización por superficies tratan de hacer una clasificación binaria, la superficie pasa por un vóxel o no lo hace. Como resultado, estos métodos a menudo presentan falsos positivos (superficies espurias) o falsos negativos (hoyos erróneos en las superficies), particularmente cuando se tiene presencia de características pequeñas o definidas pobremente [25].

Como su nombre lo indica, la visualización directa de volúmenes trata con los datos de una imagen tridimensional de forma directa, por lo tanto presenta información tanto de las superficies como de las estructuras internas del modelo, preservando la continuidad de los datos en 3D [29]. Por esta razón, esta técnica ha sido ampliamente usada para visualizar modelos que no tienen superficies tangibles, tales como nubes y niebla [30].

La visualización directa de volúmenes omite una representación geométrica intermedia, por lo que las imágenes son creadas otorgando propiedades visuales a todas las muestras de datos y proyectándolas sobre un plano (típicamente una pantalla). Es

importante hacer notar que la falta de geometría explícita no excluye la posibilidad de desplegar superficies. La mejora clave de la visualización directa de volúmenes es que provee una forma de desplegar superficies débiles o difusas, ya que no es necesario decidir si la superficie está presente o ausente de un vóxel dado [25], aunque lo anterior no evita la presencia del efecto de volumen parcial, ya que estamos trabajando con datos discretizados. Por lo tanto, si el vóxel definido en (2.2) no es lo suficientemente pequeño como para muestrear únicamente el material de un solo objeto dentro de él, puede ser que se muestreen varios materiales. El efecto de volumen parcial puede aminorarse si se asignan propiedades del material con más contribución en aquellos vóxeles que presenten el efecto de volumen parcial, el nuevo problema consiste en cómo elegir si el vóxel pertenece a uno u otro material. Una imagen obtenida por medio de esta metodología puede observarse en la Fig. 2.3.

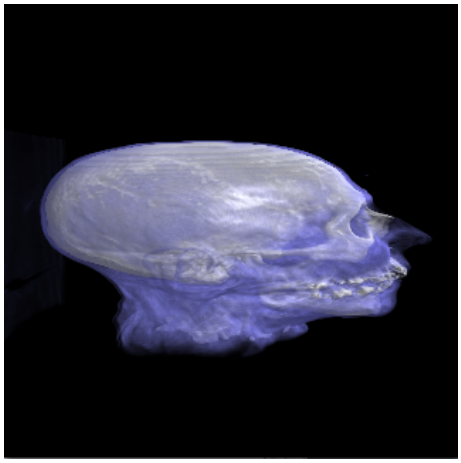


Figura 2.3: Imagen de un conjunto de datos de una cabeza humana, obtenida por medio del método estándar de visualización directa de volúmenes, implementado en el programa *Voreen* [2]

2.2.2.1. Principio Básico

Uno de los primeros algoritmos para realizar la visualización directa de volúmenes fue propuesto en [22]. En él, se asume que el arreglo de volumen fue muestreado por encima de la frecuencia de Nyquist [31], o bien, que un filtro pasa-bajas removió las frecuencias altas que producen *aliasing*.

El primer paso del algoritmo consiste en clasificar los materiales encontrados en los datos volumétricos de entrada. Muchas técnicas de clasificación pueden utilizarse y la elección de cual usar depende de los datos de entrada. Normalmente, cambios de densidades o gradientes son usados para estimar superficies entre materiales [29]. Es importante recalcar que estos métodos son también utilizados en técnicas básicas de segmentación de imágenes y señales [11].

Posteriormente, se eligen propiedades visuales (típicamente, un color y una transparencia, para cada material) para asignar dichas propiedades apropiadamente a cada vóxel dependiendo de su clasificación. Se puede considerar que se hacen los siguientes mapeos

$$\varphi : \Gamma_\Delta \longrightarrow \Theta \tag{2.6}$$

y

$$\alpha : \Gamma_\Delta \longrightarrow [0, 1]\,, \tag{2.7}$$

donde Θ representa vectores del tipo $(r,\ g,\ b)$, un tipo especial de vectores donde $r, g, b \in [0, 1]$ representan las intensidades de luz roja, verde y azul, respectivamente; mientras que α representa la opacidad. Un valor de $\alpha = 1$ implica que el material es totalmente opaco y un valor de $\alpha = 0$ implica que es completamente transparente.

La imagen final de *volume rendering* se forma conceptualmente de la siguiente manera (ver Fig. 2.4): se asume que el conjunto v se encuentra entre una fuente de luz ℓ y el plano P (la pantalla) y para cada píxel de la pantalla se envía un rayo de luz desde ℓ a P. Dicho rayo entra al vóxel más cercano a la fuente de luz ℓ (parte de atrás de la imagen 3D) con una intensidad de entrada I_i y sale por el vóxel mas cercano al plano P (parte del frente de la imagen 3D) con una intensidad de salida I_i'. En un material real, la intensidad de la luz cambia debido a los siguientes efectos: (i) los materiales pueden actuar como filtros traslúcidos, absorbiendo la luz entrante; (ii) pueden ser luminosos y emitir luz, o bien, (iii) pueden contener superficies o partículas dispersoras que atenúen la luz entrante y que también reflejen luz de las fuentes de luz. Sin embargo, en visualización directa de volúmenes solo se asume una dispersión simple de la radiación de la fuente de luz; es decir, no existe ni emisión ni reflexión

de luz. De igual forma, los rayos de luz no son atenuados, por lo que la intensidad de salida se calcula como una integral de línea usando el color y opacidad asignados a cada vóxel; típicamente, este proceso es llamado *composición*. Adicionalmente, el volumen iluminado es transformado y remuestreado para desplegarlo en el sistema de coordenadas de la pantalla.

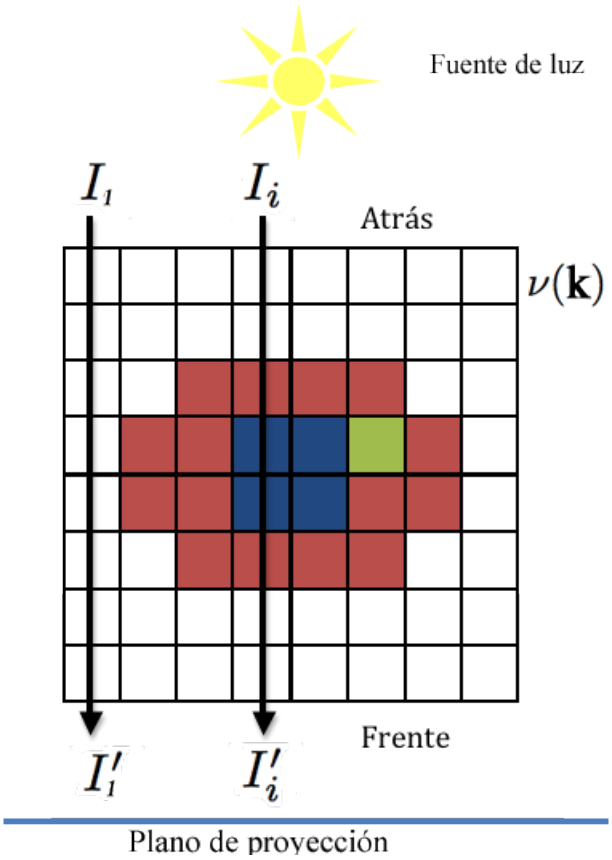


Figura 2.4: Integral de línea usada por el modelo de iluminación en *volume rendering*, donde I_i es la intensidad de luz de entrada, mientras que I_i' es la intensidad de salida. La intensidad de salida es calculada con base en las propiedades asignadas a cada vóxel de $v(\mathbf{k})$ que es atravesado por el rayo de luz.

2.2.3. Composición

Como se mencionó anteriormente, la composición (también conocida como *alpha blending*) es el proceso que nos permite obtener los valores finales de la proyección

del volumen en pantalla. Durante esta etapa se unen los colores del fondo con los colores y opacidades asignados en la etapa anterior, haciendo la suposición de que los valores del fondo se encuentran detrás del conjunto de datos (tomando como frente la pantalla de proyección).

Generalmente, la composición se hace de atrás hacia adelante (*back to front*) y se realiza haciendo uso de técnicas como el *raycasting* para enviar un rayo por cada píxel, ya sea en perspectiva o en paralelo, aunque esta última es más común y es el caso que usamos en este trabajo. Para un plano discretizado $P_\Delta \subset \mathbb{Z}^2$ (en otras palabras, la pantalla) de forma similar a la discretización realizada usando el subconjunto Γ_Δ de (2.1), se asigna un color a cada píxel por medio de la función $g : R_\mathbf{u} \to (\Theta, [0,1])$ donde $R_\mathbf{u}$ es un subconjunto de los vóxeles de la escena que se obtiene como sigue. Para el píxel $\mathbf{u} \in P_\Delta$ se usa la línea recta (la cual representa a un rayo paralelo al plano) definida por

$$\mathbf{x}_\mathbf{u} = \mathbf{u} + \rho\vec{\mathbf{o}}, \tag{2.8}$$

donde $\vec{\mathbf{o}}$ es un vector de dirección (en otras palabras, un vector unitario) tal qué $\vec{\mathbf{o}} \perp P_\Delta$ y ρ es un número real positivo. Para un rayo paralelo, que parte de un píxel $\mathbf{u}$, nos referimos a los vóxeles de Γ_Δ intersecados por la línea $\mathbf{x}_\mathbf{u}$ de (2.8) como el conjunto $R_\mathbf{u} = \{\mathbf{k}_1, \ldots, \mathbf{k}_l, \ldots, \mathbf{k}_L\}$, donde L puede considerarse como la longitud del rayo en vóxeles. Los vóxeles del conjunto $R_\mathbf{u}$ cumplen $dist\,(\mathbf{u}, \mathbf{k}_l) \leq dist\,(\mathbf{u}, \mathbf{k}_{l+1})$ (se encuentran ordenados de menor a mayor con respecto a la distancia del vóxel a la pantalla).

El color final para un píxel $\mathbf{u} \in P_\triangle$ se calcula tomando en cuenta las contribuciones individuales de las propiedades visuales de cada vóxel perteneciente al conjunto $R_\mathbf{u}$. Una fórmula ampliamente utilizada para realizar la composición de los vóxeles en un conjunto $R_\mathbf{u}$ es la propuesta en [25]

$$g\,(R_\mathbf{u}) = \sum_{l=1}^{L} \left[\varphi\,(\mathbf{k}_{L-l+1})\,\alpha\,(\mathbf{k}_{L-l+1}) \prod_{m=l+1}^{L} (1 - \alpha\,(\mathbf{k}_{L+l+1-m})) \right], \tag{2.9}$$

donde $\varphi\,(\mathbf{k}_L)$ es igual al color del fondo y $\alpha\,(\mathbf{k}_L) = 1$. Esta ecuación se deriva con-

siderando que solo ocurren cambios locales en la iluminación. Por lo tanto, se puede analizar el cambio de iluminación dentro de un cilindro muy delgado cuyo eje se encuentra centrado sobre la línea $\mathbf{x_u}$ y el cual atraviesa el volumen, donde φ representa la emisión, o luminosidad, y α representa la atenuación [32]. La luz entra al cilindro por la parte posterior del volumen y emerge del cilindro en la parte anterior del volumen para ayudar a determinar el color que se debe asignar al píxel $\mathbf{u}$. Debido a que se usa un cilindro muy delgado se puede considerar que los cambios en la iluminación ocurren solo a lo largo del cilindro, ver la Fig. 2.5.

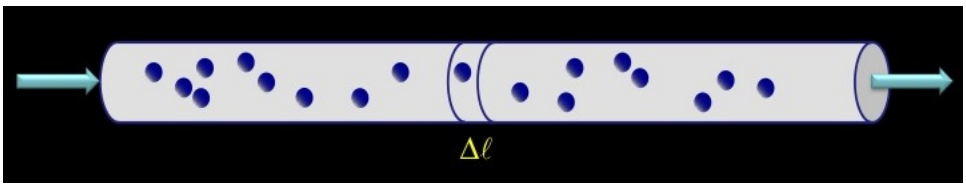


Figura 2.5: Corte cilíndrico hipotético del conjunto de datos que sirve para el modelo de partículas en visualización directa de volúmenes.

El comportamiento de la intensidad de un rayo de luz que pasa a través del volumen de la Fig. 2.5 se define como [33]

$$\frac{dI}{ds} = L_e(s)\,\eta(s) - I(s)\,\eta(s), \tag{2.10}$$

donde η es el coeficiente de atenuación (la cantidad de luz entrante que se espera sea extinguida por unidad de longitud). Por lo tanto, el cambio de intensidad de un rayo de luz I al pasar a través del volumen es igual a la luz emitida L_e por el corte axial en la posición s menos la cantidad de luz entrante que ha sido atenuada hasta ese punto. La ecuación diferencial (2.10) tiene la solución

$$I(\ell) = I_0 e^{-\int_0^\ell \eta(t)dt} + \int_0^\ell L_e(s)\,\eta(s)\,e^{-\int_s^\ell \eta(t)dt} ds. \tag{2.11}$$

El primer término representa la cantidad de luz entrante I_0 que llega al punto de longitud ℓ. El segundo término representa la cantidad de luz emitida en cada punto del rayo en su camino (tomando en cuenta la cantidad de atenuación en cada uno de sus puntos). Se puede ver que la ecuación (2.9) es la solución discreta de la ecuación

(2.11). La composición puede tomarse como una integral de línea, donde se suma la contribución de cada vóxel a la imagen final según ciertos parámetros.

2.3. *Rendering Pipeline*

El proceso por el que pasa el modelo hasta lograr la producción de la imagen final es conocido como *rendering pipeline* o proceso de renderizado. Durante este proceso se hacen operaciones que afectarán la apariencia de la imagen final, puede considerarse análogo a tomar una foto. Aunque existen distintas variaciones e implementaciones de este proceso, los siguientes pueden considerarse los pasos básicos.

- *Transformaciones de modelo*: son transformaciones geométricas realizadas sobre los objetos (modelos) presentes en la escena.
- *Transformaciones de vista*: son transformaciones realizadas sobre la escena de acuerdo a la orientación y posición de una cámara virtual, incluye eliminar aquellas partes de los objetos que no son visibles desde la cámara (*clipping*).
- *Rasterización* (*Rastering*): es el conjunto de procesos que producen los valores finales de cada píxel en una región de memoria conocida como *frame buffer*.
    - *Determinación de superficie visible*. Este proceso consiste en la eliminación de las regiones de los objetos que se encuentran detrás de otros objetos en relación con el espectador (*z-buffer*). Este proceso no es necesario para llevar a cabo visualización directa de volúmenes.
    - *Proyección*. Este proceso consiste en proyectar el modelo 3D sobre un plano. La proyección puede ser ortogonal o en perspectiva. Cuando se hace una proyección ortogonal el tamaño de los objetos no se altera en función de la distancia al espectador; cuando se realiza en perspectiva, los objetos cambian su tamaño de acuerdo a la distancia a la que estén con respecto al espectador.

- *Iluminación y sombreado*. Estos procesos consisten en modelar la interacción de la luz con los materiales de los objetos y por lo tanto dar una apariencia mas realista a la imagen final.

- *Despliegue de la imagen*: los píxeles finales son mostrados en la pantalla o cualquier otro dispositivo de despliegue, está etapa también es conocida como *renderizado* (*rendering*).

2.3.1. Iluminación

El modelo de la interacción de la luz con los materiales de los objetos es importante en graficación por computadora para dar realismo a la imagen final. Para obtener un resultado visualmente más agradable y realista, la apariencia de cada objeto debe tomar en cuenta el tipo de fuentes de luz que los iluminan, de sus propiedades (tales como color, textura, reflectancia), así como su posición y orientación con respecto a las fuentes de luz, el observador y otros objetos. Esta variación en la iluminación es una indicación de la estructura en 3D del objeto [10].

Para tal propósito se han modelado varios tipos de fuentes de luz. La *luz ambiental* es aquella que afecta al objeto en todas direcciones y es el tipo de fuente de luz más fácil de modelar, porque se asume que produce iluminación constante en todas las superficies (sin importar su posición u orientación); sin embargo, por si sola, no produce imágenes realistas. Una fuente de *luz puntual*, es aquella cuyos rayos emanan desde un solo punto; un caso particular es la *luz focal*, donde los rayos viajan en una dirección preferente formando un cono. Ambas, la luz puntual y focal, pueden usarse para aproximar un foco. Finalmente, una fuente de *luz direccional* es aquella cuyos rayos vienen de la misma dirección y son paralelos entre sí (se asume que es una fuente puntual que está infinitamente distante), puede usarse para representar un sol distante.

2.3.1.1. Modelo de Iluminación de Phong

El comportamiento de la luz es un fenómeno físico difícil de modelar; sin embargo, existen varios modelos que son comunes porque producen resultados atractivos y útiles con cómputo mínimo, aunque no sean físicamente correctos. El modelo de iluminación de Phong es de los modelos más empleados debido a su simpleza y resultados [34]. Este modelo asume que la intensidad de la luz y su reflejo pueden calcularse como la suma de tres componentes de intensidades de luz independientes: la componente ambiental, la componente difusa y la componente especular. De igual forma, supone que los objetos están hechos de un material particular y cada material tiene cierta sensibilidad a cada componente de luz. Por lo tanto, el color final de un píxel que representa un objeto de una escena dada se calcula en función del material y las propiedades de las luces que interactúan con él.

Adicionalmente, toma como convención que el color se percibe por medio de la superposición de tres longitudes de onda (rojo, verde y azul), llamados canales, y que los cálculos de iluminación pueden hacerse para cada uno de estos canales de forma independiente. Por lo tanto, puede representarse la intensidad de luz como un vector $\mathbf{I} = (I_r,\ I_g,\ I_b)$, donde I_r , I_g y I_b son las intensidades de los canales rojo, verde y azul, respectivamente.

Aunque este modelo es ampliamente usado en visualización por superficies, se puede ver de la discusión presentada en la Sección 2.2.3 que las componentes más importantes para el proceso de visualización directa de volúmenes son las componentes difusa y especular.

Reflexión Difusa. Para una superficie dada, la brillantez depende del ángulo θ entre la dirección $\boldsymbol{\ell}$ de la fuente de luz con respecto a cada punto sobre la superficie y la normal de la superficie $\vec{\boldsymbol{n}}$ en dichos puntos. Suponiendo que los vectores $\boldsymbol{\ell}$ y $\vec{\boldsymbol{n}}$ han sido normalizados, la ecuación de iluminación está dada por

$$r_d = I_0 k_d \left(\vec{\boldsymbol{n}} \cdot \boldsymbol{\ell}\right), \tag{2.12}$$

donde I_0 es la intensidad de la fuente de luz y k_d es el coeficiente de reflexión difusa el cual es una constante entre 0 y 1 que varia de un material a otro.

Reflexión Especular. Puede observarse en las superficies brillantes, donde los rayos llegan al objeto y son reflejados en una dirección preferente. En un espejo la luz es reflejada solo en la dirección del reflejo $\mathbf{e}$, que es $\boldsymbol{\ell}$ (dirección de la fuente de luz) reflejada desde la normal $\vec{\boldsymbol{n}}$. Por lo tanto, el observador solo puede ver la luz reflejada especularmente desde un espejo (y por lo tanto la reflexión especular es máxima) cuando el ángulo ϕ entre $\mathbf{e}$ y la dirección hacia el punto de vista $\mathbf{v}$ es igual cero, tomando γ como el exponente de reflexión especular, el modelo de la reflexión especular puede escribirse como:

$$r_e = I_0 k_e \left(\mathbf{e} \cdot \mathbf{v}\right)^{\gamma}, \tag{2.13}$$

donde I_0 es la intensidad de la fuente de luz y k_e es el coeficiente de reflexión especular el cual es una constante en el rango de 0 a 1 que depende del material.

Finalmente, el modelo completo de iluminación para obtener el color y la intensidad final I de cada píxel se obtiene sumando los componentes descritos en las ecuaciones (2.12) y (2.13).

2.3.2. Mejoras al principio básico de Visualización Directa de Volúmenes

Una de las ventajas de la visualización directa de volúmenes es que separa las operaciones de iluminación de las operaciones de clasificación, lo que permite desacoplar la fase de renderizado. Por esa misma razón, el proceso de visualización se vuelve independiente e insensible a la complejidad de la escena y del objeto, puesto que se proyecta todo el volumen. Además, permite desplegar superficies difusas o débiles y, en principio, también permite el despliegue de geometrías complejas [25]. Sin embargo, este método posee dos debilidades importantes: las transformaciones y la iluminación son realizados en el espacio discreto, lo que puede provocar problemas de muestreo y *aliasing*; además, es computacionalmente demandante, ya que requiere

grandes volúmenes de espacio en memoria y muchos recursos de procesamiento para proyectar todos los vóxeles.

A pesar de estas debilidades, el uso del *volume rendering* se ha vuelto más factible en la actualidad, ya que las enormes cantidades de datos pueden ser procesadas con el poder de cómputo de las nuevas tecnologías como las tarjetas gráficas programables, que además se han vuelto bastante asequibles, así como el uso de diversas técnicas de graficación por computadora, como el mapeo de texturas.

2.3.2.1. Mapeo de Texturas

Tradicionalmente, el mapeo de texturas se ha utilizado para agregar realismo a las imágenes generadas por computadora. Sin embargo, el *hardware* utilizado para el mapeo de texturas tiene varias aplicaciones adicionales [35, 36]. El mapeo de texturas tradicional consiste en cambiar el color de los píxeles de un objeto generado por computadora por el color de una imagen externa o textura. Generalmente, este proceso requiere de varios pasos ya que la imagen textura es guardada en varios arreglos a diferentes niveles de muestreo y debe ser reconstruida de las muestras y transformada para que coincida con los píxeles que representan al objeto que está siendo desplegado.

Una de las principales formas en que el mapeo de texturas puede ser usado para visualización directa de volúmenes consiste en considerar que el volumen puede ser representado dibujando rebanadas del objeto de atrás hacia adelante [35]. Primero, se genera una imagen de textura del plano correspondiente muestreando los datos que representan al volumen a lo largo del plano deseado, de esta forma se obtienen imágenes como las de la Fig. 2.1. Para poder visualizar el plano, la textura es mapeada sobre un polígono. El volumen completo puede desplegarse en la pantalla mediante la combinación de los diferentes planos que lo conforman, cada plano es combinado con los planos previamente dibujados usando transparencias y la fórmula de la composición presentada en (2.9), o bien, un concepto similar. El proceso de despliegue puede observarse en la Fig. 2.6.

Otro método permite el uso de mapeo de texturas de 3D. Este método es muy similar al anterior, la diferencia consiste en que los datos volumétricos son copiados en

una imagen de textura en 3D, es decir, se genera un cubo compuesto con cada plano. Solo los planos perpendiculares al observador son dibujados y cada uno es tratado como un polígono con una textura mapeada; sin embargo, son las coordenadas de textura en los vértices de los polígonos las que determinan un plano a través de la imagen de textura 3D. Este método requiere que la tarjeta gráfica tenga la capacidad de hacer mapeo de texturas en 3D, pero tiene la ventaja de que la memoria de textura que se tiene que cargar es solo una sin importar el punto de vista. En caso de que el tamaño de los datos fuera grande, se puede realizar la visualización del volumen completo por partes, poniendo solo una porción del volumen en la imagen de textura en cada paso.

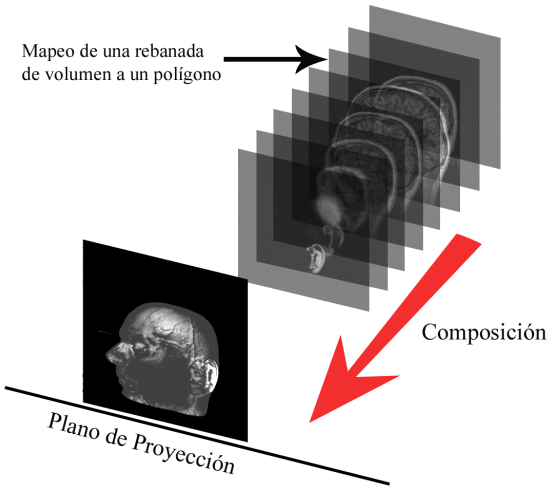


Figura 2.6: Utilización de mapeo de texturas para generar imágenes por medio de visualización directa de volúmenes.

2.3.2.2. Funciones de Transferencia

En principio, la visualización directa de volúmenes deja al observador la tarea de reconocer las estructuras contenidas dentro de una imagen 3D, aunque esta tarea puede resultar difícil debido a la oclusión de objetos [37]. Puede notarse que el principio básico de la visualización directa de volúmenes no siempre permite ver todos los objetos que son de interés. En el ejemplo de la Fig. 2.4, si se hace la suposición de que el objeto de interés es el vóxel verde y se tiene que la intensidad y opacidad

de los vóxeles rojos es mayor a la intensidad y opacidad del vóxel verde, este no podrá observarse en la visualización. Para resolver este problema se puede realizar un "cambio de ventana" de los valores pero existe la posibilidad de que el rango de valores entre el objeto de interés y el resto sea tan pequeño que el cambio no sea notable. De ahí surgió la idea de manipular la opacidad durante la trayectoria, lo que derivó en la creación del concepto de *función de transferencia*. Por lo tanto, existen enfoques que permiten asignar opacidades a los vóxeles de forma que no se obstruyen regiones "importantes" del volumen y a la vez se permite una mejor interpretación de la información contenida en las regiones visibles [38, 39].

En análisis de sistemas, una entrada se relaciona con la respuesta de un sistema por medio de una función de transferencia, este mismo concepto se aplica a visualización directa de volúmenes al momento de asignar color y opacidad a los vóxeles, según el tipo y la cantidad de material presente en los mismos. Es decir, la función de transferencia se da con base en las funciones φ de (2.6) y α de (2.7), y es la encargada de hacer el mapeo de los valores de la imagen υ a los vectores Θ. Las funciones de transferencia son particularmente importantes para la calidad y el reconocimiento de los componentes de las imágenes producidas por visualización directa de volúmenes, ya que hacen que un conjunto de los datos del volumen sea visible al asignarle propiedades ópticas (tales como color y opacidad) [39]. Por ejemplo, en la Fig. 2.7(a) los datos de interés (el diente y sus componentes) no pueden observarse porque la opacidad de los vóxeles que rodean al objeto es más alta que la opacidad de los vóxeles del mismo, cambiando la función de transferencia se puede modificar esa asignación de valores, y en algunos casos mejorar la visualización de los datos de interés, como puede observarse en las imágenes de la Fig. 2.7(b), la Fig. 2.7(c) y la Fig. 2.7(d).

Desafortunadamente, la popularidad del uso de *volume rendering* ha sido obstaculizada por la dificultad de crear funciones de transferencia efectivas y fáciles de usar [8]. Dentro de las metodologías existentes para la generación de funciones de transferencia, las que diseñan la función de forma manual pueden ser tardadas y tediosas, mientras que aquellas de generación automática ahorran tiempo en la tarea de diseño de la función de transferencia, pero a cambio, evitan la retroalimentación del usuario

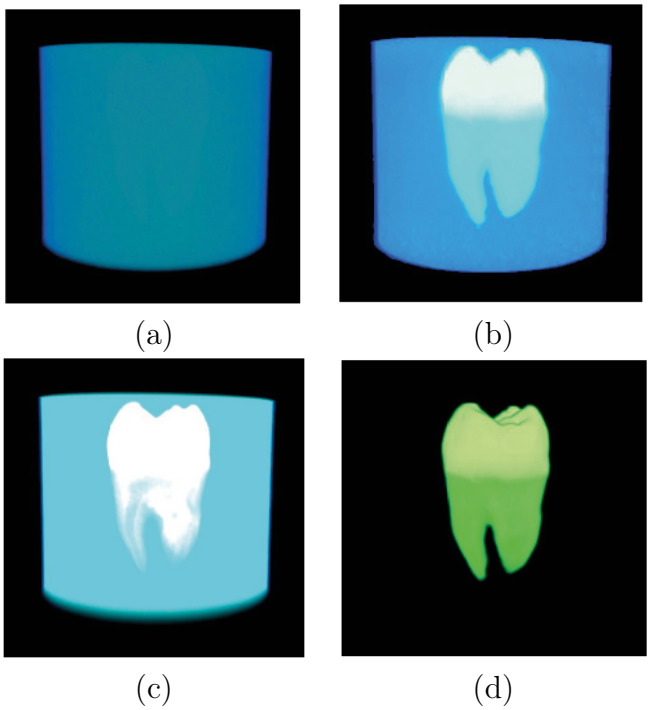


Figura 2.7: Visualización de datos volumétricos de un diente cuyos valores fueron asignados por distintas funciones de transferencia: (a) usando los valores de intensidad originales, (b) usando máxima intensidad a través de la trayectoria de integración del rayo, (c) usando suma a lo largo de la trayectoria de integración del rayo y (d) usando búsqueda manual de la función de transferencia (Imágenes obtenidas con *Amira*®).

que podría mejorar la imagen resultante, o incluso pueden generar una imagen que no sea útil para el usuario. En general, la elección del método adecuado depende de la aplicación particular.

Existen funciones de transferencia básicas en la literatura, como lo son las funciones de máxima intensidad (en esta metodología se toma el valor máximo en la trayectoria de integración, véase la Fig. 2.7(b)), valor mínimo (en esta metodología se toma el valor mínimo en la trayectoria de integración de la composición) o el promedio (en esta metodología se asigna el promedio de los vóxeles en la trayectoria de integración de la composición). Estas funciones operan bien en los casos específicos para los que fueron diseñadas. Para casos más complejos, se han desarrollado diversos métodos para diseñar funciones de transferencia; según [8], los principales enfoques para el diseño de funciones de transferencia son (*i*) prueba y error, con ayuda mínima de la computadora, (*ii*) centrado en los datos, sin modelo subyacente supuesto, (*iii*)

centrado en los datos, usando un modelo subyacente y (*iv*) centrado en la imagen, usando muestreo organizado. Dentro del enfoque (*i*), se considera que la creación de la función de transferencia es una parte del proceso de exploración de los datos y que la intervención de la computadora debe ser mínima, utilizando sistemas como el *VolView* [40] de *Kitware* presentado en [4] para diseñar manualmente la función de transferencia.

El enfoque (*ii*) utiliza el cálculo de métricas sobre el campo escalar para definir un conjunto de funciones de transferencia; particularmente, se ha propuesto el uso del "espectro de contorno" (*contour spectrum*, donde cada isocontorno puede considerarse equivalente a una isosuperficie) para diseñar una interfaz (en 2D) que despliega un árbol de contornos [41], donde se puede elegir la isosuperficie a mostrar y una curva de gradiente integral permite separar los distintos materiales; esta aproximación también ha sido paralelizada [5].

El enfoque (*iii*) considera que lo importante es resaltar aquellos valores asociados a los límites entre materiales, la implementación en [6] y [42] hace uso de un histograma de volumen y conceptos de detección de bordes (creando un mapa de distancias entre el histograma y la primera y segunda derivadas para representar la relación entre los valores y la cercanía a un borde), además de proveer una interfaz para manejo de funciones de transferencia multidimensionales (implementada en el programa *Simian* [43]).

El enfoque (*iv*) consiste en crear "galerías" de funciones de transferencia, donde el usuario puede recorrer la galería y elegir la visualización que considere más acorde a sus necesidades, tal y como se presenta en [7].

Adicionalmente, se ha propuesto el uso de histogramas de visibilidad [18], y el uso de semillas para seleccionar áreas de interés [44] y hacer renderizado local. En esta última aproximación los valores de opacidad pueden ser cambiados como una función de distancia desde la ubicación de la semilla, pero el renderizado es estático y puede considerarse un antecedente directo al proyecto presentado en este documento. Asimismo, existe software comercial para realizar volume rendering (por ejemplo *Amira*® [45]); sin embargo, el estándar para la elección de funciones de transferencia

sigue siendo la asignación manual de propiedades.

Se puede afirmar que el problema de diseñar una función de transferencia adecuada equivale a un problema de segmentación, puesto que ambos buscan identificar y resaltar vóxeles con propiedades similares.

Este trabajo pretende contribuir a elegir una buena función por medio de la segmentación y de esta forma dar propiedades visuales comunes a aquellos vóxeles que representan un objeto. Además, se demuestra que se pueden generar funciones de transferencia utilizando un método de segmentación difusa cuya implementación es semi-automática y relativamente rápida, lo cual tiene como beneficio permitir la interactividad en la creación de las funciones de transferencia. Las bases del algoritmo de segmentación se describirán con mayor detalle en el siguiente capítulo.

Capítulo 3

Segmentación Difusa

Se le llama segmentación de imágenes a aquellos métodos que permiten separar, extraer, definir o etiquetar (y por lo tanto identificar) objetos o regiones de interés en una imagen. La segmentación es un proceso fundamental en diversas áreas que, ya sea, hacen uso, analizan, procesan o visualizan imágenes digitales ya que ayuda al proceso de entendimiento de una imagen.

La segmentación de imágenes (sobre todo aquellas que no son triviales) es una de las tareas más difíciles del procesamiento de imágenes, puesto que su precisión determina el eventual éxito o fracaso de los procedimientos de análisis [11]. A continuación se hará una breve revisión de algunos métodos de segmentación, para posteriormente centrarnos en una implementación basada en conceptos de lógica difusa y su relación con la visualización directa de volúmenes. Aunque en principio se hablará de imágenes bidimensionales, los conceptos y metodologías presentadas pueden, en muchos casos, extenderse a tres dimensiones.

3.1. Métodos de Segmentación

Comúnmente, los métodos de segmentación agrupan partes de una imagen en unidades que son homogéneas con respecto a una o más características (están basados ya sea en propiedades de discontinuidad o en propiedades de similitud presentes en la imagen) [46]. Aunque se han propuesto varios métodos para realizar la segmentación de imágenes, estos enfoques pueden agruparse en dos tipos de técnicas: las técnicas

basadas en fronteras (o bordes) y técnicas basadas en regiones.

3.1.1. Técnicas basadas en Frontera

El proceso realizado por los esquemas basados en frontera puede generalizarse en dos pasos: primero realizan una detección de los bordes locales usando algún tipo de diferenciación y después se realiza una agrupación de esos bordes locales en contornos que separen los vóxeles objeto de los vóxeles del fondo. Los métodos más establecidos implican detección de bordes, pero también destacan los modelos deformables y los modelos basados en morfología [47].

La base de los modelos deformables se encuentra en la confluencia de la geometría, la física y la teoría de aproximación [14]. La infraestructura geométrica de los modelos deformables permite cubrir una amplia variedad de formas utilizando representaciones que permitan varios grados de libertad (por ejemplo, los *splines*), y que están gobernadas por principios físicos y generalmente no evolucionan independientemente. La interpretación física ve a los modelos deformables como cuerpos elásticos que responden naturalmente a fuerzas aplicadas bajo ciertas restricciones. Usualmente se incluyen términos que restrinjan la suavidad o la simetría del modelo. El objetivo final es minimizar la energía asociada al contorno actual del modelo, como la suma de energía interna y externa. Se asume que la energía externa es mínima cuando la forma del modelo está sobre la posición del borde del objeto que se está buscando y que la energía interna es mínima cuando la forma del modelo es similar la forma del objeto buscado.

Por ejemplo, los modelos *active shape* [48] aprovechan el conocimiento que se tiene de la forma y estructura de la clase de objetos que se desean segmentar para generar un modelo plantilla. Este modelo puede deformarse para ajustar mejor las variaciones de los objetos presentes en la imagen con respecto al promedio, pero al mismo tiempo restringe las deformaciones para que los modelos siempre presenten características específicas de esa clase de objetos.

3.1.2. Técnicas basadas en Regiones

La meta de las técnicas basadas en regiones es usar las características de la imagen para mapear los píxeles de una imagen de entrada a conjuntos llamados regiones. Dentro de estas técnicas se encuentran los métodos de *clustering*, o agrupamiento, tales como el k-medias, métodos de *split-and-merge* y métodos basados en teoría de gráficas, entre otros. Por ejemplo, el procedimiento de crecimiento de regiones agrupa píxeles o subregiones en regiones más grandes según un criterio predefinido. El enfoque básico es empezar con un conjunto de puntos "semilla" y hacer que crezcan regiones a partir de ellas añadiendo a cada semilla aquellos píxeles/vóxeles, conectados de alguna forma a ella, que tengan propiedades similares a la semilla. La selección del criterio de similitud depende no solo del problema, si no del tipo de datos de imagen disponibles.

3.1.2.1. Transformada *Watershed*

Una de las técnicas emblemáticas de segmentación basada en el crecimiento de regiones es la transformada *watershed* presentada en [49, 50], el principio básico consiste en considerar una imagen discreta $v(\mathbf{k})$ como una superficie topológica (creada al utilizar las dos coordenadas espaciales y una coordenada de niveles de gris). Este método utiliza la analogía de que los valores de la imagen representan alturas y por lo tanto las regiones homogéneas se pueden considerar como cuencas rodeadas por elevaciones. Para hallar mínimos, máximos y puntos de inflexión se utiliza el gradiente de la función v.

Una analogía del proceso de segmentación es la siguiente. Cuando la función $|\nabla v(\mathbf{k})|$ tiene varios mínimos y máximos locales y se tira una gota de agua virtual sobre la superficie formada por v, ésta fluirá hasta alcanzar un mínimo de altura local. El conjunto de todos los puntos de la superficie que son mínimos locales, y por lo tanto acumulan gotas tiradas desde varios puntos, se conoce como cuencas de captación. Por lo tanto, hay tantas cuencas de captación como mínimos locales en $|\nabla v(\mathbf{k})|$ y cada uno tiene una etiqueta única. Si suponemos que estas cuencas reci-

ben agua constantemente, varias de ellas pueden llegar a traslaparse y unirse, esto sucede cuando existen puntos en común (es decir, puntos que equidistan de mínimos locales distintos) entre dichas cuencas. Para evitar esta unión de cuencas el algoritmo construye diques entre ellas.

El principal inconveniente de la segmentación con *watershed* es que produce muchas regiones y resultados sobre-segmentados, lo cual podría provocar que el resultado no sea útil [11, 51, 52], una solución a este problema es marcar las cuencas importantes. A menudo, la transformación *watershed* es considerada un paso intermedio en un método híbrido de segmentación. Este es el caso del método de segmentación por crecimiento de regiones basado en *watershed* y presentado en [51]. Inicialmente, se eligen algunas cuencas "semilla" y el algoritmo empieza a unir otras cuencas a las regiones iniciales para llevar a cabo el crecimiento de regiones. Para tratar de conservar la homogeneidad de la región se eliminan aquellas cuencas cuya desviación estándar con respecto a la región actual sea mayor a cierto umbral. La secuencia se repite hasta que el resultado de dos iteraciones consecutivas es idéntico.

De igual forma, se ha demostrado la posibilidad de implementar la transformación *watershed* para volúmenes, como es el caso de los mapas de densidad de electrones obtenidos por medio de microscopía electrónica de transmisión [52]. Este algoritmo asume que la altura de un relieve topográfico de cuatro dimensiones corresponde a los valores de densidad en los datos 3D del volumen original, este relieve es sumergido en el equivalente en 4D a un lago. Esta operación se empieza en el mínimo de densidad global y se intensifica en intervalos regulares determinados previamente. En cada paso, se evalúa la conexión de cada vóxel encontrado en el nivel de densidad actual a las cuencas de captación existentes y su cercanía a algún dique. Si el vóxel evaluado no es parte de una cuenca ni de un dique, se crea una cuenca nueva en esa posición. Una vez procesados todos los vóxeles de un nivel de densidad, se incrementa el nivel y se repite el procedimiento.

3.1.2.2. Conectividad Difusa

Los algoritmos de segmentación basados en los principios de la lógica difusa, y más específicamente en el concepto de *conectividad difusa*, han demostrado cierta eficacia bajo diferentes condiciones de ruido, textura y artefactos [17, 53, 54, 55]. Este conjunto de algoritmos es conocido comúnmente como métodos de *segmentación difusa*.

Estos métodos suponen que las propiedades que definen la pertenencia de un punto a un objeto, en una imagen, cambian suavemente. De este modo, puede asignarse un valor dentro del intervalo [0, 1] al grado de pertenencia de un punto a un objeto, donde un valor de 1 indica que el punto es miembro del conjunto de puntos del objeto con toda certeza y un valor de 0 indica que el punto no es miembro del conjunto del objeto.

En general, estos algoritmos requieren una intervención mínima del usuario para identificar puntos, comúnmente denominados *semillas*, dentro de los objetos de interés, para después calcular la similitud entre las semillas y el resto de los puntos de la imagen. La similitud entre cualquier par de puntos de la imagen se calcula usando una *función de conectividad difusa*, ésta es una función global que depende de una relación de *afinidad difusa*, la cual mide localmente la similitud entre dos puntos adyacentes. Típicamente, una relación de afinidad difusa tiene dos componentes, uno para capturar las características de los objetos, y otro para capturar su homogeneidad.

La robustez de la teoría base de los algoritmos de segmentación difusa y las pocas restricciones que existen para designar una relación de afinidad, permite diseñar funciones de afinidad apropiadas para la tarea en turno.

3.2. Principios de la Segmentación Difusa

Este trabajo utiliza el método de segmentación presentado en [16, 17, 56], el cual permite la segmentación difusa simultánea de múltiples objetos. Siguiendo con las definiciones presentadas en (2.1) y (2.4), tenemos que $v(\mathbf{c})$ es una función discreta

que representa valores de intensidad para un punto $\mathbf{c}$ y está definida en un subconjunto de la rejilla Γ_Δ. El algoritmo de segmentación difusa es una función σ que para M objetos mapea cada elemento $\mathbf{c} \in \Gamma_\Delta$ a un vector de $(M+1)$ dimensiones $\sigma^{\mathbf{c}} = (\sigma_0^{\mathbf{c}}, \sigma_1^{\mathbf{c}}, \ldots, \sigma_M^{\mathbf{c}})$ de forma tal que $\sigma_0^{\mathbf{c}} \in [0,1]$. En este vector por lo menos existe algún valor de m, en el rango $1 \leq m \leq M$, para el cual $\sigma_m^{\mathbf{c}} = \sigma_0^{\mathbf{c}}$ y para cualquier otra posición m del vector el valor es cero o $\sigma_0^{\mathbf{c}}$.

La segmentación difusa asume que el valor de densidad de un vóxel puede definir si es miembro o no de un objeto y que esa propiedad cambia suavemente, de igual forma, supone que la imagen o volumen puede representarse como un gráfico con nodos y aristas, tal y como se muestra en la Fig. 3.1. La teoría de conjuntos difusos indica que el grado de pertenencia a un conjunto puede ser determinado por una función que indica el grado de certeza con que el elemento cumple la condición de pertenencia al conjunto. En este caso particular, esa relación difusa puede representarse por medio de una gráfica, donde los vóxeles son los nodos y los grados de pertenencia representan los pesos de las aristas, como se muestra en la Fig. 3.2.

En esta representación se puede definir una *cadena* como una secuencia $\mathcal{P}\left(\mathbf{c}^{(0)}, \mathbf{c}^{(L)}\right) = \left(\mathbf{c}^{(0)},\ \mathbf{c}^{(1)},\ \ldots,\ \mathbf{c}^{(L)}\right)$ de $(L+1)$ nodos, donde sus *ligas* son dos nodos consecutivos $\left(\mathbf{c}^{(l-1)},\ \mathbf{c}^{(l)}\right)$ en la secuencia. Una *función de afinidad difusa* ψ permite asignar el *peso de las ligas*; mientras que el *peso de la cadena* se determina con el peso de su liga más débil. Para dos puntos cualquiera $\mathbf{c}$ y $\mathbf{d} \in \Gamma_\Delta$ su similitud, o conectividad, se encuentra definida por el peso de la cadena más fuerte que conecta a esos dos puntos. La conectividad entre dos puntos $\mathbf{c}$ y $\mathbf{d}$ se define por medio de la siguiente función

$$\mu\left(\mathbf{c},\, \mathbf{d}\right) = \max_{\substack{\mathcal{P}\left(\mathbf{c}^{(0)}, \mathbf{c}^{(L)}\right) \\ \mathbf{c}=\mathbf{c}^{(0)}, \mathbf{d}=\mathbf{c}^{(L)}}} \left[\min_{1 \leq l \leq L} \psi\left(\mathbf{c}^{(l-1)}, \mathbf{c}^{l}\right) \right]. \tag{3.1}$$

Un algoritmo de segmentación difusa calcula la conectividad difusa de cada nodo en la gráfica (en otras palabras, cada píxel o vóxel en la imagen) al conjunto de nodos *semilla*, los cuales representan puntos en la imagen que pertenecen con toda certeza a los objetos a ser segmentados. Normalmente, la tarea de selección de semillas es realizada por un usuario entrenado debido a que, típicamente, el ser humano puede

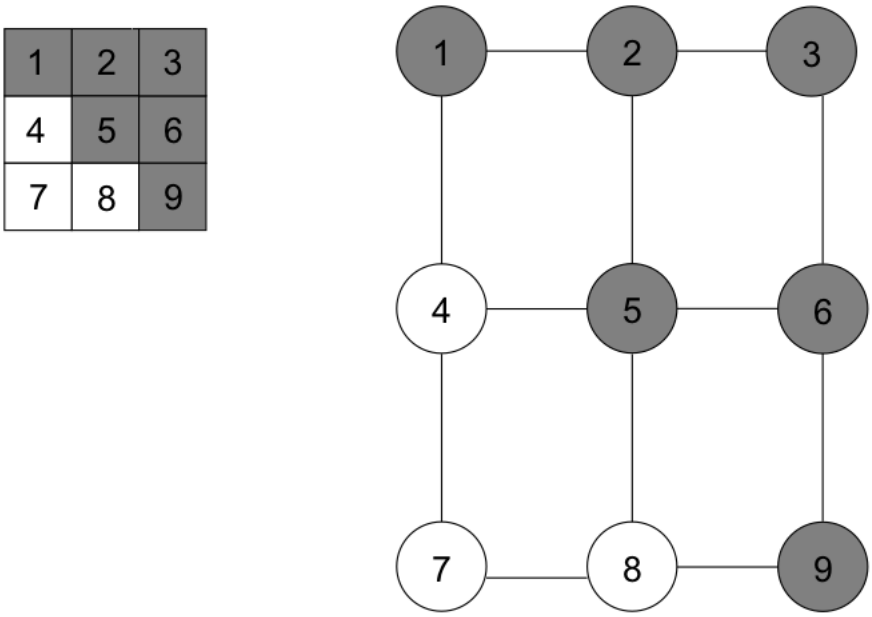


Figura 3.1: Representación de una imagen de tamaño 3×3 como una gráfica, donde cada píxel corresponde a un nodo y las aristas representan un tipo de adyacencia entre los nodos (por "cara", en este caso).

reconocer mejor los objetos de una imagen que una computadora.

Buscar la conectividad de todos los puntos de la imagen a los puntos semilla es la tarea mas cara del proceso de segmentación difusa, los algoritmos propuestos en [57] y [58] lo hacen de manera eficiente usando programación dinámica y algoritmos avaros (*greedy*). Es importante notar que estos algoritmos buscan la conectividad de los puntos semillas, que representan solo un objeto, al resto de los puntos en la imagen. La extensión para segmentar varios objetos simultáneamente fue presentada en [56] y en [59]. Para acelerar el algoritmo presentado en [56], Carvalho et al. [16] sugieren un algoritmo que discretiza el rango $[0, 1]$ y éste es el algoritmo que usamos en este trabajo.

Este algoritmo de segmentación difusa procede asignando las conectividades entre cada punto de la imagen y los puntos semilla en pasos iterativos discretos. El algoritmo también asume que es posible hallar un camino entre cualquier par de puntos de una

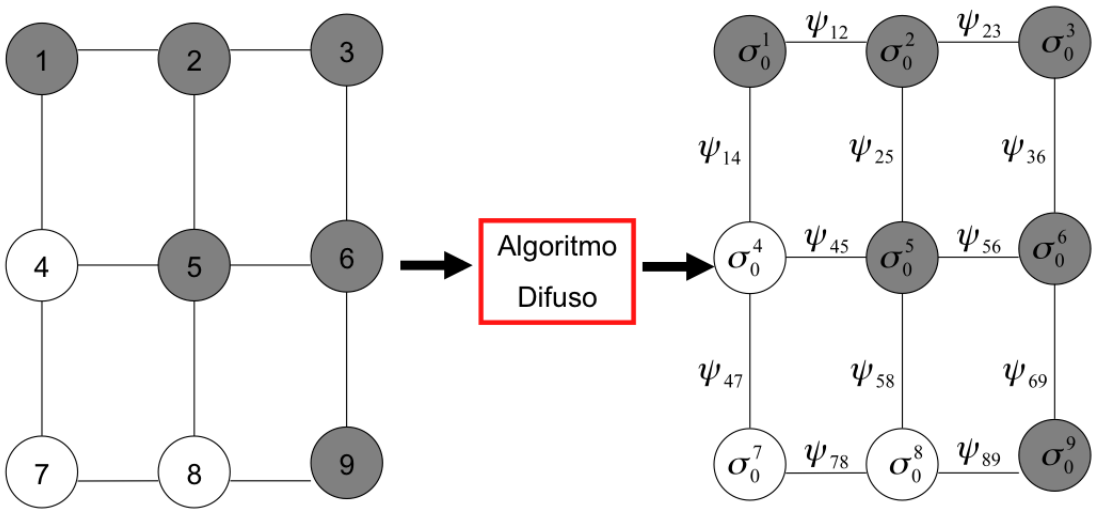


Figura 3.2: El algoritmo de segmentación difusa realiza un mapeo de la gráfica que representa la imagen a una gráfica difusa, donde a cada nodo corresponde el valor $\sigma_0^{\mathbf{c}}$ y cada arista tiene un peso $\psi_{\mathbf{c},\mathbf{d}}$.

imagen. El rango $[0, 1]$ de conectividades se discretiza en el rango $[0, \ldots, MAX]$, donde MAX es un número entero positivo. Al inicio del algoritmo se asigna una conectividad 0 para todos los puntos de la imagen excepto para los puntos semillas, a los cuales se les asigna el valor MAX. En cada iteración k aquellos puntos cuya conectividad sea igual a $MAX + 1 - k$ trataran de hacer crecer la región en la que se encuentran al enviar exploradores a otras regiones de la imagen. Cuando los exploradores llegan a una nueva región tratarán de reclamar los puntos de esa región como parte de sus regiones respectivas. Primero, los exploradores calculan el valor de conectividad de los nuevos puntos. El valor de conectividad de los nuevos puntos explorados es igual al mínimo entre el valor $MAX + 1 - k$ y el valor de afinidad que existe entre los puntos de la región ya ocupada y la región recién explorada. Los exploradores toman posesión de los puntos nuevos de la siguiente forma. Si el valor de conectividad que tienen actualmente los puntos en la nueva región es menor que la nueva conectividad calculada por los exploradores, entonces los exploradores reclaman los puntos para cada una de sus regiones con la conectividad recién calculada. Si la conectividad calculada por los exploradores es menor que la conectividad existente en los puntos

nuevos, entonces las regiones recién exploradas no cambian de dueño. Finalmente, si la conectividad calculada por los exploradores es igual a la ya existente en las regiones recién exploradas, entonces los puntos de éstas son repartidos equitativamente entre las regiones de los exploradores y las regiones que ya los poseían. El algoritmo termina en la iteración MAX y provee para cada clase un conjunto de puntos los cuales tienen asignados los valores de conectividad a sus puntos semillas.

La implementación del algoritmo permite segmentar M objetos con la posibilidad de que cada objeto (o clase) tenga su propia función de afinidad ψ y su propio conjunto de nodos semilla. El resultado final define a un objeto como el conjunto de elementos que están más fuertemente conectados a una de las semillas de ese objeto que a cualquier semilla de otro objeto. Para la segmentación de M objetos, el algoritmo recibe como entrada los M conjuntos de semillas $(S_1,\ S_2,\ \ldots,\ S_M)$, las M funciones de afinidad $(\psi_1,\ \psi_2,\ \ldots,\ \psi_M)$ y el volumen v a segmentar, y como salida se obtiene la segmentación difusa única $(\sigma_1,\ \sigma_2,\ \ldots \sigma_M)$ que es consistente con los datos de entrada [16, 56].

3.2.1. Afinidad Difusa

Es evidente de (3.1) que la función de afinidad ψ es crucial para el funcionamiento de los algoritmos de segmentación difusa. La importancia de esta relación reside en su influencia en los resultados y el desempeño de los algoritmos. En el caso de la rejilla Γ, una relación de afinidad difusa ψ es una función que tiene las siguientes propiedades

$$\psi : \Gamma \times \Gamma \to [0,\ 1] \tag{3.2}$$

y

$$\psi\left(v(\mathbf{c}),\ v(\mathbf{d})\right) = 0,\ \text{si } \mathbf{c} \text{ y } \mathbf{d} \text{ no son adyacentes.} \tag{3.3}$$

Para definir a la función ψ solo se necesita que ésta sea monotónica y normal (aunque hay quiénes creen que además debe ser simétrica [60, 61]), así como que considere

las características y la homogeneidad de los objetos. La propiedad (3.3) es conocida como *localidad*. La función de afinidad difusa tiene la siguiente forma general [55, 57]

$$\psi\left(\mathbf{c},\ \mathbf{d}\right) = \psi_A\left(\mathbf{c},\ \mathbf{d}\right)\left[w_H\psi_H\left(\mathbf{c},\ \mathbf{d}\right) + w_O\psi_O\left(\mathbf{c},\ \mathbf{d}\right)\right], \tag{3.4}$$

donde ψ_A es una relación de adyacencia entre los píxeles $\mathbf{c}$ y $\mathbf{d}$ que típicamente tiene la forma

$$\psi_A\left(\mathbf{c},\ \mathbf{d}\right) = \begin{cases} 1, & \text{si } \left(\mathbf{c},\ \mathbf{d}\right) \in \rho, \\ 0, & \text{en cualquier otro caso,} \end{cases}$$

donde ρ es una relación binaria entre dos puntos $\mathbf{c}$ y $\mathbf{d}$. Para nuestra aplicación esta relación se define como $\left(\mathbf{c},\ \mathbf{d}\right) \in \rho \Leftrightarrow \|\mathbf{c}-\mathbf{d}\| = 1$ (típicamente conocida como vecindad por cara o adyacencia). Por otra parte, el componente ψ_O de (3.4) es el encargado de medir el grado de similitud de las características de intensidad de los objetos esperados. Finalmente, ψ_H mide el grado de homogeneidad entre las intensidades de los puntos $\mathbf{c}$ y $\mathbf{d}$. Las funciones ψ_O y ψ_H las hemos definido como [16, 53, 56, 57, 58]

$$\psi_O(v(\mathbf{c})+v(\mathbf{d})) = e^{-\frac{(v(\mathbf{c})+v(\mathbf{d})-m_o)^2}{2\varsigma_o^2}} \quad \text{y} \quad \psi_H(|v(\mathbf{c})-v(\mathbf{d})|) = e^{-\frac{\left(|v(\mathbf{c})-v(\mathbf{d})|-m_H\right)^2}{2\varsigma_H^2}}, \tag{3.5}$$

donde m_o y ς_o son la media y la desviación estándar de la suma $v(\mathbf{c})+v(\mathbf{d})$, mientras que m_H y ς_H son la media y la desviación estándar de $|v(\mathbf{c})-v(\mathbf{d})|$. Las funciones ψ_O y ψ_H son ponderadas por w_O y w_H que son dos constantes a las que normalmente se les da el valor de 0.5, aunque se han investigado los efectos de estas constantes y la forma de ajustarles durante el proceso de segmentación [62].

En [60, 61] se desarrolla y demuestra el término *equivalencia de afinidades*, que implica que a pesar del uso de funciones diferentes, se obtenga una misma segmentación. El análisis presentado en esos artículos muestra que desde el punto de vista de las metodologías de conectividad difusa, el único elemento esencial de una función es su orden y plantea la importancia de generar funciones y combinaciones de afinidades que en realidad sean diferentes unas de otras.

3.2.2. Medida de textura

La segmentación basada en textura es importante, especialmente para detectar información donde los defectos y anormalidades en las estructuras de los objetos se muestran como una diferencia en textura [46].

La componente de homogeneidad definida en (3.5) mide el grado de similitud entre valores de v para puntos adyacentes, de tal forma que cuando los valores de intensidad de los puntos adyacentes son más cercanos, el valor de ψ_H es mayor. No existe una definición estándar para describir texturas, pero se ha reconocido que la idea de textura es una función de variación espacial en los valores de los puntos de una imagen, así como una relación espacial entre ellos. Por lo tanto, la componente ψ_H se puede considerar como una medida simple de textura y hace posible usar otras medidas para realizar una mejor medición de textura.

Una aproximación estadística puede ser apropiada para describir texturas puesto que éstas pueden considerarse como la realización de un proceso estocástico subyacente. Los métodos estadísticos analizan la distribución espacial de los valores de los píxeles, al calcular las características locales en cada punto de la imagen y derivar/generar un conjunto de estadísticas a partir de las distribuciones de estas características locales. Para calcular las estadísticas podemos tomar dos enfoques, la estadística de primer orden y la estadística de segundo orden. La estadística de primer orden es calculada a partir de la probabilidad de observar el valor de un punto seleccionado de la imagen en forma aleatoria. La estadística de primer orden depende solo de los valores individuales de los puntos y no de interacción con los valores de los puntos vecinos. La estadística de segundo orden es calculada a partir de la probabilidad de observar los valores de una pareja de puntos en la imagen que están apartados por cierta distancia.

Un problema importante que se presenta cuando se desea incorporar una descripción de textura en el componente ψ_H es el grado de complejidad computacional, el cual debe ser relativamente bajo dado que el cálculo de este componente solo es una parte de todo el sistema de segmentación difusa.

3.2.3. Algoritmo de segmentación difusa

A continuación, presentamos el algoritmo de segmentación difusa utilizado en esta tesis y propuesto en [16]. El algoritmo recibe como entradas la funcion discretizada υ, el conjunto de funciones de afinidad $\{\psi_m\}$ y el conjunto de semillas $\{S_m\}$ para $m \in [1, \ldots, M]$, donde M es el número total de objetos a segmentar. La función $\sigma_m^{\mathbf{c}}$ representa el grado de pertenencia del vóxel $\mathbf{c}$ al objeto m, mientras que $\sigma_0^{\mathbf{c}}$ es el valor de pertenencia máximo que se le ha dado al vóxel $\mathbf{c}$ (el cual es proporcional a la fuerza de conexión del vóxel $\mathbf{c}$ en la clase m). La clase m lo reclama si, y sólo si, $\sigma_m^{\mathbf{c}} = \sigma_0^c > 0$.

Para hacer más rápido el algoritmo, se hace la suposición de que el conjunto de afinidades difusas es siempre un subconjunto fijo $A \cup \{1\}$ con cardinalidad K y sean $1 = a_1 > a_2 > \ldots > a_K > 0$ los elementos de A. Además, en muchas aplicaciones la calidad de una segmentación no se ve significativamente afectada si se redondea cada afinidad difusa a tres decimales, entonces se puede tener que $A = \{0{,}001,\ 0{,}002,\ \ldots,\ 0{,}999,\ 1{,}000\}$, de forma que $K = MAX$ y los elementos de A sean $a_k = 1{,}001 - k/MAX$. Adicionalmente, se tiene un arreglo U con tamaño $M \times K$, donde $U\,[m]\,[k]$ representa el conjunto de vóxeles que están ocupados por la clase del objeto m y cuya fuerza actual es $MAX + 1 - k$.

El proceso se inicializa en los Pasos 1-9, donde se le da a $\sigma_m^{\mathbf{c}}$ el valor de 0 por cada vóxel $\mathbf{c}$ para $0 \leq m \leq M$ y se asegura de que los conjuntos $U\,[m]\,[k]$ estén vacíos. Luego, para cada vóxel $\mathbf{c} \in S_m$ se les incluye en el conjunto $U\,[m]\,[1]$ y se asigna a $\sigma_0^{\mathbf{c}}$ y $\sigma_m^{\mathbf{c}}$ un valor igual a 1.

Posteriormente, se realiza el ciclo principal del algoritmo en los Pasos 11-23, para cada iteración k hasta K y para cada objeto m hasta M, se lleva a cabo la actualización de los valores de $\sigma_m^{\mathbf{c}}$. Un valor actual es reemplazado por uno mayor si se encontró que existe una cadena del objeto m desde una semilla en S_m hasta $\mathbf{c}$ con una fuerza mayor al valor anterior, y es reemplazado por 0 si se encuentra que (para toda $n \neq m$) hay una cadena desde una semilla en S_n hasta $\mathbf{c}$ con una fuerza mayor al valor anterior de $\sigma_m^{\mathbf{c}}$.

Algoritmo 1 El algoritmo Secuencial, y Rápido, de Segmentación Difusa (FSFS) presentado en [16].

1: **para** $\mathbf{c} \in \Gamma$ **hacer**
2: **para** $m \leftarrow 0$ **a** M **hacer**
3: $\sigma_m^{\mathbf{c}} \leftarrow 0$
4: **para** $m \leftarrow 1$ **a** M **hacer**
5: **para** $k \leftarrow 1$ **a** K **hacer**
6: $U[m][k] \leftarrow \emptyset$
7: **para** $c \in S_m$ **hacer**
8: $\sigma_0^{\mathbf{c}} \leftarrow \sigma_m^{\mathbf{c}} \leftarrow 1$
9: $U[m][1] \leftarrow S_m$
10: **para** $k \leftarrow 1$ to K **hacer**
11: **para** $m \leftarrow 1$ to M **hacer**
12: **mientras** $U[m][k] \neq \emptyset$ **hacer**
13: **retirar** un punto $\mathbf{d}$ de $U[m][k]$
14: $C \leftarrow \{\mathbf{c} \in \Gamma \,|\, \sigma_m^{\mathbf{c}} < \text{mín}\,(a_k, \psi_m\,(\mathbf{d}, \mathbf{c}))$ y $\sigma_0^{\mathbf{c}} \leq \text{mín}\,(a_k, \psi_m\,(\mathbf{d}, \mathbf{c}))\}$
15: **mientras** $C \neq \emptyset$ **hacer**
16: **retirar** un punto $\mathbf{c}$ de C
17: $t \leftarrow \text{mín}\,(a_k, \psi_m\,(\mathbf{d}, \mathbf{c}))$
18: **si** $\sigma_0^c < t$ **entonces hacer**
19: **retirar** $\mathbf{c}$ de cada conjunto en U que lo contiene
20: **para** $n \leftarrow 1$ **to** M **hacer**
21: $\sigma_n^{\mathbf{c}} \leftarrow 0$
22: $\sigma_0^{\mathbf{c}} \leftarrow \sigma_m^{\mathbf{c}} \leftarrow t$
23: **insertar** $\mathbf{c}$ en el conjunto $U[m][l]$, donde l es el entero tal que $a_l = t$

Al inicio de cada iteración k, para cada índice l en el rango $k \leq l \leq K$, el conjunto $U\,[m]\,[l]$ contiene todos aquellos puntos que están ocupados por la clase m y cuya fuerza actual es $MAX + 1 - l$. Durante la iteración k, exploradores de la clase m son enviados desde cada punto en $U\,[m]\,[k]$ a todos los otros puntos (Pasos 11-14). Cuando un explorador de la clase m tiene éxito en ocupar otro punto $\mathbf{c}$, y la fuerza resultante de $\mathbf{c}$ es $MAX + 1 + l$, se inserta $\mathbf{c}$ en el conjunto $U\,[m]\,[l]$ (Paso 23). Sin embargo, si $\mathbf{c}$ es tomado porque la fuerza anterior era menor a $MAX + 1 - l$, entonces es necesario remover primero a $\mathbf{c}$ de cualquier conjunto que lo contenga (Paso 19).

Finalmente, pueden obtenerse como resultado las segmentaciones σ_1, σ_2, $\ldots$, σ_M recorriendo los elementos en los conjuntos $U\,[1]\,,\ U\,[2]\,,\ \ldots\,,\ U\,[M]$. La Fig. 3.3 muestra el resultado de una segmentación realizada mediante la aplicación de este método.

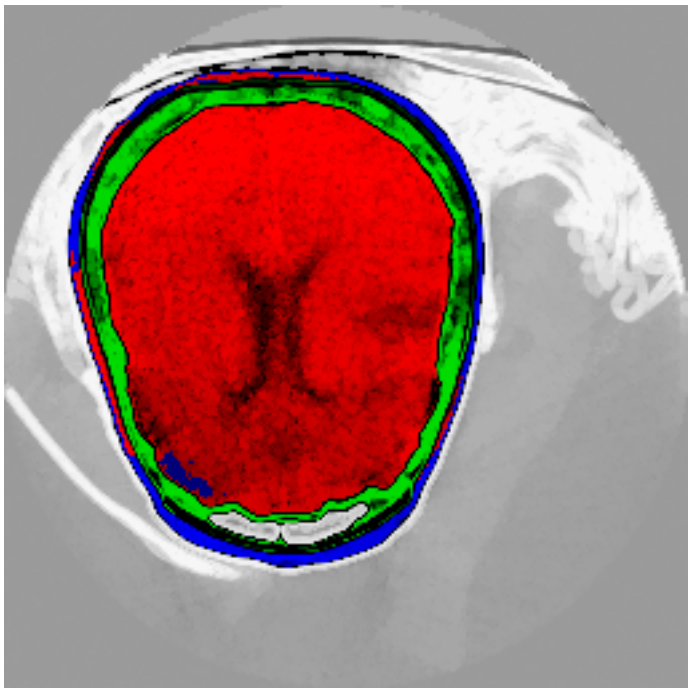


Figura 3.3: Corte del resultado de la aplicación del algoritmo de segmentación difusa sobre datos volumétricos del estudio de una cabeza por tomografía computarizada (CAT), el mismo conjunto utilizado en la Fig. (2.2). Pueden observarse cuatro clases: cerebro (rojo), cráneo (verde), piel (azul) y fondo (gris). Las tonalidades oscuras indican menor grado de pertenencia que las tonalidades claras.

3.3. Diseño de Funciones de Transferencia

Como se mencionó en el Capítulo 2, el problema del diseño de funciones de transferencia es equivalente al problema de realizar una segmentación, puesto que se trata de identificar objetos que comparten ciertas propiedades. Identificar regiones de interés dentro de un volumen está muy relacionado con el problema de identificar regiones de interés en una imagen [44].

De forma similar al argumento empleado para el diseño de funciones de transferencia, es deseable que los algoritmos de segmentación faciliten y agilicen el proceso de delimitar e identificar objetos, ya que la segmentación manual (el estándar en muchos procesos de diagnóstico médico) es un procedimiento que tiende a ser tedioso, tomar mucho tiempo y, de alguna forma, ser subjetivo [52]. El método de segmentación elegido es semi-automático, aunque existe la posibilidad de automatizar el proceso de selección de semillas; sin embargo, eliminar por completo la participación humana no

es del todo deseable puesto que se supone que un ser humano experimentado es mejor para reconocer en donde están los objetos, mientras que típicamente las computadoras superan a los humanos en delinear dichos objetos [57].

Una característica particular de esta implementación de segmentación difusa es que se obtiene un resultado diferente para cada configuración de semillas (tanto número como posición) y de funciones de afinidad, a diferencia de otras implementaciones [59], donde no importa donde se pongan las semillas siempre se obtiene la misma segmentación como resultado (siempre y cuando las semillas se coloquen dentro de la misma región). Aunque esta característica podría llegar a considerarse como una desventaja, en realidad representa una de las razones de peso para utilizarlo en la tarea de diseño de funciones de transferencia. En los otros métodos de segmentación difusa (como el presentado en [59]), si el resultado es incorrecto, no existe manera de corregirlo sin hacer uso de algún otro método o truco.

Claramente, si se eligen semillas adecuadas (que representan objetos de un tipo) y se les asignan propiedades gráficas, sería de esperar que la visualización resultante fuera buena. Además, asumiendo que se obtiene una buena segmentación, pero una mala asignación de propiedades visuales, sólo se hace necesario cambiar las propiedades para ver los objetos de interés. Viendo la situación desde el caso contrario, si se obtiene una mala segmentación, pero una buena asignación de propiedades basta con mover las semillas y generar una nueva segmentación.

La finalidad de realizar visualización directa de volúmenes es realizar exploración de datos en tres dimensiones, usar un método de segmentación para generar las funciones de transferencia permite realizar esta exploración de forma más sencilla sin necesidad de usar un sistema experto, puesto que se puede variar el comportamiento del resultado modificando el conjunto de semillas, lo cual vuelve el proceso semi-interactivo. Adicionalmente, existe la posibilidad de paralelizar el algoritmo de segmentación como se demuestra en [63], aprovechando el poder de las nuevas tarjetas gráficas, lo que permitiría lograr un desempeño interactivo.

Capítulo 4

Experimentos y Resultados

En este capítulo se llevará a cabo la descripción de la metodología utilizada para la implementación del proyecto, de los programas utilizados, así como de los experimentos realizados y resultados obtenidos.

4.1. Implementación

Para esta tesis obtuvimos una implementación del algoritmo descrito en la Sección 3.2.3 la cual fue desarrollada en el lenguaje C++ y consta de dos programas separados: un programa con interfaz gráfica para recopilar los datos de entrada y un programa que realiza propiamente el Algoritmo 1. La interfaz gráfica permite definir el número M de clases en las que se va a segmentar la imagen o volumen, así como escoger los conjuntos S_m de puntos semilla y ψ_m de funciones de afinidad para cada clase. La selección de semillas se realiza mediante la exploración del volumen plano por plano, dicha exploración puede realizarse en cualquiera de los tres ejes coordenados. Finalmente, el número de clases, el tipo de función de afinidad y su adyacencia correspondiente, el número de semillas para cada clase y la localización de cada semilla son guardados en un archivo con formato especial, el cual puede volver a leerse en el mismo programa para hacer modificaciones. El programa de segmentación difusa lee el volumen a segmentar y el archivo creado con el programa de interfaz gráfica para realizar el proceso descrito en el Algoritmo 1, la segmentación resultante puede ser almacenada en archivos de distintos formatos de imagen, tales como MRC [64] o

TIFF.

Para la realización de este proyecto fue necesario, en primer lugar, analizar y adaptar los programas que realizan la segmentación difusa para que ellos pudiesen leer y guardar otros formatos de imagen y datos, esto con la finalidad de poder utilizar formatos más estándar, además de generar una salida que pudiera ser utilizada como entrada para el proceso de visualización por *volume rendering*. La información que produce el programa de segmentación difusa que se utiliza en el algoritmo de visualización es el vector $\sigma^{\mathbf{c}} = (\sigma_0^{\mathbf{c}}, \sigma_1^{\mathbf{c}}, \ldots, \sigma_M^{\mathbf{c}})$ para todo $\mathbf{c} \in \Gamma_\Delta$, descrito en la Sección 3.2 (en otras palabras, produce una imagen o volumen donde cada uno de sus puntos tiene el valor de conectividad de ese punto al grupo de semillas de clase m).

Para realizar la visualización por medio del método de visualización directa de volúmenes se hizo una búsqueda de programas que implementasen dicha metodología. Se decidió tomar esta ruta porque la principal motivación de esta tesis es demostrar la utilidad de los resultados de la segmentación difusa en la selección de propiedades visuales para la visualización directa de volúmenes y no el desarrollo de un visualizador. Además, en el grupo de trabajo no se tenía una plataforma de desarrollo que implementase la visualización directa de volúmenes, así que implementar uno hubiese consumido todo el tiempo dispuesto para el desarrollo de todo el proyecto sin la garantía de poder incluir nuestra aportación. Finalmente, las implementaciones existentes poseen funcionalidades que probablemente no se habrían podido implementar en nuestro programa con el tiempo disponible, tales como selección de cualquier punto de vista o visualización usando mapeo de texturas.

Para la selección del programa de visualización directa de volúmenes se contemplaron los siguientes programas y bibliotecas: CUDA™ *Software Development Kit* de la compañía NVIDIA® [65], VTK (*Visualization ToolKit*) [66], *Simian* [43] y *Voreen* (*Volume Rendering Engine*) [2]. El ejemplo provisto en CUDA™ tenía poca flexibilidad de modificación y era dependiente del volumen de ejemplo (por ejemplo, el tipo de datos y el tamaño del conjunto), incluyendo la función de transferencia, lo cual dificultaba su modificación para adaptarlo a las necesidades de la tesis. La biblioteca VTK presentó incompatibilidades con el *hardware* y *software* de la computadora

empleada para los experimentos (además de que el ejemplo para poder desarrollar un visualizador directo de volúmenes también estaba demasiado ligado a los datos de entrada de ejemplo como para ser modificado fácilmente). Por otra parte, el programa *Simian*, el cual fue desarrollado por la Universidad de Utah, parecía ser una buena opción puesto que es un programa general dedicado a la visualización directa de volúmenes. Sin embargo, este programa también fue descartado porque su interfaz gráfica no es muy intuitiva, utiliza un formato de datos muy específico y poco conocido, y su código no está lo suficientemente documentado y organizado como para facilitar su modificación. Por lo tanto, la opción elegida para trabajar en este proyecto fue *Voreen* en su Versión 2.6.1.

4.1.1. *Voreen*

Voreen es un programa de código abierto (*open source*) para visualización directa de volúmenes implementado en el lenguaje de programación C++, usa *OpenGL* y GLSL (*GL Shading Language*) para las tareas de graficación, y *Qt* para la interfaz gráfica; todos estándares para el desarrollo de aplicaciones de graficación por computadora. Además, los autores de este programa también lo concibieron como una biblioteca y por lo tanto provee formas de implementar funcionalidades nuevas a través del desarrollo de módulos. Nos inclinamos por el uso de este programa puesto que es una implementación general, permite el uso de formatos de datos más estándar y su estructura y herramientas facilitan la adición e integración de nuevos componentes. Adicionalmente, su interfaz gráfica es más intuitiva que la de otros programas.

Voreen sigue un concepto de flujo de datos en el cual cada procedimiento básico dentro de un módulo es ejecutado por un programa llamado *procesador*; por lo tanto, una tarea es llevada a cabo conectando aquellos procesadores que sean apropiados para solucionar el problema. Cada procesador se comunica con otros a través de *puertos* y cada procesador tiene *propiedades* específicas que permiten modificar el comportamiento del mismo.

Particularmente, existen varios procesadores que realizan *raycasting*, un proceso

que se lleva a cabo en GPU, ya que pueden implementarse ciclos y ramificaciones dinámicas dentro de los programas escritos en GLSL (programas conocidos como *shaders*) que son ejecutados en el procesador gráfico.

4.1.1.1. *Shaders*

Un *shader* es un programa usado para calcular y modificar los efectos de renderizado en el *hardware* gráfico, utilizando el *rendering pipeline* programable de las GPU's. Los *shaders* son llamados desde un programa normal (por ejemplo, un programa escrito en C++) pero pueden ser compilados de forma independiente al programa que lo manda ejecutar. Los *shaders* permiten realizar operaciones de transformación geométrica comunes y funciones de sombreado de píxel, así como personalizar otros efectos [67].

Los *shaders* pueden describir las características de vértices o de píxeles y su implementación depende de las especificaciones de la tarjeta gráfica. Sin embargo, hay tres tipos de shaders de uso común: los *shaders* de píxel o *pixel shaders*, los *shaders* de vértices o *vertex shaders* y los *shaders* geométricos o *geometric shaders*.

Los *vertex shaders* se encargan de describir, entre otras características, la posición, las coordenadas de textura y color de un vértice y es ejecutado una vez por cada vértice dado al procesador gráfico. Su propósito es transformar cada posición 3D de un vértice en una posición virtual en el espacio de las coordenadas 2D en el cual aparece en la pantalla. Aunque los *vertex shaders* pueden manipular las propiedades del vértice, no pueden crear nuevos vértices.

Los *geometric shaders* pueden generar nuevas primitivas gráficas, tales como puntos, líneas y triángulos, a partir de aquellas primitivas enviadas al inicio del *pipeline* gráfico. Los programas *shader* geométricos son ejecutados después de los *vertex shaders*. Toman como entrada una primitiva completa, posiblemente con información de adyacencia. El *shader* puede emitir más primitivas, o ninguna, que son rasterizadas y sus fragmentos son finalmente pasados a los programas *pixel shader*.

Los *pixel shaders*, también conocidos como *fragment shaders*, son los encargados de calcular el color y otros atributos de cada píxel. La complejidad de los *pixel shaders*

puede ser tan simple como siempre asignar el mismo color o tan complicados como aplicar un valor de iluminación, transparencia, sombra o algún otro fenómeno por medio de funciones complejas. Estos *shaders* también pueden alterar la profundidad del píxel (para *z-buffering*) o dar como salida más de un color si varios blancos de despliegue están activos. No produce efectos complejos pero puede operar a nivel de un solo píxel, sin necesidad de tener conocimiento de la geometría de la escena.

4.1.1.2. Implementación de *raycasting*

En la Sección 2.3.2.1 se mencionó como el uso de texturas puede ayudar a realizar visualización directa de volúmenes. Adicionalmente, la evolución del *hardware* para gráficos ha permitido cambiar el *pipeline* descrito en la Sección 2.3, de operaciones relativamente fijas a algo completamente programable con ayuda de los *shaders*. Ahora es posible realizar operaciones aritméticas, de extracción de texturas por fragmento o de reemplazo arbitrario de los valores calculados para profundidad, entre otras. Esto ha permitido mejorar las implementaciones de visualización directa de volúmenes que utilizan las texturas como apoyo.

La implementación del algoritmo de *raycasting* dentro de *Voreen* está basada en las metodologías presentadas en [3, 68]. En el pasado no había ramificaciones (*threading*) reales ni soporte para ciclos en los programas de *shading* disponibles a nivel fragmento. Por lo tanto, las soluciones anteriores (tales como la empleada en [3]) implementan el algoritmo de *raycasting* haciendo múltiples pasos de renderización y pruebas de píxeles adicionales. Las nuevas metodologías permiten hacer solo una pasada para realizar *raycasting* lo cual elimina la necesidad de lecturas y escrituras intermedias de *buffer*; además, como solo se necesita renderizar un polígono para generar los fragmentos necesarios, se necesita poco procesamiento geométrico y poca generación de fragmentos. Por último, permite que el tamaño de pasos sea adaptativo, y por definición, muestrea el volumen a distancias iguales. Además, las implementaciones de *raycasting* en tarjetas gráficas modernas tienen más precisión que el renderizado basado en rebanadas puesto que el algoritmo completo de *raycasting* se realiza con precisión de punto flotante.

El método básico de *raycasting* presenta características que se ajustan bien en la idea de procesamiento paralelo, el cual es intrínseco de los procesadores gráficos. Para cada píxel de la imagen final, un solo rayo es enviado desde la pantalla y su recorrido e interacción a través del volumen es independiente de los demás rayos. La integral de línea de *volume rendering* para cada píxel puede evaluarse de forma aproximada, muestreando el rayo en un número finito de posiciones dentro del volumen. Las contribuciones de todas las muestras a lo largo del rayo se acumulan sobre el color y la opacidad total. Sin embargo, pueden evitarse operaciones que no contribuyen a la imagen final (por ejemplo, no es necesario hacer la composición de aquellas estructuras que no son relevantes y su opacidad es igualada a cero).

En *Voreen* el procesador que lleva a cabo el algoritmo del *raycasting* inicia estableciendo la ecuación paramétrica del rayo, ya que con ella es posible muestrear el volumen. Para determinar la dirección del rayo de visión para el píxel respectivo, es necesario definir el punto de entrada del rayo. Este punto corresponde a la primera intersección con la caja que delimita al volumen (un paralelepípedo recto que cubre el soporte del conjunto de datos). Tomando en cuenta que el volumen se encuentra delimitado por una caja , los puntos de entrada se encuentran tomando las coordenadas 3D de las caras anteriores de esta caja y tratando dichas coordenadas como si fueran componentes de un vector de color. El resultado es una textura 2D que puede observarse en la Fig. 4.1(a) y que tiene la misma resolución que la ventana de vista actual. Los componentes de color en la textura corresponden a lo primeros puntos de intersección entre los rayos y el volumen y sus coordenadas están dadas con respecto al espacio de textura.

De igual forma, pueden determinarse los puntos de salida de los rayos, para esto se usan las caras posteriores de la caja delimitadora para crear una textura 2D, el resultado se puede observar en la Fig. 4.1(b). El *fragment shader* obtiene para cada fragmento los valores respectivos del punto de entrada $\mathbf{p}_{in}$ y el punto de salida $\mathbf{p}_{out}$ y calcula la dirección de rayo como

$$\vec{\mathbf{r}} = \frac{\mathbf{p}_{out} - \mathbf{p}_{in}}{|\mathbf{p}_{out} - \mathbf{p}_{in}|}. \tag{4.1}$$

Sustituyendo los valores $\mathbf{p}_{in}$ y $\vec{\mathbf{r}}$ en (2.8) tenemos que para cada píxel $\mathbf{u}$ de la pantalla la ecuación

$$\mathbf{x_u} = \mathbf{p}_{in} + \rho\vec{\mathbf{r}}, \ \forall\rho < |\mathbf{p}_{out} - \mathbf{p}_{in}| \tag{4.2}$$

proporciona la posición $\mathbf{x_u}$ de la muestra sobre el rayo en coordenadas de textura local. La posición de muestra $\mathbf{x_u}$ es movida a lo largo del rayo usando un tamaño de paso discreto $\Delta_{raycasting}$, incrementando ρ en $\Delta_{raycasting}$. En cada paso a lo largo del rayo, los valores actuales del vóxel en el punto de muestra $\mathbf{x_u}$ son obtenidos del mapa de textura 3D que se usa para almacenar el volumen de entrada y el cual también se utiliza para aplicar la función de transferencia. La función de transferencia también puede estar almacenada en una textura que actúa como *look-up table* (LUT), de donde se obtienen los valores visuales que se asignan al vóxel $\mathbf{k}_l$ asociado a la posición $\mathbf{x_u}$. Entonces, los valores de color y opacidad acumulados para el fragmento hasta cierto punto se actualizan usando la contribución del vóxel $\mathbf{k}_l$. Para evaluar eficientemente la composición definida en (2.9), se cambia el orden de la composición de adelante hacia atrás, así puede terminarse el rayo antes (por ejemplo, si se satura la opacidad) y definir la siguiente fórmula para el cálculo del color y opacidad para el píxel $\mathbf{u}$

$$g(R_\mathbf{u}) = [c(\mathbf{k}_L),\, a(\mathbf{k}_L)], \tag{4.3}$$

donde $c(\mathbf{k}_l)$ y $a(\mathbf{k}_l)$, son la acumulación del color y la opacidad hasta el vóxel $\mathbf{k}_l$, respectivamente. Estas funciones están dadas por

$$c(\mathbf{k}_l) = c(\mathbf{k}_{l-1}) + (1 - a(\mathbf{k}_{l-1}))\varphi(\mathbf{k}_l)\hat{\alpha}(\mathbf{k}_l), \tag{4.4}$$

y

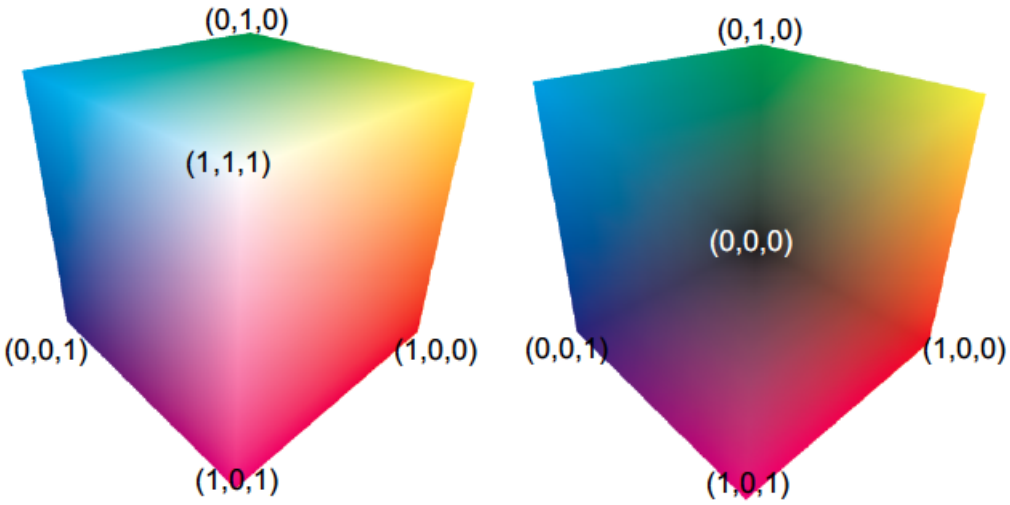
$$a(\mathbf{k}_l) = a(\mathbf{k}_{l-1}) + (1 - a(\mathbf{k}_{l-1}))\hat{\alpha}(\mathbf{k}_l), \tag{4.5}$$

donde las funciones φ y α se encuentran definidas en (2.6) y (2.7), respectivamente. Como la opacidad asignada también depende de la tasa de muestreo, no se usa $\alpha(\mathbf{k}_l)$

directamente, pues al usar menos rebanadas, la opacidad debe escalarse para que la intensidad total de la imagen permanezca igual. La ecuación usada para corregir la opacidad de la función de transferencia cuando se cambia la tasa de muestreo Δ_s con respecto a la tasa de muestreo de referencia Δ_{s_0} [69] está dada por

$$\hat{\alpha}(\mathbf{k}_i) = 1 - (1 - \alpha(\mathbf{k}_i))^{\Delta_{s_0}/\Delta_s}. \tag{4.6}$$

El ciclo por fragmento se termina una vez que el rayo a dejado el volumen (es decir, la posición $\mathbf{x_u}$ supera la longitud $|\mathbf{p}_{out} - \mathbf{p}_{in}|$ calculada inicialmente), o anticipadamente cuando existe algún otro criterio (dependiendo del modelo óptico escogido) que se haya cumplido; por ejemplo, que se haya alcanzado la opacidad máxima.



(a) (b)

Figura 4.1: Texturas usadas para calcular los puntos de entrada (a) y salida (b) de los rayos (imagen tomada de [3]).

4.1.1.3. Modificaciones

Para poder utilizar los mapas de conectividad para generar funciones de transferencia se construyó un módulo nuevo en donde se agruparon todos aquellos códigos fuente de utilidad para el proyecto. Primeramente, fue necesario desarrollar una clase para realizar la lectura del mapa de conectividad generado por el programa de

segmentación difusa y almacenarlo en una estructura de datos que facilitara su manipulación dentro del programa de visualización. Posteriormente, y con base en los procesadores de renderizado que ya estaban implementados, se generó una clase para utilizar el mapa de conectividad para la renderización.

Como se mencionó en la Sección 4.1.1.2, el proceso de composición se lleva a cabo en el programa *shader*, por lo que también fue necesario crear uno basándose en los que ya se encontraban implementados. El mapa de conectividad es enviado al programa *shader* en dos texturas 3D (una para el valor de la clase a la que pertenece el vóxel y la otra para el valor de la conectividad del vóxel a esa clase). Inicialmente, sólo se usaron los valores respectivos a la clase para realizar la renderización, con el fin de determinar si era posible elegir distintas propiedades de visualización según la clase del vóxel a renderizar. Si se usa únicamente el valor de la clase para definir las propiedades visuales que deben asignarse es posible que no se obtenga suficiente información del volumen, puesto que se pierden variaciones dentro de un mismo material al tratarlo de manera uniforme, véase la Fig. 4.3(a).

Posteriormente, se programó una clase para el manejo de múltiples funciones de transferencia de una dimensión. La idea es crear una función de una dimensión por cada una de las clases en las que se segmentó el volumen. Como el proceso de asignación de propiedades también se lleva a cabo en el *shader*, es necesario que las funciones de transferencia sean almacenadas en una textura 2D (en donde cada hilera representa la clase u objeto y cada columna un valor de conectividad), de esta forma, se mandan todas las funciones de transferencia al *shader* en una sola variable.

Originalmente, las funciones de transferencia de una dimensión utilizaban como entrada el valor de intensidad del vóxel. La asignación de valores visuales se lleva a cabo usando unos puntos llamados *mapping keys*, los cuales son utilizados de forma similar a una LUT. Cada *mapping key* está asociado a un valor de intensidad que funciona como llave de la tabla y realiza un mapeo a los valores de color y opacidad asignados a esa llave. Si no existe una llave definida para un valor de intensidad dado, las propiedades visuales se calculan utilizando interpolación lineal entre la llave menor y la llave mayor que se encuentren más cercanas al valor buscado. Finalmente,

las *mapping keys* son las que se guardan en una textura de 1D correspondiente a la función de transferencia (el largo de la textura corresponde al valor de intensidad de la llave). En nuestro caso, nos interesa más el valor de conectividad con la que el vóxel pertenece a una clase, por lo que este valor se uso en lugar del valor de intensidad.

Si sólo se usan los valores de conectividad existe la posibilidad de que aparezcan con mayor opacidad sólo aquellos vóxeles que tienen una conectividad similar a la de los puntos semilla y se pierdan algunos vóxeles del objeto cuyo valor de conectividad no sea tan alto, sin embargo esto puede ponderarse usando la función de transferencia. Por lo tanto, la función de transferencia inicial se calcula de la siguiente manera: Se busca el valor de conectividad mínimo dentro de cada clase (éste puede ser distinto de cero), se calcula el valor medio entre el valor mínimo encontrado y el valor de conectividad máximo (igual al valor presente en las semillas) y, finalmente, se crean *mapping keys* para mapear cada una de estas intensidades (incluyendo cero si no es el valor mínimo). Los valores visuales a los que se mapean dichos valores de conectividad se calculan con base en el número M de clases en las que está segmentada la imagen, esto con la finalidad de poder asignar colores y opacidades distintas a cada una de las clases sin necesidad de la intervención directa del usuario. Para ello, se multiplica por el número m de la clase para la cual se está creando la función por el factor $v = 1/M$ para generar un número $d = \frac{m}{M}$ distinto por clase. Los dos primeros *mapping keys* (correspondientes a los valores de conectividad mínimos) se inicializan con color blanco ($r = 1$, $g = 1$, $b = 1$) y una opacidad cero, el punto medio (correspondiente al valor de conectividad entre el mínimo y el máximo) con color $r = 1 \times d$, $g = 1 \times (1 - d)$, $b = 1/(v \times (m - 1))$ con una opacidad igual a $(m + d)/2$, mientras que el punto con la conectividad más alta recibe el color $r = 1 \times d$, $g = 1 \times (1 - d)$, $b = 1/v \times (m - 1)$ con una opacidad $m + d$.

En un principio y con fines de prueba, las funciones de transferencia eran fijas, posteriormente se modificó el *widget* (interfaz de comunicación con el programa) utilizado en el entorno gráfico para leer y modificar las funciones de transferencia de acuerdo a nuestras necesidades. El *widget* consta de los siguientes componentes y puede observarse en la Fig. 4.2: un eje coordenado, donde el eje ordenado corres-

ponde a los valores de conectividad y el eje de las abscisas al valor de la opacidad, una ventana para selección de colores y pestañas para moverse entre clases. El *widget* funciona de la siguiente manera, se selecciona la pestaña de la clase que se desea modificar, después se pueden colocar *mapping keys* haciendo click sobre la gráfica de ejes coordenados, para posteriormente seleccionar el color que se desea para ese valor de conectividad y la altura correspondiente con la opacidad deseada. Estos puntos pueden crearse, moverse o eliminarse en cualquier momento y de igual manera pueden cambiarse sus propiedades de color.

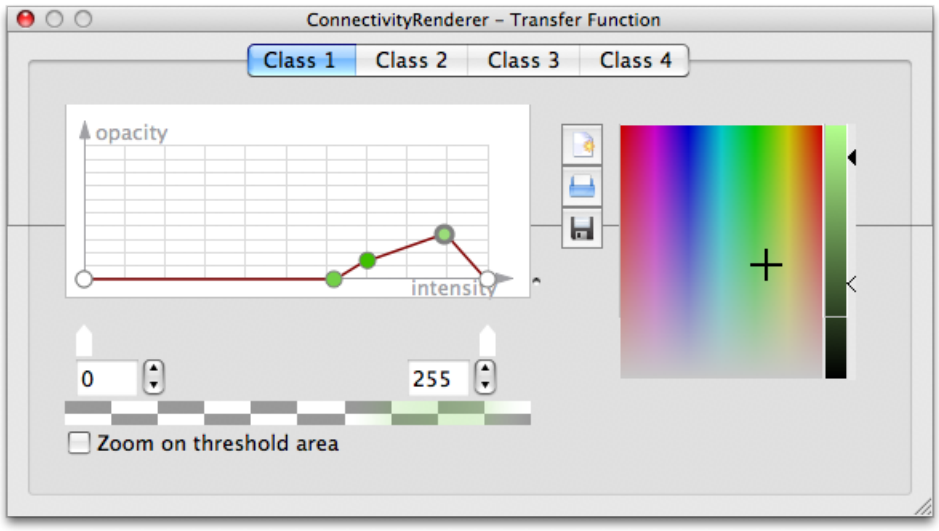


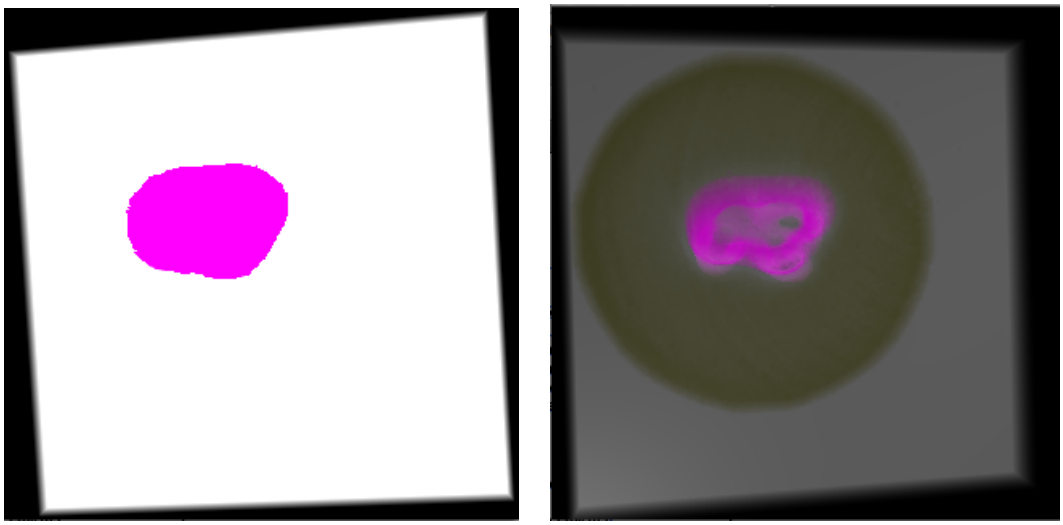
Figura 4.2: Interfaz para diseño de funciones de transferencia de *Voreen.*

4.2. Resultados

Todos los programas fueron instalados y ejecutados en una computadora con procesador Intel® Core 2 Duo a 2.26 GHz, con 2GB de memoria RAM, una tarjeta gráfica NVIDIA® GeForce 9400M y bajo el sistema operativo Mac OS 10.4 (Mac OS Leopard®). Para nuestros experimentos usamos nueve conjuntos de datos disponibles en la base de datos pública The Volume Library [70].

4.2.1. Muela

Este conjunto de datos representa una reconstrucción tri-dimensional de una muela generado por medio de tomografía de rayos-X (*GE Aircraft Engines*, Evendale, Ohio, EE.UU.). El conjunto de datos tiene dimensiones $128 \times 128 \times 128$ vóxeles. Las imágenes de la Fig. 4.3 se realizaron con un subconjunto de los datos, (específicamente $128 \times 128 \times 32$ vóxeles), la imagen de la Fig. 4.3(a) asigna valores usando solo el valor de la clase a la que pertenece el vóxel, la Fig. 4.3(b) utiliza adicionalmente el valor de conectividad. En este caso, se segmentó el volumen en dos clases (muela y fondo) y se utilizó una función de transferencia fija.

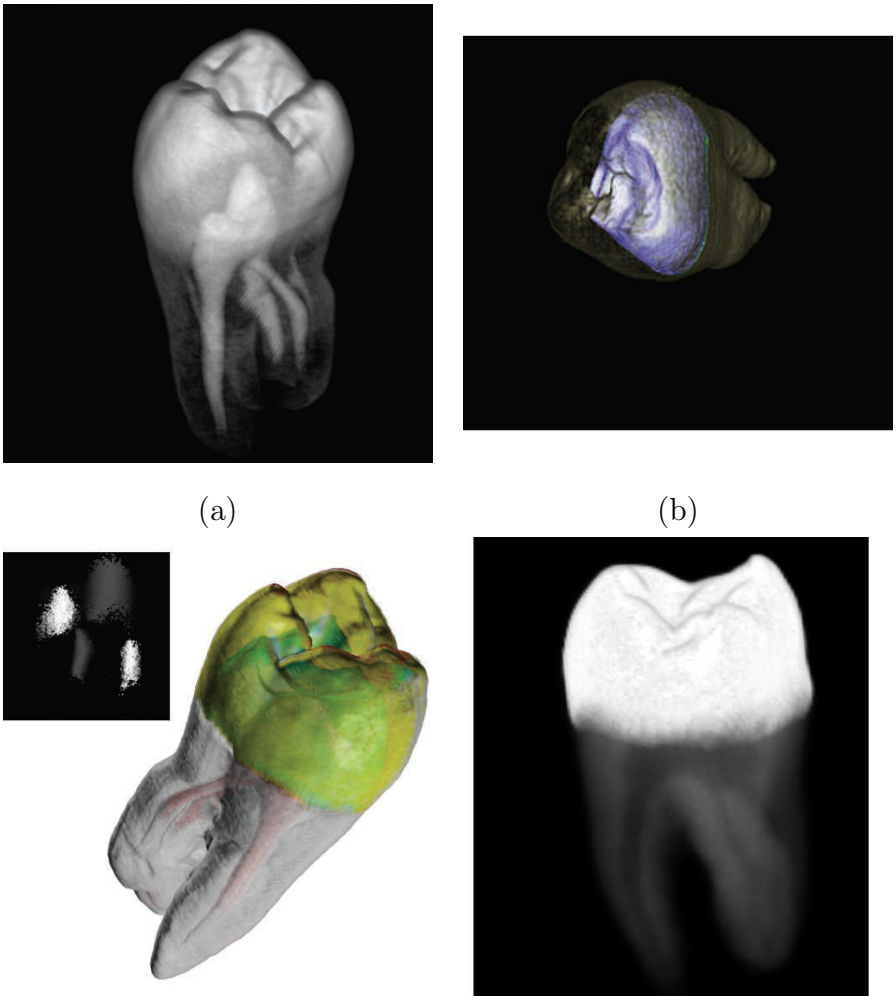


(a) (b)

Figura 4.3: Visualización de un fragmento de volumen usando (a) sólo información de clase e (b) información de clase y conectividad. Si no se usan los valores de conectividad, no es posible distinguir variaciones internas dentro de cada clase.

Este conjunto de datos fue utilizado en [8] como parte de una comparación de diferentes métodos para la visualización directa de volúmenes, cuatro de los resultados utilizados para esta comparación se muestran en la Fig. 4.4. En el patrón de la Fig. 4.4(d) se utilizó una metodología que genera automáticamente una galería de funciones de transferencia y se eligió una para visualizar este conjunto de datos, por lo tanto nosotros tratamos de reproducir el mismo patrón con la metodología expuesta arriba.

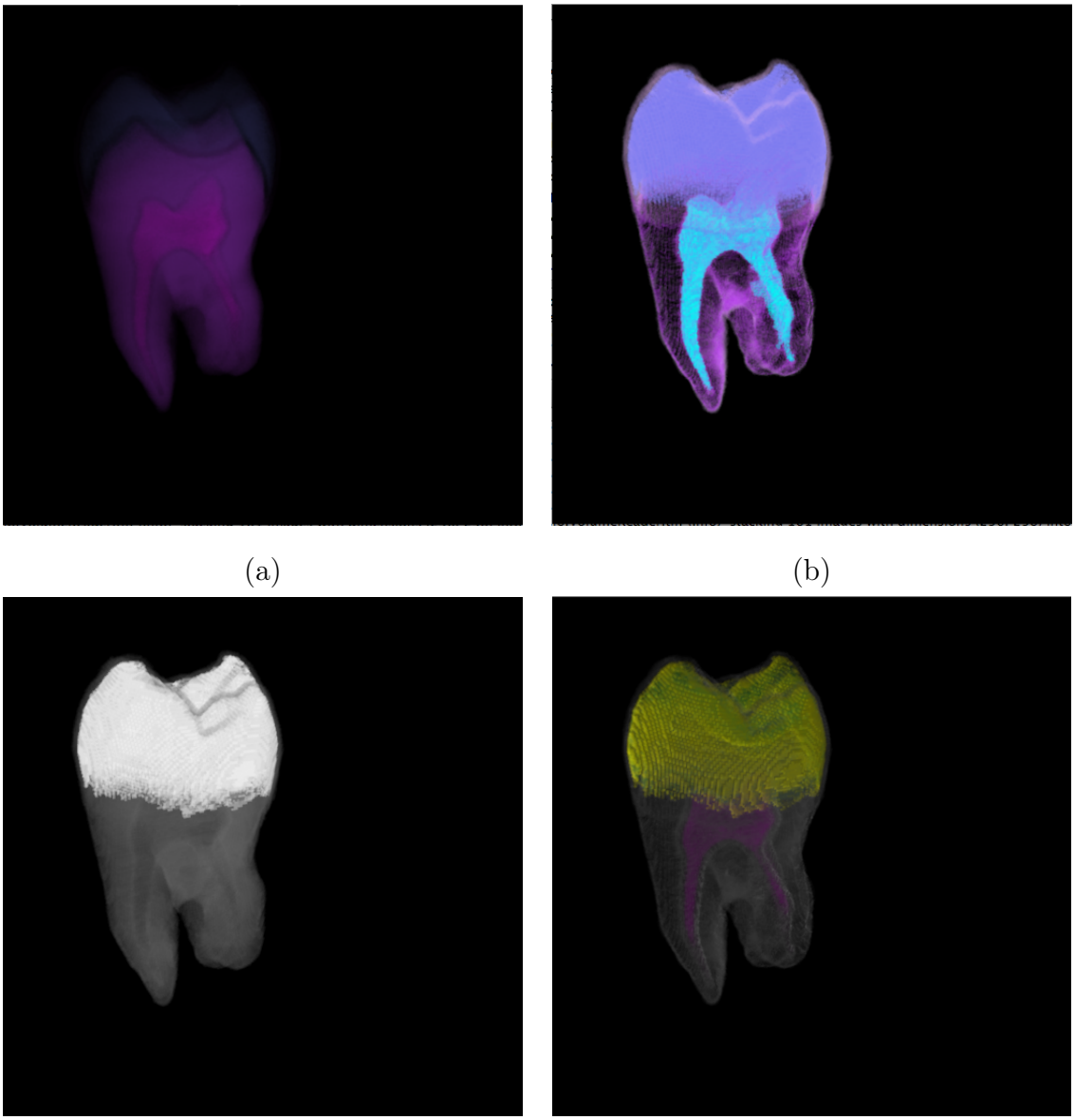
Los resultados de la visualización con nuestra metodología se muestran en la Fig. 4.5, específicamente en la imagen de la Fig. 4.5(c). El conjunto de datos fue segmentado en cuatro clases (raíces, pulpa, corona y fondo), la Fig. 4.5(a) muestra las funciones de transferencia iniciales, la imagen de la Fig. 4.5(b) muestra un conjunto de funciones que permite destacar la raíz y la Fig. 4.5(d) adiciona el cálculo de gradientes para los cálculos de iluminación, también debe notarse que esto produce imágenes donde los bordes de los vóxeles son más visibles.



(a) (b)

(c) (d)

Figura 4.4: Visualización del conjunto de datos por medio de (a) prueba y error [4], (b) espectro de contorno [5], (c) histogramas de volumen [6] y (d) selección de una galería de funciones de transferencia generada automáticamente [7] (Imágenes tomadas de [8]).



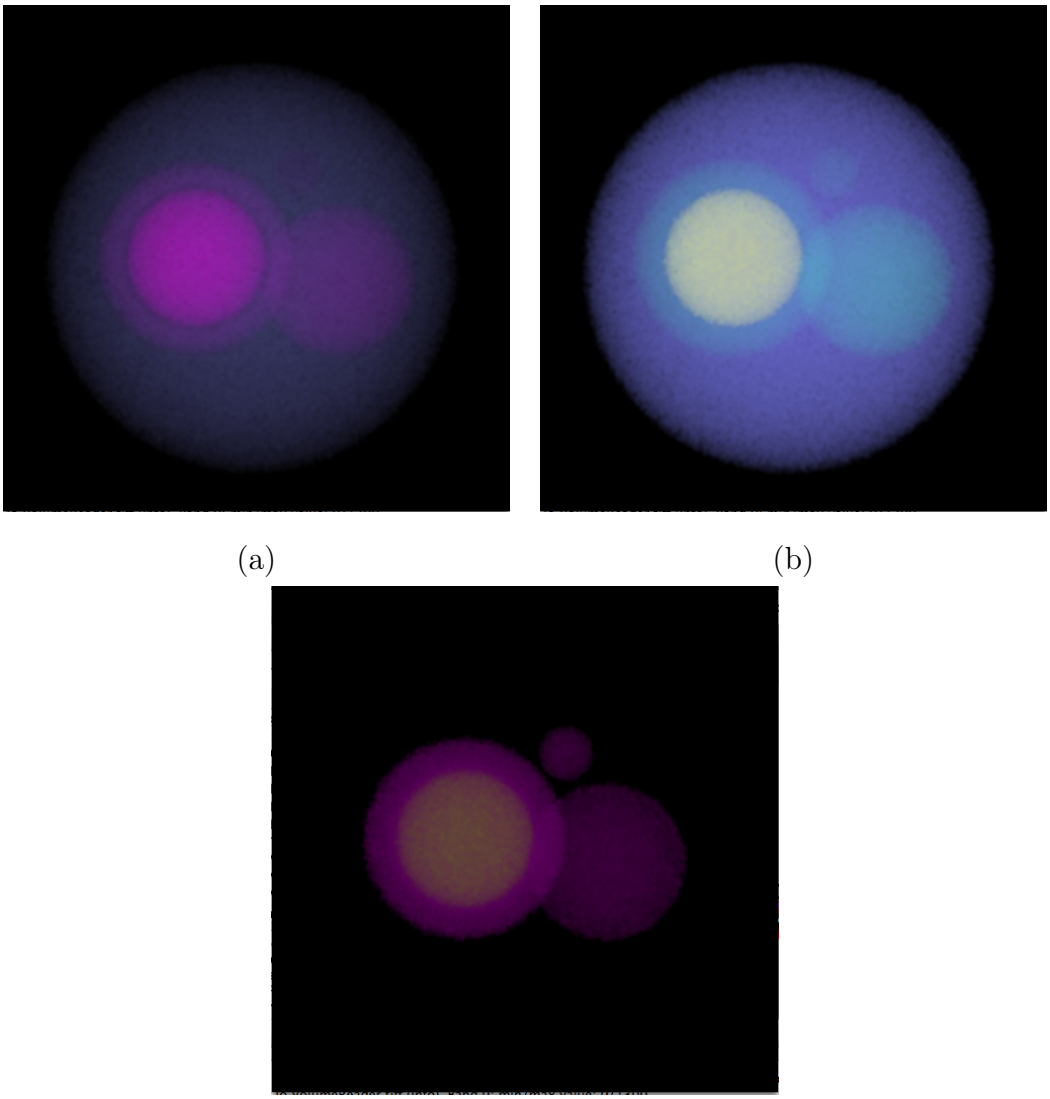
(a) (b)

(c) (d)

Figura 4.5: Distintas renderizaciones de la mismo conjunto de datos (muela) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra el conjunto con modificaciones ligeras a las funciones, (c) muestra un patrón similar al de la Fig. 4.4(d) y (d) incluye cálculo de gradientes para la iluminación.

4.2.2. Elipses

Este es un conjunto de datos artificial, representa un conjunto de esferas tridimensionales (Stefan Roettger del *Computer Graphics Group*, *University of Erlangen*, Alemania). El conjunto de datos tiene dimensiones $128 \times 128 \times 128$ vóxeles. Los resultados de la visualización con nuestra metodología se muestran en la Fig. 4.6.



(a) (b)

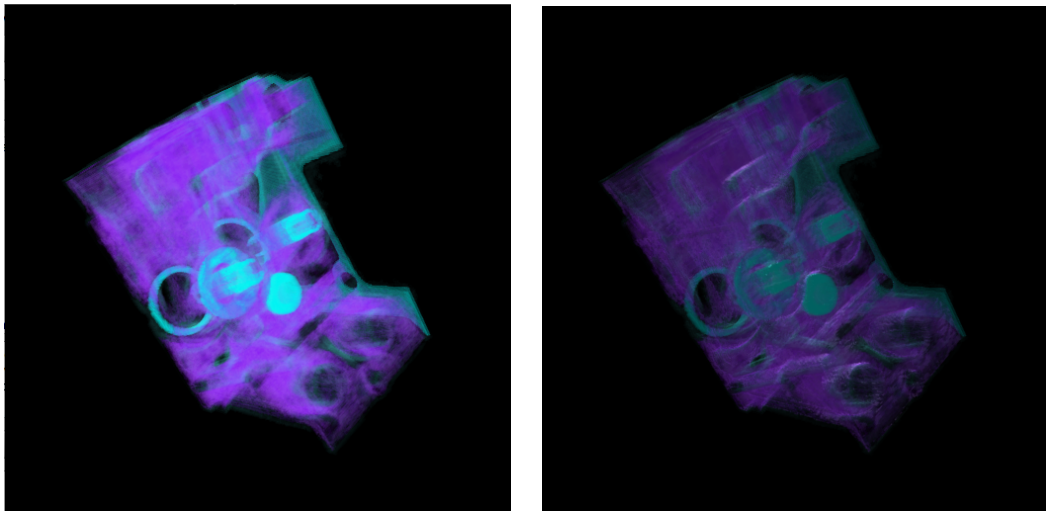
(c)

Figura 4.6: Distintas visualizaciones del mismo conjunto de datos (esferas) y funciones de transferencia diferentes. La imagen (a) muestra las funciones de transferencia iniciales, la imagen (b) una modificación de esas mismas funciones y la imagen (c) oculta una de las clases y se realizó incluyendo el cálculo de gradientes para la iluminación.

El conjunto de datos se segmentó en cuatro clases (esfera exterior, esferas internas, esfera más interna y fondo) y la Fig. 4.6(a) muestra las funciones de transferencia iniciales, la Fig. 4.6(b) una modificación de esas mismas funciones y la Fig. 4.6(c) oculta la esfera externa y se realizó incluyendo el cálculo de gradientes para realizar la iluminación.

4.2.3. Motor

Este conjunto de datos representa una reconstrucción tri-dimensional que muestra dos cilindros de un bloque de motor. El conjunto fue obtenido por medio de tomografía computarizada de rayos-X (*General Electric*, EE.UU.). El conjunto de datos tiene dimensiones $256 \times 256 \times 256$ vóxeles y fue segmentado en tres clases. Los resultados de la visualización con nuestra metodología se muestra en la Fig. 4.7, la imagen en la Fig. 4.7(a) muestra una modificación a las funciones de transferencia iniciales y la Fig. 4.7(b) muestra el mismo patrón utilizando el cálculo de gradientes para la iluminación.

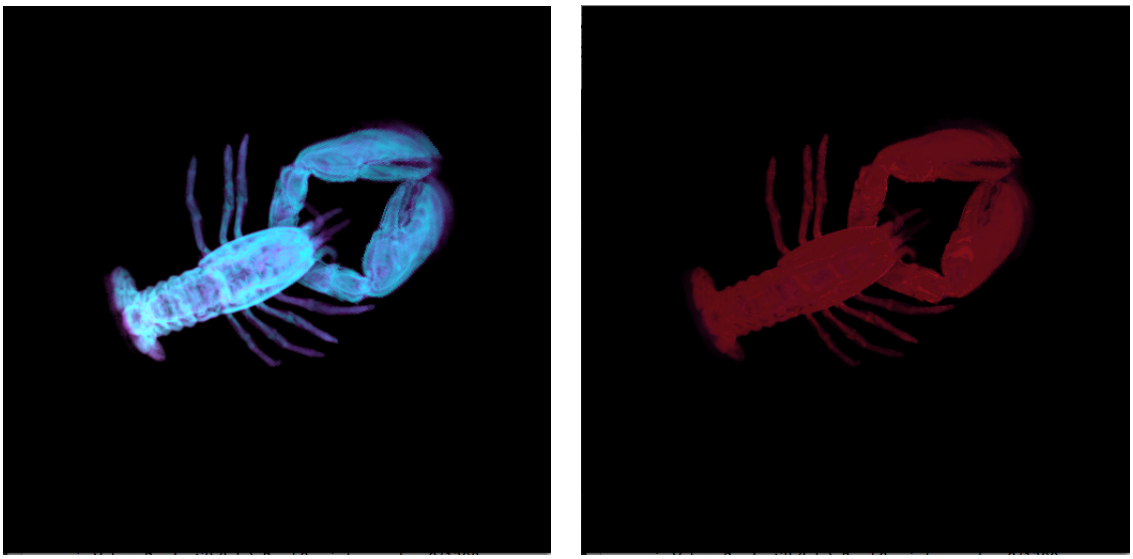


(a) (b)

Figura 4.7: Distintas visualizaciones del mismo conjunto de datos (motor) y funciones de transferencia diferentes. La imagen (a) muestra una modificación de las funciones de transferencia iniciales y la imagen (b) esas mismas funciones incluyendo el cálculo de gradientes para la iluminación.

4.2.4. Langosta

Este conjunto de datos representa una reconstrucción tri-dimensional de una langosta contenida en un bloque de resina. El conjunto fue obtenido por medio de tomografía computarizada de rayos-X (*VolVis* distribución de SUNY Stony Brook, NY, EE.UU.). El conjunto de datos tiene dimensiones $301 \times 324 \times 56$ vóxeles. Los resultados de la visualización con nuestra metodología se muestra en la Fig. 4.8, el conjunto fue segmentado en dos clases, la Fig. 4.8(a) muestra una modificación a las funciones de transferencia iniciales y la Fig. 4.8(b) muestra otro patrón utilizando el cálculo de gradientes para la iluminación.



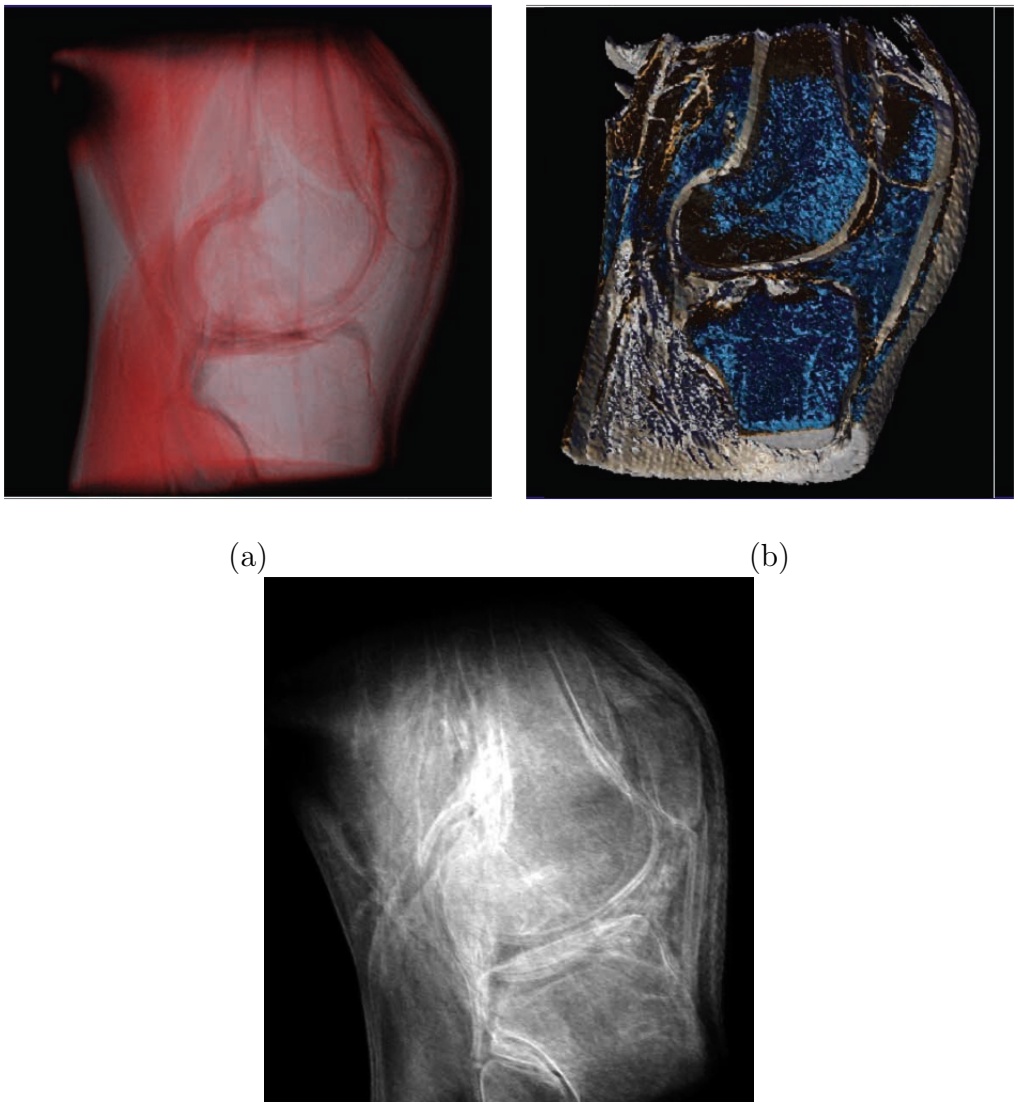
(a) (b)

Figura 4.8: Distintas visualizaciones del mismo conjunto de datos (langosta) y funciones de transferencia diferentes. La imagen (a) muestra una modificación de las funciones de transferencia iniciales y la imagen (b) esas mismas funciones incluyendo el cálculo de gradientes para la iluminación.

4.2.5. Rodilla

Este conjunto de datos representa una reconstrucción tri-dimensional de una rodilla obtenido por medio resonancia magnética (*Brigham and Women's Hospital Surgical Planning Laboratory*). El conjunto de datos tiene dimensiones $512 \times 512 \times 87$ vóxeles

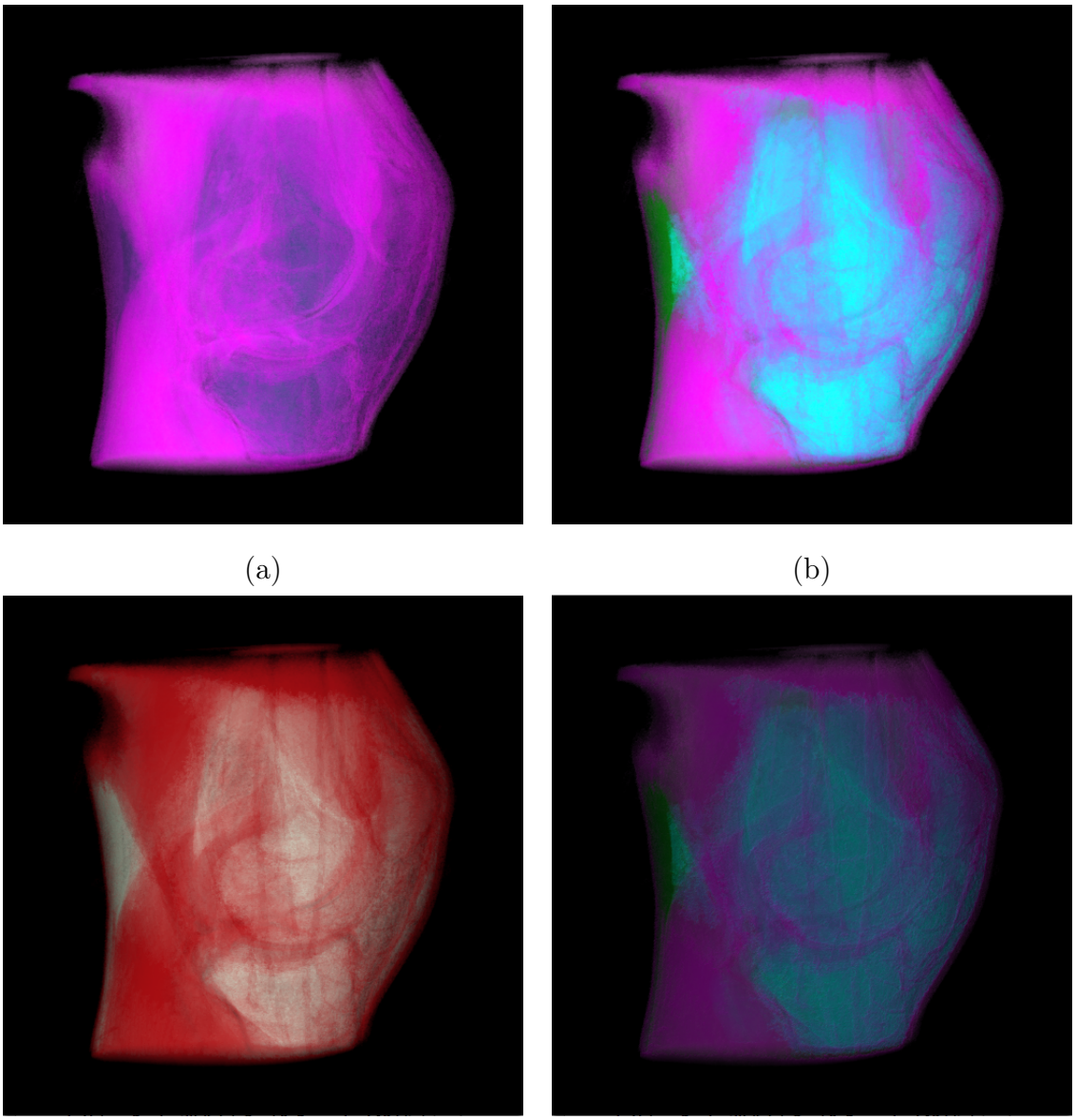
y fue utilizado en [8] como parte de una comparación de diferentes métodos para la visualización directa de volúmenes. En la Fig. 4.9(a) se utilizó una función de transferencia generada manualmente para visualizar este conjunto de datos, por lo tanto nosotros tratamos de reproducir el mismo patrón con la metodología expuesta arriba, el resultado puede observarse en la Fig. 4.10(c). La Fig. 4.9(b) utilizó histogramas de volumen y la Fig. 4.9(c) fue seleccionada de una galería generada automáticamente.



(a) (b)

(c)

Figura 4.9: Visualización del conjunto de datos por medio de (a) prueba y error [4], (b) histogramas de volumen [6] y (c) selección de una galería de funciones de transferencia generada automáticamente [7] (Imágenes tomadas de [8]).



(a) (b)

(c) (d)

Figura 4.10: Distintas renderizaciones del mismo conjunto de datos (rodilla) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra el conjunto con modificaciones ligeras a las funciones, (c) muestra un patrón similar a la Fig. 4.9(a) y (d) incluye cálculo de gradientes para la iluminación.

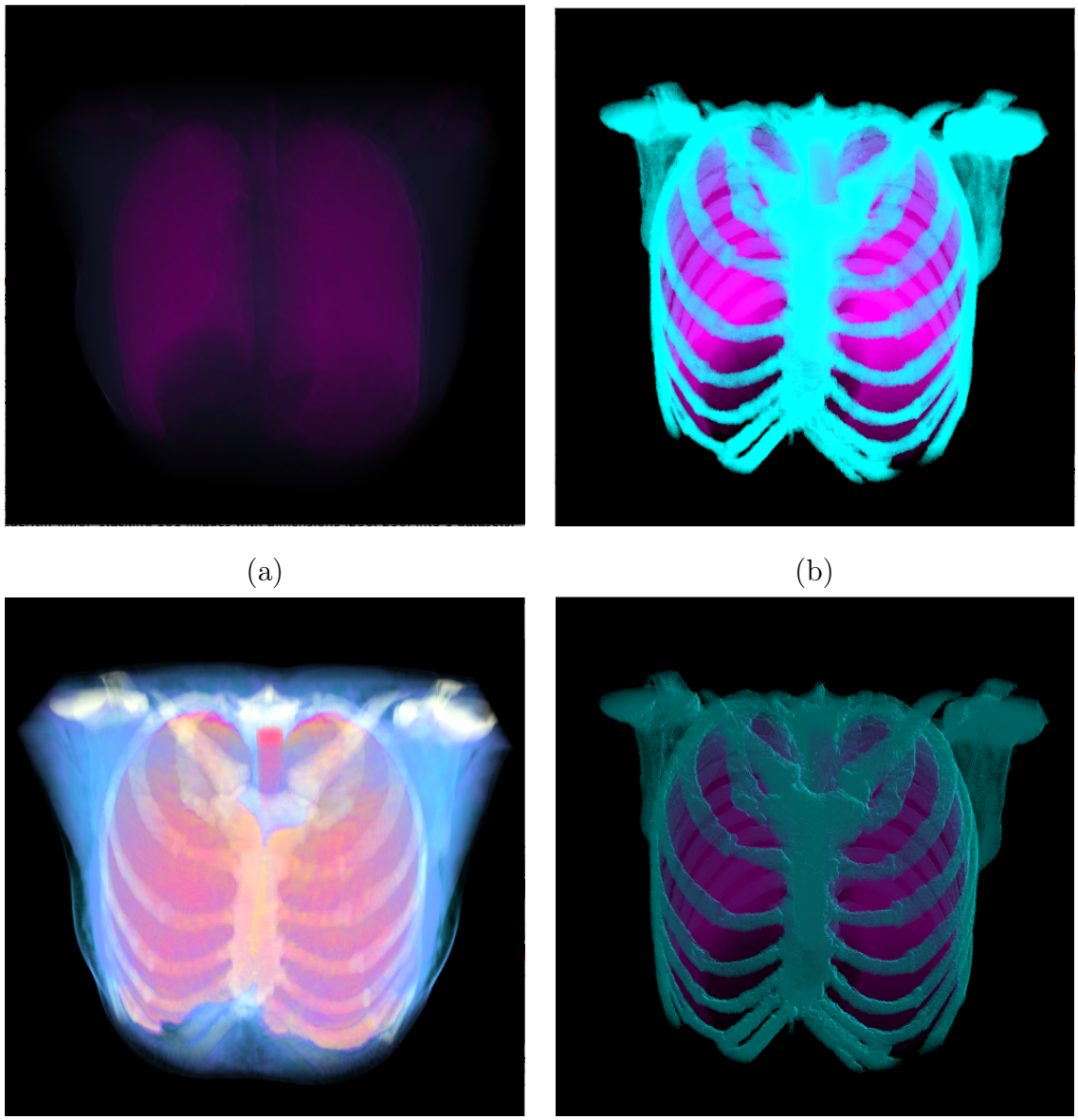
Los resultados de la visualización con nuestra metodología se muestran en la Fig. 4.10, donde el conjunto de datos fue segmentado en cuatro clases y la Fig. 4.10(a) muestra las funciones de transferencia iniciales, la Fig. 4.10(b) muestra modificaciones

con el fin de resaltar los huesos y la Fig. 4.10(d) muestra el uso del cálculo de gradientes para la iluminación.

4.2.6. Tórax

Este conjunto de datos representa una reconstrucción tri-dimensional del tórax de una mujer generado por medio de tomografía de rayos-X (*Department of Radiology, University of Iowa*). El conjunto de datos tiene dimensiones $384 \times 384 \times 240$ vóxeles. Para su visualización fue segmentado en 4 clases (pulmones, huesos, piel y otros órganos, y fondo).

Los resultados de la visualización con nuestra metodología se muestran en la Fig. 4.11, donde la Fig. 4.11(a) muestra la imagen generada con las funciones de transferencia iniciales, la Fig. 4.11(b) muestra modificaciones para resaltar los huesos, la Fig. 4.11(c) muestra modificaciones para observar también los pulmones y la Fig. 4.11(d) muestra el mismo patrón que la Fig. 4.11(b) pero usando cálculo de gradientes para la iluminación.



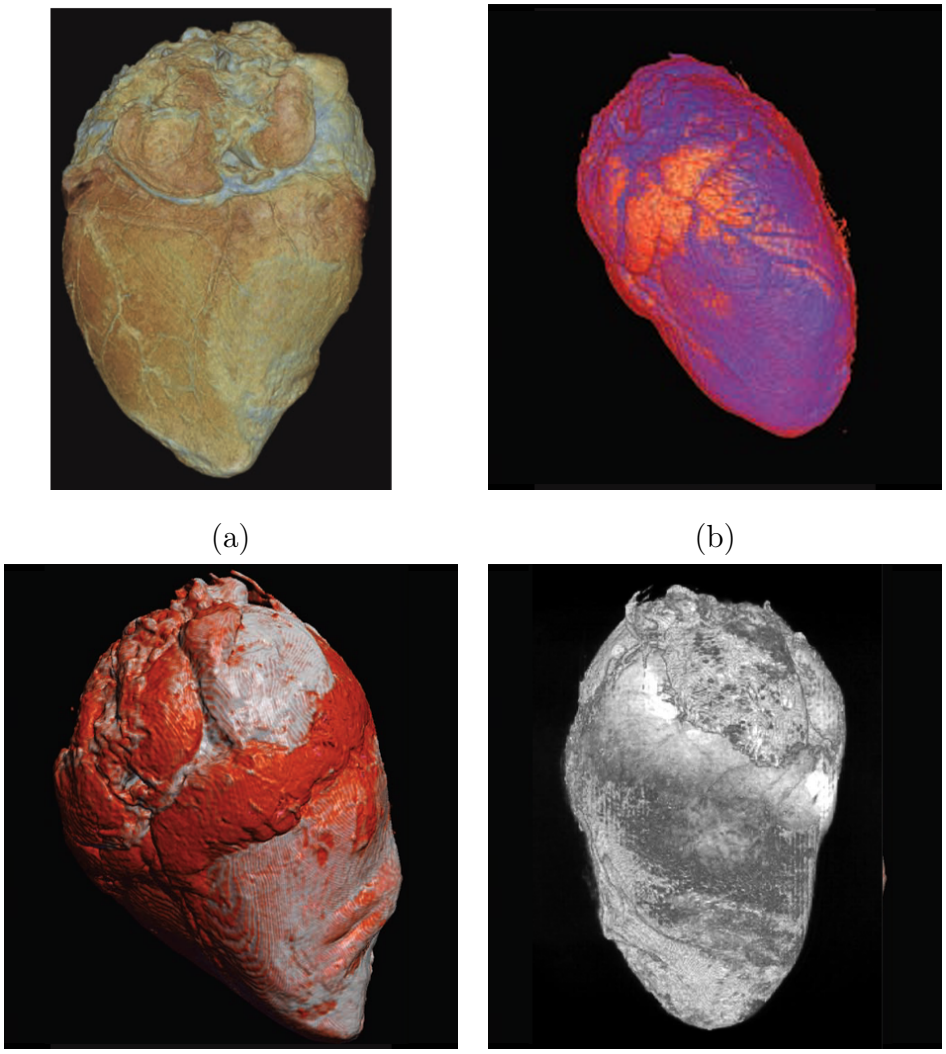
(a) (b)

(c) (d)

Figura 4.11: Distintas renderizaciones del mismo conjunto de datos (tórax) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra modificaciones para resaltar las costillas, (c) muestra modificaciones para resaltar pulmones y (d) muestra el patrón (b) incluyendo cálculo de gradientes para la iluminación.

4.2.7. Corazón

Este conjunto de datos representa una reconstrucción tri-dimensional del corazón de una oveja obtenido por medio de resonancia magnética (*Center for In-Vivo Microscopy*, *Duke University*, Carolina del Norte, EE.UU.).



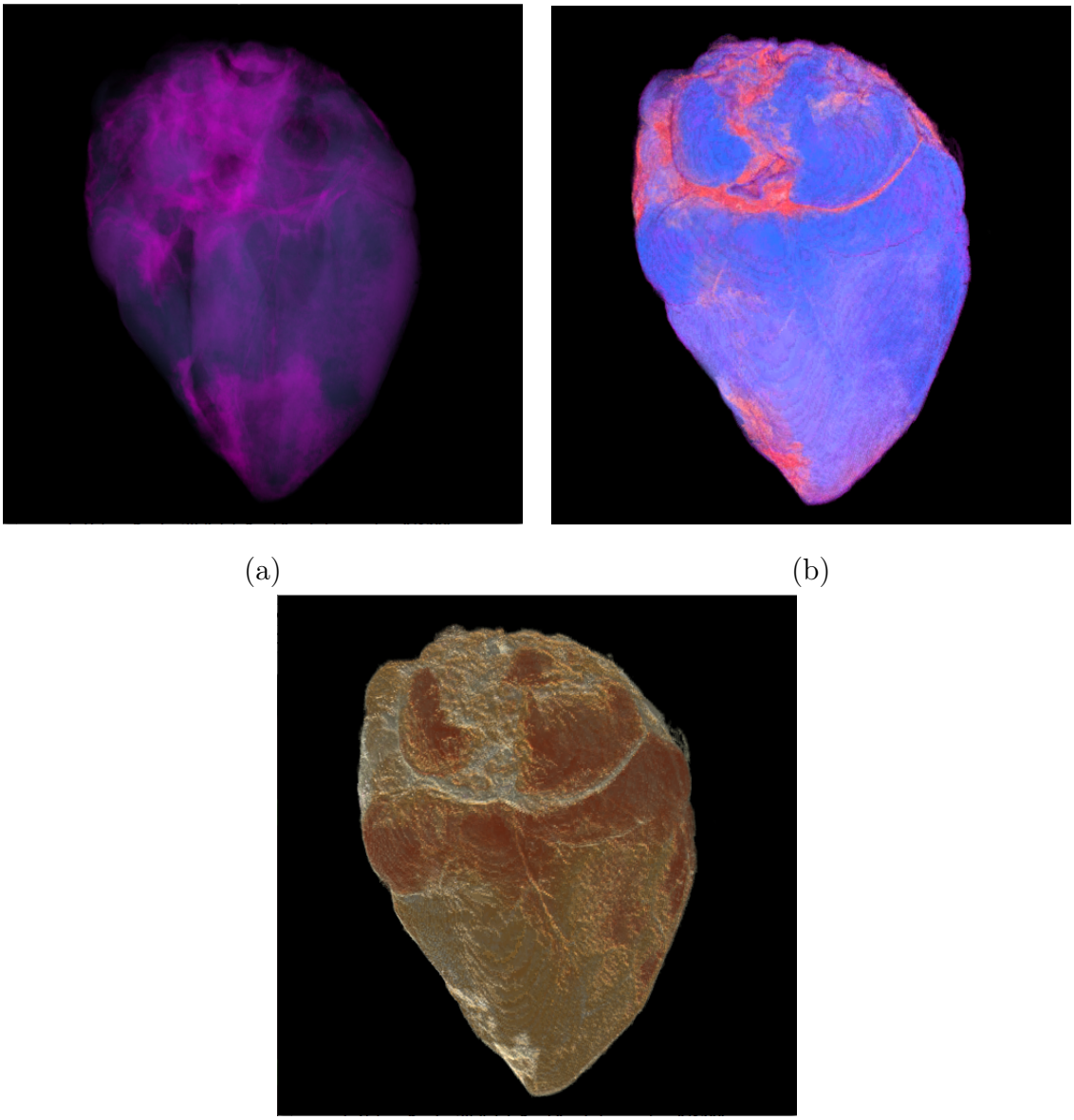
(a) (b)

(c) (d)

Figura 4.12: Visualización del conjunto de datos por medio de (a) prueba y error [4], (b) espectro de contorno [5], (c) histogramas de volumen [6] y (d) selección de una galería de funciones de transferencia generada automáticamente [7] (Imágenes tomadas de [8]).

El conjunto de datos tiene dimensiones $352 \times 352 \times 256$ vóxeles. Este conjunto

de datos fue utilizado en [8] como parte de una comparación de diferentes métodos para la visualización directa de volúmenes, resultados presentados en dicho trabajo se muestran en la Fig. 4.12.



(a) (b)

(c)

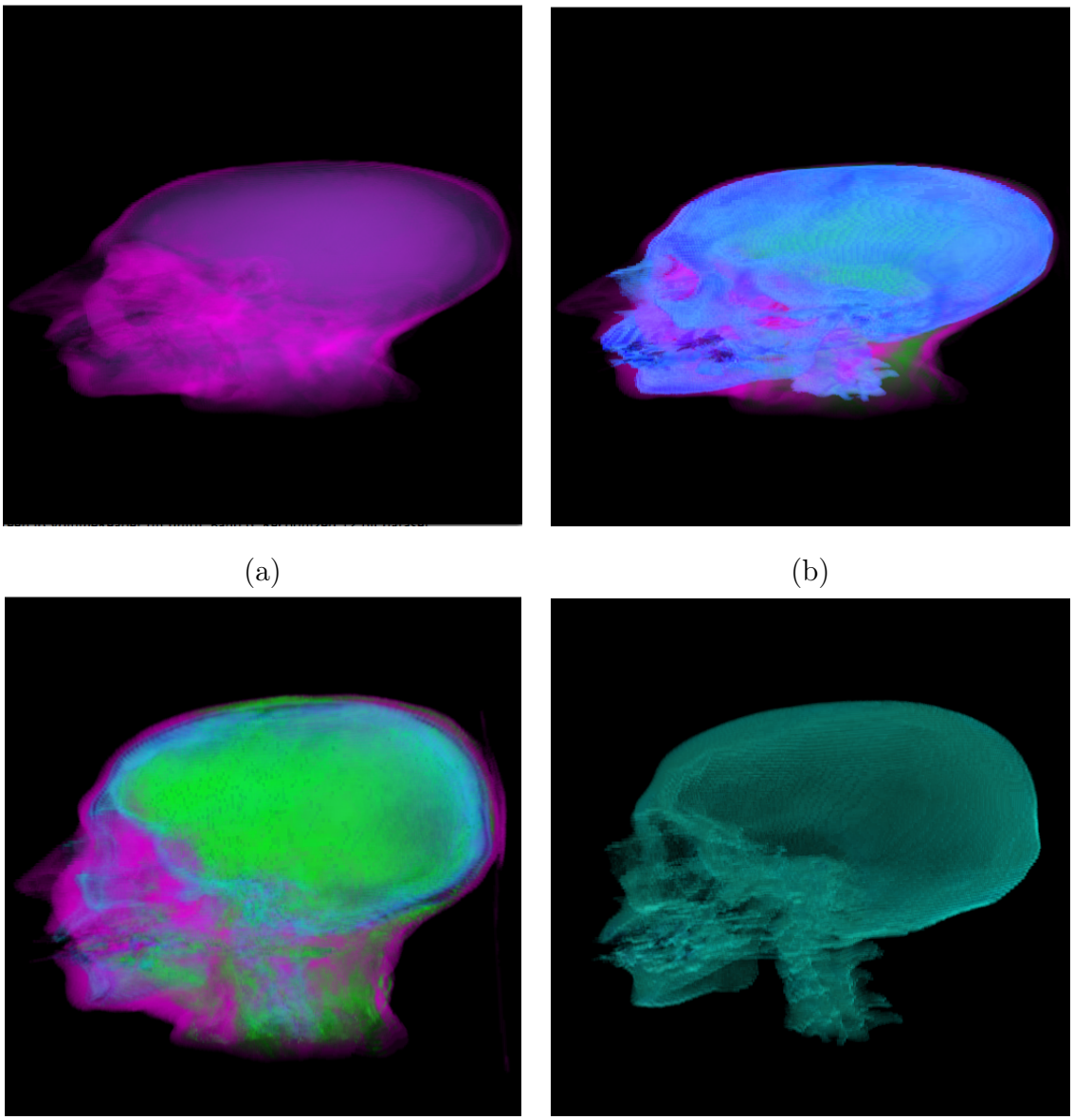
Figura 4.13: Distintas renderizaciones del mismo conjunto de datos (corazón) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra el conjunto tratando de imitar el patrón de la Fig. 4.12(a) y (c) incluye cálculo de gradientes para la iluminación.

El patrón en la Fig. 4.12(a) se generó creando una función de transferencia manual para visualizar este conjunto de datos, por lo tanto nosotros tratamos de reproducir el mismo patrón con la metodología expuesta arriba, el resultado puede observarse en la Fig. 4.13(b).

Los resultados de la visualización con nuestra metodología y segmentado el conjunto de datos en cuatro clases se muestran en la Fig. 4.13, donde la Fig. 4.13(a) muestra las funciones de transferencia iniciales y la Fig. 4.13(c) representa el mismo patrón de la Fig. 4.13(b) incluyendo información de gradientes para realizar la iluminación.

4.2.8. Cabeza

Este conjunto de datos representa una reconstrucción tri-dimensional de la cabeza del cadáver de una mujer, fue generado por medio de tomografía de rayos-X (Marc Levoy del *Computer Graphics Laboratory*, *Stanford University*, EE.UU. cortesía del *North Carolina Memorial Hospital*). El conjunto de datos tiene dimensiones $256 \times 256 \times 113$ vóxeles. Los resultados de la visualización con nuestra metodología se muestran en la Fig. 4.14 y se generaron segmentando el conjunto de datos en cuatro clases (piel, cráneo, cerebro y fondo), la Fig. 4.14(a) muestra las funciones de transferencia iniciales, la Fig. 4.14(b) modifica las funciones para resaltar cráneo y cerebro, la Fig. 4.14(c) muestra un poco más de cerebro y la Fig. 4.14(d) muestra el cráneo incluyendo gradientes para la iluminación.



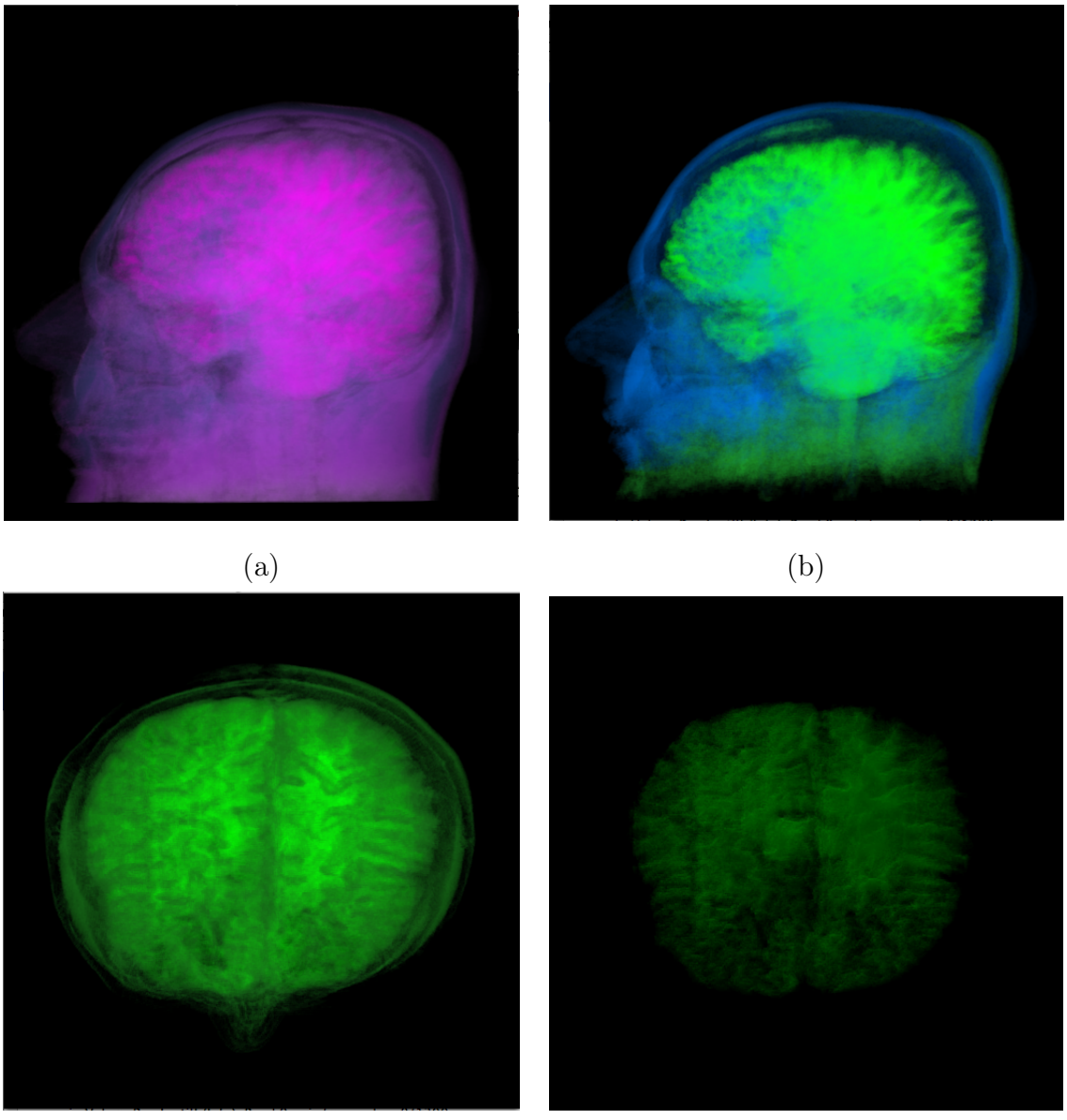
(a) (b)

(c) (d)

Figura 4.14: Distintas renderizaciones de la mismo conjunto de datos (cabeza obtenida por tomografía computarizada) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra el conjunto con modificaciones para resaltar cráneo, (c) muestra modificaciones para resaltar cerebro y (d) muestra sólo el cráneo e incluye cálculo de gradientes para la iluminación.

También se utilizó otro conjunto de datos de una cabeza obtenido por medio de resonancia magnética (*Computer Graphics Group*, *University of Erlangen*, Alemania).

El conjunto de datos tiene dimensiones $256 \times 256 \times 256$ vóxeles. Los resultados de la visualización con nuestra metodología se muestran en la Fig. 4.15.



(a) (b)

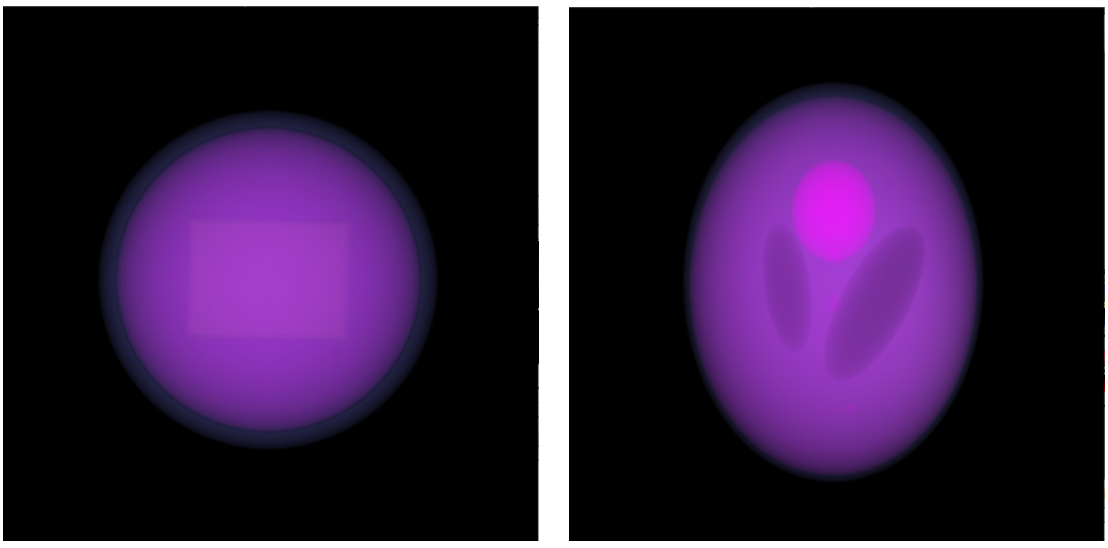
(c) (d)

Figura 4.15: Distintas renderizaciones de la mismo conjunto de datos (cabeza obtenido por resonancia magnética) usando funciones de transferencia diferentes, (a) muestra las funciones de transferencia iniciales, (b) muestra el conjunto con modificaciones para resaltar el cerebro, (c) muestra el cerebro visto desde arriba y (d) usa otra función de transferencia e incluye cálculo de gradientes para la iluminación.

Los resultados fueron obtenidos segmentando en cuatro clases, sin embargo, está segmentación fue más difícil de realizar que las otras. La Fig. 4.15(a) muestra las funciones de transferencia iniciales, la Fig. 4.15(b) muestra modificaciones para resaltar el cerebro y la Fig. 4.15(c) muestra el cerebro (visto desde arriba) y la Fig. 4.15(d) tiene la misma orientación que la imagen anterior pero utiliza una función de transferencia distinta y el cálculo de gradientes adicional para la iluminación.

4.3. Validación

Para realizar la validación de los resultados, se optó por utilizar dos volúmenes con datos sintéticos (*phantoms*). El volumen de la Fig. 4.16(a) fue generado utilizando el programa Xmipp [71], y consiste de dos esferas de distintas densidades que generan un "hueco" dentro del cual hay un ladrillo, mientras que el volumen de la Fig. 4.16(b) corresponde a un *phantom Shepp-Logan* modificado (un modelo que trata de imitar las densidades del cerebro humano, muy usado en reconstrucción de volúmenes por tomografía computarizada), dicho volumen se obtuvo de [72].



(a) (b)

Figura 4.16: Visualización de los conjuntos de datos sintéticos utilizados para la validación del proyecto, (a) es el volumen generado por Xmipp y (b) el volumen *Shepp-Logan*.

Volumen/Ruido (Relación señal a ruido)	Caso (i)	Caso (ii)	Caso (iii)
Phantom Xmipp	1.5075	2.1647	2.4236
Shepp-Logan	1.4131	1.7821	1.9716

Tabla 4.1: Relación señal a ruido para cada uno de los casos y volúmenes.

Al volumen de la Fig. 4.16(a) se le agregó ruido gaussiano para generar otros tres volúmenes de la siguiente manera: (i) usando la desviación estándar de todo el volumen, (ii) usando la desviación estándar del volumen sin tomar en cuenta el fondo y (iii) usando la desviación estándar de sólo dos de los objetos dentro del volumen. De forma similar, se generaron otros tres volúmenes para la Fig. 4.16(b), sin embargo se utilizó un múltiplo de la desviación estándar obtenida del caso (iii) para generar el ruido de los otros dos casos. La Tabla 4.1 muestra la relación señal a ruido calculada como $SNR = \mu_s/\sigma_s$, donde μ_s y σ_s son la media y la desviación estándar de los volúmenes con ruido, respectivamente.

Para evaluar la *precisión* de la segmentación y la visualización, se calculó la proporción de la discrepancia entre la segmentación del volumen original (utilizado como *estándar de oro* o *ground truth*) y la segmentación del volumen con ruido utilizando la siguiente medición:

$$precisión = \frac{|O_i \cap O_i^g|}{|O_i \cup O_i^g|}, \tag{4.7}$$

donde O_i^g representa la segmentación de la clase i del volumen sin ruido y O_i representa la segmentación de la misma clase. De igual forma, se midió la *sensibilidad* o *factor de verdaderos positivos* (proporción de verdaderos positivos en la segmentación) utilizando la siguiente definición,

$$sensibilidad = TPF = \frac{|O_i \cap O_i^g|}{|O_i^g|}, \tag{4.8}$$

donde, para cada clase i, TFP representa la superposición entre la segmentación del volumen con ruido y la segmentación del estándar de oro. También se puede medir la fracción de vóxeles falsamente identificados como parte del objeto en la clase i, o *factor de falsos positivos*, por medio de la siguiente ecuación

$$FPF = \frac{|(O_i \cup O_i^g) - O_i^g|}{|O_i^g|}, \tag{4.9}$$

donde la diferencia entre las segmentaciones resultantes se da en vóxeles. La *especificidad* se mide con el traslape de los verdaderos negativos en ambas segmentaciones y puede calcularse como

$$especificidad = 1 - FPF, \tag{4.10}$$

sin embargo, puede ser más interesante conocer la fracción de vóxeles que se identificaron falsamente como no pertenecientes al objeto i. Esa medición se calcula utilizando

$$FNF = \frac{|(O_i \cup O_i^g) - O_i|}{|O_i^g|}. \tag{4.11}$$

Las mediciones se hicieron por cada clase en la que se segmento el volumen, los resultados de estas medicionesen los dos volúmenes por cada nivel de ruido pueden ser consultados en las Tablas 4.2, 4.3 y 4.4, respectivamente. La evaluación se realizó usando todos los valores de conectividad presentes en cada segmentación.

Phantom	**Cuerpo**	**Precisión**	**Sensibilidad**	**FPF**	**FNF**	**Especificidad**
Propio	Fondo	0.99457	0.99503	0.001416	0.004964	0.99858
	Esfera 1	0.85838	0.92598	0.005664	0.074018	0.99433
	Esfera 2	0.96073	0.98994	0.005932	0.010057	0.99406
	Ladrillo	0.97648	0.98060	0.0000707	0.019396	0.99992
Shepp-Logan	Fondo	0.99632	0.99741	0.003224	0.002582	0.99677
	Elipse 1	0.97675	0.99285	0.000552	0.007146	0.99944
	Elipse 2	0.50091	0.51002	0.004814	0.489979	0.99518
	Elipses 4	0.07898	0.79084	0.103115	0.209158	0.896885

Tabla 4.2: Medidas obtenidas de la comparación entre las segmentaciones de los volúmenes con ruido del caso (i) y las segmentaciones sin ruido.

Phantom	Cuerpo	Precisión	Sensibilidad	FPF	FNF	Especificidad
Propio	Fondo	0.99209	0.99299	0.002780	0.007000	0.9972
	Esfera 1	0.84050	0.93580	0.008156	0.064195	0.99184
	Esfera 2	0.96114	0.98468	0.004777	0.015318	0.99406
	Ladrillo	0.96756	0.97719	0.0001668	0.022806	0.99983
Shepp-Logan	Fondo	0.99995	0.99996	0.0000468	0.0000305	0.99995
	Elipse 1	0.99999	1.00000	1.913×10^{-7}	0.00000	1.00000
	Elipse 2	0.99802	0.99937	0.000359	0.0006236	0.99964
	Elipses 4	0.96676	0.97689	0.000119	0.0231022	0.99988

Tabla 4.3: Medidas obtenidas de la comparación entre las segmentaciones de los volúmenes con ruido del caso (ii) y los objetos estándar.

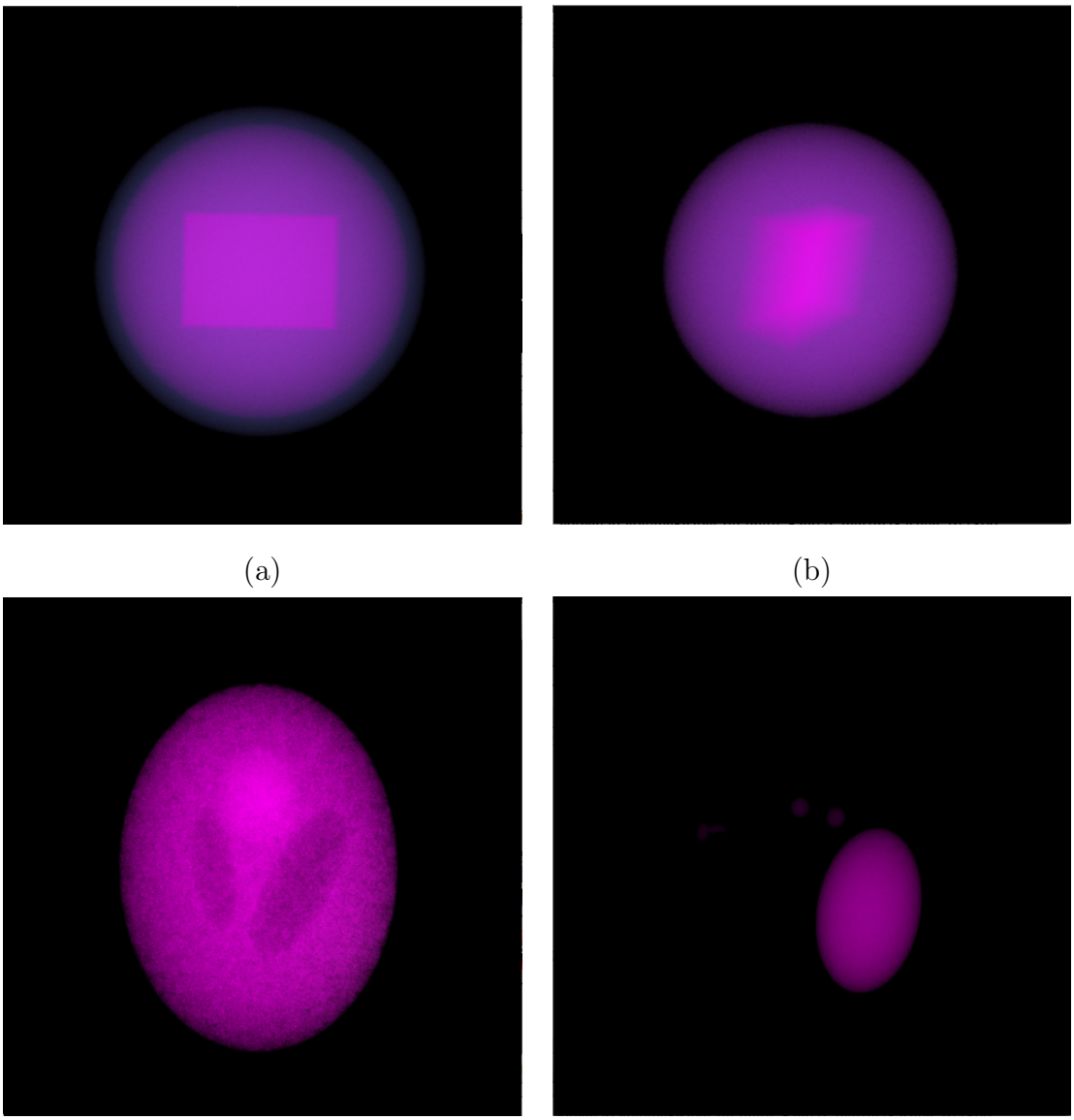
Phantom	Cuerpo	Precisión	Sensibilidad	FPF	FNF	Especificidad
Propio	Fondo	0.99400	0.99403	0.0000919	0.005965	0.999908
	Esfera 1	0.88191	0.95463	0.005931	0.045366	0.994068
	Esfera 2	0.97435	0.99326	0.003786	0.006737	0.996214
	Ladrillo	0.98759	0.99112	0.0000599	0.008873	0.99994
Shepp-Logan	Fondo	1.00000	1.00000	0.00000	0.00000	1.00000
	Elipse 1	0.99998	1.000000	4.976×10^{-7}	0.00000	0.99999
	Elipse 2	0.99988	0.99993	0.0000120	0.000067	0.99998
	Elipses 4	0.99794	0.99915	0.0000138	0.000841	0.99998

Tabla 4.4: Medidas obtenidas de la comparación entre las segmentaciones de los volúmenes con ruido del caso (iii) y los objetos estándar.

Los resultados de estas tablas concuerdan con los resultados reportados en artículos anteriores [16, 17, 53]. Aunque los niveles de ruido de estos experimentos no son tan altos, el algoritmo demuestra cierta robustez para operar en ambientes con ruido y datos de origen diverso. Además, existe la posibilidad de umbralizar los resultados de la segmentación difusa, y por lo tanto, discriminar (o tratar como objetos distin-

tos) aquellos vóxeles que posean bajos valores de conectividad, por ejemplo, en caso de una sobre-segmentación.

Adicionalmente, se realizaron visualizaciones de los volúmenes con ruido (i) y (iii) de cada *phantom*, algunas de las cuales pueden observarse en la Fig. 4.17. Estas proyecciones se utilizaron para calcular las mismas métricas definindas en (4.7), (4.8), (4.9), (4.10) y (4.11) usando una proyección de los *phantom* sin ruido como estándar de oro. Para cada posición, se evaluaron proyecciones que desplegaban todas las clases juntas, y cada una de las clases por separado, el promedio de los resultados por volumen para cada proyección y nivel de ruido pueden observarse en las Tablas 4.5 y 4.6.



(a) (b)

(c) (d)

Figura 4.17: Visualización de los conjuntos de datos sintéticos utilizados para la validación del proyecto, (a) y (b) muestran el volumen Xmipp con diferente orientación, ruido y clases desplegadas, mientras que (c) y (d) muestran el volumen *Shepp-Logan* con diferente orientación, ruido y clases desplegadas.

Las Tablas 4.5 y 4.6 reflejan que los resultados obtenidos en proyecciones bidimensionales de los volúmenes son muy similares a los resultados presentados en las Tablas 4.2, 4.3 y 4.4, que fueron obtenidas por medio de los volúmenes completos, esta similitud es consecuencia de que la visualización se encuentra delimitada por el

Phantom	Orientación	Precisión	Sensibilidad	FPF	FNF	Especificidad
Propio	Frente	0.95580	0.99934	0.016191	0.000650	0.98380
	Proyección	0.89769	0.93733	0.01175	0.062660	0.98824
Shepp-Logan	Frente	0.69465	0.95976	0.096591	0.040238	0.90340
	Proyección	0.71493	0.97466	0.100569	0.025390	0.89942

Tabla 4.5: Medidas obtenidas de la comparación entre las segmentaciones de los volúmenes con ruido y los objetos estándar.

Phantom	Orientación	Precisión	Sensibilidad	FPF	FNF	Especificidad
Propio	Frente	0.97447	0.99985	0.009062	0.000145	0.99093
	Proyección	0.95899	0.99520	0.010183	0.004797	0.98981
Shepp-Logan	Frente	0.99721	0.99961	0.0000781	0.000382	0.99992
	Proyección	0.99793	0.99960	0.000105	0.000398	0.99989

Tabla 4.6: Medidas obtenidas de la comparación entre las segmentaciones de los volúmenes con ruido y los objetos estándar.

proceso de segmentación. Evaluaciones más detalladas del método de segmentación difusa se han realizado en diversas publicaciones [17, 53, 54, 55]. Por el momento, sólo hemos validado que los resultados de la visualización son consistentes con los resultados de la segmentación. Sin embargo, validar aspectos tales como la facilidad para el diseño de las funciones de transferencia o verificar la utilidad de las mismas requiere otro tipo de estudio, en donde se pida a varias personas utilizar el programa y realizar varias visualizaciones con este método. La realización de esta evaluación queda como trabajo futuro, puesto que en el estado actual del proyecto aún no existen las condiciones apropiadas para llevarla a cabo.

Capítulo 5

Conclusiones

Es común que los métodos empleados para el diseño de funciones de transferencia utilicen métodos de segmentación básicos y que, en general, tengan resultados satisfactorios, pero suele ser tedioso encontrar la función de transferencia adecuada. Por lo tanto, es natural pensar que utilizar resultados de métodos de segmentación más elaborados puede ayudar a mejorar el tiempo empleado en la creación de una función de transferencia que provea la visualización más adecuada y resalte los objetos de interés. Como se muestra en los resultados de la Sección 4.2, el programa de segmentación difusa descrito en la Sección 3.2.3 facilita la tarea de aislar objetos, provee un criterio de decisión para casos donde hay más de un material por vóxel y ayuda a generar una función de transferencia inicial por cada objeto en el que fue segmentado el objeto.

Aunque puede comprobarse que el problema del diseño de funciones de transferencia es en esencia un problema de segmentación, no existen muchas referencias en la literatura que ataquen el problema desde ese punto de vista. Precisamente, esa es la principal aportación de este trabajo, mostrar que las técnicas de visualización directa de volúmenes pueden verse beneficiadas al utilizar los avances en el área de segmentación para facilitar el diseño de funciones de transferencia.

Un aspecto importante de tener una función de transferencia por cada clase, es el hecho de que modificar la función de una clase no implica que se modifiquen las otras funciones y, en consecuencia, la visualización de las mismas no se ve afectada (esto se aprecia claramente en las imágenes de la Fig. 4.6). Esta situación no se presenta en

los otros métodos ya que las modificaciones se hacen a todos los vóxeles del volumen que cumplen cierta condición, lo cual llega a provocar que objetos que ya se habían visualizado de forma adecuada vuelvan a perder definición o relevancia.

Adicionalmente, la interfaz gráfica provee una herramienta que facilita la modificación de las funciones de transferencia. Sin embargo, como la asignación de propiedades visuales está en función de la variación en los valores de conectividad, es importante encontrar un punto medio para evitar que se pierdan vóxeles relevantes de la clase que no tienen una conectividad alta sin perder las diferencias locales del objeto que proporcionan información relevante de la estructura del mismo.

Podemos afirmar que la visualización resultante depende en gran medida de la segmentación. Obviamente, la calidad de la misma obedece a la forma de adquisición de los datos y a las propiedades de los materiales en el volumen y su eficacia ha sido estudiada en diversas publicaciones [17, 53, 54, 55]. Vale la pena resaltar que los resultados visuales son subjetivos y dependen en gran medida de las necesidades del usuario; sin embargo, en este proyecto se presenta una solución general que puede adaptarse a dichas necesidades.

Finalmente, es importante recalcar la importancia de las tarjetas gráficas programables. Su poder de procesamiento y costo relativamente bajo ha hecho posible grandes avances en campos que requieren el procesamiento de grandes cantidades de datos, permitiendo realizar tareas de manera paralela, incluyendo aquellas que típicamente no se realizan dentro del procesador gráfico de las mismas. Por ejemplo, la posibilidad de llevar a cabo la evaluación de la función de transferencia, el *raycasting* y la composición de forma paralela.

5.1. Trabajo Futuro

Uno de los principales objetivos a futuro es terminar de acoplar el programa de segmentación difusa y el programa visualizador, es decir, hace falta agregar los módulos y procesadores que permitan llamar al programa de segmentación desde el programa *Voreen* y así todo el proceso se puede llevar a cabo bajo una misma interfaz.

De igual forma, otro objetivo importante es paralelizar el algoritmo de segmentación difusa (implementándolo en GPU de forma similar a lo realizado en [63]), ya que es la tarea que más tiempo consume de todo el proceso. Este paso es fundamental para lograr que la generación de funciones de transferencia y, por lo tanto, todo el proceso de visualización pueda hacerse de forma interactiva.

Debido a la flexibilidad con la que se pueden definir las funciones de transferencia para cada clase, también es posible probar con la creación de otras funciones de transferencia iniciales. Por ejemplo, utilizar una aproximación parecida al proceso de ecualización del histograma para los valores de conectividad, o bien, incluir alguna transformación de potencia para dar más flexibilidad al proceso de enfatizar ciertas características del volumen. Otra opción interesante consiste en incluir los valores de intensidad originales para generar los valores de color y opacidad finales.

También es importante realizar varias pruebas del programa con usuarios entrenados, una vez que ambos programas se encuentren bajo la misma interfaz y, de esta forma, verificar si se facilita el trabajo de encontrar una función de transferencia que cumpla con un objetivo establecido.

Finalmente, la elección del método de segmentación se realizó con base en la experiencia que se ha tenido con ese algoritmo particular. Sin embargo, no se descarta la posibilidad de experimentar con otros métodos de segmentación que sean más rápidos o provean mejores resultados para tipos de datos particulares.

Bibliografía

[1] J. A. García, "Visualización suavizada de superficies obtenidas por rastreo de fronteras aplicado a volúmenes discretizados," Master's thesis, Universidad Nacional Autónoma de México, 2011.

[2] http://voreen.org/.

[3] J. Kruger and R. Westermann, "Acceleration techniques for GPU-based volume rendering," in *Proceedings of the 14th IEEE Visualization 2003*, 2003.

[4] W. Schroeder, L. Avila, K. Martin, W. Hoffman, and C. Law, "The visualization toolkit user's guide," 01 2001.

[5] C. Bajaj, S. Park, and A. G. Thane, "Parallel multi-pc volume rendering system," cs & ticam technical report, University of Texas at Austin, 2002.

[6] G. Kindlmann and J. W. Durkin, "Semi-automatic generation of transfer functions for direct volume rendering," *Proceedings of the IEEE Symp. Volume Visualization*, pp. 79–86, 1998.

[7] J. Marks, B. Andalman, P. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber, "Design galleries: A general approach to setting parameters for computer graphics and animation," *SIGGRAPH Conference Proceedings*, pp. 389–400, 1997.

[8] H. Pfister, B. Lorensen, and C. Bajaj, "The transfer function bake-off," *IEEE Computer Graphics and Applications*, pp. 16–22, 2001.

[9] H. Hagen, A. Ebert, R. H. van Lengen, and G. Scheuermann, "Scientific visualization: Methods and applications," *Proceedings 19th Spring Conference on Computer Graphics*, pp. 23–33, 2003.

[10] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, and R. L. Phillips, *Introduction to Computer Graphics.* Addison-Wesley, 1993.

[11] R. C. Gonzalez and R. E. Woods, *Digital Image Processing.* Prentice-Hall, 2001.

[12] J. Freixenet, X. Munoz, D. Raba, J. Marti, and X. Cufi, "Yet another survey on image segmentation: Region and boundary information integration," in *Computer Vision - ECCV 2002 PT III*, vol. 2352, pp. 408–422, 2002.

[13] R. M. Haralick and L. G. Shapiro, "Image segmentation techniques," *Computer Vision, Graphics, and Image Processing*, vol. 29, pp. 100–132, 1985.

[14] T. Mcinerney and D. Terzopoulos, "Deformable models in medical image analysis: A survey," *Medical Image Analysis*, vol. 1, pp. 91–108, 1996.

[15] N. R. Pal and S. K. Pal, "A review on image segmentation techniques," *Pattern Recognition*, vol. 26, pp. 1277–1294, 1993.

[16] B. M. Carvalho, G. T. Herman, and T. Y. Kong, "Simultaneous fuzzy segmentation of multiple objects," *Discrete Applied Mathematics*, vol. 151, pp. 55–77, 2005.

[17] E. Garduño, M. Wong-Barnum, N. Volkmann, and M. H. Ellisman, "Segmentation of electron tomographic data sets using fuzzy set theory principles," *Journal of Structural Biology*, vol. 162, pp. 368–379, 2008.

[18] C. D. Correa and K.-L. Ma, "Visibility-driven transfer functions," *IEEE Pacific Visualization Symposium*, pp. 177–184, 2009.

[19] G. T. Herman, *Geometry of Digital Spaces.* Birkhaüser Boston, 1998.

[20] E. Garduño, G. T. Herman, and R. Davidi, "Reconstruction from a few projections by ℓ_1-minimization of the haar transform," *Inverse Problems*, vol. 27, 2011.

[21] R. Strand and G. Borgefors, "Distance transform for three-dimensional grids with non-cubic voxels," *Computer Vision and Image Understanding*, vol. 100, pp. 294–311, 2005.

[22] R. A. Debrin, L. Carpenter, and P. Hanrahan, "Volume rendering," *Computer Graphics*, vol. 22, pp. 65–74, 1988.

[23] J. F. Blinn, "A generalization of algebraic surface drawing," *ACM Transactions on Graphics*, vol. 1, pp. 235–256, 1982.

[24] E. Garduño and G. T. Herman, "Implicit surface visualization of reconstructed biological molecules," *Elsevier Theoretical Computer Science*, vol. 346, pp. 281–299, 2005.

[25] M. Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics and Applications*, vol. 8, pp. 29–37, 1988.

[26] E. Catmull and J. Clark, "Recursively generated b-spline surfaces on arbitrary topological meshes," *Computer Aided Design*, vol. 10, pp. 350–355, 1978.

[27] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer Graphics*, vol. 21, pp. 163–169, July 1987.

[28] A. Litvinenko and V. Usov, "About interactive modeling system with the help of implicit function," in *International Conference Graphicon*, 2005.

[29] K. A. Frenkel, "Volume rendering," *Communications of the ACM*, vol. 32, pp. 426–435, 1989.

[30] T. He and A. E. Kaufman, "Fast stereo volume rendering," in *Proceedings of the 7th conference on Visualization '96*, pp. 49–56, 1996.

[31] H. D. Luke, "The origins of the sampling theorem," *Communications Magazine, IEEE*, pp. 106–108, 1999.

[32] E. Cerezo, F. Pérez, X. Pueyo, F. J. Seron, and F. X. Sillion, "A survey on participating media rendering techniques," *The Visual Computer*, vol. 21, pp. 303–328, 2005.

[33] K. D. Moreland, *Fast high accuracy volume rendering.* Tesis doctoral, University of New Mexico, Albuquerque, New Mexico, USA, July 2004.

[34] B. T. Phong, "Illumination for computer generated pictures," *Communications of the ACM*, vol. 18, pp. 311–317, June 1975.

[35] P. Haeberli, M. Segal, and S. G. C. System, "Texture mapping as a fundamental drawing primitive," *Proceedings Fourth Eurographics Workshop on Rendering*, pp. 259–266, 1993.

[36] N. Max and B. Becker, "Flow visualization using moving textures," *Proceedings of ICASE/LaRC Symposium on Visualizing Time Varying Data*, 1995.

[37] I. Viola, A. K. M. E. Gröller, A. Kanitsar, and M. E. Gröller, "Importance-driven volume rendering," in *In Proceedings of IEEE Visualization*, pp. 139–145, 2004.

[38] G. Kindlmann and D. Weinstein, "Hue-balls and lit-tensors for direct volume rendering of diffusion tensor fields," in *Proceedings of the conference on Visualization '99: celebrating ten years*, pp. 183–189, 1999.

[39] G. Kindlmann, D. Weinstein, and D. Hart, "Strategies for direct volume rendering of diffusion tensor fields," *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, pp. 124–138, 2000.

[40] http://www.kitware.com/products/volview.html.

[41] C. L. Bajaj, V. Pascucci, and D. R. Schikore, "The contour spectrum," *Proceedings of theIEEE Visualization Conf*, pp. 167–173, 1997.

[42] J. Kniss, G. Kindlmann, and C. Hansen, "Multidimensional transfer functions for interactive volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, pp. 270–285, 2002.

[43] http://www.cs.utah.edu/ jmk/simian/.

[44] M. F. Cohen, J. Painter, M. Mehta, and K.-L. Ma, "Volume seedlings," *Symposium on Interactive 3D Graphics*, pp. 139–145, 1992.

[45] http://www.amira.com/.

[46] R. Acharya, R. P. Menon, A. Singh, D. Goldgof, and D. Terzopoulos, "A review of biomedical image segmentation techniques," in *Deformable Models in Medical Image Analysis*, IEEE Computer Society, 1998.

[47] P. Maragos and R. W. Schafer, "Morphological systems for multidimensional signal processing," *Proceedings of the IEEE*, vol. 78, pp. 690–710, 1990.

[48] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham, "Active shape models-their training and application," *Computer Vision and Image Understanding*, vol. Vol. 61, No. 1, pp. 38–59, 1995.

[49] S. Beucher and C. Lantuejoul, "Use of watersheds in contour detection," in *International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation*, 1979.

[50] S. Beucher and C. D. M. Mathmatique, "The wathershed transformation applied to image segmentation," in *Scanning Microscopy*, pp. 299–314, 1991.

[51] J. Smolka, "Watershed based region growing," in *Annales UMCS Informatica AI 3*, pp. 169–178, 2005.

[52] N. Volkmann, "A novel three-dimensional variant of the watershed transform for segmentation of electron density maps," *Journal of Structural Biology*, vol. 138, pp. 123–129, 2002.

[53] B. M. Carvalho, E. Garduño, and G. T. Herman, "Multiseeded fuzzy segmentation on the face centered cubic grid," in *Proceedings Second International Conference on Advances in Pattern Recognition ICAPR 2001*, 2002.

[54] S. Dellepiane and F. Fontana, "Extraction of intensity connectedness for image processing," *Pattern Recognition Letters*, vol. 16, pp. 313–324, 1995.

[55] J. K. Udupa and P. K. Saha, "Fuzzy connectedness and image segmentation," *Proceedings of the IEEE*, vol. 91, pp. 1649–1669, 2003.

[56] G. T. Herman and B. M. Carvalho, "Multiseeded segmentation using fuzzy connectedness," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, pp. 460–474, 2011.

[57] J. K. Udupa and S. Samarasekera, "Fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation," *Graphical Models and Image Processing*, vol. 58, pp. 246–261, 1996.

[58] B. M. Carvalho, C. J. Gau, G. T. Herman, and T. Y. Kong, "Algorithms for fuzzy segmentation," *Pattern Analysis and Applications*, vol. 2, pp. 73–81, 1999.

[59] J. K. Udupa, P. K. Saha, and R. A. Lotufo, "Relative fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 1485–1500, 2002.

[60] K. C. Ciesielski and J. K. Udupa, "Affinity functions in fuzzy connectedness based image segmentation I: Equivalence of affinities," *Computer Vision and Image Understanding*, vol. 114, pp. 146–154, 2010.

[61] K. C. Cielsielski and J. K. Udupa, "Affinity functions in fuzzy connectedness based image segmentation II: Defining and recognizing truly novel affinities," *Computer Vision and Image Understanding*, vol. 114, pp. 155–166, 2010.

[62] A. S. Pednekar and I. A. Kakadiaris, "Image segmentation based on fuzzy connectedness using dynamic weights," *IEEE Transactions on Image Processing*, vol. 15, pp. 1555–1562, 2006.

[63] Y. Zhuge, R. W. Miller, Y. Cao, and J. K. Udupa, "Parallel fuzzy connected image segmentation on GPU," *Medical Physics*, vol. 38, 2011.

[64] http://en.wikipedia.org/wiki/MRC_(file_format).

[65] http://developer.nvidia.com/category/zone/cuda-zone.

[66] http://www.vtk.org/.

[67] R. Marroquim and A. Maximo, "Introduction to GPU programming with GLSL," in *XXII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI Tutorials) 2009*, pp. 3–16, October 2009.

[68] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl, "A simple and flexible volume rendering framework for graphics-hardware-based raycasting," in *Proceedings of the International Workshop on Volume Graphics '05*, pp. 187–195, 2005.

[69] F. Randima, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004.

[70] S. Roettger, "The volume library." http://www9.informatik.uni-erlangen.de/External/vollib/.

[71] http://xmipp.cnb.uam.es/twiki/bin/view/Xmipp/WebHome.

[72] http://tomography.o-x-t.com/.