UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE UNA APLICACIÓN PARA LA VISUALIZACIÓN, MANIPULACIÓN Y SEGMENTACIÓN DE VOLÚMENES

T E S I S

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTAN:
OSCAR ROMERO HERNÁNDEZ
ENRIQUE RAMÓN TAPIA PATLÁN

DIRECTOR DE TESIS:
DR. BORIS ESCALANTE RAMÍREZ

MÉXICO, D. F. 2007

Agradecimientos

A mis padres, quienes han sido mi principal e incondicional apoyo durante toda mi vida, y especialmente, en esta etapa. Este logro también es suyo. Gracias.

A mi hermana y su esposo por todo el apoyo y la comprensión que me han tenido.

A mi sobrino. Espero que algún día logres esto y más.

A todos mis familiares que confiaron y creyeron en mí. Espero seguir siendo alguien de quien puedan sentirse orgullosos.

Al Dr. Boris Escalante Ramírez por todo su apoyo y orientación durante el desarrollo de este proyecto. Y también, por haberme introducido en el interesante campo del Procesamiento Digital de Imágenes. Gracias por todo.

A Enrique, mi compañero de tesis, por su trabajo duro y constante en este proyecto. Y sobre todo, por ser un gran amigo.

A mis compañeros de laboratorio, por su ayuda y consejos en temas difíciles durante el desarrollo de este trabajo. Y, desde luego, por brindarme su amistad y compañía durante todo este tiempo.

A mis amigos por su apoyo y consejos. Especialmente a Adriana e Iván. Ojalá también alcancen sus metas.

A la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería, magníficas instituciones que me brindaron mi formación académica y personal. Siempre les estaré agradecido.

A mis profesores por haberme transmitido sus conocimientos. Son un ejemplo a seguir.

A Héctor Barrón por toda su ayuda para solucionar los inconvenientes que se presentaron en el Observatorio iXtli.

A la Dirección General de Servicios de Cómputo Académico por los cursos y las facilidades brindados para poder trabajar en el Observatorio iXtli.

Al proyecto iXtli IN500604 por el apoyo económico brindado durante la elaboración de este trabajo.

Oscar Romero Hernández

Agradecimientos

A mis padres, Ramón y María Esther, por todo el apoyo y los consejos que me han proporcionado tanto en mi formación profesional como en la vida misma. Gracias por ayudarme a salir adelante para cumplir finalmente con este objetivo. Saben que todos mis éxitos son también de ustedes.

A mis abuelos, José y Luciana, por acompañarme y ayudarme a lo largo de mis estudios profesionales.

A mis tías, Graciela y María Luz, que también me acompañaron en esta etapa de mi vida.

A mis tíos, Salvador y Ana María, porque siempre he sentido su apoyo.

A mis familiares que siempre han estado al pendiente de mí y que me han otorgado su ayuda incondicional.

Al Dr. Boris Escalante Ramírez, por su orientación y consejos en la elaboración de este trabajo y además por despertar en mí un interés especial en el procesamiento digital de imágenes.

A mi compañero de tesis, Oscar, con quien hice un gran equipo para poder lograr la realización de este proyecto. Gracias por tus puntos de vista y en especial, por brindarme tu amistad.

A mis compañeros del Laboratorio de Procesamiento Digital de Imágenes, por hacer más amenas las horas de trabajo. Gracias por su compañía y amistad.

A la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería, por la formación académica recibida. Nunca olvidaré mi paso por estas grandes instituciones.

A la Dirección General de Servicios de Cómputo Académico, por complementar mi formación y por permitirme el poder trabajar en el Observatorio iXtli.

A Héctor Barrón, por toda la ayuda prestada en las etapas iniciales de este proyecto.

Un agradecimiento especial al proyecto iXtli IN500604 por el apoyo económico recibido durante el desarrollo de este trabajo.

Enrique Ramón Tapia Patlán.

CONTENIDO

RESUMEN

En esta tesis se presenta la elaboración de una aplicación para la visualización, manipulación y segmentación de volúmenes capaz de funcionar en diversas plataformas (entiéndase como *volumen* el conjunto de datos que representan una entidad de tres dimensiones). Este proyecto surge de la necesidad de ampliar y mejorar el software desarrollado en el laboratorio de Procesamiento Digital de Imágenes que tenía el objetivo de analizar, usando la transformada de Hermite, secuencias de imágenes considerándolas como un volumen da datos. La ampliación consiste, principalmente, en la inclusión de más herramientas para dicha aplicación, además de la anexión de un módulo de segmentación (en términos del procesamiento digital de imágenes).

Partiendo de lo anterior, a la aplicación que aquí se presenta se le dio el enfoque del análisis de volúmenes en sí. En resumen, se añadieron módulos para un manejo más cómodo de los volúmenes cargados: mejor control de rotaciones, aplicación de planos de corte, manejadores de umbralización, visión de líneas de guía y una mejora en los parámetros de visualización. La descripción de dichos módulos, así como su implementación, es el tema central de este escrito.

Dado que, primordialmente, se le ha dado mayor prioridad a la segmentación de volúmenes basados en imágenes médicas, se buscó implementar una técnica especializada (pero no exclusiva) para ese tipo de imágenes y, además, ampliándola a tres dimensiones. *Fuzzy c-means con información espacial* es la técnica elegida, por lo que es la que se explica de manera más amplia, detallando aspectos de su implementación en la aplicación.

Este documento parte de los fundamentos básicos del procesamiento digital de imágenes, para después proceder con la segmentación de imágenes y así brindar de un marco teórico sólido a la aplicación. A continuación se muestra un análisis de requerimientos, el diseño de la misma y la implementación de los diversos módulos. La aplicación finalizada y los resultados que arroja constituyen las secciones finales del trabajo.

INTRODUCCIÓN.

En el ámbito del procesamiento digital de imágenes existe una gran cantidad de aplicaciones que realizan diversos procedimientos como el mejoramiento de la nitidez, la detección de contornos, el cambio de tonalidades de luminiscencia o la compresión de la información visual por mencionar sólo algunos de estos procedimientos. Sin embrago, únicamente aplicaciones especializadas manejan reconstrucciones y análisis de datos en tercera dimensión.

La visualización de volúmenes es una de las mejores maneras para entender los largos y complejos conjuntos de datos producidos por las observaciones o simulaciones de procesos de ciencia o ingeniería. Las técnicas de visualización de volúmenes presentan los datos de tal manera que aprovechan la innata capacidad del sistema de visión humano para distinguir diversos patrones, tendencias y anomalías en entornos visuales. Por otra parte, estas técnicas habilitan al usuario para que interprete los datos a través de sus conocimientos de ciencia o ingeniería lo cual ayuda a científicos e ingenieros a crear mejores soluciones de manera rápida para los diversos problemas a los que se enfrentan.

El desarrollo de un software que permita el manejo de información visual volumétrica con el objetivo de detectar diversas características de dicho volumen es, en ocasiones, exclusivo de aquellas personas u organizaciones que manejan una gran cantidad de datos y que, a su vez, necesitan visualizarlos para detallar los aspectos más íntimos de su información.

La aplicación que se presenta en el presente trabajo está encaminada a la reconstrucción en tercera dimensión de información médica, principalmente de resonancia magnética y de tomografía computarizada[1], aunque puede generalizarse a cualquier tipo de información que pueda visualizarse *ya que se trata de una aplicación con fines, principalmente, didácticos*.

[1] Todas las imágenes de tomografía computarizada y resonancia magnética presentes en esta tesis son cortesía del Instituto de Óptica de Madrid, España.

El objetivo de desarrollar una aplicación de este tipo consiste en proporcionar una herramienta multiplataforma de código original que pueda ser utilizada en cualquier computadora con los recursos suficientes, que procese información volumétrica y que dé paso al análisis y entrega de resultados visuales que proporcionen un apoyo útil para el usuario final.

Entre los principales aspectos desarrollados se encuentra la codificación de diferentes algoritmos de segmentación de imágenes, la manipulación del volumen de datos en el espacio de tres dimensiones controlando (principalmente) rotaciones y translaciones y el seccionado de dicho volumen aplicando cortes transversales y diagonales. El hecho de incluir como parte fundamental el segmentado del volumen de datos como un todo invita a hacer parte de este proyecto a la teoría del reconocimiento de patrones, la cual nos permite a través del uso de la computadora digital detectar las diferentes clases de datos relacionadas del propio volumen.

La elección de las técnicas utilizadas para la segmentación del volumen de datos obedece a diversos aspectos. En primer lugar se tomaron en cuenta a aquellas técnicas que fueran no supervisadas y que, de cierta manera, siguieran un proceso generalizado para la segmentación. También se dejaron de lado varias técnicas clásicas de segmentación ya que se buscaba el uso de una técnica novedosa que pudiera aplicarse a diversos ámbitos; se buscó, además, que la complejidad computacional fuese sencilla y que requiriera de tiempos de cómputo relativamente cortos.

Por otro lado, para generalizar la aplicación, se elaboraron funciones para poder cargar diferentes tipos de volúmenes de datos, de esta manera se puede visualizar tanto información médica como cualquier otro conjunto de imágenes que pueden ser modelos de distribución de probabilidad de los electrones en un átomo de hidrógeno o secuencias de movimiento, por mencionar algunos ejemplos.

El uso de la transformada Hermitiana para la detección de estructuras y contornos es útil, por ejemplo, para delimitar las secciones de un aneurisma, observar exclusivamente cambios de contraste en las tres direcciones espaciales que se manejan, u observar las estructuras detectadas en su conjunto por medio de la combinación de los diferentes coeficientes de dicha transformada. Se demostrará que esta herramienta matemática permite obtener más información visual del volumen de datos original cargado en la aplicación.

Resulta importante mencionar que el presente trabajo surgió como complemento del proyecto "*Análisis de secuencias de imágenes utilizando la transformada hermitiana y visualización en 3D*"[2] que tiene cimentado su origen en la colaboración de la Facultad de Ingeniería con el observatorio de visualización *iXtli* de la Universidad Nacional Autónoma de México a través del programa PAPIIT y es la expansión de dicho análisis a volúmenes. De esta forma, este proyecto pretende abarcar algo más que secuencias espacio-temporales.

En los capítulos siguientes se tratan los distintos aspectos involucrados con la elaboración de la aplicación. Primeramente, en el capítulo uno se presenta el sistema de visión humano como punto de partida para la comprensión de cómo los seres humanos visualizamos las cosas; enseguida se aborda el filtrado espacial comenzando con los filtros que se relacionan directamente con las derivadas de funciones Gaussianas para después continuar con la transformada Hermitiana, ya que ésta se emplea como herramienta adicional de visualización dada su estrecha relación con los campos receptivos existentes en el sistema de visión humano. Seguidamente se abordan en el capítulo dos conceptos sobre la segmentación de imágenes, detallando aspectos importantes tomados en cuenta para la realización de la aplicación, principalmente detalles de los métodos por agrupamiento. A continuación se presenta, en el capítulo tres, un análisis particular referente al método c medias difuso con información espacial. Después, en el capítulo cuatro, se muestra el desarrollo de la aplicación mencionando su diseño y estructura completa, para finalmente, en el capítulo número cinco establecer conclusiones sobre la aplicación final.

[2] Suárez González, C .E., Juárez Acosta, G., Wong Mosqueda J. A. "*Análisis de secuencias de imágenes utilizando la transformada hermitiana y visualización 3D*". Tesis de Licenciatura (Ingeniero en Computación). Director: Dr. Boris Escalante Ramírez. Universidad Nacional Autónoma de México, Facultad de Ingeniería, 2006.

1. CONCEPTOS PRELIMINARES

1.1. El sistema de visión humano [1].

A pesar de que el procesamiento digital de imágenes está construido sobre fundamentos matemáticos, en muchas ocasiones, al momento de seleccionar alguna técnica en lugar de otra, los juicios visuales subjetivos que aporta algún observador humano son de gran relevancia. Es por esto que resulta importante el estudio de los aspectos básicos del sistema de visión humano, especialmente las características en sus etapas más tempranas.

1.1.1. Estructura del ojo humano.

El ojo humano puede modelarse como si éste fuese una esfera cuyo diámetro aproximado es de 20 milímetros. En la figura 1.1 puede observarse la estructura del ojo humano de manera simplificada.

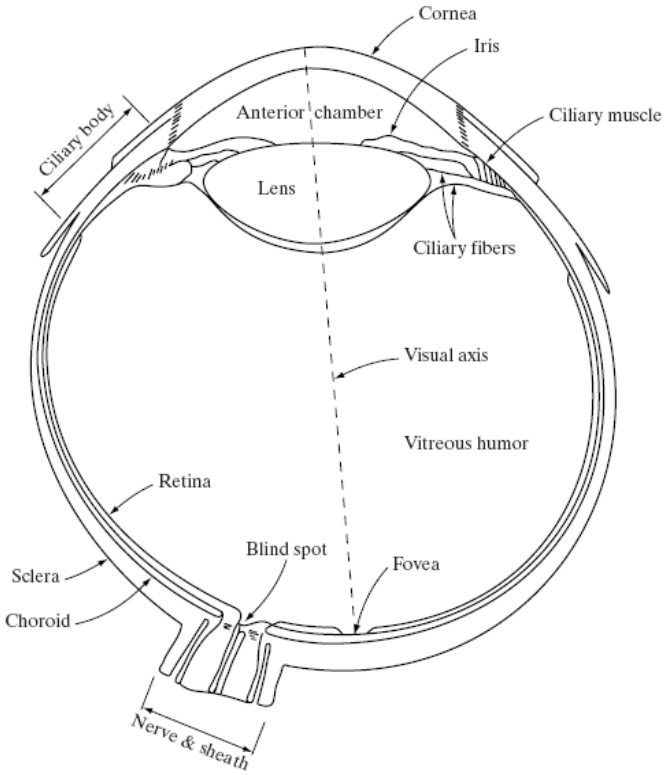


Figura 1.1. Diagrama del ojo humano [1]

El ojo humano está encapsulado por tres membranas: la córnea junto con la esclerótica en la parte exterior, la coroides y la retina. La córnea es un tejido transparente y resistente que cubre la superficie anterior del ojo. En conexión con la córnea y sobre el mismo nivel de capas se tiene a la esclerótica, que es una membrana oscura que envuelve al resto de componentes del globo ocular.

Debajo de la esclerótica se localiza la coroides, la cual contiene una red de vasos sanguíneos cuya función es la de nutrir al globo ocular, principalmente a la retina. Esta capa además se encuentra altamente pigmentada y ayuda a reducir el ingreso de luz adicional al ojo, regulando de esta forma la cantidad de energía luminosa que incide en el ojo. En su parte anterior la coroides se divide en dos partes: el cuerpo ciliar y el diafragma del iris. El diafragma del iris se contrae o se expande para regular la cantidad de luz que entra directamente al ojo. La pupila, que es la entrada primaria del iris, varía aproximadamente de dos a ocho milímetros en su diámetro. La parte frontal del iris contiene la pigmentación visible del ojo mientras que la parte posterior contiene un pigmento negro.

La lente o bien, el cristalino, está compuesto por capas concéntricas de células fibrosas y se encuentra sostenido por fibras que unen esta estructura con el cuerpo ciliar. El cristalino está compuesto aproximadamente en su 60% o 70% por agua, cerca del 6% son lípidos y contiene más proteínas que cualquier otro tejido del ojo humano. Este tejido está coloreado de manera suave por un pigmento amarilloso el cual se incrementa con la edad. El cristalino absorbe aproximadamente el 8% del espectro de luz visible. Sin embargo, tanto la luz ultravioleta como la luz infrarroja pueden ser absorbidas apreciablemente por proteínas contenidas en el cristalino, lo cual puede ser dañino si la absorción de dicho tipo de luz es excesiva.

La retina es la membrana ocular que se localiza en la parte interior del ojo y recubre toda la pared interna del mismo en su parte posterior. Cuando el ojo es correctamente enfocado, la luz proveniente de un objeto en el exterior es reflejada en la retina. El patrón de visión es producido por la distribución de receptores de luz discretos ubicados en la superficie de la retina. En el ojo humano existen dos clases de receptores de luz: los conos y los bastones.

Los conos alcanzan un número que va de 6 a 7 millones de elementos por cada ojo y se encuentran localizados principalmente en la porción central de la retina llamada fóvea y son muy sensibles al color. Gracias a estos elementos se pueden resolver detalles finos puesto que cada cono tiene su propia terminal nerviosa. Músculos que controlan el movimiento del globo ocular, hacen rotar al ojo hasta que la imagen del objeto de interés cae directamente sobre la fóvea, con lo cual se logran enfocar los detalles de dicha imagen. A este tipo de visión se le denomina visión fotópica o de luz brillante.

El número de bastones es mucho mayor con respecto al número de conos, teniéndose aproximadamente alrededor de 75 o 150 millones de bastones distribuidos en toda la superficie de la retina de cada ojo humano. Su gran distribución y el hecho de que un número alto de bastones se conectan en conjunto a una sola terminal nerviosa disminuyen el número de detalles que pueden ser identificados por los bastones. Los bastones sirven para otorgar una panorámica total y general del campo de visión; no se involucran en la recepción del color y son sensibles a niveles bajos de iluminación. Así, por ejemplo, comúnmente por la noche, los diversos objetos lucen colores grisáceos ante el ojo humano ya que sólo son estimulados los bastones. Este fenómeno es conocido como visión escotópica o de luz débil. La retina posee una región en la cual no tiene receptores de ningún tipo, la cual es llamada “punto ciego”. En la figura 1.2 se aprecia la densidad de distribución de los elementos receptores de luz en la retina:

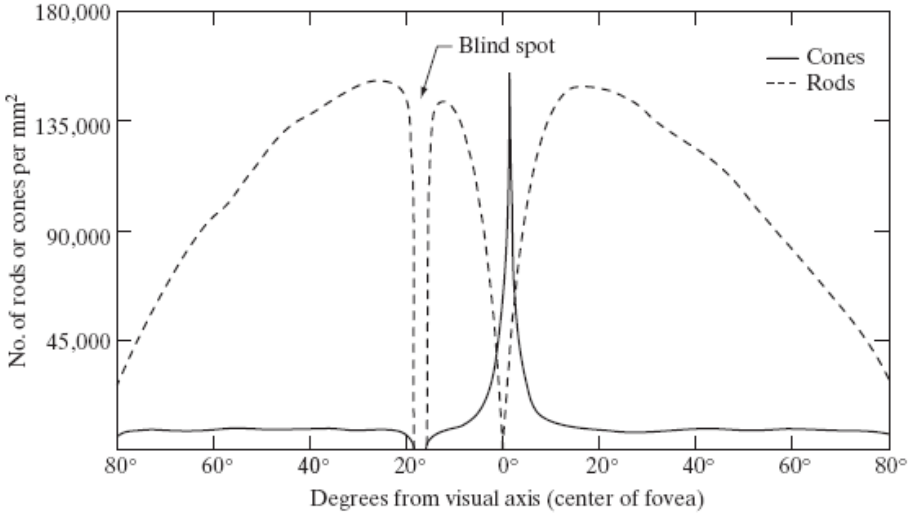


Figura 1.2. Distribución de conos y bastones en la retina [1]

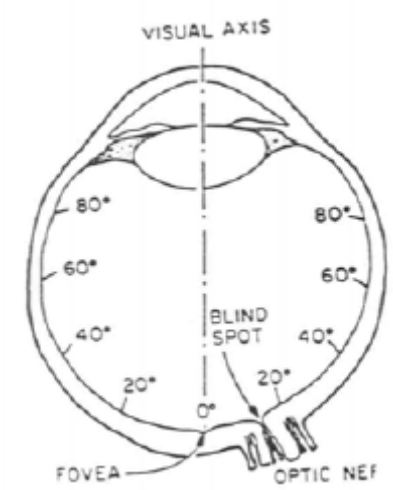


Figura 1.3. Ordenamiento en grados de la superficie de la retina [2]

De la figura 1.2 se observa que la distribución de los receptores es radialmente simétrica alrededor de la fóvea y la distribución de los receptores se mide en grados a partir de ella (0°) cómo se muestra en la figura 1.3. Puede notarse que los conos se encuentran en su mayoría en la parte central de la retina (fóvea), mientras que los bastones aumentan en intensidad a aproximadamente 20° desde la fóvea, apreciándose también, que el número de bastones decae en densidad hacia los extremos, en la periferia de la retina.

En su conjunto, conos y bastones proveen una visión de una alta resolución, capaz de percibir el color y que además posee una alta sensitividad para las frecuencias espaciales intermedias (por ejemplo, los bordes o contornos) [3].

1.1.2. La inhibición lateral.

Resulta importante estudiar los mecanismos de la percepción visual humana, ya que éstos han proporcionado información importante para el desarrollo de técnicas y algoritmos de tratamiento de imágenes. Además, comprender su funcionamiento ayuda a establecer la relación concreta entre estos mecanismos y la composición de los filtros basados en derivadas de la función Gaussiana.

Las células fotosensibles (conos y bastones) se encuentran conectadas con las células ganglionares a través de células intermediarias. Las células ganglionares funcionan de acuerdo a la estimulación de su campo receptivo, el cual se compone por el área de la retina que dicha célula controla, es decir, un conjunto de conos y bastones.

Dado que los conos funcionan mal en condiciones de baja luminosidad, son, a su vez, altamente eficaces en la detección de contornos, bordes y contrastes cuando hay condiciones de alta luminosidad. Por tal razón, en lo que respecta a la detección de estructuras, la información que procede de los conos debe ser procesada de manera más selectiva, y es por esto que, en condiciones normales, cada cono está conectado a una célula ganglionar a través de una célula bipolar mientras que un conjunto de bastones es conectado con una sola célula ganglionar. De lo anterior, resulta que el campo receptivo de una célula ganglionar conectada con bastones es mayor al de una célula ganglionar conectada con un cono [4].

Las células fotosensibles actúan como transductores que transforman la luz incidente a impulsos eléctricos. La información proporcionada por dichos impulsos eléctricos es dirigida hacia las células bipolares, las cuales transmiten dicha información

hacia las células ganglionares. Sin embargo, otras estimulaciones provenientes de las células horizontales (también conectadas con los fotorreceptores) y las células amácrinas forman en conjunto la estimulación del campo receptivo (el conjunto de estas células se aprecia en la figura 1.4). La conjunción de estímulos que recibe la célula ganglionar define el comportamiento de su campo receptivo, que puede ser idealizado como una forma circular compuesta de un centro de excitación y una periferia de inhibición. En dicho campo receptivo de una célula ganglionar se produce el fenómeno conocido como inhibición lateral.

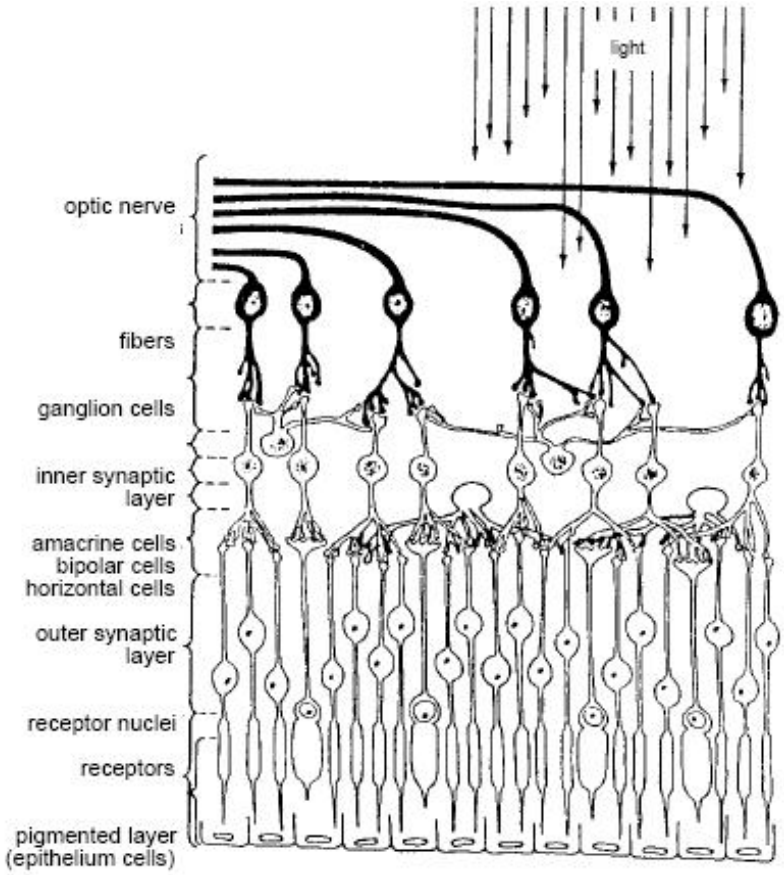


Figura 1.4. Células ubicadas en la retina [5]

Las activaciones provenientes del centro y la periferia del campo receptivo se suman, pero la activación general de la célula ganglionar implica el cómputo diferencial de las dos partes (un gráfico de esto puede apreciarse en la figura 1.5).

Si las dos partes del campo receptivo reciben el mismo tipo de estimulación sus efectos son sumados y se alcanza un nivel alto de activación ganglionar; por el contrario, si se producen efectos inversos en el centro y periferia del campo receptivo, las dos regiones son antagonistas y la respuesta ganglionar es baja [4]. Psicológicamente, esto es importante ya que de esta manera el cerebro no es bombardeado con grandes cantidades de imágenes innecesarias [6]. Esta diferencial de estímulos es conocida como inhibición lateral. Dado este fenómeno, la visión humana es sensible a contrastes de luminiscencia más que a niveles absolutos de luminiscencia.

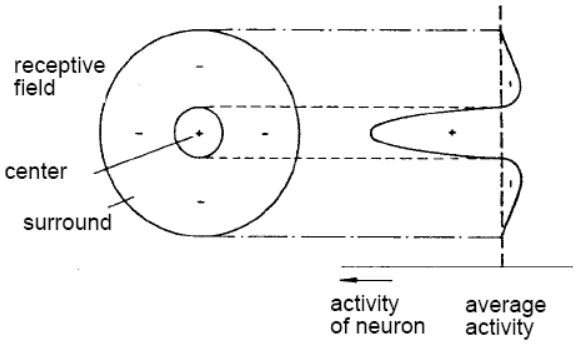


Figura 1.5. Campos Receptivos [5]

1.2. Filtros basados en derivadas de la función Gaussiana [7].

El proceso de filtrado es muy común (y casi esencial) en el procesamiento digital de imágenes, sin embargo, para este trabajo, interesan principalmente los filtros provenientes del modelado discreto de las derivadas de la función Gaussiana ya que, además de tener un fundamento matemático, su comportamiento es congruente con el sistema de visión humano y se especializan en la detección de cambios bruscos, como lo son los bordes o contornos. El fenómeno de la inhibición lateral presente en los campos receptivos de las células ganglionares puede ser modelado como una diferencia de Gaussianas [8]. Una función Gaussiana tiene la siguiente expresión:

$$f(x) = N \exp(-ax^2) \tag{1}$$

Donde N es una constante de normalización que depende de *a* pero no de *x*. Existen varias formas para el exponente *a*. La forma estándar es $a = \frac{1}{2\sigma^2}$. La primera derivada de Gaussiana tiene la siguiente forma:

$$f(x) = -N2a(x)\exp(-ax^2) \tag{2}$$

Mientras que la segunda derivada tiene la siguiente expresión:

$$f(x) = N2a(2ax^2 - 1)\exp(-ax^2) \tag{3}$$

A continuación, en la figura 1.6 se muestran las gráficas para las cuatro primeras derivadas de la función Gaussiana con los parámetros *N=a=1*.

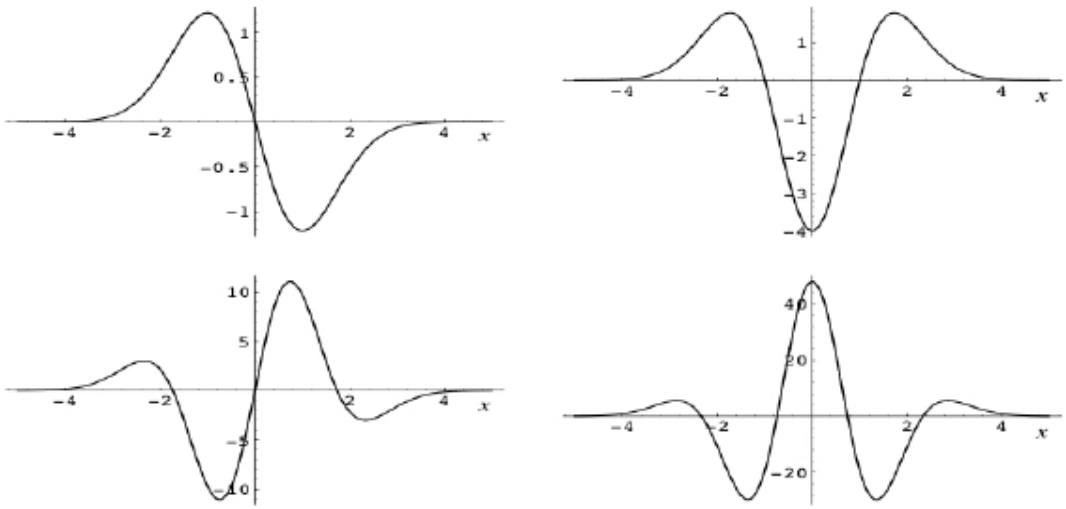


Figura 1.6. Las cuatro primeras derivadas de la función Gaussiana [7]

Las derivadas de las funciones continuas son definidas en las funciones discretas como funciones de diferencias. La forma discreta de la primera derivada de Gaussiana puede entonces escribirse como:

$$f(x) - f(x-1) \tag{4}$$

Por otro lado, se desea diseñar un filtro espacial $g(x) = h(x) * f(x)$ tal que $g(x) = f(x) - f(x-1)$. Para encontrar el filtro deseado se efectúan las siguientes operaciones:

Se tiene que:

$$g(x) = h(x) * f(x) \tag{5}$$

$$g(x) = f(x) - f(x-1) \tag{6}$$

Entonces, aplicando transformada de Fourier a las ecuaciones (5) y (6):

$$G(\omega) = H(\omega)F(\omega) \tag{7}$$

$$G(\omega) = F(\omega) - e^{-j\omega}F(\omega) \tag{8}$$

Igualando las ecuaciones (7) y (8) se tiene que:

$$H(\omega)F(\omega) = F(\omega)(1 - e^{-j\omega}) \tag{9}$$

$$H(\omega) = 1 - e^{-j\omega} \tag{10}$$

Entonces, al aplicar transformada inversa de Fourier, el filtro *h(x)* tiene la siguiente expresión:

$$h(x) = \delta(x) - \delta(x-1) \tag{11}$$

La figura 1.7 es la representación de dicho filtro en el plano:

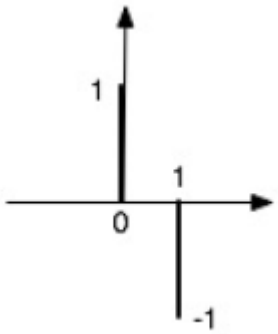


Figura 1.7. Filtro $\delta(x)-\delta(x-1)$ [7]

Enseguida, el filtro obtenido se lleva a la forma matricial, ya que sólo interesan sus coeficientes:

$$h(x)=\begin{bmatrix}1 & -1\end{bmatrix} \tag{12}$$

Por otro lado, el filtro h(x) podría reescribirse como $h(x)=\delta(x+1)-\delta(x-1)$, cuya representación en el plano se muestra en la figura 1.8:

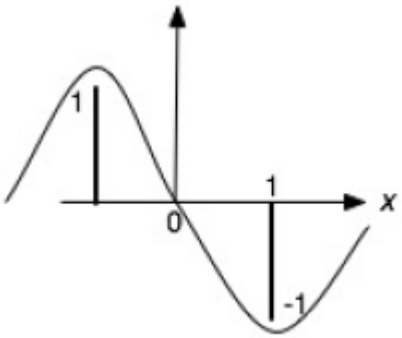


Figura 1.8. Filtro $\delta(x+1)-\delta(x-1)$ [7]

Llevando este filtro a su forma matricial se obtendría lo siguiente:

$$h(x)=\begin{bmatrix}1 & 0 & -1\end{bmatrix} \tag{13}$$

De esta manera, el filtro h(x) ya posee un comportamiento que se ajusta a la primera derivada de Gaussiana en el plano continuo y, además, es un modelo discreto, por lo tanto, ahora es posible detectar cambios en la dirección de la variable *x*. Los filtros detectores de bordes basados en derivadas de Gaussianas tienen su aplicación *en tareas previas a la segmentación* como detectores de bordes.

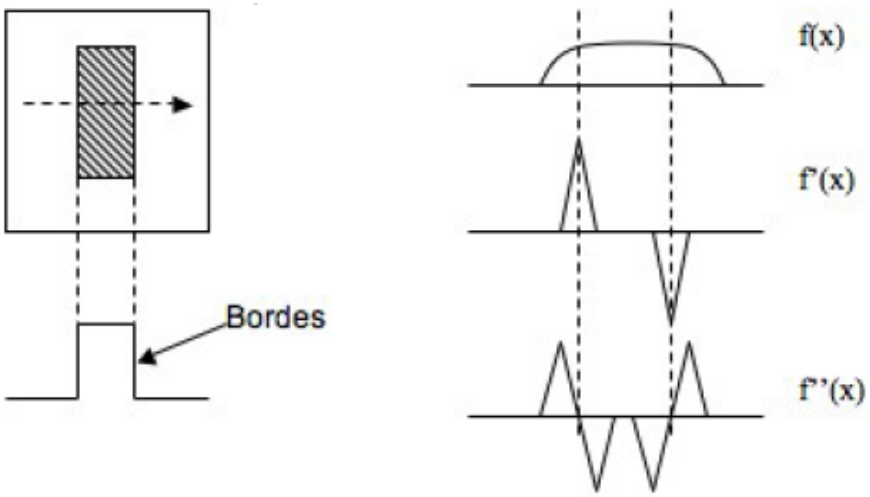


Figura 1.9. La detección de bordes [7]

En el tratamiento de imágenes conviene emplear la siguiente máscara para la detección de bordes en la dirección horizontal:

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

El filtro espacial anterior es conocido comúnmente como filtro Prewitt. Una variación de este filtro es el que se muestra a continuación conocido como filtro Sobel:

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

El filtro Sobel también detecta bordes en la dirección horizontal y el hecho de que tenga una ponderación de dos unidades en el centro radica en que se busca tener un cierto grado de suavizado para lograr reducción de ruido. En sus inicios estos filtros fueron obtenidos heurísticamente, sin embargo, después se verificó que su funcionamiento correspondía a la aproximación de una Binomial de primer orden para una primera derivada de Gaussiana. Para obtener los filtros Sobel y Prewitt en la dirección vertical simplemente se transponen los filtros de detección horizontal mostrados anteriormente:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Es posible aproximar las derivadas de Gaussianas empleando la función Binomial. La forma general de la función Binomial para cualquier orden *r* de diferencia de Gaussiana y para cualquier longitud de filtro *N* es la siguiente:

$$f^{r}_{N}(x) = \Delta^{r}\binom{N-r}{x} \tag{14}$$

donde:

$$x = 0...N \tag{15}$$

$$r \leq N \tag{16}$$

$$\Delta[f(x)] = f(x) - f(x-1) \tag{17}$$

Siendo *N* un número entero positivo, *x* sólo puede tomar valores enteros positivos. De tal forma que si, por ejemplo, se desea obtener el filtro Binomial de dimensión *N=2* con derivada *r=0*, se tiene lo siguiente:

$$f^{0}_{2}(x) = \Delta^{0}\binom{2}{x} = \frac{2!}{x!(2-x)!} \tag{18}$$

Entonces:

$$f_2^0(0) = \binom{2}{0} = 1 \qquad f_2^0(1) = \binom{2}{1} = 2 \qquad f_2^0(2) = \binom{2}{2} = 1 \tag{19}$$

Por lo tanto:

$$f_2^0(x) = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \tag{20}$$

Por otro lado, si se desea encontrar el filtro Binomial de orden 2 de primera derivada (r=1), se tiene lo siguiente:

$$f_2^1(x) = \Delta^1 \binom{1}{x} = \binom{1}{x} - \binom{1}{x-1} \tag{21}$$

Entonces:

$$f_2^1(0) = \binom{1}{0} - \binom{1}{-1} = 1 \qquad f_2^1(1) = \binom{1}{1} - \binom{1}{0} = 0$$

$$f_2^1(2) = \binom{1}{2} - \binom{1}{1} = -1 \tag{22}$$

Por ende, se tiene que:

$$h(x) = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \tag{23}$$

De esta manera, los filtros Sobel y Prewitt pueden obtenerse por medio de los filtros Binomiales; por ejemplo, para obtener el filtro Sobel, que detecta bordes en la dirección horizontal, se multiplica la aproximación de la Gaussiana por la aproximación de su primera derivada:

$$h_X(x,y) = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{24}$$

Cabe señalar que las primeras derivadas de Gaussianas se utilizan para la detección de bordes mientras que las segundas derivadas de Gaussianas indican si un píxel determinado perteneciente a un borde está dentro del nivel de gris oscuro o claro, a lo cual se le denomina *cruces por cero*.

A continuación, en las figuras 1.11 y 1.12, se muestran los resultados que se obtienen al aplicar los filtros Sobel y Prewitt sobre una imagen (figura 1.10):

Figura 1.10 Imagen Original [9]

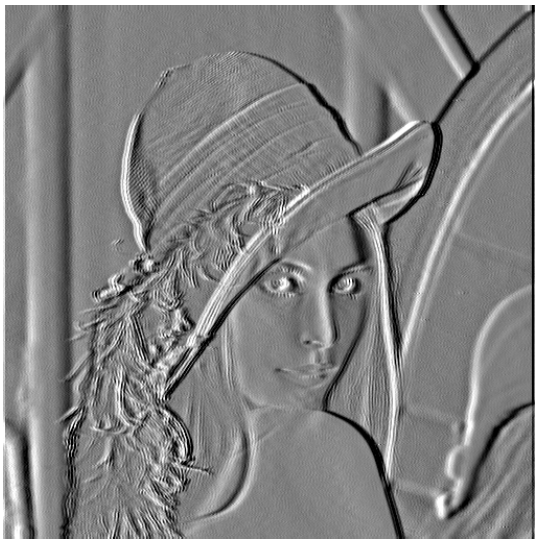


Figura 1.11. Resultado filtro Sobel horizontal

Figura 1.12. Resultado filtro Prewitt horizontal

1.3. La transformada de Hermite.

La transformada de Hermite, un caso particular de la transformada polinomial, no es en sí un preliminar para este trabajo, sin embargo la aplicación se ayuda de esta transformada para la detección de contornos y estructuras, lo cual en ocasiones es muy útil, principalmente en las imágenes médicas. El hecho de contrastar los contornos con las regiones homogéneas de una imagen ayuda a tener una visión más clara de los objetos inmersos en la misma.

Una de las metas que tiene la aplicación involucrada con el presente escrito es la segmentación de volúmenes y todo lo que esto implica, pero, en ocasiones basta aplicar técnicas derivativas para aislar elementos en las imágenes o volúmenes; González por ejemplo, ubica el uso del gradiente como una técnica de segmentación [1]. Puesto que se trata de una herramienta muy útil y biológicamente congruente dado que se basa en derivadas de la función Gaussiana [10], la transformada de Hermite fue incluida en la aplicación.

Las imágenes pueden considerarse como arreglos rectangulares compuestos de valores de intensidades; en la mayoría de las aplicaciones relacionadas con el ámbito visual estas intensidades tienen que ser interpretadas por medio de patrones visuales significativos, para lo cual es necesario determinar relaciones espaciotemporales entre dichas intensidades.

Para lograr lo anterior, debe involucrarse en el procesamiento un análisis local, es decir, la señal original se multiplica por una ventana de cierto tamaño y forma para determinar el número de puntos que contribuyen en la operación y establecer los pesos relativos para fijar la contribución de cada punto respectivamente. Para obtener una descripción completa de la imagen, el procesamiento local se repite un número suficiente de posiciones de la ventana. Pero, para conseguir esto deben elegirse en primera instancia la forma y tamaño de la ventana, además de la elección del operador básico que se empleará para cada posición de la ventana [11]. Para realizar una óptima elección de estos parámetros suele considerarse como referencia el sistema de visión humano puesto que se busca encontrar una representación de imágenes apta y apropiada para su exhibición a observadores humanos, pues datos biológicos sugieren que la mejor manera de obtener esta representación es usando funciones base que provengan de una clase especial de formas funcionales que aproximen a aquellas usadas por las neuronas visuales de los primates [12].

En sus inicios, las investigaciones sobre el sistema de visión humano se concentraban en las funciones de Gabor para la descripción de los procesos de la retina y la corteza cerebral, basándose en el modelado de los perfiles de los campos receptivos por medio de estas funciones. Posteriormente se determinó que los perfiles de los campos receptivos pueden modelarse de manera más eficiente y con mayor facilidad con las derivadas de Gaussianas [10].

La transformación polinomial es una técnica de descomposición de señales en la cual las señales son aproximadas localmente por medio de polinomios [11]. En resumen, la transformación polinomial consiste en convolucionar una señal original $L(x)$ con filtros $D_n(x)$ para posteriormente submuestrear el resultado de la convolución y obtener los

coeficientes polinomiales $L_n(x)$. La reconstrucción de la señal se logra sobremuestreando los coeficientes para convolucionarlos con filtros $P_n(x)$ y los resultados de estas convoluciones se suman para obtener la señal original. Esto se ilustra en la figura 1.13:

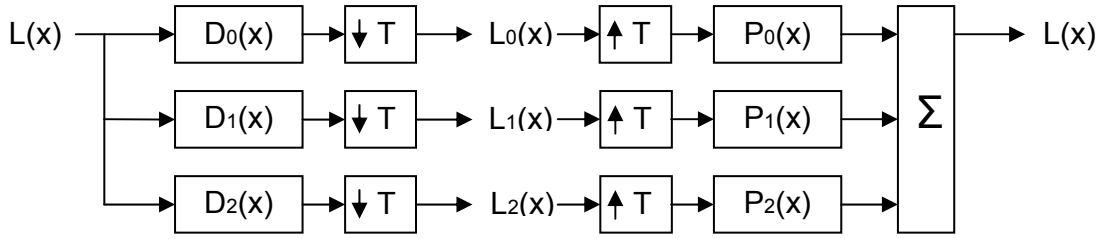


Figura 1.13. La Transformada Polinomial

Como ya se mencionó, la transformada de Hermite es un caso particular de la transformada polinomial. La característica esencial radica en que la función ventana que se utiliza es una Gaussiana en donde el factor de normalización es tal que $V^2(x)$ tiene energía unitaria:

$$V(x) = \frac{1}{\sqrt{\sqrt{\pi\sigma}}} \exp(-x^2 / 2\sigma^2) \tag{25}$$

Los polinomios ortonormales que se asocian con $V^2(x)$ se denominan polinomios de Hermite y a la técnica de descomposición local que utiliza estos polinomios se le conoce como la transformada de Hermite [11].

Sean $G_n(x)$ los polinomios ortonormales referidos en el párrafo anterior. Los coeficientes $L_n(x)$ pueden derivarse de una señal original L(x) convolucionando dicha señal con las funciones de los filtros $D_n(x)$ [11] determinados por:

$$Dn(x) = Gn(-x)V^2(-x) \tag{26}$$

Los filtros de la ecuación (26) pueden muestrearse en múltiplos de un factor de muestreo T para cubrir a toda la señal L(x). Las funciones de los filtros determinan qué información será explícita en los coeficientes de la transformada de Hermite. Las propiedades principales de la transformada de Hermite son determinadas por las funciones que describen estos filtros [11]. De la expresión general de la ecuación (26) se derivan los filtros usados en la transformada de Hermite:

$$Dn(x) = \frac{(-1)^n}{\sqrt{2^n n!}} \frac{1}{\sigma\sqrt{\pi}} Hn\left(\frac{x}{\sigma}\right) \exp(-x^2/\sigma^2) \tag{27}$$

En la ecuación anterior, la función $H_n(x/\sigma)$ representa a los polinomios de Hermite. Puede demostrarse que las funciones de los filtros $D_n(x)$ son iguales a la derivada de Gaussiana de orden *n* dada la propiedad que muestran los polinomios de Hermite descrita en la ecuación (28) [13], con lo que se obtiene la equivalencia mostrada en la ecuación (29).

$$\exp(-x^2) Hn(x) = (-1)^n \frac{d^n}{dx^n} \exp(-x^2) \tag{28}$$

$$Dn(x) = \frac{1}{\sqrt{2^n n!}} \frac{d^n}{d\left(\frac{x}{\sigma}\right)^n} \left[\frac{1}{\sigma\sqrt{\pi}} \exp(-x^2/\sigma^2)\right] \tag{29}$$

Entonces, la transformada de Hermite involucra el uso de filtros que corresponden a derivadas de la función Gaussiana que, como se ha mencionado anteriormente, son útiles en el modelado de los campos receptivos del sistema de visión humano. Las gráficas de las funciones de los filtros D_0 a D_4 en el dominio espacial con $\sigma=1$ se muestran en la figura 1.14.

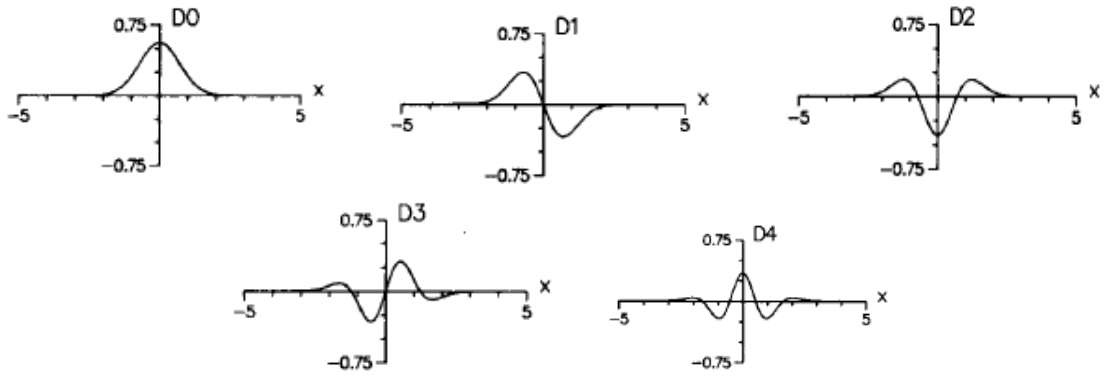


Figura 1.14. Funciones de los filtros en el dominio espacial con σ=1 [11]

Para el caso discreto, como se constató en la sección anterior, la función Binomial es la correspondiente a la función Gaussiana; por este motivo, el análisis gira en torno a la ventana Binomial:

$$V^2(x) = \frac{1}{2^M} C_M^X \tag{30}$$

Para la ecuación (30) x adopta valores desde 0 hasta M [11], siendo C_M^X otra notación para las combinaciones referidas en la ecuación (14). Los polinomios discretos ortonormales asociados con la ventana Binomial son conocidos como los polinomios de Krawtchouk [11], los cuales convergen a los polinomios de Hermite cuando el número de puntos discretos tiende al infinito [12], por lo tanto, el modelo discreto está completo. Los polinomios de Krawtchouk tienen la siguiente forma:

$$Gn(x) = \frac{1}{\sqrt{C_M^X}} \sum_{k=0}^{n} (-1)^{n-k} C_{M-X}^{n-k} C_X^k \tag{31}$$

Concentrándose en el caso de que M sea par, el filtro puede ser centrado en el origen al desplazar la función Binomial sobre M/2 [11], lo cual conlleva a la siguiente definición para las funciones de los filtros de la transformada de Hermite discreta:

$$Dn(x) = Gn\left(\frac{M}{2} - x\right)V^2\left(\frac{M}{2} - x\right) \tag{32}$$

Cabe recalcar que para la aplicación interesa solamente la descomposición en coeficientes de la señal, ya que lo que se busca es observar dichos coeficientes para extraer características de los volúmenes. El proceso de descomposición consiste en el mapeo de los datos originales por medio de la transformada discreta de Hermite a un conjunto de coeficientes espectrales.

La transformada de Hermite, en su forma discreta, se basa en filtros Binomiales que tienen la forma descrita en la ecuación (33) para n, x = 0,..., N, siendo n el orden de la derivación, N+1 la longitud del filtro y Δ^n el operador de diferencia de orden n; nótese la inmediata relación de la ecuación (33) con la ecuación (14).

$$B_{n,x} = (-1)^n 2^{-N} \sqrt{C_N^n}\, \Delta^n \{C_{N-n}^x\} \tag{33}$$

También, por otra parte, pueden definirse las funciones Binomiales de igual manera para n, x = 0,…, N:

$$A_{n,x} = \sqrt{2^{-N}}\, \Delta^n \{C_{N-n}^x\} \tag{34}$$

Las funciones Binomiales están relacionadas con los filtros Binomiales mediante la siguiente ecuación:

$$B_{n,x} = W_x A_{n,x} \tag{35}$$

Donde:

$$W_x = \sqrt{2^{-N} C_N^x} \tag{36}$$

Llevando esto al espacio de las matrices, la transformación definida por la operación **B = WA** da como resultado la salida de los filtros Binomiales en una posición genérica, siendo $\mathbf{W = diag(W_X)}$ para x = 0,…,N, una matriz diagonal que representa la ventana Binomial. Si **F** es una matriz cuadrada de (N+1)x(N+1) que representa las intensidades de una imagen digital, entonces, genéricamente, $\mathbf{G = BFB^T}$ son los coeficientes de la transformada de Hermite discreta [14]. Desde luego, esto puede ampliarse para conjuntos de datos de tres dimensiones.

Recuérdese que la ecuación (33), que define los filtros binomiales utilizados por la transformada de Hermite, involucra el uso de derivadas de la función Gaussiana de orden *n*; si dicho orden es cero, la transformada hará un suavizado de la imagen ya que dicho filtro de orden cero tendría un comportamiento paso bajas descrito en [7].

2. SEGMENTACIÓN DE IMÁGENES

Para comprender a fondo el funcionamiento de la aplicación descrita en esta tesis es necesario tener presentes los fundamentos a partir de los cuales se ha realizado todo el desarrollo de este trabajo. Para ello, es necesario mencionar, en primer lugar, algunos conceptos teóricos básicos de la segmentación de imágenes, y después, algunos métodos para llevarla a cabo.

2.1. Segmentación de imágenes

La segmentación de imágenes se refiere a la partición de una imagen en regiones no solapadas, las cuales son homogéneas con respecto a alguna característica, por ejemplo la intensidad o la textura. Es decir que, idealmente, un método de segmentación encuentra aquellos conjuntos que corresponden a distintas estructuras o regiones de interés en la imagen, por lo que regiones adyacentes deberían tener características diferentes y claras.

En determinados problemas de análisis de imágenes la etapa de segmentación resulta ser muy importante. Sin embargo, existen situaciones en las que no es factible o, simplemente, no es posible llevarla a cabo manualmente y es necesario implementar alguna solución que permita realizarla de forma automática, es decir, con ayuda de un sistema computarizado.

Además, algunas “buenas” soluciones para problemas de reconocimiento de patrones corresponden a “buenas” soluciones al problema de optimización matemática del modelo elegido para representar el proceso físico. Matemáticamente, los modelos representan una idea bien definida (y muy rígida) y a menudo son completamente diferentes de lo que piensan los observadores humanos [15]. Debido a esto, se deben definir las características deseables de la segmentación de acuerdo a cada caso.

Además, un punto que se vuelve un requerimiento importante es el que tiene que ver con las expectativas sobre los resultados, pues se espera que éstos deban ser lo más parecidos a los que el ser humano obtiene a través de su sistema de visión. Y esto se debe a que el procesamiento de imágenes se ha inspirado en gran medida en el modelo de visión humano, especialmente en la etapa más temprana, la cual es la responsable de detectar estructuras como puntos, líneas, bordes y texturas con el fin de facilitar interpretaciones subsecuentes de estas estructuras en etapas posteriores del sistema de visión [16].

Las aplicaciones de este proceso son muy variadas, pero un campo en el que se emplea típicamente es la Medicina. En esta área el reto es obtener información acerca de estructuras anatómicas del cuerpo humano, a partir de imágenes de tomografía computarizada (CT), resonancia magnética (MR), tomografía por emisión de positrones (PET), etc., con el fin de ayudar en los diagnósticos y tratamientos de enfermedades.

2.1.1. Definición formal de segmentación de imágenes

Antes de definir segmentación se debe tener presente lo que significa *región conexa*:

Se dice que una región R es conexa si dados 2 pixeles cualquiera (x_A, y_A), (x_B, y_B) *de R pueden ser conectados por la ruta* (x_A, y_A), … (x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1}), …, (x_B, y_B), *en donde cada punto* (x_i, y_i) *debe pertenecer a R y ser adyacente al punto anterior y siguiente de la ruta.*

Tomando en cuenta lo anterior, el concepto de *segmentación* se puede definir formalmente [15]:

La segmentación de imágenes es el proceso de particionar una imagen $\boldsymbol{P_{IJ}}$ *en c subregiones conexas tales que cada región* R_i *es homogénea con respecto a algún predicado, esto es,*

$$\bigcup_{i=1}^{c} R_i = P_{IJ} \tag{1}$$

$$R_i \cap R_j = \varnothing \quad \forall\, i, j, \quad i \neq j$$

$$R_i,\ i = 1, \ldots, c \ \text{son conexas}$$

$$P(R_i) = \text{Verdadero} \quad \forall i$$

$$P(R_i \cup R_j) = \text{Falso} \quad \text{si } i \neq j \text{ y } R_i \text{ es adyacente a } R_j$$

donde *IJ* denota el dominio bidimensional de la imagen.

Estrictamente, al eliminar la restricción de que las regiones sean conexas, a la acción de determinar los conjuntos R se le llama clasificación del pixel y a los conjuntos se les llama *clases* [17].

Sin embargo, en la literatura se puede encontrar, acerca de este tema, que ésta no es la principal o única diferencia (como se verá en la siguiente sección). Por lo anterior, en este trabajo se continuará utilizando el concepto de segmentación para referirse al proceso de obtención de las regiones; a los grupos resultantes se les llamará clusters o clases.

2.1.2. Técnicas clásicas de segmentación de imágenes.

A continuación se mencionan algunas clasificaciones hechas por algunos autores con el propósito de mostrar algunos enfoques y criterios, y desde luego, la variedad de metodologías propuestas para tratar este tema.

Liu *et. al*. [18] clasifican los algoritmos de segmentación en tres categorías: agrupamiento, detección de bordes, extracción de regiones. Báez J. J. [19] *et. al.* dividen los métodos en cuatro grandes grupos dependiendo de las características de la imagen que usan: valores de píxel, área, orillas, basadas en la física. La clasificación de Ho y Lee [20] es la siguiente: basada en bordes, agrupamiento, por regiones y división y fusión.

González [1] menciona que los algoritmos de segmentación se basan en detectar dos propiedades: discontinuidades y semejanzas de los valores de niveles de gris de los pixeles.

La siguiente clasificación, la cual fue propuesta por Pham, Xu y Prince [17], se cita con un poco más de detalle debido a que describe estrategias específicas enfocadas al área médica. Los autores dividen los métodos de segmentación en 8 categorías:

- **Umbralización** (*Thresholding*). Los métodos que implementan esta estrategia crean una partición binaria de las intensidades de la imagen. Un procedimiento de umbralización intenta determinar un valor de intensidad llamado umbral, el cual separa las clases.

- **Crecimiento de regiones** (Region growing). Los algoritmos que usan esta técnica extraen regiones conexas de la imagen según un criterio predefinido.

- **Clasificadores**. Estos métodos son técnicas de reconocimiento de patrones que buscan particionar un espacio de características derivado de la imagen usando datos con etiquetas conocidas. Estos métodos son supervisados.

- **Agrupamiento** (*Clustering*). Básicamente, estos métodos funcionan del mismo modo que los clasificadores, pero sin usar datos de entrenamiento, por lo cual son métodos no supervisados. Estos métodos se describen con mayor detalle en la sección siguiente.

- **Campos aleatorios de Markov** (MRF - *Markov Random Fields*). Estrictamente los algoritmos que emplean esta idea no son métodos de segmentación pero es un modelo estadístico que puede ser usado. Los MRF modelan relaciones espaciales entre vecinos o pixeles cercanos y estas correlaciones locales proporcionan un mecanismo para modelar una gran variedad de propiedades de la imagen.

- **Redes de neuronas artificiales** (ANN - *Artificial Neural Networks*). Son redes paralelas de elementos de procesamiento o nodos que simulan el aprendizaje biológico. Su mayor uso en aplicaciones médicas es como un clasificador, donde los pesos son determinados empleando datos de entrenamiento y después se emplea para segmentar nuevos datos.

- **Modelos deformables**. Se emplean para delinear límites de regiones usando curvas o superficies paramétricas cerradas que se deforman bajo la influencia de fuerzas internas y externas.

- **Guiados por plantillas**. Estos métodos son una poderosa herramienta cuando un mapa o plantilla está disponible. El mapa se genera por medio de información compilada de la anatomía que requiere segmentación. Después, este mapa se usa como un marco de referencia para segmentar nuevas imágenes.

Otros métodos que mencionan los autores de esta clasificación son ajuste al modelo (*model-fitting*) y el algoritmo *watershed*, el cual tiene fundamentos matemáticos y morfológicos.

2.2. Valoración y elección del algoritmo de segmentación

Para seleccionar el algoritmo se han considerado varios aspectos referentes a las características de las imágenes y también los posibles escenarios en los que funcione esta aplicación.

En primer lugar, es importante resaltar las ventajas y desventajas de las técnicas mencionadas en la sección anterior para después elegir una que se adecue a los requerimientos de la aplicación que se desarrolla en esta tesis.

La umbralización es muy útil para segmentar imágenes en las que los objetos tienen intensidades de gris, u otra característica, muy distintas. Entre las desventajas más evidentes es que, por lo general, es un proceso interactivo, es decir, requiere intervención por parte del usuario; otras limitantes de la técnica original son que no puede ser aplicado a imágenes multicanal y no toma en cuenta las características espaciales de la imagen, lo cual lo hace sensible a la presencia de ruido.

En el caso de la técnica *Region Growing*, la ventaja es que, una vez que se tiene una semilla, el algoritmo obtiene todos los píxeles conectados a ella según un criterio establecido. Las desventajas de esta opción es que la semilla debe ser designada por el usuario. Otra restricción de esta técnica es que, al igual que la umbralización, es sensible al ruido.

Respecto a los clasificadores, una característica importante es que no son iterativos, lo cual los hace eficientes desde el punto de vista computacional; un atributo más es que puede ser aplicado a imágenes multicanal. Entre los inconvenientes que tiene, uno de los más significativos es que requiere de datos de entrenamiento, lo que, en cierta medida, representa una participación importante por parte del usuario. Al igual que las técnicas anteriores, no consideran información espacial.

Los métodos basados en agrupamiento, a diferencia de los clasificadores, no requieren datos de entrenamiento sino iteraciones, lo cual puede representar un aumento en el procesamiento computacional. Un inconveniente importante es que los modelos originales no consideran las características espaciales, lo cual los hace susceptibles a ruido.

El modelado por MRF ofrece como ventaja la robustez frente al ruido puesto que se encarga de modelar las interacciones espaciales entre píxeles vecinos. Los MRF son complementados generalmente con un algoritmo de agrupamiento como K medias. Sin embargo, los MRF requieren una apropiada elección inicial para los parámetros que controlarán la fuerza de las interacciones espaciales y, además, usualmente requieren algoritmos computacionalmente intensos.

Como ya se mencionó anteriormente las redes neuronales artificiales son empleadas en la segmentación de imágenes como clasificadores presentando similares ventajas y desventajas con éstos, sin embargo dadas las múltiples conexiones que se presentan en una red neuronal, la información espacial puede ser fácilmente incorporada. Pero, por otro lado, el procesamiento de las redes neuronales es inherentemente paralelo, su procesamiento es usualmente simulado en una computadora estándar reduciendo así el potencial computacional que es su principal ventaja.

De los párrafos anteriores se puede establecer que el método a elegir debe: ser robusto frente al ruido, prestar atención a la información espacial, reducir lo más posible la interacción con el usuario (o bien, hacer más sencilla esta interacción). Y dada la dificultad de obtener datos de entrenamiento confiables para la segmentación de imágenes médicas, es preferible optar por un método que no requiera dicho entrenamiento y que de preferencia no presente algoritmos muy intensos con miras a reducir los tiempos de programación y la puesta a punto de la aplicación.

Es por ello que se optó por un método de agrupamiento (*clustering*) que fue modificado para tomar en cuenta la información espacial y así, de esta forma, hacer frente al ruido.

2.3. Agrupamiento o clustering de datos [15]

La estrategia elegida para cumplir el objetivo de esta tesis es el agrupamiento o clustering. Por esto, se presentan los conceptos básicos de esta herramienta empleada en el reconocimiento de patrones. Además, también se presenta el diagrama que ilustra el proceso de análisis de problemas desde la perspectiva del *clustering*.

2.3.1. Análisis empleando agrupamientos (cluster)

El análisis empleando clusters involucra tres problemas: evaluación, agrupamiento y validación (ver figura 2.1).

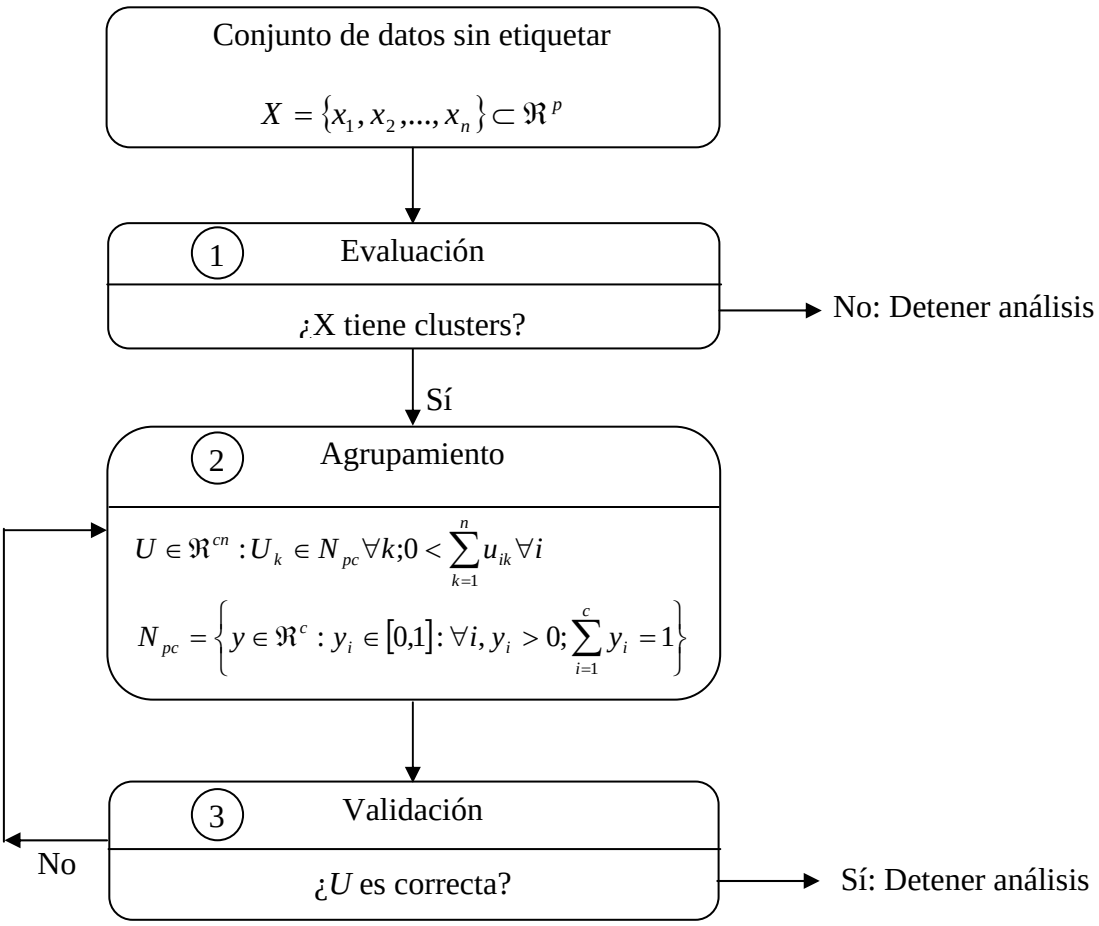


Figura 2.1. Análisis empleando agrupamientos

De acuerdo al diagrama anterior, a partir de un conjunto de datos no etiquetados se determina si existen estructuras a detectar, lo cual puede volverse un problema considerable. Una vez que se ha decidido formar clusters, se necesita seleccionar un modelo cuya medida de similitud matemática pueda percibir una estructura en la forma en que un humano la percibe. Seleccionar un criterio de similitud es un problema fundamental

de todos los modelos de agrupamiento por la cuestión previamente mencionada referente a las expectativas sobre los resultados.

El problema 2 del diagrama es el agrupamiento o *clustering* (aprendizaje no supervisado) del conjunto de datos no etiquetados $X = \{x_1, x_2, ..., x_n\}$, el cual es la asignación de vectores de etiquetas a los elementos $\{x_k\}$.

También es importante decir que diferentes algoritmos producen diferentes particiones de los datos y no es fácil determinar cuál puede ser el más útil y debido a esto resulta de mucha importancia validar los clusters mediante algún criterio o regla.

Como resultado del proceso de clustering se obtienen vectores de etiquetas.

2.3.2. Modelos C-Means (C-Medias).

Las familias *c-means* son las más conocidas y mejor desarrolladas familias de modelos de clustering. Esto se debe a que son modelos de mínimos cuadrados.

La expresión para representar el problema de optimización que define este tipo de modelos es:

$$\underbrace{\min}_{(U,V)}\left\{ J_m(U,V) = \sum_{i=1}^{c}\sum_{k=1}^{n} u_{ik}^m D_{ik}^2 \right\}, \text{ donde} \qquad (2)$$

$U \in$ M_{hcn} o M_{fcn} para el modelo *hard c-means* (HCM) o *fuzzy c-means* (FCM) respectivamente;

$V = (v_1, v_2, \ldots, v_c) \in \Re^{cp}$; $v_i \in \Re^{p}$ es el i-ésimo punto prototipo;

$m \geq 1$ es el grado de fuzzificación

y $D_{ik} = \|x_k - v_i\|_A$ es la medida de similitud.

En la ecuación (2) se obtiene un mínimo local para J_m, el cual resultará en la obtención de buenos clusters U y en la de buenos prototipos (centroides) V para representar a los clusters.

Profundizando sobre la definición de partición, se dice que una partición c de X es una matriz de tamaño $c \times n$ definida como $U = [U_1\ U_k\ ...\ U_n] = [u_{ik}]$, donde U_k denota la k-ésima columna de U. La razón de que las matrices se llamen particiones se debe a la interpretación de sus entradas. Si U es de tipo tradicional o difusa, u_{ik} se toma como la membresía de x_k en el i-ésimo cluster de X.

Las expresiones para calcular U y V en el caso de HCM (*Hard C-Means*) son:

$$u_{ik} = \begin{Bmatrix} 1; & D_{ik} \le D_{ij}, j \ne i \\ 0; & otro\ caso \end{Bmatrix} \quad \forall i,k \tag{3}$$

$$v_i = \frac{\sum_{k=1}^{n} u_{ik} x_k}{\sum_{k=1}^{n} u_{ik}} = \frac{\sum_{x_k \in x_i} x_k}{n_i} = \bar{v}_i \quad \forall i \tag{4}$$

Mientras que para el modelo FCM (*Fuzzy C-Means*) son:

$$u_{ik} = \left[\sum_{j=1}^{c} \left(\frac{D_{ik}}{D_{ij}} \right)^{\frac{2}{m-1}} \right]^{-1} \quad \forall i,k \tag{5}$$

$$v_i = \left(\frac{\sum_{k=1}^{n} u_{ik}^m x_k}{\sum_{k=1}^{n} u_{ik}^m} \right) \quad \forall i \tag{6}$$

La medida de similitud más común es la distancia euclidiana (8), sin embargo otras medidas empleadas son la *Manhattan* (*city block*) (7) y la distancia *Chebychev* (9).

Distancia *Manhattan* $$\|x - v\| = \left(\sum_{j=1}^{p} |x_j - v_j| \right) \quad (7)$$

Distancia euclidiana $$\|x - v\| = \left(\sum_{j=1}^{p} |x_j - v_j|^2 \right)^{\frac{1}{2}} \quad (8)$$

Distancia *Chebychev* $$\|x - v\| = \underbrace{\max}_{1 \le j \le p} \{ |x_j - v_j| \} \quad (9)$$

Finalmente, el algoritmo resultante que hace posible la implementación computacional de los modelos *c-means* se muestra a continuación:

1. Almacenar los datos no etiquetados $X \subset \Re^p$
2. Definir
   - el número de clusters: $1 < c < n$
   - el número máximo de iteraciones: T
   - el valor del exponente de peso: $1 \le m$
   - la medida de similitud $D_{ik} = \|x_k - v_i\|_A$
   - la medida de término: $E_t = \|V_t - V_{t-1}\|$
   - el umbral de término: $0 < \varepsilon$
3. Inicializar los valores de los prototipos (centroides): $V_0 = \{v_{1,0}, ..., v_{c,0}\} \in \Re^{cp}$
4. Iterar

   $t \leftarrow 0$

   Repetir

   $t \leftarrow t + 1$

   Calcular U_t

   Calcular V_t

   Hasta ($t = T$ o $E_t \le \varepsilon$)

   $(U, V) \leftarrow (U_t, V_t)$

2.4. Segmentación de imágenes por medio de técnicas de agrupamiento difuso

En general, la intensidad del píxel no está directamente relacionada con los grados de membresía del píxel a las regiones u objetos. Por lo tanto, se necesita generar funciones de membresía para las regiones en la imagen. Si se extraen *p* características del píxel X_i, entonces X_i puede ser representado por un vector de características x_i en $\Re^p$. Los componentes de x_i pueden ser intensidades (en el caso de imágenes multiespectrales), o funciones de las intensidades, o ambas. Por otra parte, si se seleccionan las características adecuadas, entonces los vectores de características correspondientes a cada región significativa pueden formar agrupamientos (o clusters) en $\Re^p$. Probablemente, los datos del objeto $X = \{x_1, x_2, ..., x_3\}$ correspondientes a los *n* pixeles de la imagen pueden ser agrupados dentro de un número especificado de *c* clusters usando un algoritmo de agrupamiento conveniente.

En ocasiones no es fácil determinar si un píxel debe ser incluido en una región o no debido a que las características usadas para determinar si existe homogeneidad pueden no tener claras transiciones en las regiones de los límites. Y esto es más común cuando las características se procesan usando ventanas locales. Para aligerar esta situación se pueden usar conceptos de conjuntos difusos al proceso de segmentación. Prewitt sugirió que los resultados de la segmentación deberían ser conjuntos difusos más que conjuntos absolutos (o no difusos). En la segmentación no difusa, la función de membresía tradicional $m_{R_i} : IJ \to \{0,1\}$ de una región R_i es

$$m_{R_i}(x,y) = \begin{Bmatrix} 1 & ; (x,y) \in R_i \\ 0 & ; (x,y) \notin R_i \end{Bmatrix} \qquad (10)$$

En una segmentación difusa, a cada píxel se asigna un valor de membresía a cada una de las regiones. El resultado de una segmentación difusa es una partición de P_{IJ} en c subconjuntos difusos $\{R_i\}$. Cada conjunto R_i está representado por su función de membresía $m_{R_i} : IJ \to [0,1]$, la cual remplaza la función de membresía de una partición no

difusa. Para $(x, y) \in P_{IJ}$, $m_{R_i}(x, y)$ representa el grado en el que (x, y) pertenece a R_i . Una segmentación difusa de una imagen en *c* regiones es una matriz de particiones difusas de tamaño $c\,x\,n$

$$U = \lfloor u_{ij} \rfloor \tag{11}$$

donde $u_{ij} = m_{R_i}(x, y)$, es decir u_{ij} es la membresía del j-ésimo píxel en la i-ésima región R_i.

En términos de generar funciones de membresía para procesos posteriores, los algoritmos de clustering basados en modelos *c-means* tienen muchas ventajas. Son no supervisados; pueden ser usados con cualquier número de características y clases; y distribuyen los valores de membresía entre clases basándose en un agrupamiento "natural" en el espacio de las características.

Los algoritmos de agrupamiento funcionan del mismo modo que los clasificadores, pero con la diferencia de que no usan datos de entrenamiento. Debido a lo anterior, para compensar la falta de tales datos, los métodos de agrupamiento iteran entre segmentar la imagen y caracterizar las propiedades de cada clase. En este sentido, los métodos de agrupamiento se entrenan a si mismos usando los datos disponibles [17].

2.5. Inclusión de información espacial a modelo *c-means* difuso (FCME) [21]

Hasta ahora se han descrito los modelos *c-means*, los cuales hacen uso de la información del espacio de las características; sin embargo, no emplean la información espacial.

En el método estándar, la función objetivo se minimiza cuando a los pixeles cercanos al centroide de su cluster se les asignan valores grandes de membresía, mientras que a los pixeles que están más lejos se les asignan valores pequeños.

Los pixeles de una imagen están altamente correlacionados, esto significa que los pixeles vecinos tienen valores de características similares, y es muy probable que pertenezcan al mismo cluster. Por lo tanto, la relación espacial que existe entre los pixeles de la vecindad puede ser de gran ayuda en el proceso de segmentación.

El método que aquí se describe es propuesto por Chuang [21] et. al. y propone que el peso de las membresías de cada cluster sea alterado después de considerar la distribución de clusters en la vecindad.

Para aprovechar la información espacial, se define la función espacial

$$h_{ij} = \sum_{k \in NB(x_j)} u_{ik} \tag{12}$$

donde $NB(x_j)$ representa una ventana cuadrada centrada en el píxel x_j en el dominio espacial. Del mismo modo que la función de membresía, la función espacial h_{ij} representa la probabilidad de que el píxel x_j pertenezca al i-ésimo cluster. El valor de la función espacial de un píxel en un cluster es mayor si la mayoría de sus vecinos pertenecen al mismo cluster.

A continuación se muestra como se incorpora la función espacial a la función de membresía

$$u'_{ij} = \frac{u_{ij}^{p} h_{ij}^{q}}{\sum_{k=1}^{c} u_{kj}^{p} h_{kj}^{q}} \tag{13}$$

donde p y q son parámetros para controlar la importancia o peso relativo de ambas funciones dentro de la toda la función.

En una región homogénea, la función espacial simplemente fortalece la membresía original. Sin embargo, para los pixeles de una imagen con ruido, se reduce la posibilidad de que sean mal clasificados.

Como resultado, el algoritmo consta de dos fases en cada iteración:

- La primera fase realiza las mismas operaciones que el algoritmo estándar de FCM con el objetivo de calcular el valor de la función de membresía en el dominio de las características.

- En la segunda fase, la información de la función de membresía de cada píxel es mapeada al dominio espacial y a partir de ella se calcula el valor de la función espacial.

Al finalizar la segunda fase, se calcula el valor de la nueva función de membresía usando el valor de la función de membresía de la primera fase y el valor de la función espacial de la segunda fase.

El proceso se detiene si la diferencia entre los centroides de dos clusters en dos iteraciones sucesivas es menor que un umbral establecido. Después de converger, se aplica la regla de defuzzificación, la cual consiste en asignar cada píxel al cluster en el que el valor de función de membresía es mayor.

La interpretación e implementación de este algoritmo elegido se abordará más adelante.

3. IMPLEMENTACIÓN DE LA TÉCNICA DE SEGMENTACIÓN.

En el capítulo 2 se expuso la segmentación de imágenes empleando los modelos de *clustering c-means*, específicamente *hard c-means* (o *k-means*), *fuzzy c-means* y *fuzzy c-means* con información espacial propuesto por Chuang [21] *et. al*.

Con respecto a las funciones de la aplicación, se incluyen los tres modelos pertenecientes a la familia *c-means*. Sin embargo, en este capítulo sólo se mostrará la implementación computacional del algoritmo *c-means* con información espacial ya que la programación de los otros dos algoritmos de *clustering* es muy similar.

El hecho de contar con los tres modelos se debió a la consideración de aspectos como el grado de segmentación deseado y el tiempo de procesamiento requerido por cada modelo.

Adicionalmente a los algoritmos de *clustering*, la aplicación también implementa una técnica de umbralización, la cual no se detalla en esta sección.

3.1. Algoritmo para el modelo *fuzzy c-means* con información espacial

El último modelo presentado en el capitulo dos fue *c-means* con información espacial, en el que puede verse que, partiendo del modelo de *c-means estándar*, no es complicado integrar las modificaciones propuestas por Chuang [21], ya que sólo se agrega el cálculo de la función espacial y la modificación del cálculo de la función de membresía. Desde luego, existen pasos adicionales, pero éstos sirven para realizar ajustes necesarios para integrar las modificaciones principales.

El punto más importante del algoritmo es programar la función espacial, pues en el artículo que describe el modelo, la función está dirigida a imágenes, es decir elementos en dos dimensiones (2D). La función espacial se basa en los valores de la función de

membresía que tengan los pixeles vecinos en una ventana cuadrada. Por ejemplo, una ventana de 5x5 centrada en el píxel para el cual se calcula la función la ventana de análisis es

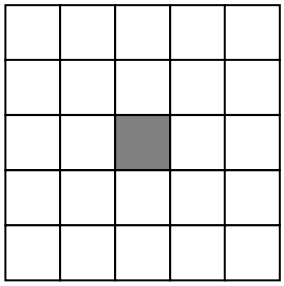


Figura 3.1. Ventana en dos dimensiones

Debido a que la aplicación construye un volumen en tres dimensiones a partir de imágenes, la vecindad también debe estar en 3 dimensiones. En el dibujo siguiente se muestra a un elemento correspondiente a un píxel con una vecindad de 5x5x5 (además, con el fin de mostrar el comportamiento en 2D, se proyecta el píxel en los planos)

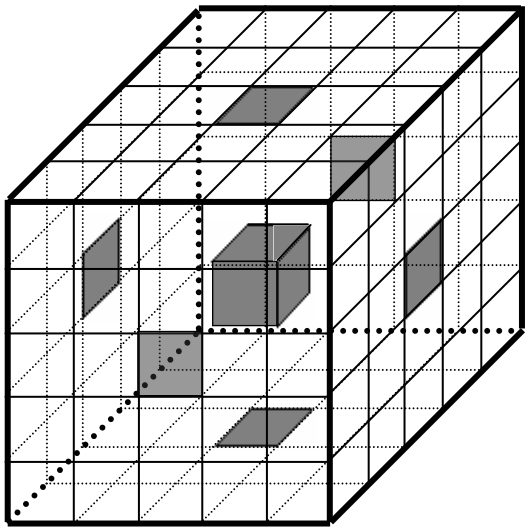


Figura 3.2. Ventana en tres dimensiones

Una vez que se ha explicado la forma en que se interpreta la función espacial, el siguiente paso es codificar el algoritmo aplicando estos últimos detalles.

3.1.1. Descripción de la técnica fuzzy c-means con información espacial

Enseguida se presentan los algoritmos de la técnica *fuzzy c-means* con información espacial. Además, por ser parte fundamental de la técnica, también se detallan las funciones de membresía.

Para hacer más comprensibles los algoritmos, se ha decidido presentar solamente pseudocódigo en lugar de código en un lenguaje de programación específico.

El algoritmo final para la técnica *fuzzy c-means* con información espacial es

1. Almacenar los datos no etiquetados $X \subset \Re^p$
2. Definir
   - el número de clusters: $1 < c < n$
   - el número máximo de iteraciones: T
   - el valor del exponente de peso: $1 \leq m$
   - la medida de similitud: $D_{ik} = \|x_k - v_i\|_A$
   - el indicador de término: $E_t = \|V_t - V_{t-1}\|$
   - el umbral de término: $0 < \varepsilon$
   - el tamaño de la ventana: W
   - el valor del exponente de peso de la función de membresía: p
   - el valor del exponente de peso de la función espacial: q
3. Inicializar los valores de los prototipos (centroides): $V_0 = \{v_{1,0}..., v_{c,0}\} \in \Re^{cp}$
4. Iterar

   $t \leftarrow 0$

   Repetir

   $t \leftarrow t + 1$

   Calcular U_t

Calcular U'_t

Calcular V_t

Hasta ($t = T$ o $E_t \leq \varepsilon$)

$(U,V) \leftarrow (U'_t, V_t)$

El pseudocódigo para obtener la función de membresía estándar U es

1. Leer

los datos originales no etiquetados $X \subset \Re$

el número de clusters k

el valor del exponente de peso: $1 \leq m$

2. Inicializar los centroides $C_0 = \{c_1 ..., c_k\} \in \Re$

3. Iterar

$i \leftarrow 0$

Repetir

$j \leftarrow 0$

Repetir

cociente $\leftarrow 0$

$g \leftarrow 0$

Repetir

Calcular $cociente \leftarrow cociente + \left(\frac{|X_i - c_j|}{|X_i - c_g|} \right)^{\frac{2}{m-1}}$

$g \leftarrow g + 1$

Hasta $g < k$

$U_{ji} \leftarrow \frac{1}{cociente}$

$j \leftarrow j + 1$

Hasta $j < k$

$i \leftarrow i + 1$

Hasta $i < total\ de\ elementos\ de\ X$

La segunda función de membresía se obtiene de manera inmediata a la función estándar, pues toma los valores que ésta calcula. Es importante mencionar que la función espacial se evalúa para cada píxel, es decir, no se puede realizar un procedimiento externo al de la función de membresía. Por lo tanto, el primer paso en este proceso es computar la función espacial.

1. Recibir y leer

   los valores de la función de membresía U

   el número de clusters k

   el tamaño de la ventana ventana

   el valor del exponente de peso: $1 \leq m$

   el valor del exponente de peso de la función de membresía p

   el valor del exponente de peso de la función espacial q

2. Iterar

   $i \leftarrow 0$

   Repetir

      Calcular h_{gi} $para\ g = 1...k$ (considerando el valor de *ventana*)

      $j \leftarrow 0$

      Repetir

         $l \leftarrow 0$

         $\text{cociente} \leftarrow 0$

         Repetir

            Calcular $\text{cociente} \leftarrow \text{cociente} + U_{\text{li}}^{\text{p}} * H_{li}^{q}$

            $l \leftarrow l + 1$

         Hasta $l < k$

         $$U'_{ji} = \frac{U_{ji}^{p} * H_{ji}^{q}}{cociente}$$

Hasta $j < k$

Hasta $i < total\ de\ elementos\ de\ X$

3.2. Pruebas de segmentación.

En la presente sección se muestran los resultados de segmentar imágenes por medio de los modelos de *clustering* mencionados en este trabajo. Sin embargo, es importante destacar que los resultados serán discutidos y evaluados más adelante, ya que en este capítulo la intención es, únicamente, mostrar el funcionamiento y resultados del proceso de segmentación.

Debido al modo en que la aplicación funciona, fue necesario crear volúmenes empleando una sola imagen; en estas pruebas se emplearon tres imágenes idénticas, a las que únicamente se les cambió el nombre de acuerdo a los requerimientos del programa.

La imagen original que se utilizó es una imagen de tomografía computarizada (CT) en formato bmp y con dimensiones de 320x320:

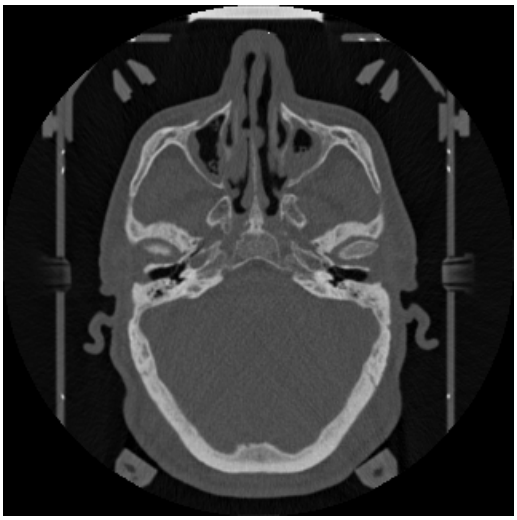


Figura 3.3. Imagen de tomografía en formato bmp

Con el fin de tener mejores condiciones de comparación, todos los algoritmos de segmentación se aplicaron para obtener tres clusters o clases. Y además, cabe aclarar que con el modelo modificado de *fuzzy c-means* se realizan tres pruebas en las que se cambian los valores de los parámetros p y q del modelo, así como el tamaño de la ventana.

Una vez que se han expuesto las condiciones de las pruebas, se procede a mostrar los resultados que los tres modelos de segmentación arrojaron para la imagen ya mencionada.

Para k-means se tiene

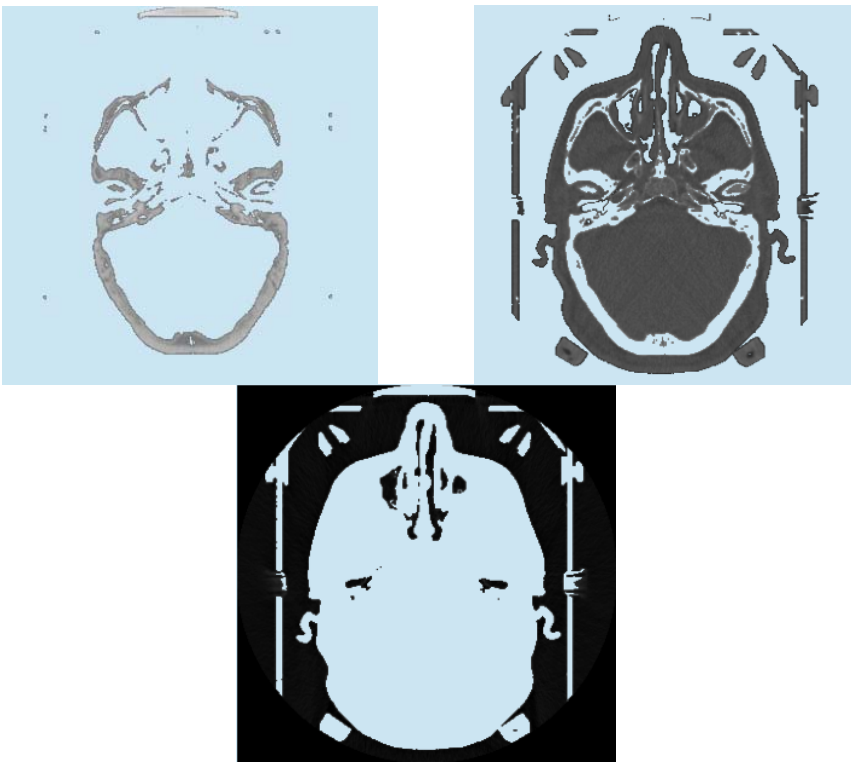


Figura 3.4. Clusters obtenidos con k-means

Para el método fuzzy c-means:

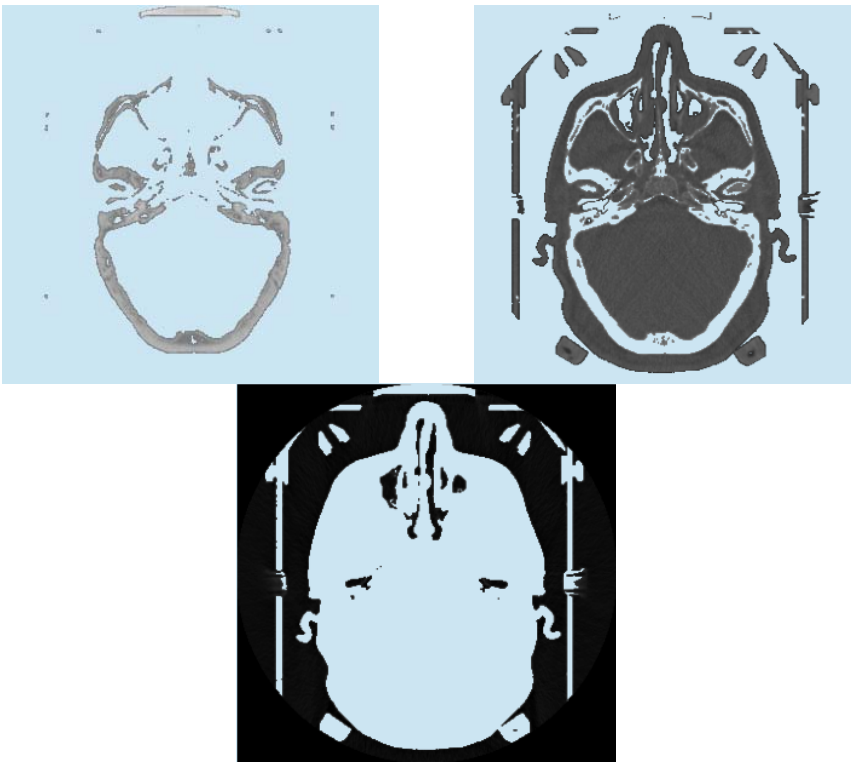


Figura 3.5. Clusters obtenidos con fuzzy c-means

Con el algoritmo modificado de fuzzy c-means con valores de *m=2*, *p=0* y *q=2* y una ventana de 3x3x3 se tiene

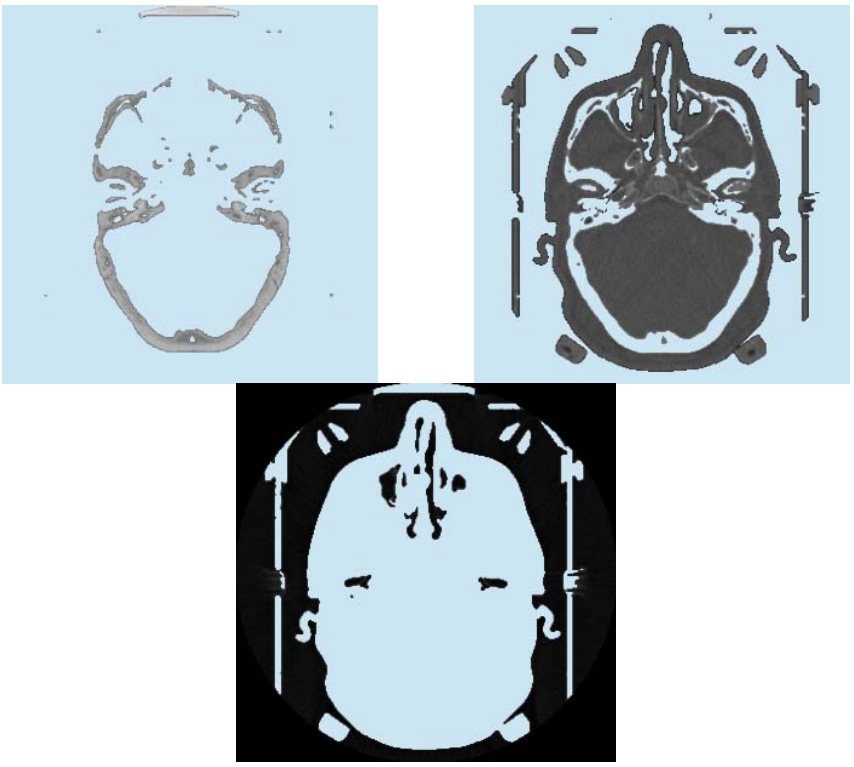


Figura 3.6. Clusters obtenidos con fuzzy c-means con información espacial (p=0, q=2, w=3)

Modificando los parámetros de *p=1* y *q=3* se obtienen los siguientes resultados

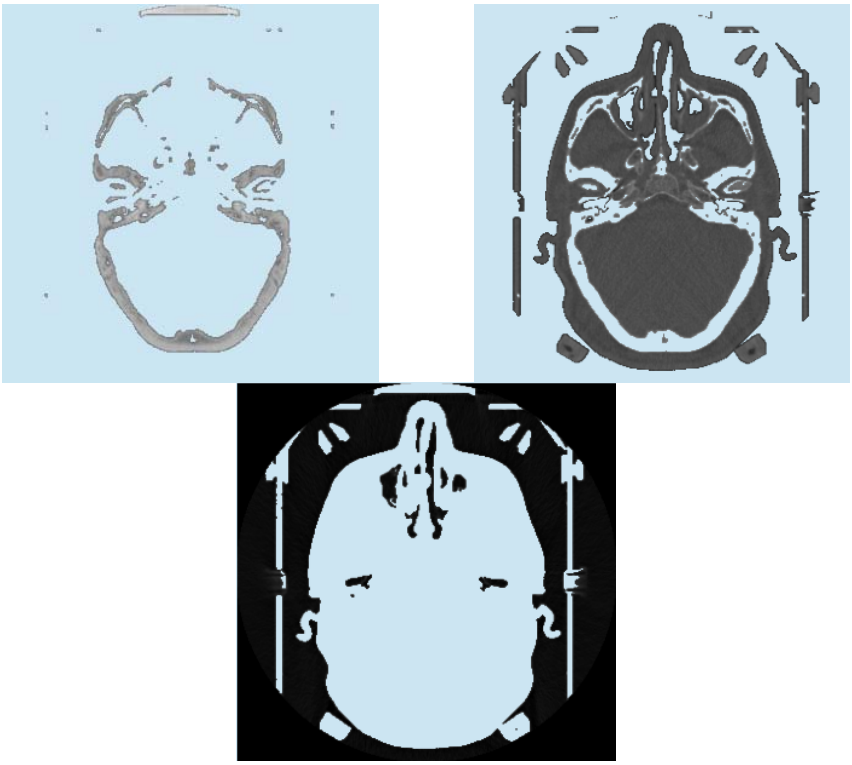


Figura 3.7. Clusters obtenidos con fuzzy c-means con información espacial (p=1, q=3, w=3)

Por otra parte, si lo que se cambia es el tamaño de la ventana y se establece en 5, permaneciendo los valores de *p=0* y *q=2*, se tiene

Figura 3.8. Clusters obtenidos con fuzzy c-means con información espacial (p=0, q=2, w=5)

El siguiente paso fue probar los métodos con imágenes con ruido. Para poder realizar comparaciones y evaluaciones visuales, se decidió generar una imagen a partir de la imagen de las pruebas anteriores agregándole ruido aditivo blanco gaussiano con media $\mu = 0$ y varianza $\sigma^2 = 0.05$ resultando la imagen siguiente

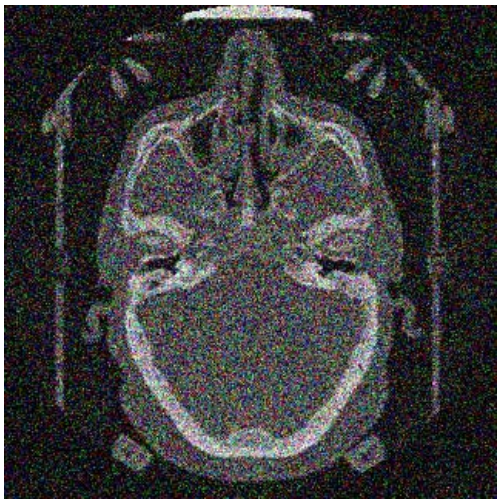


Figura 3.9. Imagen de tomografía con ruido blanco gaussiano aditivo

Los resultados de la segmentación de esta imagen, del mismo modo que las pruebas anteriores, se obtuvieron para 3 clusters o clases.

Empleando k-means se tienen las siguientes clases:

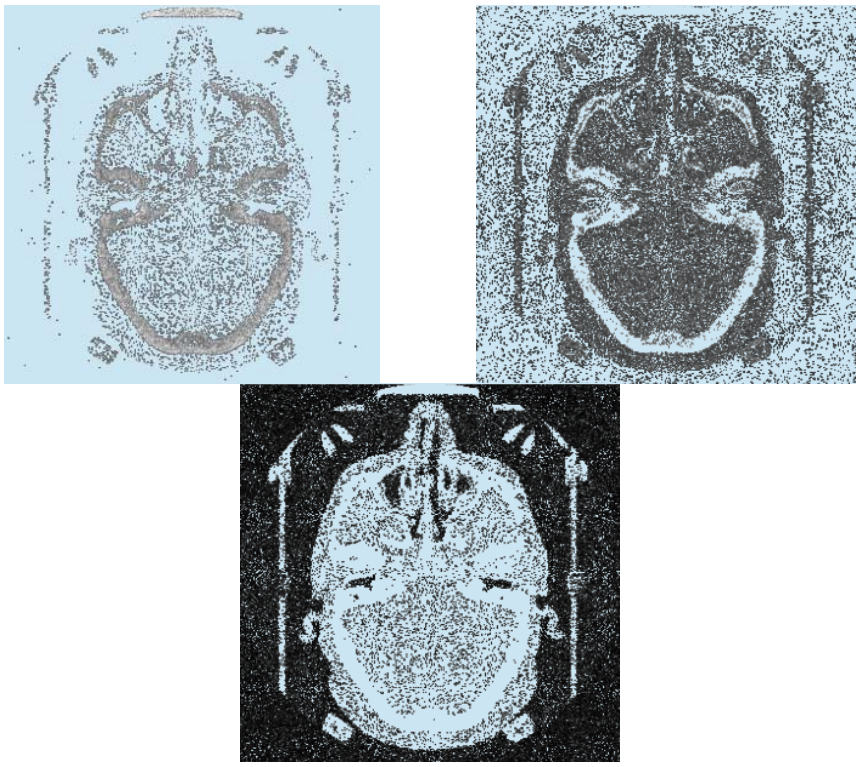


Figura 3.10. Clusters obtenidos con k-means

Por el algoritmo de fuzzy c-means resultan los clusters siguientes

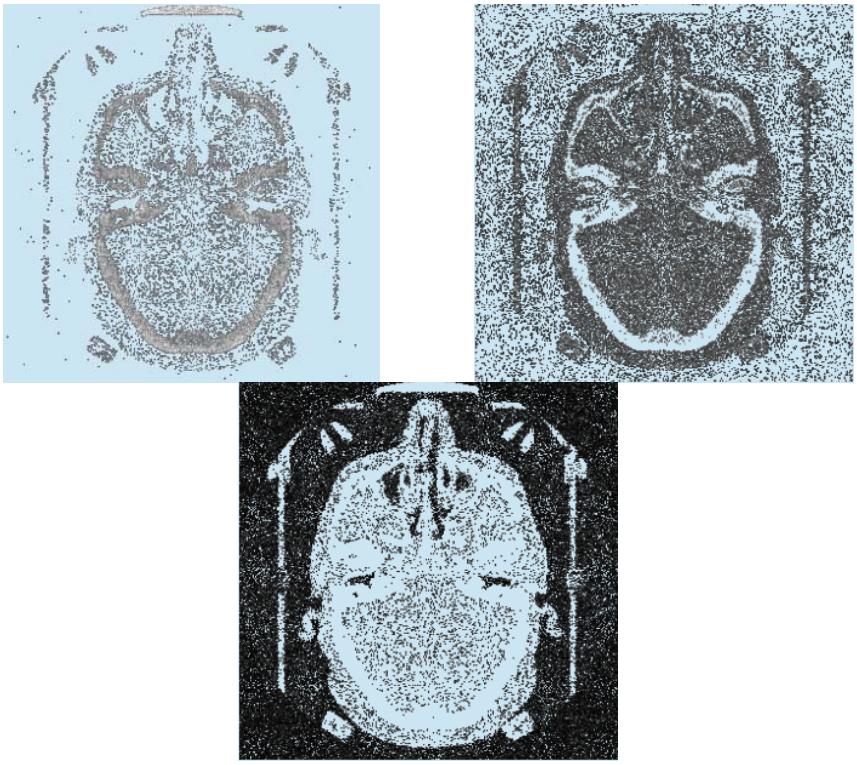


Figura 3.11. Clusters obtenidos con fuzzy c-means

Enseguida se muestran los resultados para distintas combinaciones de parámetros del algoritmo *fuzzy c-means* con información espacial. Comenzando con los parámetros *p=0*, *q=2* y *ventana=3* se tiene

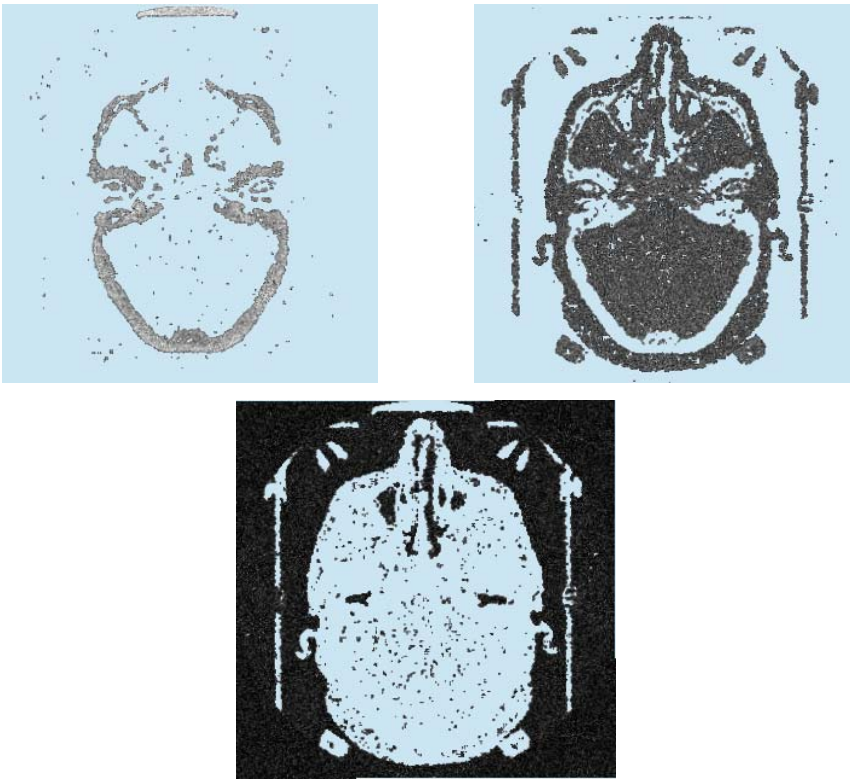
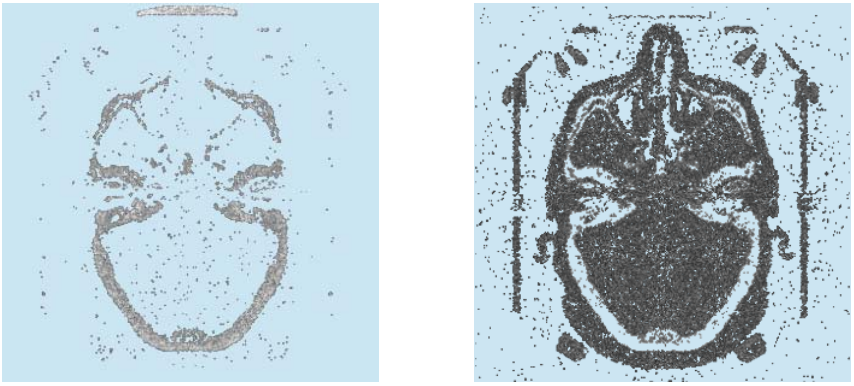


Figura 3.12. Clusters obtenidos con fuzzy c-means con información espacial (p=0, q=2, w=3)

Modificando los parámetros para tomar en cuenta la función espacial y darle más peso se toman los valores *p=1* y *q=3*, con lo que los clusters resultantes son



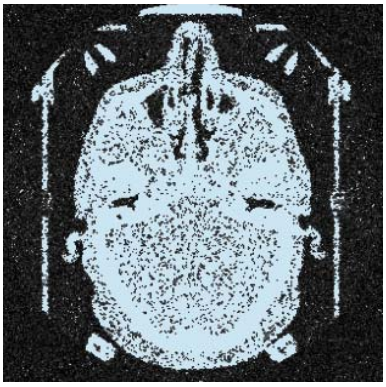


Figura 3.13. Clusters obtenidos con fuzzy c-means con información espacial (p=1, q=3, w=3)

Finalmente, en lugar de modificar los parámetros *p* y *q* (es decir, se mantienen *p=0*, *q=2*), se aumenta la ventana a 5, obteniéndose las clases siguientes

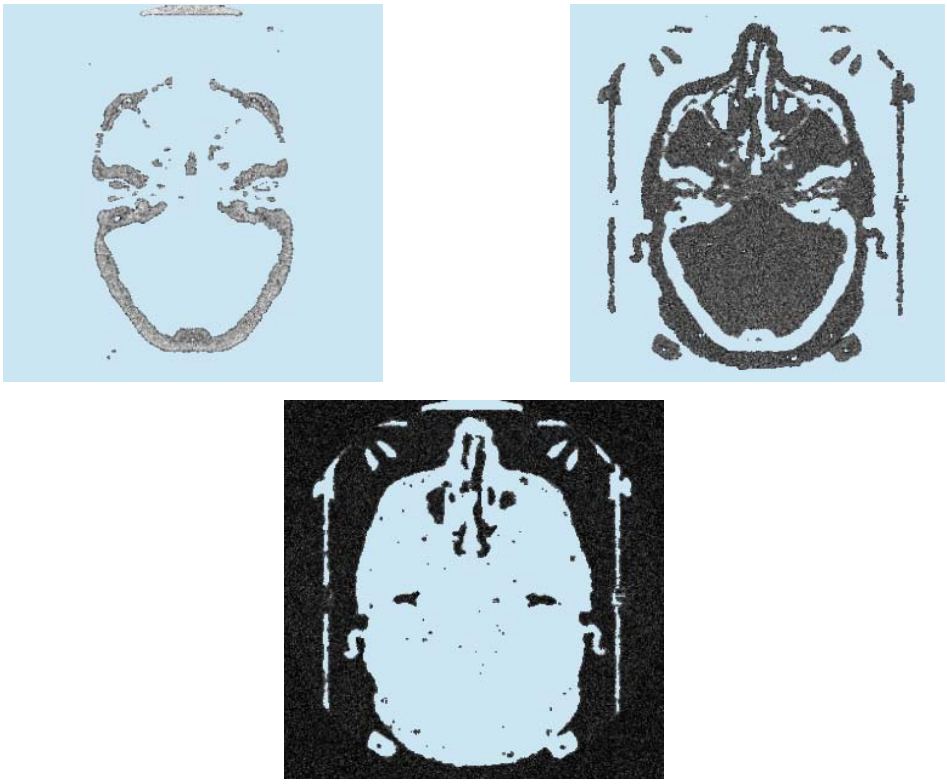


Figura 3.14. Clusters obtenidos con fuzzy c-means con información espacial (p=0, q=2, w=5)

4. DESARROLLO DE LA APLICACIÓN DE VISUALIZACIÓN.

4.1. Estructura general de la aplicación.

El objetivo de esta sección es proporcionar una breve introducción de los procedimientos involucrados en la aplicación de visualización, manipulación y segmentación de volúmenes; aquí se definen las partes principales que componen la aplicación sin todavía presentar la modulación completa de la misma, simplemente a forma de introducción se muestra la composición requerida para su estructuración.

El desarrollo de la aplicación involucró la organización de los diversos módulos a implementar para obtener un esquema global de la composición de la aplicación. De manera muy general se mencionan a continuación los principales componentes de la aplicación, los cuales en su enunciado explican de manera resumida su labor:

- Entrada de parámetros por parte del usuario.
- Obtención de los datos de volumen a través del manejo de diferentes tipos de archivo.
- Transformación de los datos originales a datos apropiados para su manejo interno.
- Determinación de las dimensiones del cubo de datos usando como insumos las entradas por parte del usuario y la lectura de los datos de volumen.
- Elaboración del conjunto de datos requerido para la geometría del volumen.
- Puesta a punto de los parámetros de visualización y de apariencia.
- Rendereo del volumen de datos original.
- Rendereo de la interfaz gráfica de usuario.
- Manejo de las peticiones enviadas por el usuario.
- Aplicar al volumen las peticiones de procesamiento provenientes del usuario.
- Aplicar a la ventana de visualización las peticiones de transformación geométrica provenientes del usuario.
- Encausar otras peticiones del usuario.

El listado de los componentes ayuda a comprender lo que la aplicación está destinada a realizar, sin embrago, es conveniente observar la manera en que se conectan estos componentes para modelar el comportamiento de la misma. Un diagrama de bloques de los componentes de la aplicación se muestra en la figura 4.1

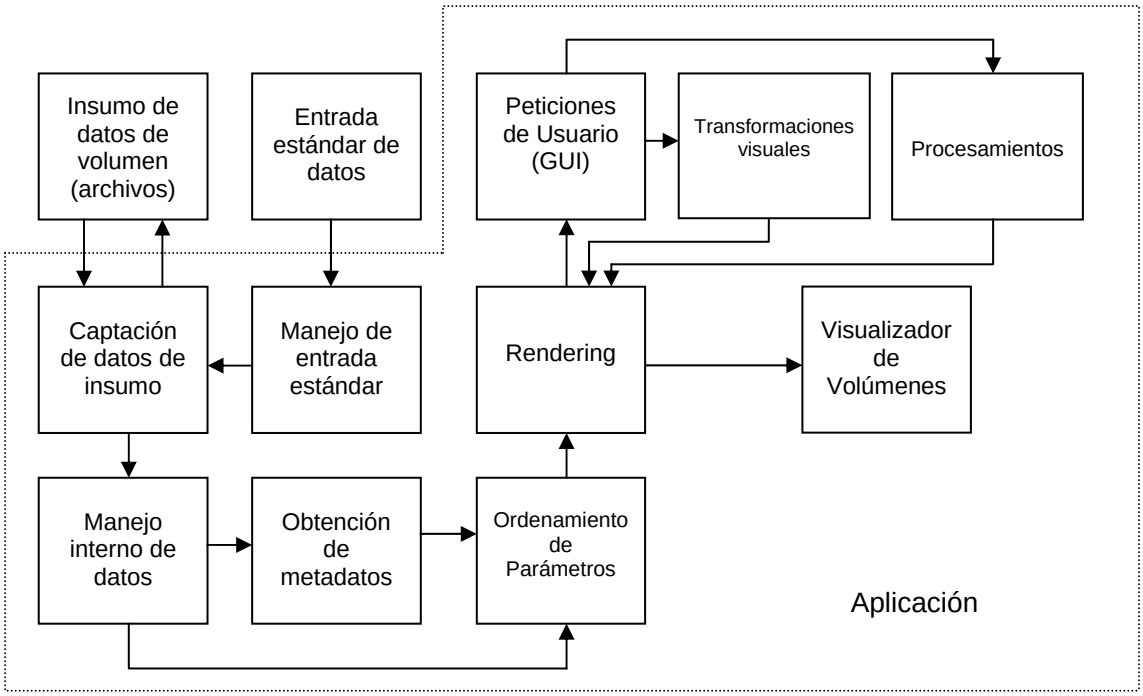


Figura 4.1. Diagrama de bloques de la aplicación

4.2. Diagramas de flujo de datos (DFD) correspondientes a los módulos.

Una vez que se ha presentado un modelo para la estructura general de la aplicación, a continuación se procede a entrar más en detalle en los diversos procesos involucrados. Una forma de lograr esto es diseñando el comportamiento de la aplicación por medio de los diagramas de flujos de datos que son una herramienta conceptual destinada a representar gráficamente los procesos y flujos de datos de un sistema de información. El uso de estos diagramas proporciona al desarrollador la libertad de realizar en forma muy temprana la implementación técnica del sistema, además, proporcionan una mayor comprensión de las interrelaciones de los diversos procesos involucrados mientras que, también, permite efectuar un análisis del sistema propuesto enfocado a determinar si se han definido todos los datos y procesos necesarios [22].

4.2.1. Definición formal y elementos básicos de los DFD.

Formalmente, un diagrama de flujo de datos (comúnmente abreviado DFD) es una técnica de análisis estructurado mediante la cual, un analista de sistemas puede reunir la representación gráfica de los procesos de datos involucrados en un sistema de información. El enfoque de flujo de datos enfatiza la lógica subyacente del sistema. Mediante el uso de combinaciones de solamente cuatro símbolos, el analista puede crear una representación pictórica de los procesos que eventualmente proporcionarán una documentación firme del sistema [22]. Los cuatro elementos básicos antes mencionados son los siguientes:

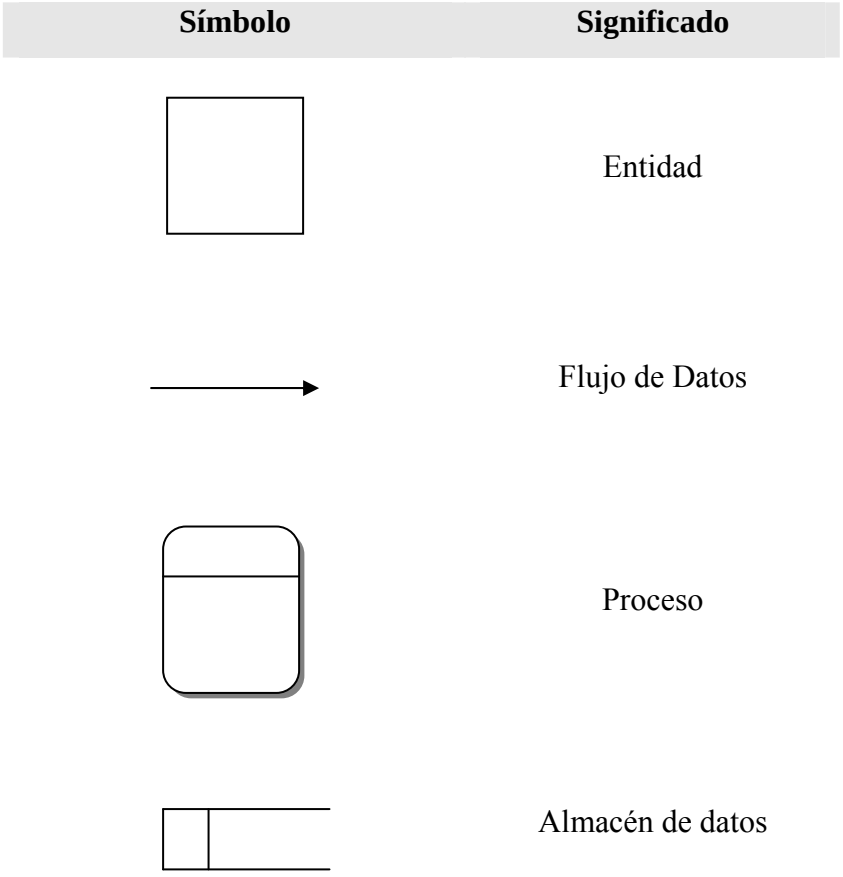


Figura 4.2. Símbolos Básicos del Diagrama de Flujo de Datos

El cuadrado es usado para representar una entidad externa (como por ejemplo, un usuario) que puede enviar o recibir datos del sistema y es considerada ajena al estudio; cada entidad externa debe ser etiquetada con un nombre adecuado. La flecha muestra el movimiento de datos de un punto a otro, señalando hacia el destino de los datos y debe ser descrita con un nombre asociado a los datos que representa. Un rectángulo con esquinas redondeadas es empleado para mostrar la aparición de un proceso de transformación, debe llevar un nombre que lo identifique de tal manera que describa la función del proceso; puede llevar también un número consecutivo que lo distinga [22]. El último símbolo de la figura 4.2 representa un almacén de datos (almacenamiento físico de datos) y es un rectángulo abierto.

Existen tres reglas básicas a seguir al momento de elaborar diagramas de flujo de datos:

1. Los flujos de datos no deben dividirse en dos o más flujos de datos diferentes.
2. Todos los flujos de datos deben iniciar o terminar en un proceso.
3. Los procesos necesitan tener al menos un flujo de datos de entrada y un flujo de datos de salida.

El uso de los diagramas de flujo de datos está originalmente destinado al modelado de los sistemas de manejo de datos de organizaciones o instituciones para su sistematización posterior, pero resultan ser muy útiles en el modelado estructural de un software particular.

4.2.2. Diagrama de contexto de la aplicación.

Una de las ventajas que proporciona el uso de los diagramas de flujo de datos es la flexibilidad que da para la creación de diagramas de diferentes niveles de expansión. En primera instancia se tiene el diagrama de contexto, después el llamado “Diagrama 0” del cual pueden elaborarse finalmente diagramas hijo que describen más detalle los procesos involucrados en el Diagrama 0.

El diagrama de flujo de datos a nivel contexto representa la panorámica de un sistema completo, en este caso, de la aplicación de visualización. El listado de las actividades que debe efectuar el sistema resulta de gran importancia al momento de elaborar el diagrama de contexto y el diagrama cero, recuérdese que este listado aparece en la sección 4.1 de este capítulo.

El diagrama de flujo de datos a nivel contexto para la aplicación se presenta en la figura siguiente:

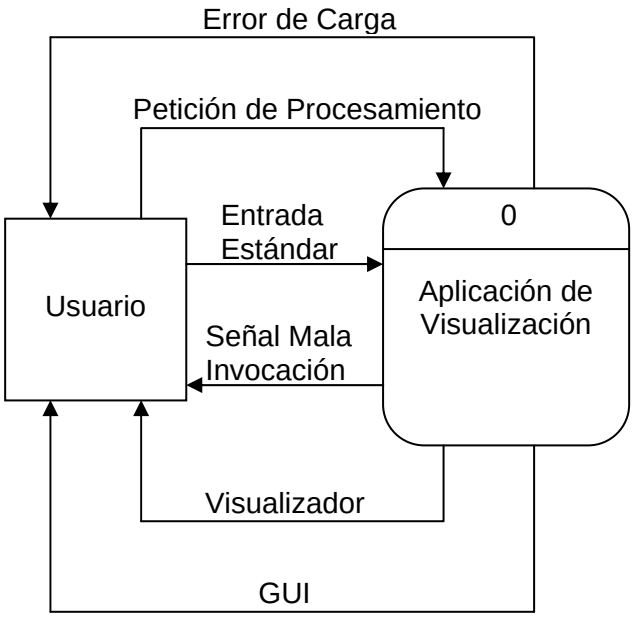


Figura 4.3. Diagrama de contexto

En el diagrama de la figura 4.3 se pueden observar las diversas interacciones entre el usuario final y la aplicación, resultando relevantes la presentación al usuario del visualizador de volúmenes y la interfaz gráfica de usuario (GUI) que le permitirá a éste efectuar las peticiones de procesamiento. La aplicación de visualización originalmente se invoca mediante la entrada estándar de datos a través de la línea de comandos de un *shell*, en la cual se dice a la aplicación que volumen será cargado y de que forma. En caso de ocurrir errores en la carga del volumen, se envían señales al usuario que indican que un error ha ocurrido.

4.2.3. Diagrama 0 y diagramas hijo de la aplicación.

El Diagrama 0 es la explosión del diagrama de contexto e involucra los principales procesos de la aplicación. Dicho diagrama se presenta a continuación:

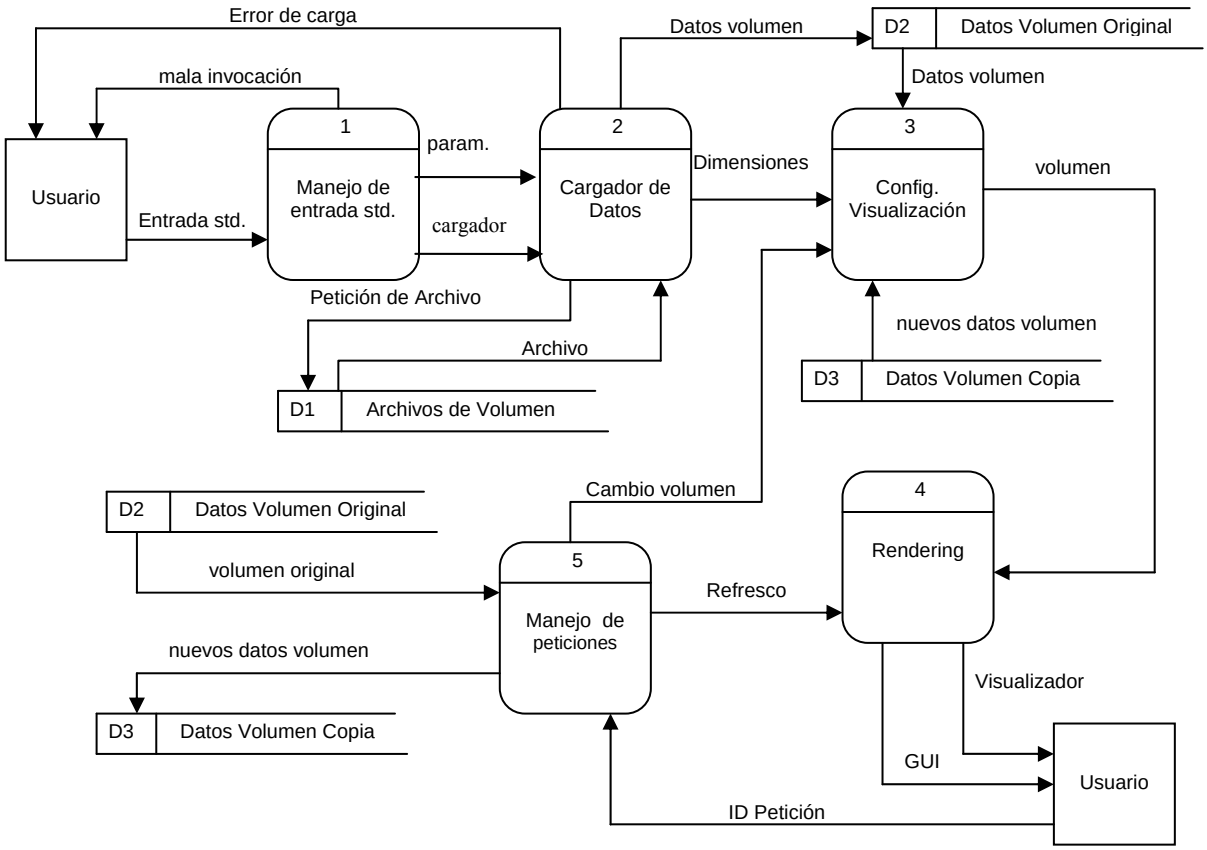


Figura 4.4. Diagrama 0

Cada proceso del Diagrama 0 puede a su vez ser explotado para crear un diagrama hijo más detallado. El proceso del Diagrama 0 que es explotado se llama proceso padre y el diagrama que resulta es llamado diagrama hijo. Un diagrama hijo no puede producir salida o recibir entrada que el proceso padre no produzca o reciba, sin embrago puede incluir almacenes de datos no descritos en el Diagrama 0. Todos los flujos de datos de entrada o salida del proceso padre deben ser mostrados entrando o saliendo al diagrama hijo [22]. A continuación se muestran los diagramas hijo para los procesos del Diagrama 0.

Diagrama hijo del proceso 1: Manejo de Entrada Estándar.

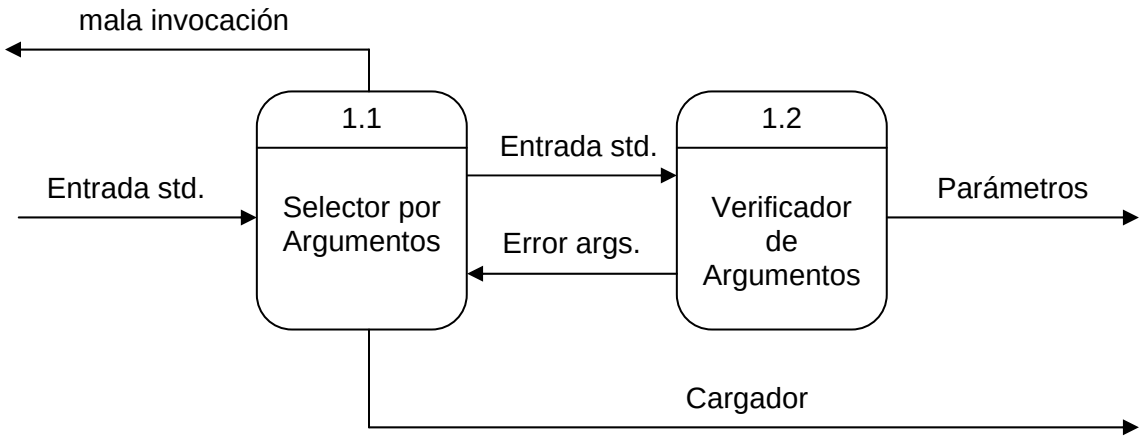


Figura 4.5. Diagrama hijo del proceso 1

Diagrama hijo del proceso 2: Cargador de datos.

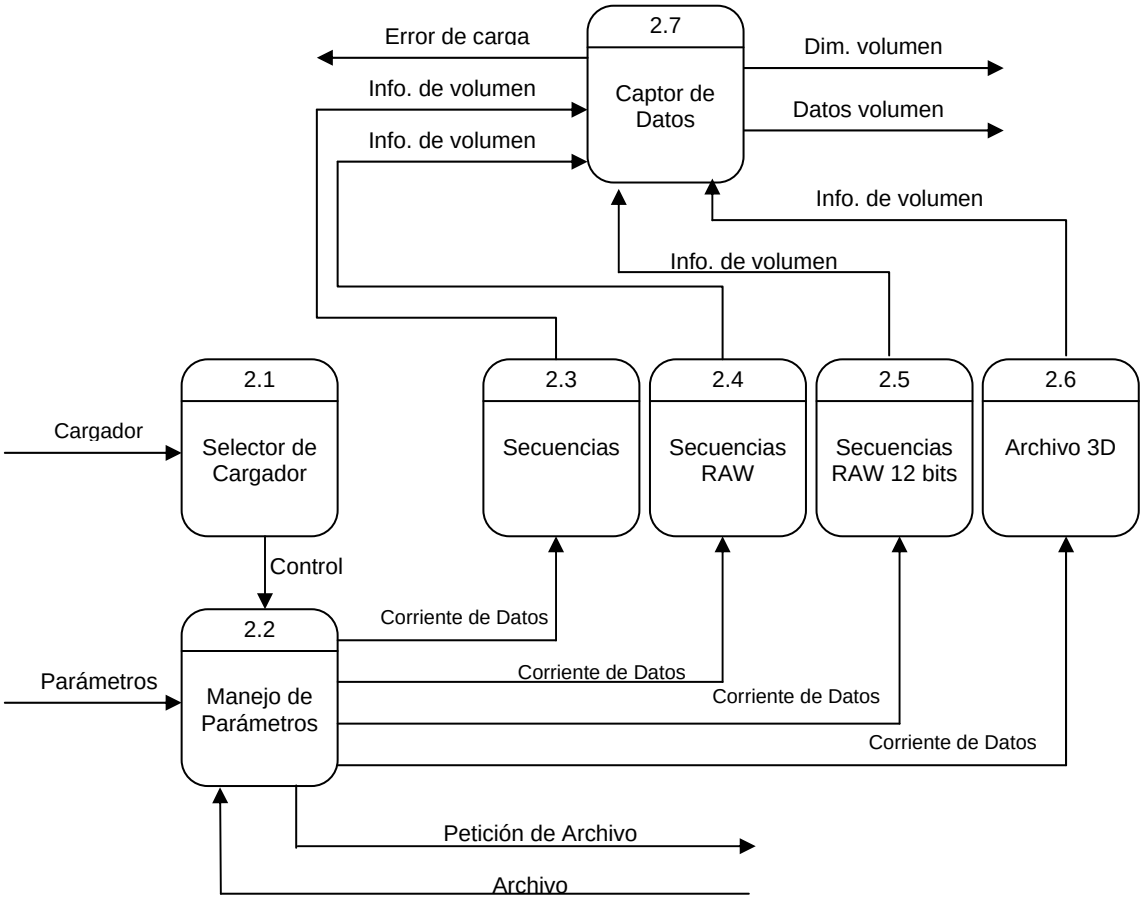


Figura 4.6. Diagrama hijo del proceso 2

Diagrama hijo del proceso 3: Configuración de Visualización.

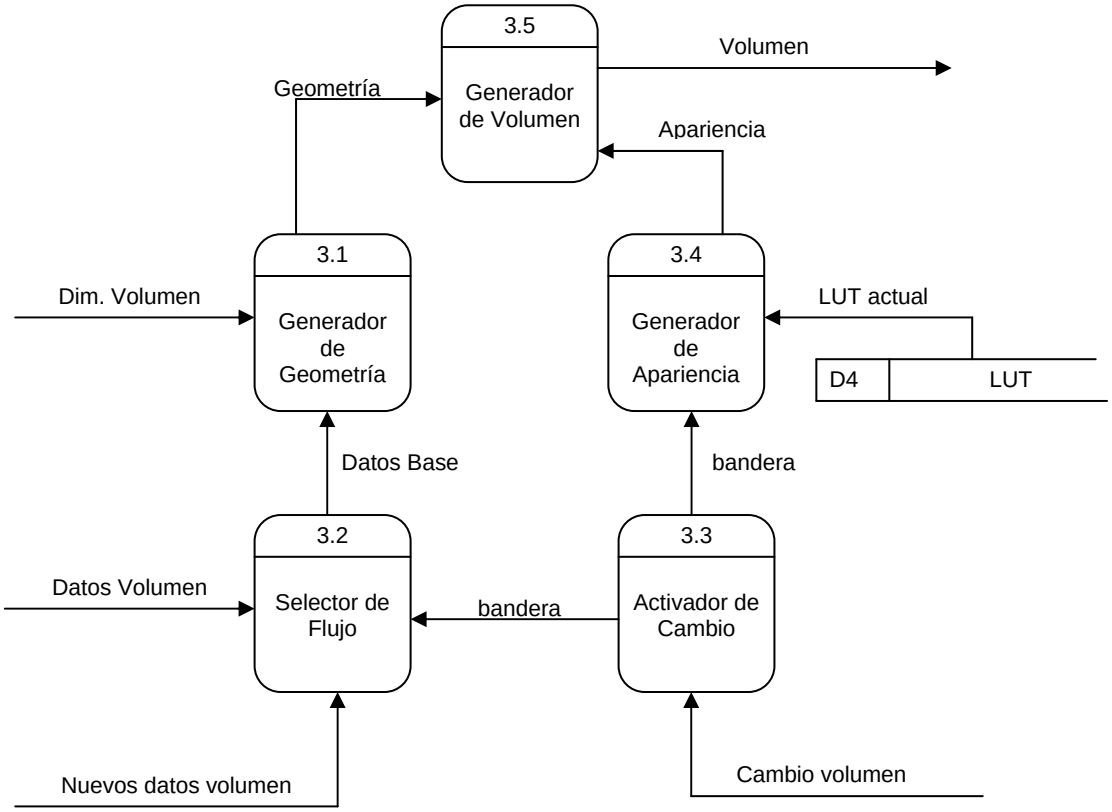


Figura 4.7. Diagrama hijo del proceso 3

Diagrama hijo del proceso 4: Rendering.

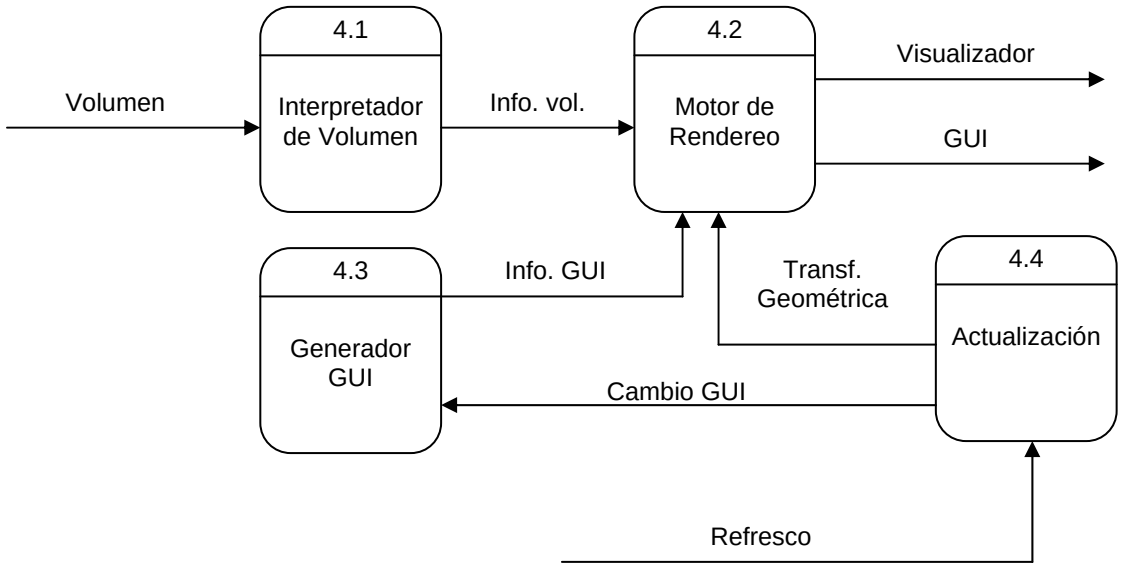


Figura 4.8. Diagrama hijo del proceso 4

Diagrama hijo Proceso 5: Manejo de Peticiones.

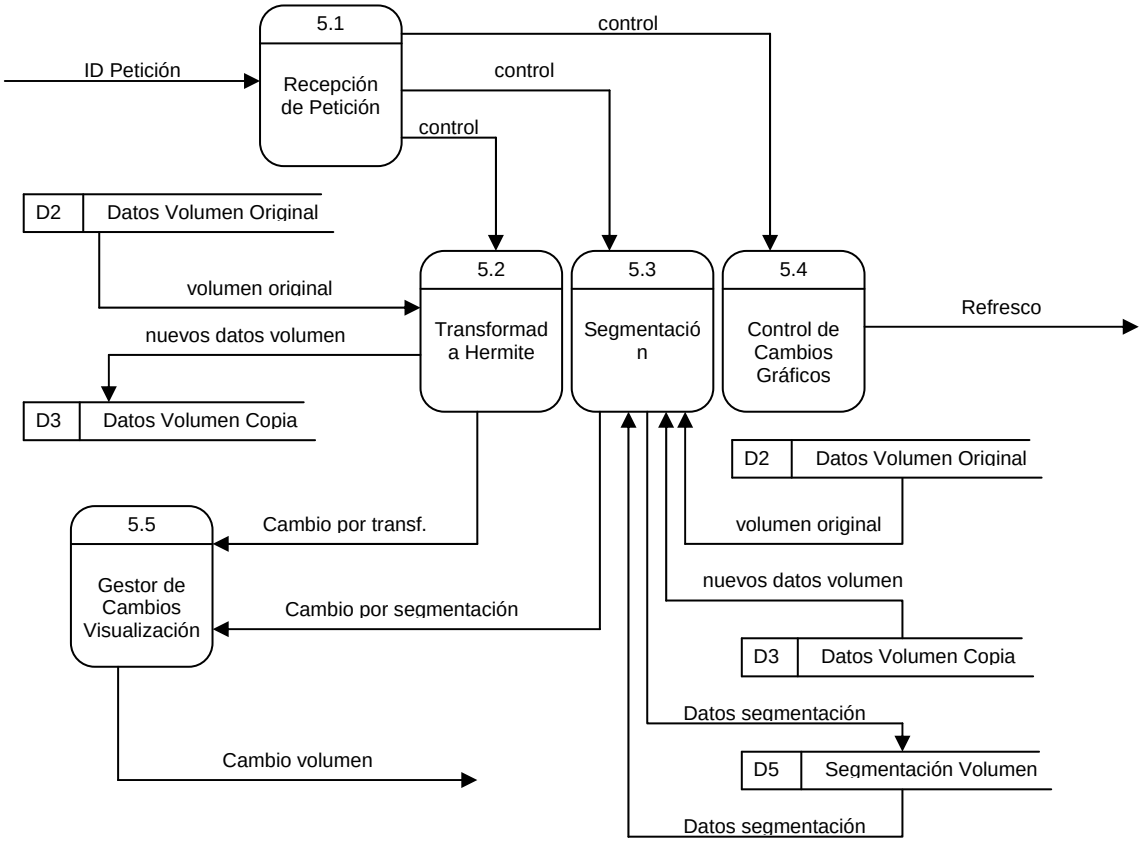


Figura 4.9. Diagrama hijo del proceso 5

Los diagramas elaborados en esta sección sirven como punto de partida para el desarrollo del código de la aplicación. Recuérdese que, en primera instancia, se efectuó un análisis de requerimientos que desembocó posteriormente en un diseño estructurado para la configuración de la aplicación, descrito, principalmente en esta sección.

4.3. Librerías

Para cumplir con los objetivos y metas de esta tesis, antes de comenzar con el desarrollo de los algoritmos propios de la segmentación de imágenes extendidos a volúmenes, se tuvo la necesidad de emplear productos de software existentes que contribuyeran a la realización de diversos módulos de la aplicación. Entre los mencionados módulos se tienen principalmente: el módulo encargado de la visualización final en pantalla

de los datos, el módulo de obtención de datos a partir de cuadros o secuencias de imágenes en formatos comerciales y, por último y no menos importante, la interfaz gráfica con el usuario.

La selección de los productos de software fue influida por el concepto de portabilidad, principalmente para las plataformas Windows e Irix, además, un factor importante tomado en cuenta fue la capacidad de integración de todas las herramientas sin la existencia de conflictos graves que impidieran dicho proceso. Por otro lado, el hecho de tener una licencia de OpenGL Volumizer en el lugar de trabajo donde se desarrolló la aplicación fue un factor de peso para decidirse por este producto de software, ya que, además de ser una importante herramienta para el despliegue de volúmenes en pantalla, presenta también diversas funcionalidades muy útiles.

Dado que OpenGL Volumizer tiene como punto de apoyo a OpenGL, se determinó el uso de OpenGL GLUT para la gestión de las ventanas de la aplicación y también para la interfaz con el usuario por medio del teclado y el ratón.

Finalmente, se decidió elaborar una interfaz gráfica de usuario más completa y hasta cierto punto intuitiva y de fácil manejo pero con la vista siempre fija en la capacidad multiplataforma que la aplicación debe tener. Una vez seleccionadas las herramientas de software necesarias y de cierta manera vitales para el soporte de los principales módulos de la aplicación, se decidió optar por la librería GLUI para la elaboración de la interfaz gráfica de usuario, tomando como razón principal el hecho de que esta librería también está sentada sobre GLUT, lo cual, permite tener una estructura piramidal para la aplicación con una base sólida y común en la mayor parte de sus componentes.

En las secciones siguientes y en el anexo A se pretende dar a conocer detalles finos de los productos de software que se involucraron con el desarrollo de esta aplicación, mencionando aspectos importantes de los mismos que fueron usados extensamente en el código para que de esta forma se logre tener un panorama completo de la arquitectura del software que se desarrolló en el presente trabajo.

4.3.1. La librería de carga de imágenes DevIL [23].

Se eligió el uso de DevIL (Developer's Image Library) dado que permite la fácil manipulación de imágenes en formatos comerciales como JPEG, BMP, TIFF o PNG. Por otra parte, permite hacer un manejo de la información completa contenida en las imágenes como lo es la resolución y los bytes por píxel; además de que separa la cabecera (presente en diversos formatos) de la información visual, lo cual sirve como base para la obtención de los datos de volumen si es que éste se encuentra representado en secuencias de imágenes. Adicionalmente, al ser de código libre o abierto, resultó ser una opción bastante viable.

El método ilLoadImage es el método que fue empleado en la aplicación desarrollada y es el método principal de DevIL para la carga de imágenes y requiere como único parámetro el nombre de la imagen que se desee cargar. A su vez, es el método más poderoso de la librería dadas sus características. Para el proyecto se empleó la versión 1.6.8 de DevIL.

4.3.2. OpenGL y OpenGL GLUT.

Antes de hablar sobre OpenGL GLUT es conveniente tratar algunos detalles sobre OpenGL, ya que dicha librería es el soporte principal sobre el cual se apoya la aplicación en lo que respecta a la parte de visualización.

4.3.2.1. OpenGL

El sistema de gráficos de OpenGL es una interfaz de software para el manejo del hardware de gráficos creada por la compañía *Silicon Graphics* y es una interfaz independiente del hardware diseñada para ser implementada en diferentes plataformas, lo que quiere decir que no incluye comandos para el manejo de ventanas en los diferentes

sistemas operativos; a su vez, tampoco soporta el manejo de entradas por parte del usuario. OpenGL no provee comandos de alto nivel para la descripción de modelos de objetos nativos de tres dimensiones, estos modelos deben construirse empleando un conjunto de primitivas geométricas como lo son puntos, líneas y polígonos.

Antes de detallar otros aspectos es importante hacer mención de dos términos que son empleados comúnmente en el argot de los gráficos por computadora:

- *Rendering*: es el proceso mediante el cual, la computadora digital crea imágenes en pantalla basándose en modelos.
- *Modelos*: los modelos u objetos son construidos a partir de primitivas geométricas que especifican los vértices de los mismos.

Dadas las limitaciones que presenta OpenGL, existen más librerías adicionales relacionadas que simplifican algunas tareas de programación, entre las principales encontramos a la librería de utilidades de OpenGL (GLU) y al kit de herramientas de utilidades para OpenGL (GLUT). GLU contiene diversas rutinas que emplean los comandos básicos de OpenGL de bajo nivel para realizar acciones como la especificación de las matrices para orientaciones de vistas, proyecciones y el trazo de superficies.

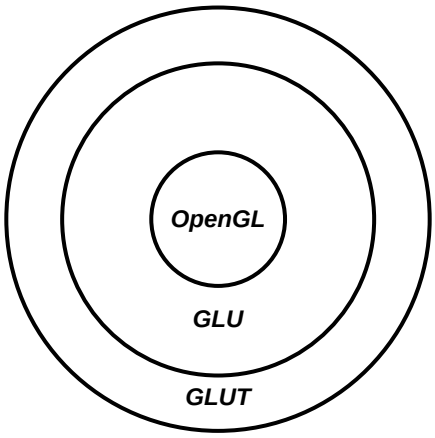


Figura 4.10. Librerías relacionadas con OpenGL

4.3.2.2. GLUT (OpenGL Utility Toolkit)

GLUT es un kit de herramientas independiente del sistema de ventanas escrito por *Mark Kilgard* [24] con el objetivo de esconder las complejidades de las interfaces de programación de aplicaciones de los diferentes sistemas de ventanas. Es, además, una interfaz de programación que proporciona vínculos para los leguajes ANSI C y FORTRAN que permiten la realización de programas independientes de la plataforma que manejen el sistema de ventanas correspondiente [25].

Las rutinas empleadas por esta librería necesitan de pocos parámetros y no regresan ningún tipo de apuntador; los únicos apuntadores que pueden ser pasados a estas rutinas son apuntadores a cadenas y a manejadores de fuentes de texto. La interfaz de programación de aplicaciones de GLUT es independiente del sistema de ventanas.

Las rutinas utilizadas son clasificadas dentro de diferentes sub-interfaces de acuerdo a su funcionalidad [25], las principales sub-interfaces de GLUT son:

- *Procesamiento de Eventos:* Esta rutina consiste en el ciclo de eventos de GLUT y no regresa nunca. Procesa las llamadas de servicio (callbacks) continuamente si es necesario. Las llamadas de servicio son rutinas especificadas por el programador y que pueden ser registradas con GLUT para responder a un tipo específico de evento.
- *Manejo de Ventanas:* Estas rutinas crean y controlan las ventanas.
- *Manejo de Recubrimientos:* Estas rutinas establecen y manejan los revestimientos de las ventanas.
- *Manejo de Menú:* Estas rutinas crean y controlan los menús.
- *Registro de las Llamadas de Servicio:* Estas rutinas registran las llamadas de servicio que necesitan ser tratadas por el ciclo de procesamiento de eventos de GLUT.
- *Rendereo de Formas Geométricas:* Estas rutinas permiten el rendereo de figuras geométricas de tres dimensiones como esferas y conos.

El hecho de que GLUT sea independiente del sistema de ventanas particular de cada sistema operativo es una característica importante que fue tomada en cuenta para su uso en este proyecto, además de que se coloca por encima del marco común que es OpenGL.

La versión usada de OpenGL GLUT es la 3.7.

4.3.3. El software de visualización OpenGL Volumizer [26].

4.3.3.1. Generalidades.

OpenGL Volumizer es, en su versión 2, un kit de desarrollo de software (SDK) de *Silicon Graphics* que proporciona principalmente una librería de clases para el lenguaje de programación de alto nivel C++, es decir, contiene una interfaz de programación de aplicaciones (API) para entornos de desarrollo C++. Las clases y métodos contenidos proporcionan facilidades para la manipulación y despliegue de conjuntos de datos volumétricos comunes en aplicaciones para medicina, ciencias e ingeniería. La librería proporciona una capa de funcionalidades que se encuentra apoyada sobre *OpenGL*. Por cuestiones de simplicidad se referirá a OpenGL Volumizer solamente como *Volumizer* de aquí en adelante en este escrito.

Figura 4.11. Logotipo de OpenGL Volumizer

Volumizer está diseñado para funcionar en las actuales y futuras plataformas de dicha compañía lo cual permite a las aplicaciones escritas sobre la librería tener la premisa de ser funcionales en los nuevos sistemas de *Silicon Graphics* con pequeñas o nulas modificaciones usando al máximo las capacidades del hardware para gráficos.

De manera particular, para el desarrollo de la presente aplicación se utilizó *Volumizer* en su versión 2.9 sobre tarjetas gráficas de la familia *NVIDIA Quadro FX* en sistemas operativos *Windows XP* empleando los compiladores de *Microsoft Visual C++* en las versiones 7 y 8. A su vez la aplicación fue probada en el sistema *Onyx* que se encuentra

en el observatorio iXtli de la Dirección General de Servicios de Cómputo Académico de la Universidad Nacional Autónoma de México. Adicionalmente, se instaló la librería GnuWIN32 Tiff [27] versión 3.6.1 debido a que permite que *OpenGL Volumizer* realice la lectura de los archivos *tiff*.

4.3.3.2. El trazo de volúmenes.

Existen diversas maneras para visualizar volúmenes usando la computadora digital, muchas de ellas usan técnicas de análisis de datos con el objetivo de localizar superficies de contorno en el volumen de interés y entonces proceden al trazo de la geometría resultante. Un método común es el uso de la técnica denominada "*ray casting*", sin embargo, *Volumizer* emplea un método diferente.

El mapeo de texturas en tres dimensiones es una técnica de visualización directa que maneja rebanadas u obleas texturizadas que una API o aplicación combina en un orden específico haciendo uso de un operador de mezcla. El objetivo es pasar de una escena compuesta absolutamente por primitivas de tres dimensiones a una imagen o arreglo de píxeles en pantalla. *Volumizer* se ayuda de las características innatas de *OpenGL* para mapear texturas al espacio de la escena (desde su versión 1.2 *OpenGL* tiene soporte para el mapeo de texturas de tres dimensiones [28]) con lo cual se obtienen atributos materiales y se permite a los objetos gráficos tener una apariencia más real.

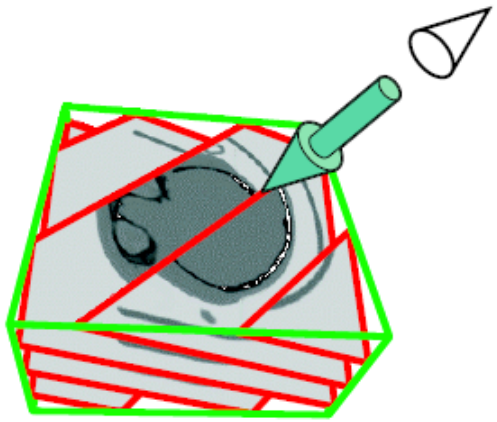


Figura 4.12. Mapeo de texturas de tres dimensiones [29]

4.3.3.3. La interfaz de programación de aplicaciones de Volumizer.

La estructuración jerárquica de una escena gráfica es usada por *Volumizer* para retener y organizar los parámetros de visualización.

En esta estructuración el nodo principal de la escena es el denominado nodo "*shape*" o nodo de forma que contiene tanto la geometría como la apariencia de la escena. La geometría del volumen define los atributos espaciales, mientras que la apariencia del volumen define diversos parámetros para la visualización. El componente geométrico del nodo de forma se describe aparte de la componente de apariencia de dicho nodo, haciendo posible la separación de la geometría o atributo espacial y la apariencia o atributo visual.

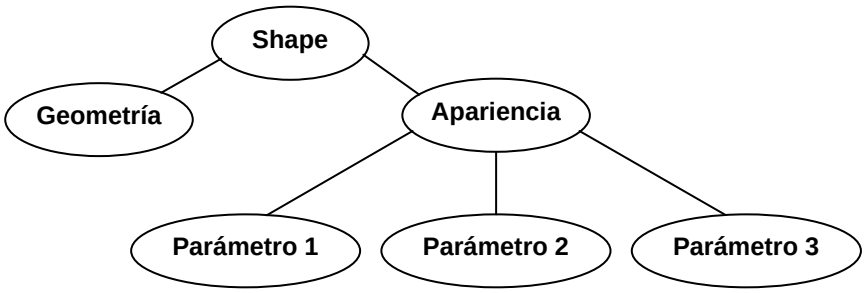


Figura 4.13. El nodo "shape" o de forma

Volumizer permite especificar arbitrariamente, usando su primitiva básica, geometrías volumétricas, las cuales van desde cubos alineados con los ejes coordenados hasta conjuntos tetraédricos. Así como los triángulos son la primitiva base usada para describir geometrías poligonales, un tetraedro es la primitiva base empleada para describir geometrías volumétricas.

La API usa el tetraedro como primitiva básica para todas sus operaciones distribuyendo diferentes tetraedros en representaciones geométricas de acoplamiento. Por ejemplo, un cubo puede ser representado con cinco tetraedros.

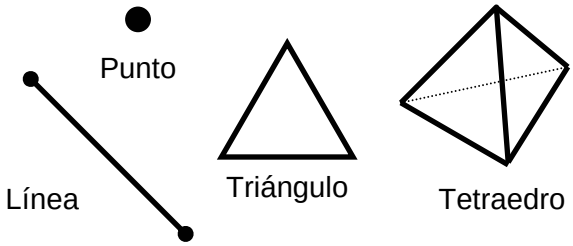


Figura 4.14. El tetraedro como primitiva básica de tres dimensiones

Para una visualización efectiva de los conjuntos de datos, los valores de dichos datos necesitan a menudo ser mapeados a diferentes valores de color o grados en la escala de grises. Ese mapeo se especifica usando tablas de apareamiento o post interpolación (*"look up tables"*, LUT) soportadas por el *pipeline* de gráficos. Diferentes valores de gris y transparencia en datos volumétricos por lo general corresponden a diferentes tipos de materiales que se encuentran en el volumen que está siendo dibujado. Una función de transferencia no lineal puede ser aplicada a los *textels* (elementos básicos de las texturas) para ayudar a analizar los datos. Por medio de valores gráficos de umbral se pueden extraer superficies en tiempo real. Volumizer implementa el parámetro LUT de post interpolación que mapea colores y niveles de gris después de la interpolación de texturas, algo similar a la transformación del histograma de una imagen.

4.3.3.4. El pipeline utilizado por el motor de gráficos de Volumizer

La secuencia de procedimientos a efectuar, utilizada por el motor de trazo de Volumizer para dibujar volúmenes, puede apreciarse en su estructura, en la figura 4.15 [29]:

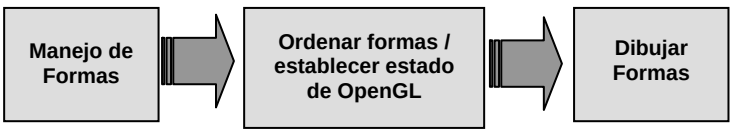


Figura 4.15. El *pipeline* de *TMRenderAction*

En primer lugar, la aplicación computa el número de formas que necesita mantener residentes en memoria de textura para una escena dada. La lista de formas puede ser resultado por ejemplo, del rayo de visibilidad en una aplicación inmersiva, la escena actual en una simulación variante en el tiempo, etc. Ya que la aplicación haya terminado de manejar y dejar de manejar las formas para la escena actual, ésta se encuentra lista para dibujar las diversas formas. Ya organizada la escena, la aplicación acomoda a OpenGL en un estado apropiado para llevar a cabo el trazo de los volúmenes.

4.3.4. La librería GLUI [30].

Muchas aplicaciones pueden ser escritas usando solamente GLUT. Sin embargo, cuando se requieren más aspectos de interacción además de los métodos de entrada estándar para el ratón y el teclado se requiere mucho esfuerzo para lograrlo sólo con las interfaces básicas que proporciona OpenGL y GLUT lo cual puede resultar en un trabajo engorroso. Precisamente, para evitar esto, se utiliza la librería GLUI, pues proporciona elementos estándar para una interfaz gráfica de usuario de fáciles manejo y comprensión para el programador.

GLUI es una librería que provee una interfaz de usuario basada en GLUT para el lenguaje de programación de alto nivel C++; además, pone a disposición del programador una interfaz simple para el control de ventanas, eventos de ratón, teclado y otros dispositivos de entrada. GLUI proporciona diversos controles como botones, casillas de verificación, radio-botones, listados y espacios de texto para aplicaciones que empleen OpenGL.

Es importante resaltar que GLUI es independiente del sistema de ventanas y resulta muy útil para la elaboración de aplicaciones multiplataforma, aspecto gracias al cual fue considerada está librería para su uso en el trabajo que aquí se presenta. Un detalle significativo a ser mencionado es que esta librería posee la ventaja de ser de código abierto.

Para el uso de GLUI se recomienda crear una función específica que maneje los eventos producidos por la interacción con los controles GLUI los cuales son distinguidos por un número de identidad. De este modo, al momento de activar un control, el método indicado llama a dicha función para que procese el evento ubicándolo por su número de identificación y así realizar las tareas que el programador indique.

La versión usada en este proyecto es la 2.2.

4.4. Compilación y ejecución de la aplicación

4.4.1. Compilación

Uno de los requerimientos de esta utilidad es que tenga la capacidad de ser multiplataforma. Por tal razón, se ha desarrollado de tal manera que pueda ejecutarse en los sistemas operativos Irix (basado en Unix) y Windows.

Específicamente, esta aplicación se ha desarrollado en dos equipos con las siguientes características:

- Equipo 1:
  - Sistema operativo Windows XP Professional SP2
  - Procesadores Intel Xeon a 2.40 GHz
  - 2GB de RAM
  - Tarjeta de video NVIDIA Quadro FX 500 (128 MB).
  - Microsoft Visual Studio 2005 versión 8.0.50727.42
  - Microsoft .NET Framework versión 2.0.50727

- Equipo 2:
    - Sistema operativo Windows XP Professional SP2
    - Procesadores Intel Xeon 2.80 GHz
    - 1GB de RAM
    - Tarjeta de video NVIDIA Quadro FX 1400 (128 MB)
    - Microsoft Development Environment 2002 versión 7.0.9466
    - Microsoft .NET framework versión 1.0.3705

Por otro lado, también se ha probado en iXtli (el Observatorio de Visualización de la UNAM), cuyas características principales son:

- SGI Onyx 350 con el sistema operativo IRIX® 6.5 (64 bits)
- Procesador MIPS R16000 64-bit a 700 Mhz con 4MB caché
- 24 GB de memoria SDRAM.
- Número de procesadores: 12
- Número de pipes gráficos: 3 con 2 RasterManager InfiniteReality4 cada uno
- Compilador de C (perteneciente a la familia MIPSpro, versión 7.4.3m).

Las líneas usadas para la compilación y ligado bajo Windows, suponiendo que los archivos del código fuente se localizan en C:\visualizador\ y las librerías en C:\librerias\, son:

```
C:\visualizador>cl.exe /c visualizador.cpp /I "C:\librerias" /D VISUALCXX

C:\visualizador>link.exe /out:visualizador.exe visualizador.obj
/NODEFAULTLIB:LIBCMTD /libpath:"C:\librerias" vzLoaders.lib vz.lib
glui32.lib opengl32.lib glut32.lib devil.lib ilu.lib
```

En el caso de Irix, y suponiendo que las librerías usadas están en un nivel inferior (pero dentro del mismo directorio), se usan estas líneas:

```
/usr/bin/CC -I. -I/src/lib/ -I./Devil/DevIL-1.6.8/include/ -nostdinc -
I/usr/include/CC -I/usr/include –I./glui_v2_2/ -mips3 -n32 -O3 -
OPT:Olimit=0 -OPT:IEEE_arithmetic=1 -OPT:roundoff=0 -TENV:X=1 -
OPT:wrap_around_unsafe_opt=off -DEBUG:optimize_space=on -OPT:space=on -
CG:unique_exit=on -OPT:unroll_times=0 -MDupdate Makedepend -c
visualizador.cpp

/usr/bin/CC -o visualizador visualizador.o -mips4 -n32 -quickstart_info -
nostdlib -L/usr/lib32/mips3 -L/usr/lib32 -L/usr/lib32/internal -lvz -
lglut -lGL -lGLU -lX11 -lXmu -lIL -lILU -rpath /lib/irixn32 -
L/lib/irixn32 -rpath ./Devil/lib/ -L./Devil/lib/ -L./glui_v2_2/lib -Wl,-
woff,31 -lvz -lglui -lglut -lGL -lGLU -lm -lX11 -lXmu -lIL -lILU
```

4.4.2. Ejecución

Para la ejecución de la aplicación se requiere tener una licencia de *Volumizer* además de las librerías DevIL, GLUT y GLUI totalmente compiladas. La compatibilidad que tenga Volumizer con el hardware de la máquina en donde se quiera ejecutar la aplicación es de suma importancia y determinará si es posible ejecutarla o no.

De manera resumida, la aplicación se llama desde la línea de comandos de un *shell* y recibe como parámetros de entrada argumentos que el usuario debe escribir junto con la invocación de la aplicación. Existen cinco maneras de invocar la aplicación de visualización que dependen de la forma en que el volumen será cargado:

1. Por secuencias de imágenes en formato comercial (JPEG, GIF, PNG, etc).
2. Por secuencias de imágenes en formato RAW de ocho bits.
3. Por secuencias de imágenes en formato RAW de doce bits.
4. Por archivo de volumen TIF.
5. Por archivo de volumen RAW.

Dependiendo de lo que se requiera cargar se deben introducir los parámetros correspondientes, por ejemplo, para la carga de un volumen constituido por secuencias de imágenes RAW de doce bits, la invocación del programa en un *shell* de MS-DOS sería la siguiente:

```
>visualizador 256 256 72 C:\Secuencias\cfzesc\cfzesc dat -d
```

Siendo los números 256, 256, 72 las dimensiones del volumen a cargar correspondiendo con el ancho horizontal y vertical de las imágenes y el número de cuadros o *"frames"* que constituyen el volumen, en este caso son 72 cuadros. Posteriormente, se coloca la ruta (absoluta o relativa) donde se localizan los archivos junto con el nombre genérico de los archivos (forzosamente necesario); en este caso, el nombre genérico es *cfzesc*. A continuación se coloca la extensión de los archivos (*dat* para este ejemplo) y dado que se trata de imágenes de doce bits se coloca la bandera *–d*. Es decir, se cargan los archivos cfzesc1.dat, cfzesc2.dat,…, cfzesc72.dat que conforman el volumen. Siguiendo los lineamientos anteriores se produce la carga del volumen:

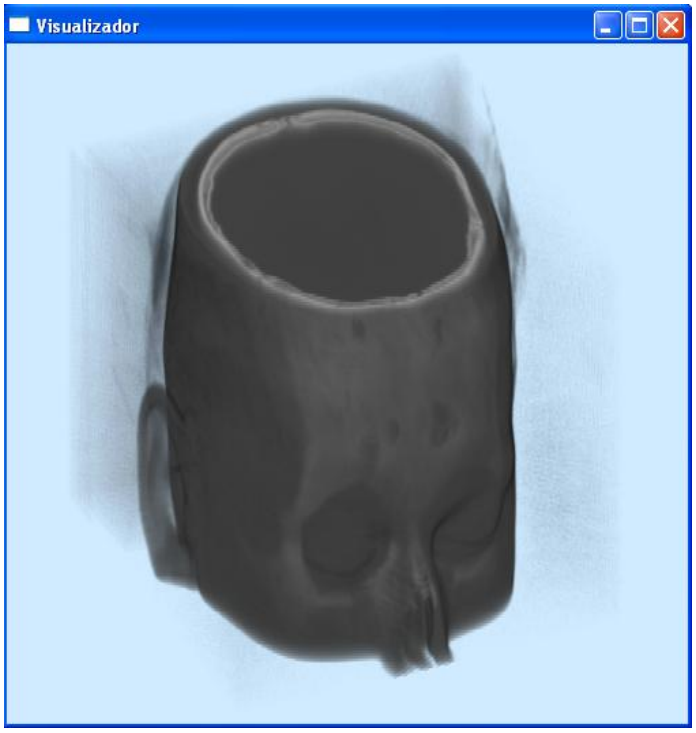


Figura 4.16. Volumen cargado mediante imágenes de 12 bits

En caso de que no se emplee ningún método de carga reconocido por la aplicación, ésta muestra un mensaje indicando al usuario la manera correcta de invocar la aplicación:

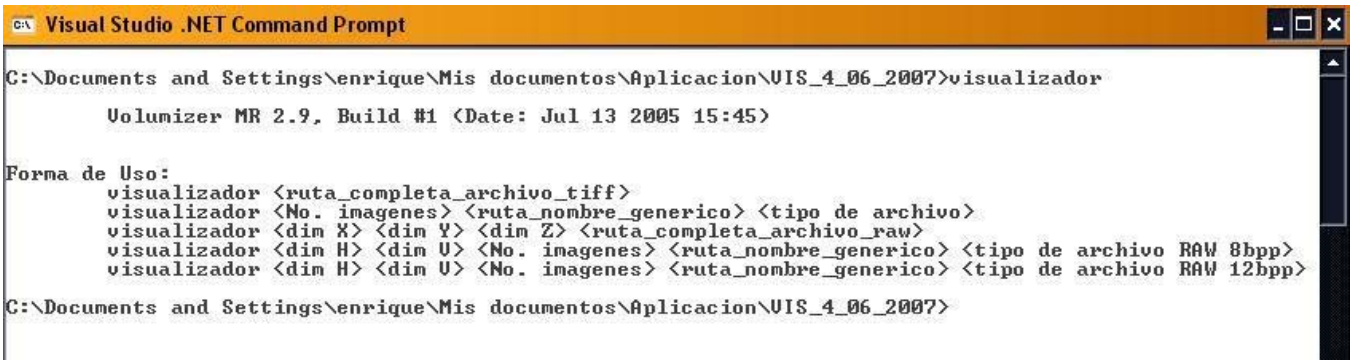


Figura 4.17. Mensaje arrojado por la aplicación

La interfaz gráfica de usuario elaborada con GLUI tiene el siguiente aspecto:

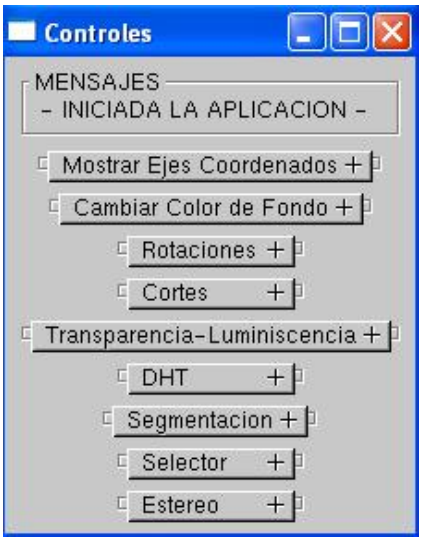


Figura 4.18. GUI paneles contraídos

Nótese que cada panel contraído muestra una función específica de la aplicación, al hacer clic sobre un panel contraído éste se expande mostrando sus diversos controles interactivos:

Figura 4.19. GUI panel expandido

4.5. Funciones implementadas.

La aplicación hace una conjunción de herramientas prefabricadas de software con métodos de código original, un diagrama de dicha conjunción es mostrado en la figura 4.20 el cual presenta la estructuración a base de niveles que tiene a la aplicación.

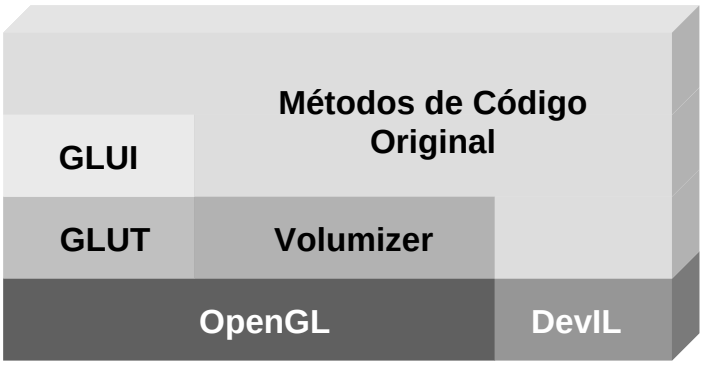


Figura 4.20. Diagrama de la conjunción de herramientas

Uno de los principales objetivos de la aplicación consiste en la segmentación de los volúmenes que se visualizan, sin embargo, hay que resaltar que hay más funcionalidades presentes en la aplicación, las cuales deben mencionarse con miras a complementar el cómo se desarrollaron por completo todos los módulos de la aplicación.

En esta sección se muestran las funcionalidades importantes que se implementaron, esto con el fin de ofrecer una mejor descripción de dichas funcionalidades.

4.5.1. Cargadores de Volumen.

Como ya se ha mencionado, existen cinco maneras de cargar un volumen en la aplicación, y en esta sección se detalla el funcionamiento del módulo que hace posible tal tarea.

Para la carga de volúmenes que se encuentran descritos por secuencias de imágenes en formatos comerciales se hace uso de los métodos de DevIL. Para la carga de archivos de volumen (un solo archivo se usa para almacenar el volumen) se hace uso de cargadores que la distribución de Volumizer trae consigo.

Sin embrago, los métodos de carga de imágenes RAW de ocho y doce bits fueron implementados por completo. Los pasos que se usan para cargar este tipo de archivos son los siguientes:

1.- Obtener *Nombre_Genérico*, *Extensión* y *Total_de_Imágenes*.
2.- Reservar Memoria.
3.- Desde $k = 1$ hasta *Total_de_Imágenes*.
    Crear nombre de archivo usando *Nombre_Genérico, k y Extensión*.
    Apuntar al archivo.
    Leer el archivo.
    Estandarizar los datos leídos en un formato interno.

Almacenar datos estandarizados en memoria reservada.

Cerrar el archivo

Fin de Ciclo.

4.- Establecer dimensiones del volumen.

Para el cargador de imágenes RAW de doce bits se sigue el mismo procedimiento solo que, en la sección de lectura del archivo se hacen los siguientes pasos adicionales:

3.- Desde *k* =1 hasta *Total_de_Imágenes*.

Crear nombre de archivo usando *Nombre_Genérico, k y Extensión*.

Apuntar al archivo.

Repetir

Leer *dato1* de 8 bits

Leer *dato2* de 8 bits

Combinar *dato1* y *dato2* en *estructura* de 16 bits

Normalizar rango de *estructura* dentro de 4096 combinaciones (12 bits)

Almacenar *estructura* normalizada en memoria reservada.

Hasta fin de archivo.

Cerrar el archivo

Fin de Ciclo.

Con la finalidad de tener congruencia en la formación del arreglo que contiene los datos de volumen se buscó estandarizar en cierta forma la manera en que los cargadores de datos leen las secuencias de imágenes. Sin embargo se tuvo presente el problema de que el método de DevIL usado para la lectura de imágenes en formatos comerciales leía los datos de manera distinta al resto de los métodos que se implementaron; este método lee la imagen por filas pero comienza por la última fila y termina en la primera:

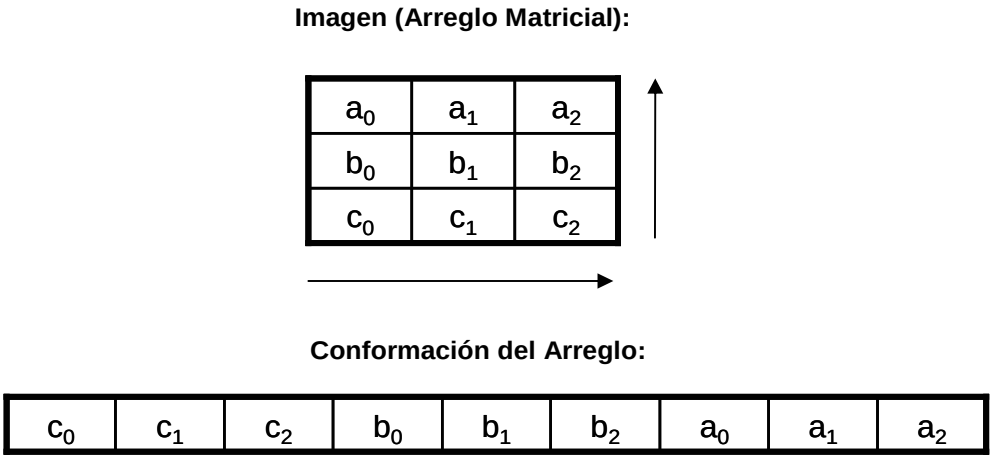


Figura 4.21. Lectura de imágenes por el método *ilLoadImage* de DevIL

Entonces, se elaboró una función de corrección para que el arreglo de datos creado por DevIL tuviera la forma del arreglo de datos creado por los cargadores de imágenes RAW implementados:

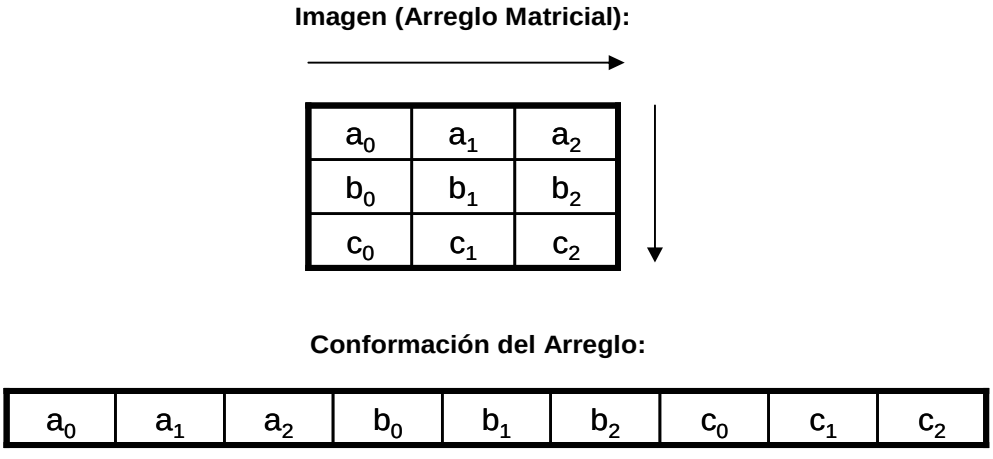


Figura 4.22. Lectura de imágenes por los métodos RAW 8b y RAW 12b

De esta manera la conformación de los datos del volumen quedó estandarizada.

4.5.2. Ejes Coordenados, Rotaciones y Color de Fondo.

Una manipulación importante del volumen consiste en tener pleno control de éste para poder visualizarlo desde diferentes ángulos.

La aplicación provee al usuario dos maneras diferentes de rotar el volumen; la primera, muy intuitiva, consiste en rotar el volumen colocando el ratón sobre el área de visualización y arrastrarlo hacia la dirección en que se desea rotar el volumen (esto se logra llamando un método de OpenGL que rota todo el sistema de coordenadas del área de visualización); la segunda forma consiste en rotar "localmente" el volumen, es decir, las transformaciones de rotación no se aplican a todo el sistema de coordenadas, solamente al volumen en sí.

Se muestran, además, líneas que representan una guía visual sobre la dirección de los ejes coordenados. Se tiene, en color rojo, al eje coordenado "X", en verde al eje coordenado "Y" y, finalmente, en amarillo al eje coordenado "Z". La visualización de estas referencias puede activarse o desactivarse a placer del usuario empleando el control correspondiente en la GUI.

Adicionalmente, por motivos de contraste visual, se implementó la funcionalidad de cambio de color de fondo. Como opciones inmediatas se colocaron tres colores de fondo predeterminados, pero también están disponibles otros controles que permiten al usuario establecer un color cualquiera de fondo por medio de combinaciones de los colores básicos del sistema RGB (rojo, verde y azul).

4.5.3. Cortes al volumen.

Se programaron dos maneras de efectuar cortes al volumen que se está observando: cortes referidos a los ejes coordenados y el plano de corte.

Los cortes referidos a los ejes coordenados funcionan cambiando los parámetros de tamaño y *offset* que definen el volumen de datos que genera *Volumizer*. En otras palabras, se implementó que el volumen quedara dentro de un cubo de visualización y así poder efectuar cortes planos paralelos a las seis caras del cubo. Por otro lado, al girar el volumen usando los controles de la GUI puede aplicarse el plano de corte, el cual irá cortando el cubo de datos con un plano perpendicular a la vista del observador, con esta herramienta pueden efectuarse cortes oblicuos.

4.5.4. El Selector de Rango.

Una herramienta implementada y muy útil para la visualización de características de los volúmenes basados en imágenes médicas es el "Selector de Rango". Esta opción consiste en agrandar una región del histograma correspondiente al volumen que se está observando con el propósito de que puedan observarse más detalles existentes en el volumen. El proceso que sigue este mecanismo es similar a la especificación de histograma, sólo que en este caso se emplea un mapeo para seleccionar el área del histograma que será ampliada.

Diseñado específicamente para un mejor análisis visual de los volúmenes de doce bits, el rango de operación de la ventana va desde un valor de 0 hasta un valor de 4095 puntos. Sin embrago esta funcionalidad también es útil para volúmenes de 8 bits. Para poder operar sin perder los datos originales del volumen, se efectúa una copia en memoria de los datos originales. Básicamente, el algoritmo empleado es el siguiente:

1.- Crear copia de datos originales de volumen. (*Copia* ← *Original*)
2.- Obtener *Total_Datos_Volumen*
3.- Leer valores de ventana: *Centro* y *Ancho*.
4.- Repetir
    Si *Copia*[*apuntador*] no está dentro de *Ventana*[*Centro, Ancho*].
        Colocar *Copia*[*apuntador*] ← 0
    Si no

Efectuar *Mapeo*(*Copia*[*apuntador*])

Mientras que (*apuntador* < *Total_Datos_Volumen*)

5.- Cargar nuevos datos de volumen.

6.- Desplegar Volumen.

4.5.5. Visualización en Estéreo.

Con la finalidad de utilizar la mayor parte de las capacidades que proporciona el observatorio iXtli se implementó la funcionalidad de desplegar en estéreo el volumen que la aplicación dibuja en pantalla. De esta manera, el volumen puede visualizarse de manera inmersiva en un ambiente virtual.

4.6. Resultados Finales.

Ya que se ha completado el desarrollo de la aplicación, el siguiente paso es probar todas sus funciones y, posteriormente, observar y evaluar los resultados. En esta sección se muestra el funcionamiento de la aplicación y el resultado de trabajar con volúmenes generados por secuencias de imágenes médicas y de secuencias de movimiento.

En primer lugar, se muestra el uso de la aplicación: desde la carga de datos hasta la manipulación de ellos haciendo uso de las funciones incluidas en la interfaz gráfica de usuario. En esta primera etapa, no se pretende evaluar resultados de segmentación, sino evaluar resultados concernientes al manejo del programa, por ejemplo el uso y funcionamiento de la interfaz; por otra parte, también se realiza una valoración de lo que se muestra en la ventana de resultados, esto es, determinar, desde un punto de vista de percepción visual del usuario, si lo que se observa es claro y coherente con los datos de la interfaz gráfica.

Después de valorar el funcionamiento, lo siguiente es evaluar los resultados de las técnicas de segmentación implementadas. Para llevar a cabo esta tarea se hace uso de dos herramientas estadísticas: la matriz de confusión y el índice *kappa* (κ).

4.6.1. Ejecución del programa: resultados y evaluaciones

Enseguida se presenta, de forma concreta, el funcionamiento de cada opción disponible en la aplicación. El orden en el que se presentan dichas funciones corresponde al orden en el que están colocados en la ventana de la interfaz (fig 4.23).

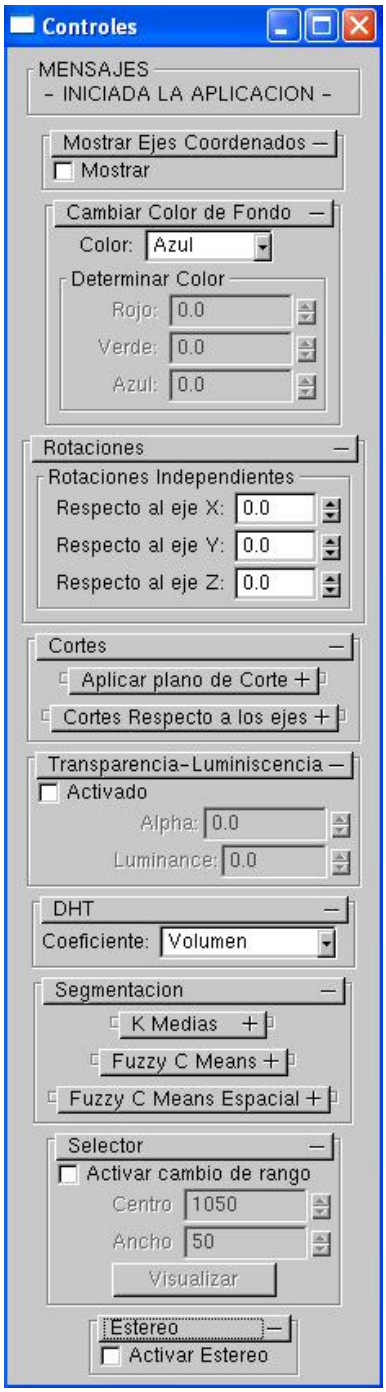


Figura 4.23. Controles de la GUI.

Por claridad, y con el objetivo de poder observar el uso de las funciones, se emplea un conjunto de imágenes que representan una secuencia de movimiento y con el que se obtiene un volumen básico. Para este caso, la línea de ejecución es:

```
C:\>visualizador 50 c:\imagenes\cono\figura bmp
```

Y el resultado es

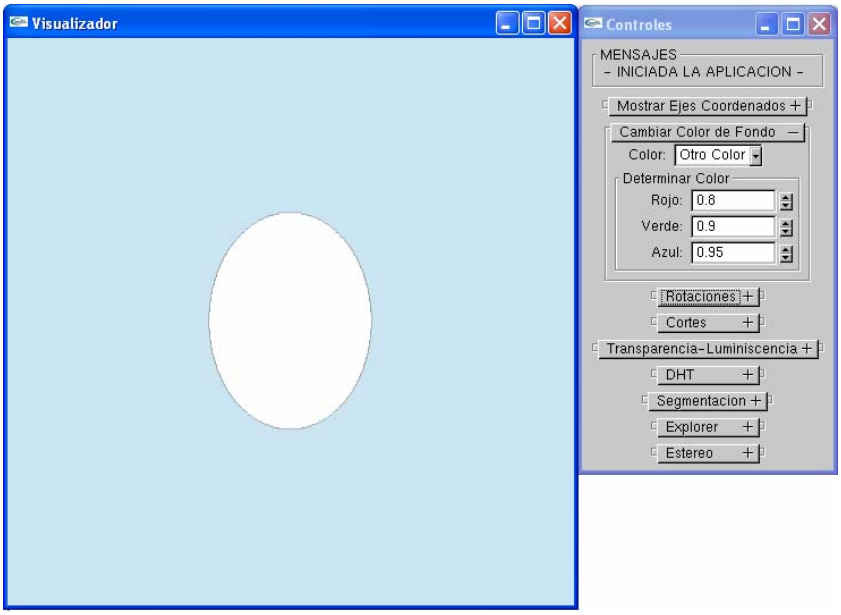


Figura 4.24. Visualización de volumen

< *Mostrar ejes coordenados* >

Al abrir la lista se pueden ver la opción que permite ver los ejes coordenados en la ventana de visualización. Con esta utilidad es posible identificar la postura del volumen por efecto de rotaciones.

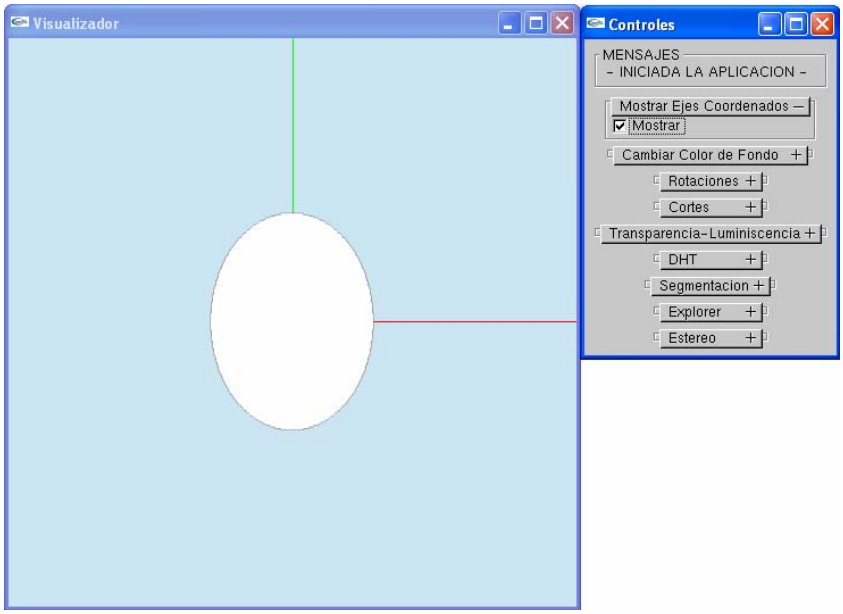


Figura 4.25. Activando los ejes coordenados

< *Cambiar color de fondo* >

Mediante esta operación el usuario puede cambiar el fondo de la ventana del visualizador para observar mejor el volumen de datos, por ejemplo para aumentar el contraste y/o evitar que se "pierda" a causa del color de fondo.

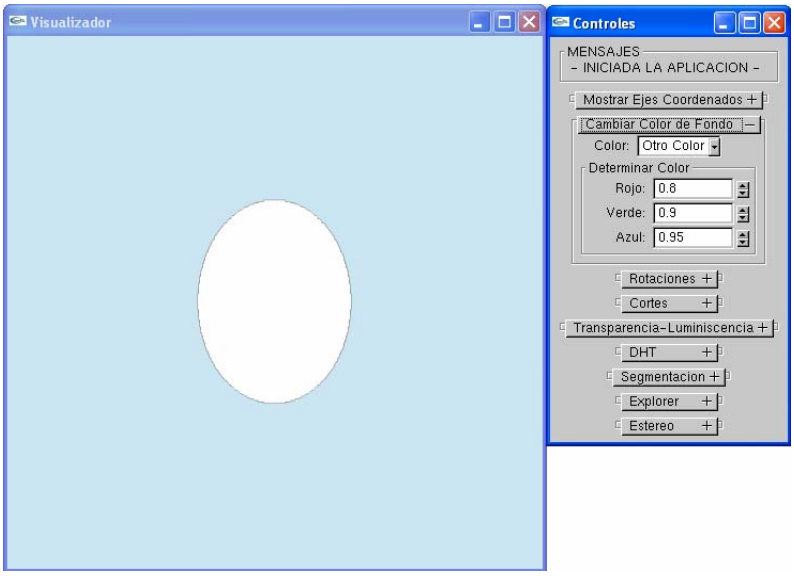


Figura 4.26. Modificando el color de fondo

< *Rotaciones* >

Cuando el usuario lo requiera, podrá rotar el volumen respecto a los ejes coordenados. La amplitud del giro se mide en grados [°] y su intervalo es [0, 360].

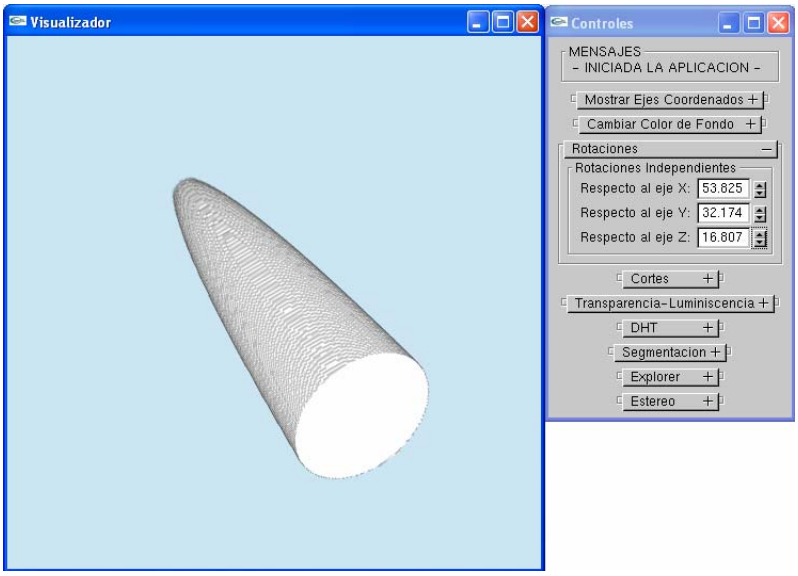


Figura 4.27. Aplicando rotaciones al volumen

< *Cortes* >

Los volúmenes pueden ser objeto de dos modos de cortes: i) aplicando un plano de corte y ii) respecto a los ejes coordenados. En el primer caso, un plano paralelo al plano de la ventana recorta el volumen; para obtener un corte específico, antes de aplicar esta opción, se debe girar el volumen. En la segunda modalidad, se pueden realizar cortes en las tres direcciones (X, Y, Z) y en cada una en ambos sentidos.

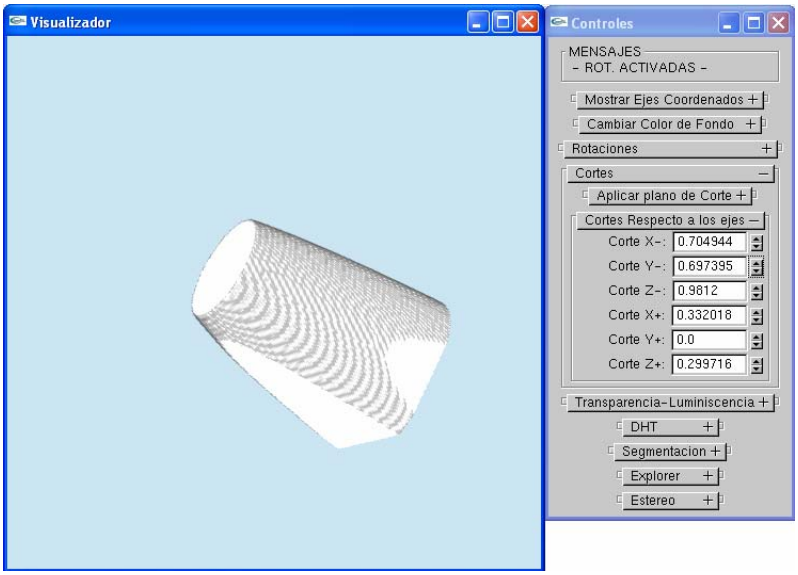


Figura 4.28. Aplicando cortes al volumen

< *Transparencia-luminiscencia* >

Con esta opción es posible modificar el mapa de colores (256 niveles de gris) empleando una *lookup table*. La opción 'Nivel de Gris' indica el valor de gris en el que se centra una ventana de 40 niveles de gris para modificar su transparencia; los valores para esta opción están en el intervalo [0, 1], donde 0 corresponde a negro (nivel 0) y 1 a blanco (nivel 255); los elementos con valores fuera de este rango no son visibles mientras esté activada esta opción. La opción 'Transparencia' trabaja sobre los elementos cuyos valores de gris están dentro de la ventana de 'Nivel de Gris' y toma valores del rango [0, 1], donde 0 es totalmente transparente (no visible) y 1 totalmente sólido (visible).

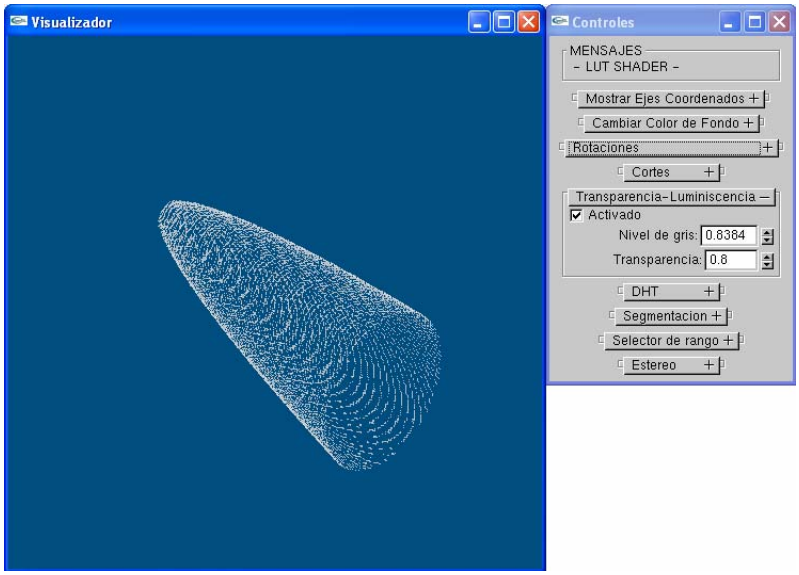


Figura 4.29. Modificando el mapa de colores

< Transformada de Hermite discreta>

A cualquier volumen se le puede calcular los coeficientes de la transformada de Hermite utilizando el método rápido para esta transformada. Los coeficientes 000, 001 (‘Coeficiente X’), 010 (‘Coeficiente Y’) , 100 (‘Coeficiente Z’), así como el gradiente (‘Gradiente’), están disponibles en la lista de esta opción.

Después de calcular los coeficientes (procedimiento que se realiza en tiempo real), se carga el coeficiente en la ventana de visualización. Los valores negativos de la transformada están asociados a niveles de gris bajos, por lo que el valor más negativo se observa con tonalidad negra; por otra parte, los valores más positivos se mapean a niveles altos de gris, lo cual resulta en la presencia de elementos blancos. Los valores ubicados entre el valor más negativo y el más positivo se mapean a niveles intermedios de gris.

Además, para mejorar los resultados visuales, la transparencia también varía dependiendo el valor de la transformada. Para los valores extremos positivo y negativo, la transparencia es 1 (sólido) mientras que para el valor intermedio (cero) la transparencia es 0 (totalmente transparente). De esta manera, se evita la presencia de grises correspondientes a datos cuya transformada fue un valor constante.

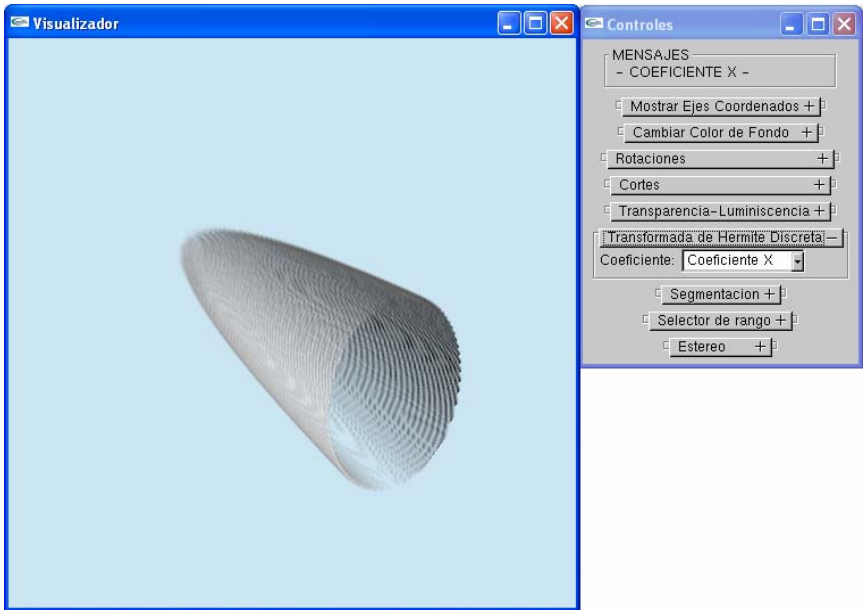


Figura 4.30. Coeficiente X

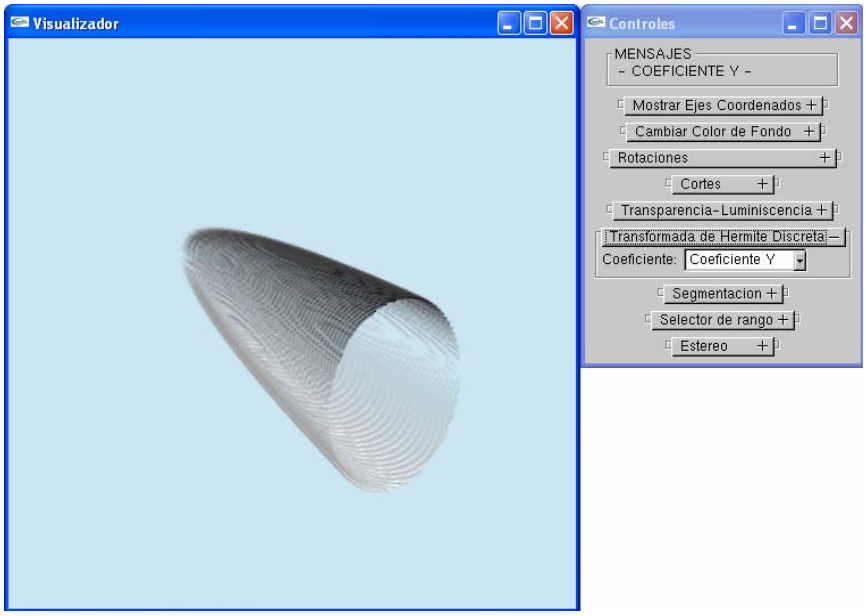


Figura 4.31. Coeficiente Y

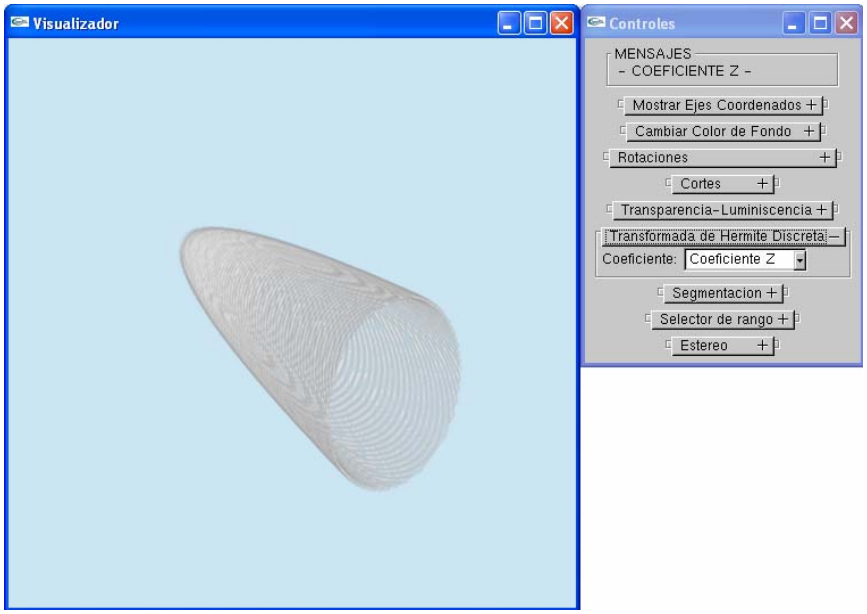


Figura 4.32. Coeficiente Z

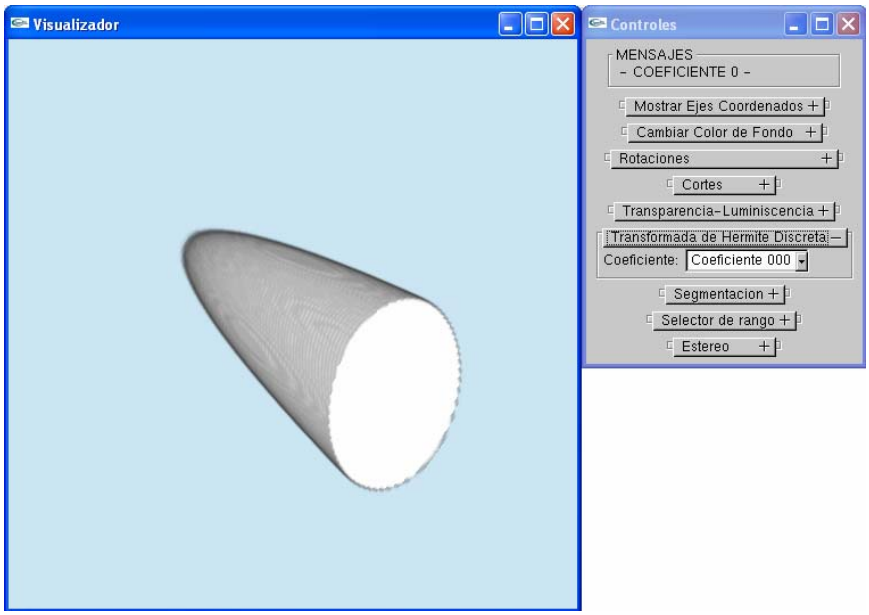


Figura 4.33. Coeficiente 000

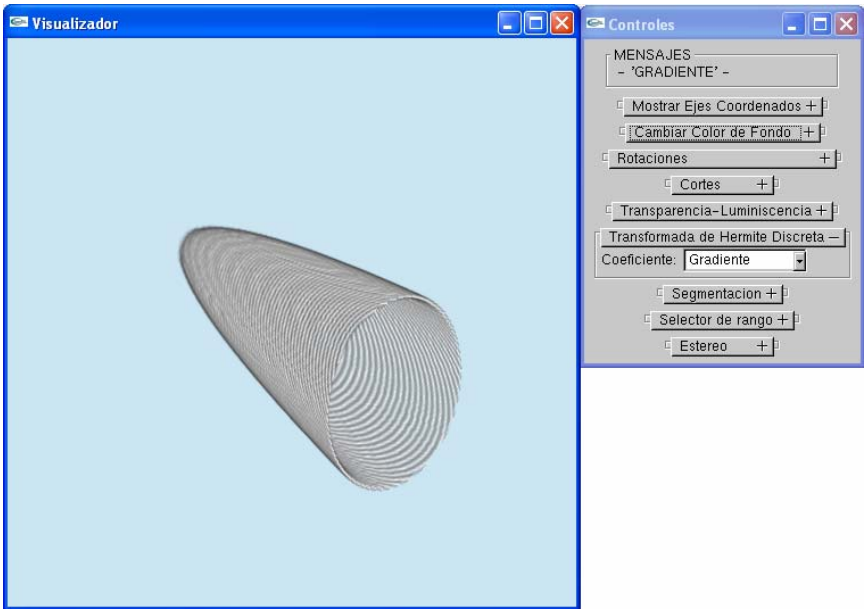


Figura 4.34. Gradiente

< *Selector de rango*>

Los formatos comunes de imágenes emplean 24 bits para almacenar cada valor de píxel. Como parte del procesamiento, esta aplicación promedia los valores de los tres canales (R, G y B) para obtener un nivel de gris comprendido de 0 a 255, lo cual puede representarse con 8 bits.

Sin embargo, algunos formatos de imágenes especializados almacenan los datos empleando más de 8 bits para almacenar intensidades de grises. Estos formatos de imágenes permiten almacenar mayor variedad de tonos de gris, lo cual evita que los datos originales sean comprimidos (como sucede con algunos formatos comunes), o que se tenga que decidir por guardar (con todo detalle) cierto rango al momento de generar las imágenes y comprimir (u omitir) lo que no esté dentro de dicho rango.

En esta aplicación se incluye el módulo que permite leer secuencias de imágenes de 12 bits (4096 niveles de gris) y se debe indicar al momento de ejecutar la aplicación con el modificador *–d*. Una vez que se hayan leído todos los archivos de las imágenes, el programa mostrará en la ventana de visualización el volumen empleando solamente 256

niveles de gris, ya que el observador humano difícilmente podría identificar los 4096 niveles.

El beneficio de contar con imágenes de 12 bits se percibe al seleccionar un rango de los 4096 niveles y mostrarlo usando 256. Precisamente ésta es la utilidad del selector.

El selector consiste en una ventana centrada en un nivel *c* de gris y con un ancho 2*a* de niveles y realiza un mapeo entre histogramas, esto es

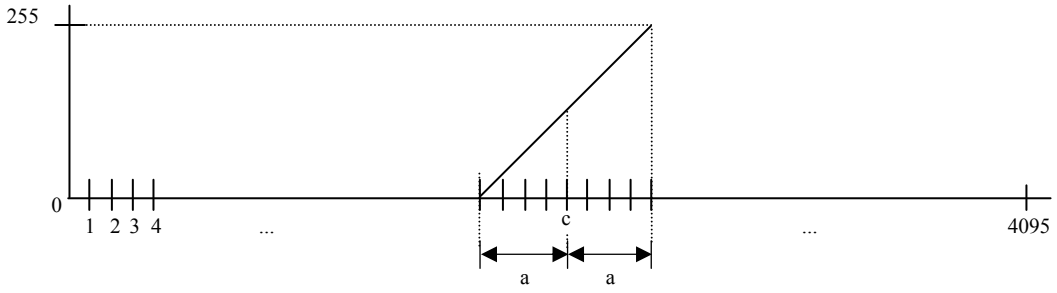


Figura 4.35. Selector

Como se puede apreciar en la figura anterior (4.35), el selector se puede mover por los 4096 niveles de gris de las imágenes y también se puede ampliar o reducir el ancho de la ventana. Lo anterior permite analizar determinadas secciones del histograma del volumen.

Por ejemplo, un volumen que se genera con una secuencia de imágenes médicas de 12 bits se muestra en la figura 4.36. A este volumen se le han realizado rotaciones y cortes con la intención de dejar visible parte de la masa encefálica y del tejido óseo y así percibir de mejor forma los efectos que se producen con el selector activado.

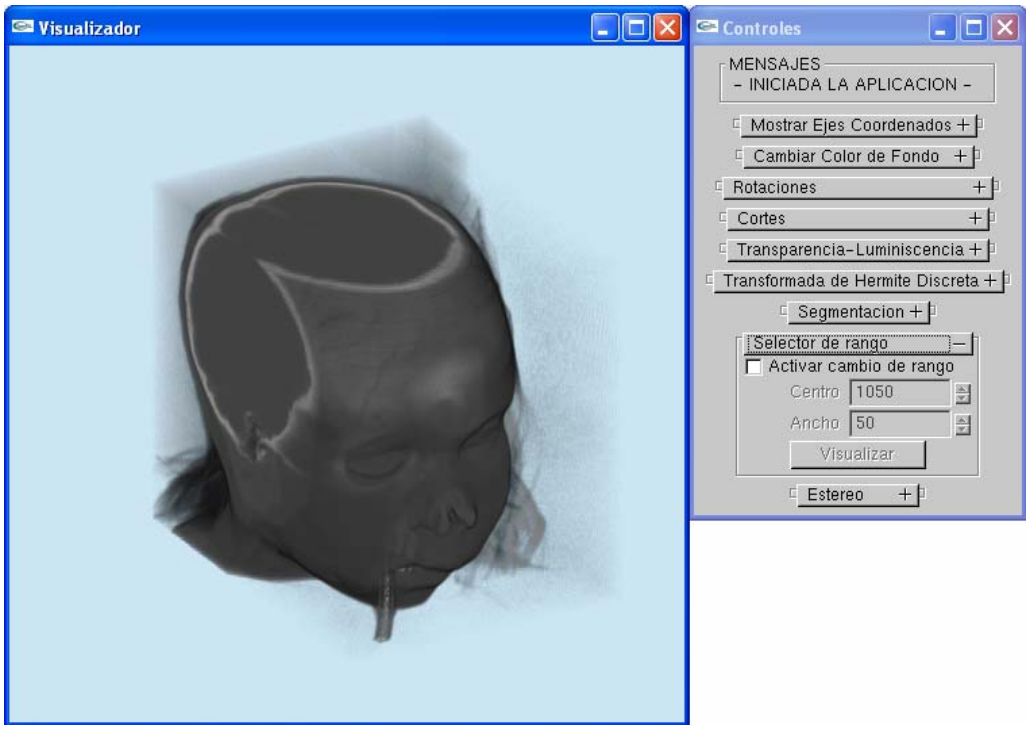


Figura 4.36. Volumen de 12 bits sin aplicar selector

En la figura 4.36 el selector aún no ha sido activado y lo que se observa es el volumen con 256 niveles de gris. Activando el selector, por default, se obtiene una ventana centrada en el nivel 1050 y de ancho tiene 100 (en la interfaz de usuario se indica el valor 'a' que se muestra en la figura 4.35). El volumen resultante se presenta a continuación

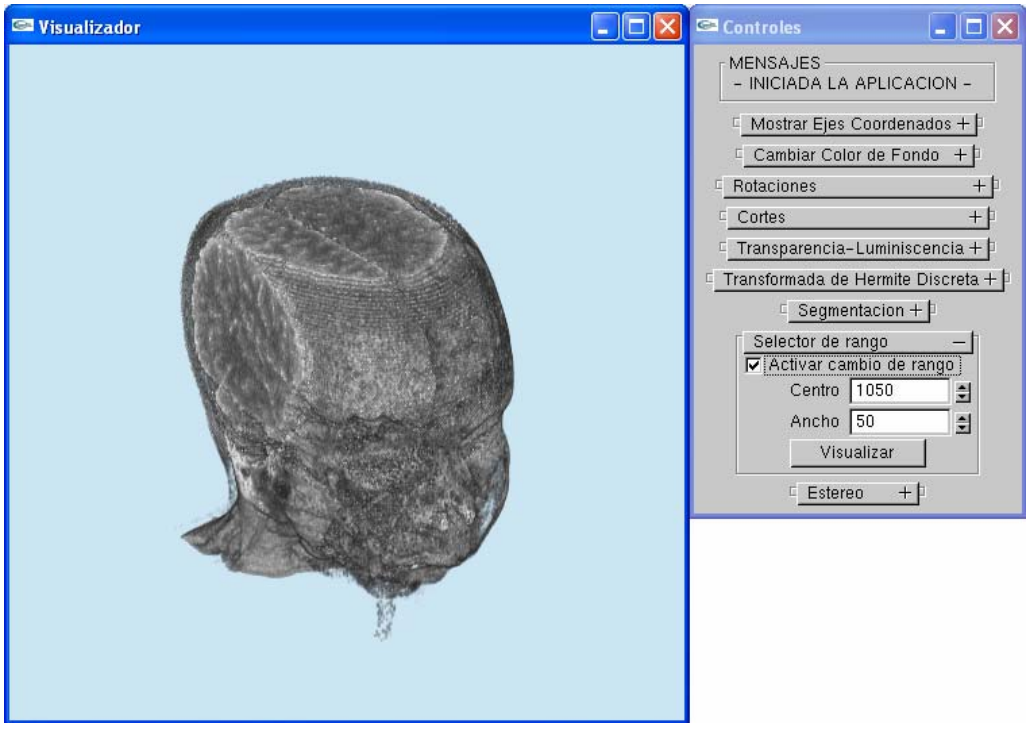


Figura 4.37. Volumen de 12 bits con el selector activado para ver masa gris

En la figura 4.37 se ha seleccionado una ventana que permite ver con detalle la masa gris (cerebro), lo cual no era posible en el volumen original (figura 4.36). Eligiendo los valores de 2600 y 1500 para el centro y ancho respectivamente, lo que se observa es el tejido óseo.

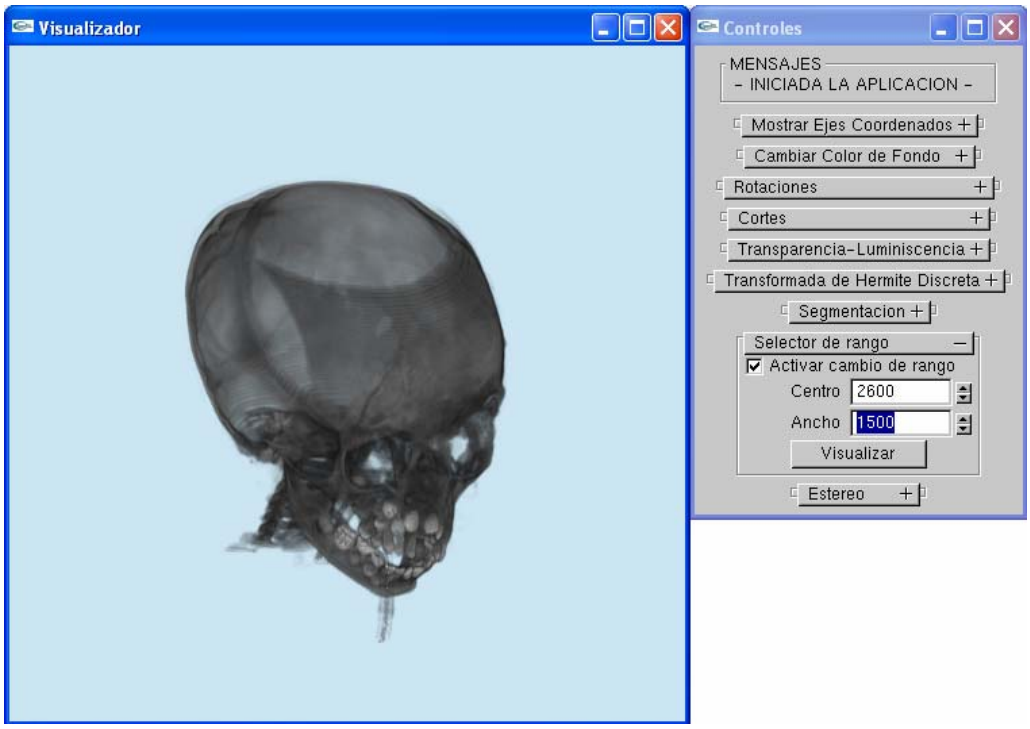


Figura 4.38. Volumen de 12 bits con el selector activado para ver masa ósea

4.6.2. Técnicas de segmentación: evaluación de resultados

El otro aspecto a evaluar de la aplicación final es el correspondiente a la segmentación. Para valorar las técnicas usadas en este trabajo se recurre a la matriz de confusión y al índice *kappa* (κ).

4.6.2.1. Matriz de confusión

La matriz de confusión [31] es una herramienta estadística en la cual se muestran las clasificaciones actual y la que se predice. Una matriz de confusión es de tamaño *l* x *l*, donde *l* es el número de valores distintos de etiquetas. Por ejemplo, una matriz de confusión de tamaño 2x2 (*l* = 2, es decir para dos clases), es

		Predicción	
		Negativo	Positivo
Actual	Negativo	*a*	*b*
	Positivo	*c*	*d*

Los valores en las celdas de la matriz tienen el siguiente significado [32]:

- *a* es el número de predicciones correctas de que una instancia es negativa
- *b* es el número de predicciones incorrectas de que una instancia es positiva
- *c* el número de predicciones incorrectas de que una instancia es negativa
- *d* es el número de predicciones correctas de que una instancia es positiva

Los siguientes términos están definidos para una matriz de confusión de 2 x 2 :

- Exactitud (a + d) / (a + b + c +d)
- Relación positivo verdadero (Sensibilidad) d / (c + d)
- Relación negativo verdadero (Especificidad) a / (a + b)
- Precisión d / (d + b)
- Relación positivo falso b / (a + b)
- Relación positivo verdadero c / (c + d)

4.6.2.2. Índice kappa (κ) [33]

El índice *kappa* (**κ**) es, básicamente, una medida que expresa el grado o magnitud de acuerdo o concordancia entre observadores. En otras palabras, se emplea para determinar qué tanto dos evaluadores que examinan los mismos datos concuerdan al asignar los datos a categorías [34].

Este índice se define de la siguiente manera:

$$\kappa = \frac{P_o - P_e}{1 - P_e} \tag{1}$$

donde P_o es la proporción de la concordancia observada y P_e es la proporción de concordancia esperada por puro azar.

Con el fin de mantener una nomenclatura consistente cuando se describe el grado de acuerdo asociado con el índice *kappa,* se han asignado etiquetas a rangos específicos del estadístico:

Índice kappa	**Grado de acuerdo**
< 0.0	Pobre (sin acuerdo)
0.0 – 0.2	Insignificante
0.21 – 0.4	Bajo
0.41 – 0.6	Moderado
0.61 – 0.8	Bueno
0.81 – 1.0	Casi perfecto

Tabla 4.1. Etiquetas Índice *kappa*

Para organizar los datos de acuerdos y desacuerdos entre dos observadores, la matriz de confusión es la estructura más adecuada.

Para obtener el valor del índice *kappa* a partir de la matriz se aplica la siguiente expresión [35]

$$\kappa = \frac{N\sum_{i=1}^{r} x_{ii} - \sum_{i=1}^{r}\left(x_{i+} * x_{+i}\right)}{N^2 - \sum_{i=1}^{r}\left(x_{i+} * x_{+i}\right)} \qquad (2)$$

donde:

r es el número de filas en la matriz,

x_{ii} es el número de observaciones en la fila i y columna i,

x_{i+}, x_{+i} es el total marginal para la fila i y columna i,

N es el número total de observaciones.

4.6.2.3. Resultados de pruebas empleando la matriz de confusión y el índice kappa

El primer paso en la evaluación consiste en elaborar una matriz de confusión empleando los resultados que la aplicación obtiene de una segmentación.

Como es necesario contar con datos "reales", se programó una función que genere aleatoriamente las coordenadas de pixeles del volumen; esta función se programó explícitamente para que genere 50 coordenadas aleatorias por cluster. Ya que se contó con las coordenadas fue posible realizar la clasificación manualmente, es decir, por un observador humano.

De manera particular, se analizó un volumen de tomografía computarizada (CT) orientada a la obtención de hueso. Obsérvese a continuación una imagen que conforma dicho volumen:

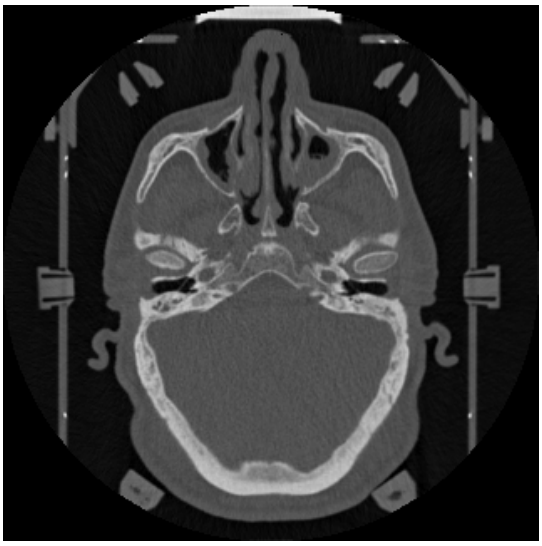


Figura 4.39. Imagen de tomografía computarizada

Claramente se puede observar que se podría segmentar esta imagen en tres clusters ya que predominan principalmente tres tipos de tonalidades: negro, grises y blanco. Es por eso que se decidió segmentarlo en tres clusters empleando el método *"C medias difuso con información espacial"*. El volumen original, basado en una secuencia de imágenes, luce de la siguiente manera:

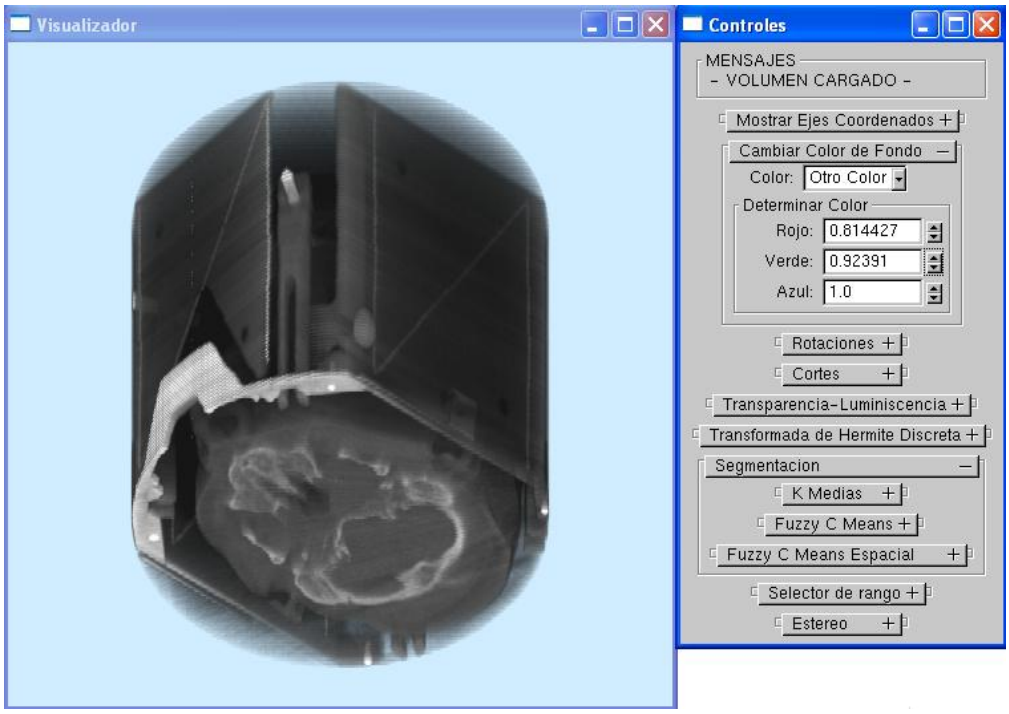


Figura 4.40. Volumen de Tomografía Computarizada.

Al segmentar el volumen como se mencionó en el párrafo anterior se obtuvieron los siguientes clusters, en los cuales se aprecia la separación de tres diferentes tonalidades, dejando claramente visible la estructura ósea:

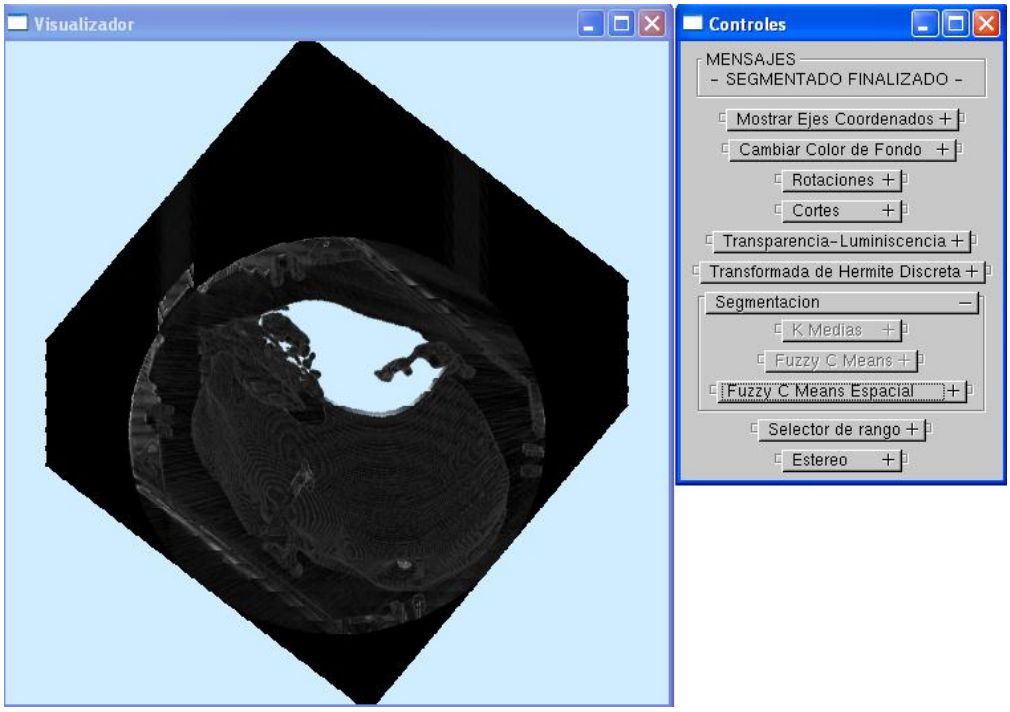


Figura 4.41. Cluster de tonalidades oscuras (aire).

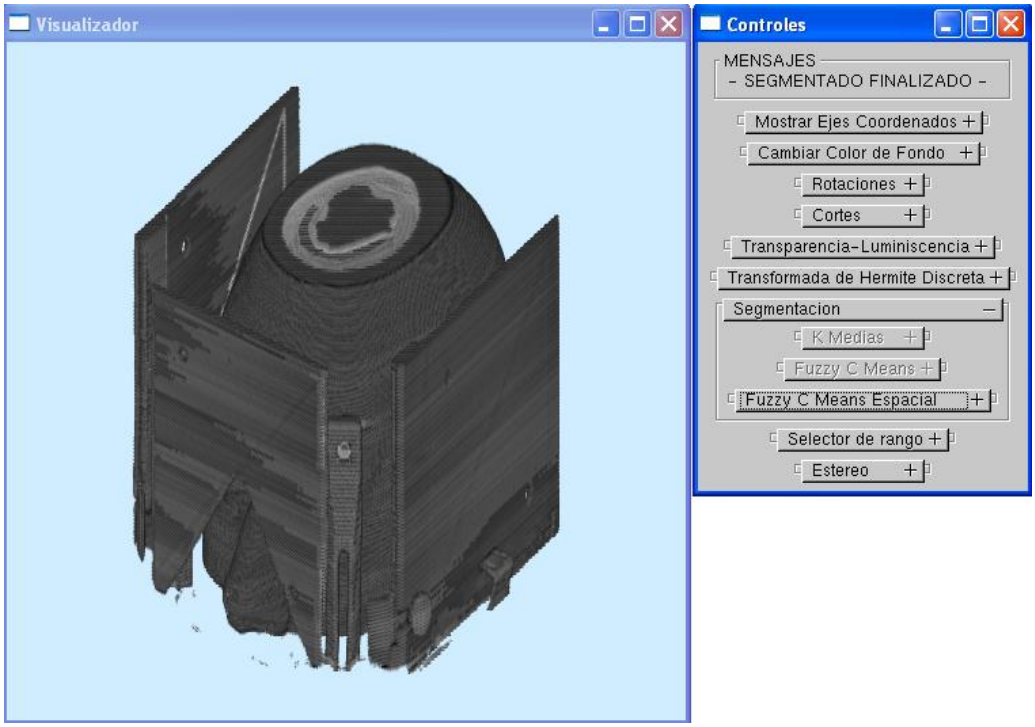


Figura 4.42. Cluster de tonalidades grises (piel y otros elementos anatómicos)

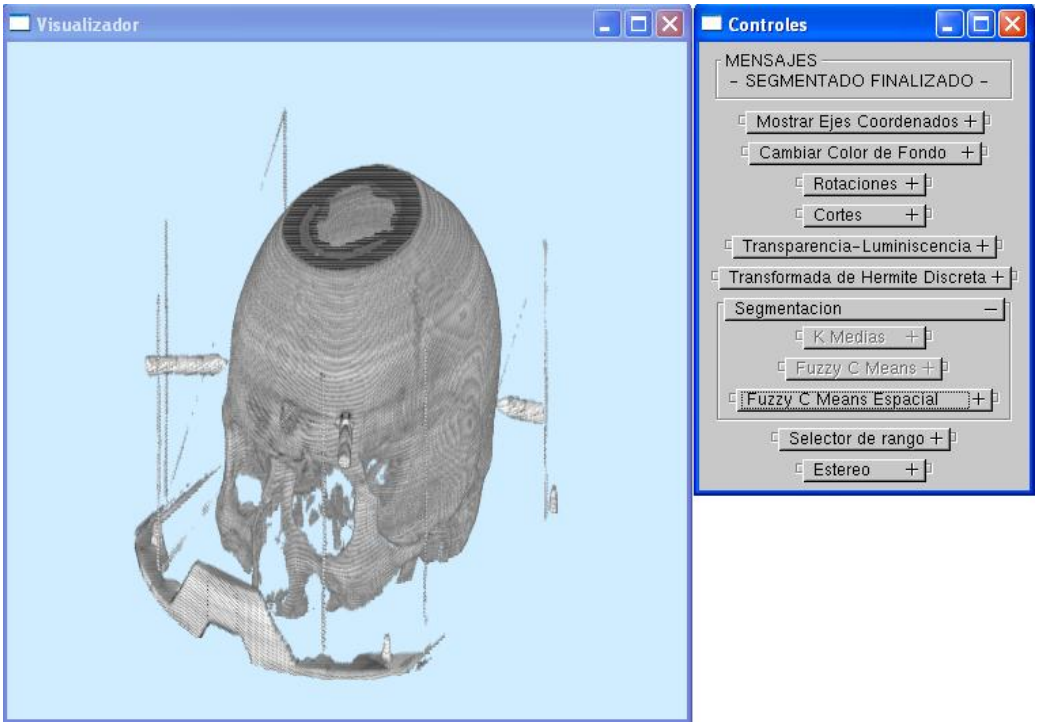


Figura 4.43. Cluster de tonalidades claras (hueso primordialmente).

Algunos elementos como las placas empleadas para la tomografía parecen tener la misma tonalidad que la piel, sin embrago esto se debe a la compresión de la cual se hacía mención en párrafos anteriores.

Para determinar que tan bien se están separando los elementos, se recurre al uso de la matriz de confusión y del índice kappa (**κ**); se muestra, en la siguiente tabla, a la matriz de confusión resultante para el análisis del presente volumen:

	Aire	*Piel*	*Hueso*	**Totales**
Aire	**45**	6	0	*51*
Piel	3	**44**	1	*48*
Hueso	2	0	**49**	*51*
Totales	*50*	*50*	*50*	**150**

Tabla 4.2. Matriz de Confusión para CT

Entonces, empleando la ecuación (2) que se mostró en la sección anterior, se obtuvo el valor del índice kappa:

$$\kappa = 0.88$$

Lo cual, indica que, para este caso, se obtuvo una muy buena clasificación de acuerdo con la tabla 4.1.

Sin embrago, este índice puede variar dependiendo del volumen al cual se le aplique la segmentación, considérese ahora un volumen de resonancia magnética (MRI) comprimido a ocho bits por píxel (escala de grises):

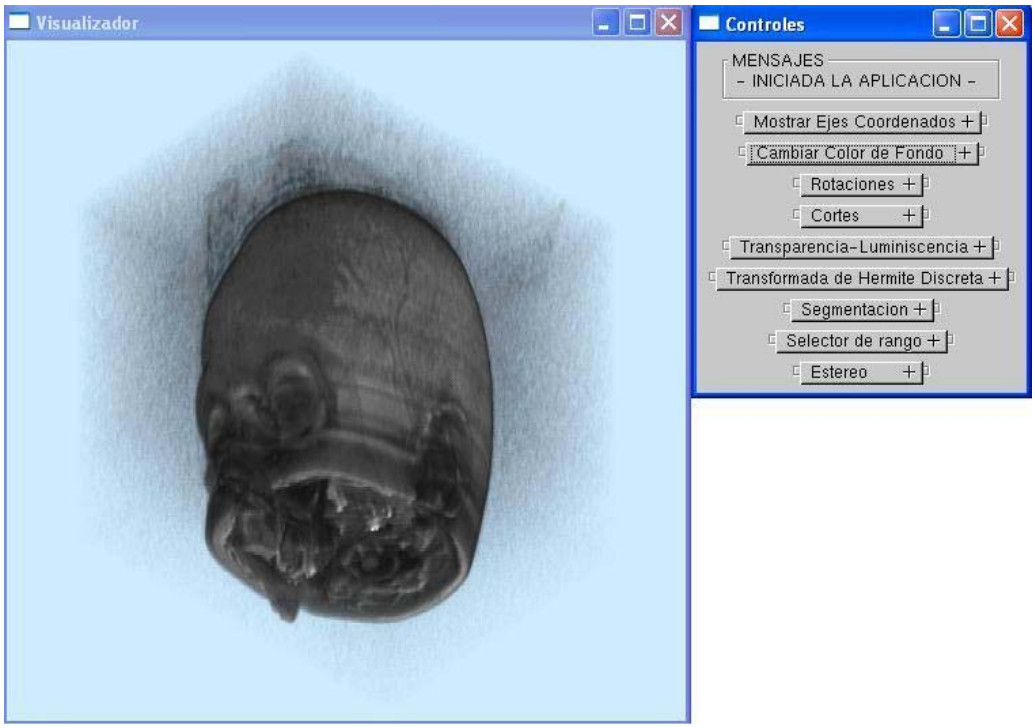


Figura 4.44. Volumen de Resonancia Magnética.

Inicialmente se buscó obtener el índice kappa para la evaluación de la segmentación de este volumen en cinco clases: aire, fluido cerebroespinal, materia gris, materia blanca y tejido óseo; sin embargo esto no pudo realizarse ya que se requiere de la presencia y evaluación de un especialista como puede ser un radiólogo o un neurólogo. Por otro lado, la evaluación de resultados empleando la matriz de confusión resulta muy complicada, pues

comprobar que la asignación de un píxel es correcta no es sencillo, sobre todo si el píxel se encuentra en las regiones de frontera de la materia gris y blanca, por ejemplo.

Para lograr un estudio completo sobre que tan bien se separan los clusters se requeriría de una especie de *"mapa de clases"* en el cual un experto haya delimitado concretamente las áreas en donde se localizan dichas clases en varias imágenes de resonancia magnética para de esta forma obtener un buen panorama sobre la delimitación de estas áreas y así reducir el posible margen de error que se pudiese tener.

Al no contar con este tipo de ayuda se recurrió a la búsqueda de apoyo en la literatura y en otro tipo de fuentes para determinar de una manera visual, empírica y heurística si los resultados para la segmentación de este volumen de resonancia magnética son coherentes. Por ejemplo, al visualizar los resultados obtenidos por Chuang [21] y Mohamed *et al*. [36] se pudo observar cierta semejanza anatómica con los resultados que se obtuvieron con la aplicación aquí desarrollada. Desde luego, se consultaron otras fuentes como el atlas de la materia blanca [37] (el cual es soportado por personalidades de diversas universidades e institutos), donde se puede apreciar de forma clara la localización de la materia blanca en diversos cortes axiales y transversales del cerebro humano; también se observó cierta concordancia con los resultados de otras aplicaciones y proyectos como lo es el proyecto BIOMORPH [38].

De esta manera, se presentarán a continuación los resultados de la segmentación para el volumen del cual se ha hecho mención. En primera instancia se presentará la segmentación de una sola imagen de resonancia magnética empleando el método FCME con parámetros m=2, p=1, q=3 y una ventana de 5x5x5 para así mostrar de manera clara las delimitaciones de las diferentes clases localizadas, como ya se mencionó se buscó segmentar en cinco clases: aire, fluido cerebroespinal, materia gris, materia blanca y tejido óseo; la imagen original que fue segmentada es la siguiente:

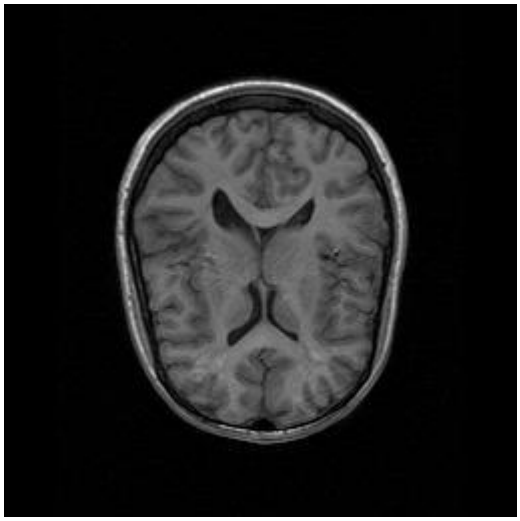


Figura 4.45. Imagen de RM a segmentar

Los clusters obtenidos se muestran a continuación, indicando la etiqueta de cada uno:

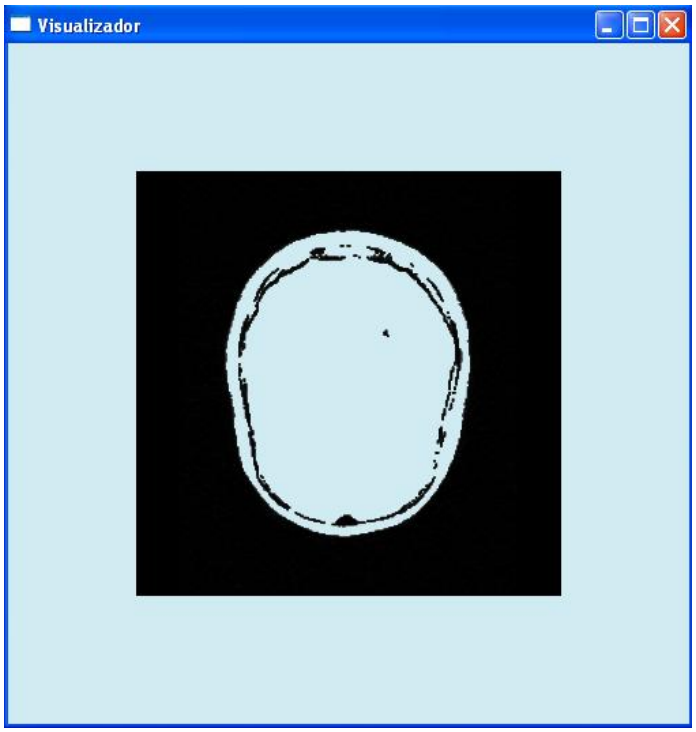


Figura 4.46. Cluster 0: Aire (fondo de la imagen)

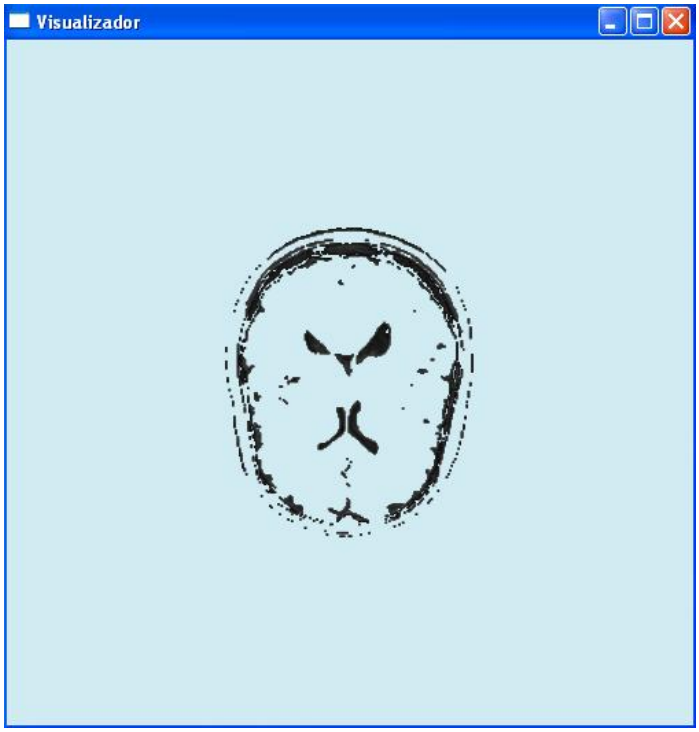


Figura 4.47. Cluster 1: Fluido Cerebroespinal

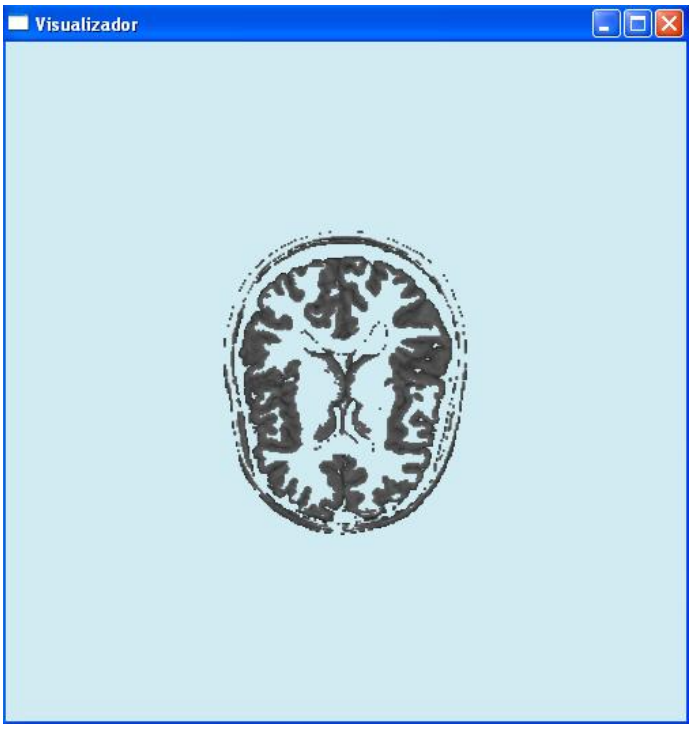


Figura 4.48. Cluster 2: Materia Gris

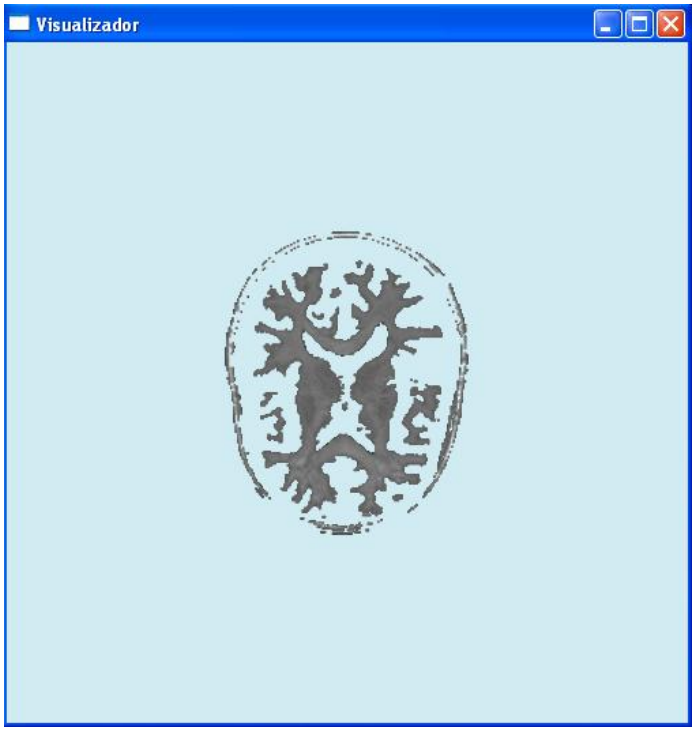


Figura 4.49. Cluster 3: Materia Blanca

Figura 4.50. Cluster 4: Tejido Óseo

De las imágenes anteriores resulta evidente que se presentan errores de clasificación en la periferia de la región de interés ya que la piel contiene todas las tonalidades de gris (desde el negro hasta el blanco) debido a la compresión de niveles a la que fue sujeta la imagen original y ésta se encuentra presente en todos los clusters, su presencia es más notoria en la segmentación del volumen completo.

Ahora, para concluir con la sección se presenta la segmentación del volumen completo con ciertos cortes para una mejor visualización buscando los mismos clusters:

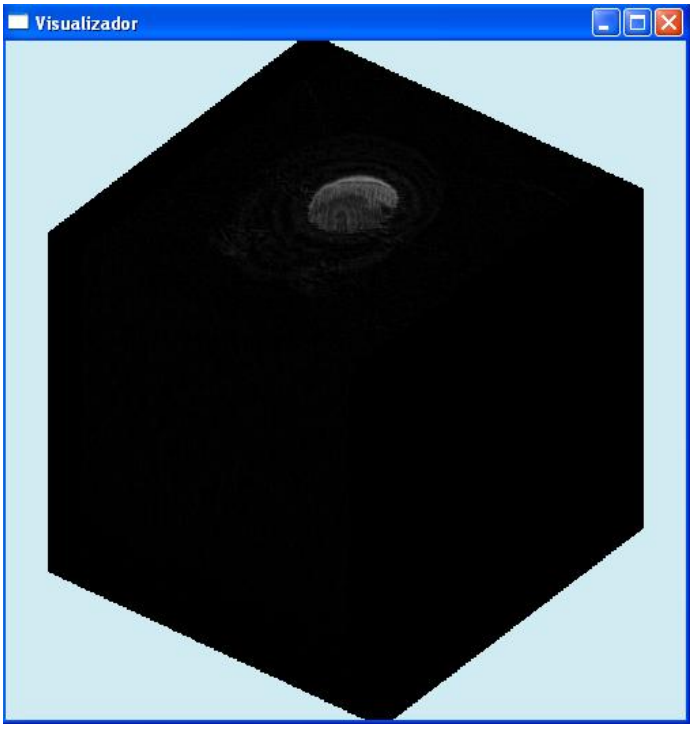


Figura 4.51. Aire (fondo de las imágenes)

Figura 4.52. Fluido Cerebroespinal

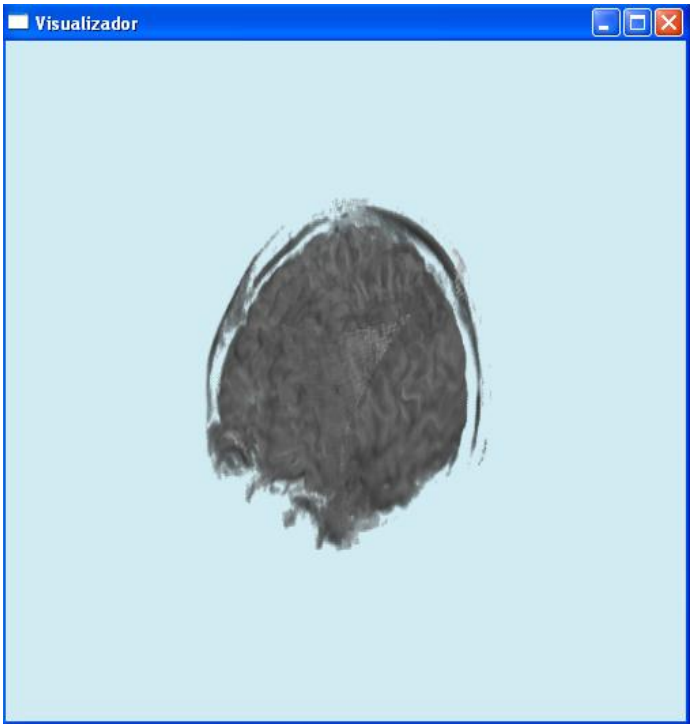


Figura 4.53. Materia Gris

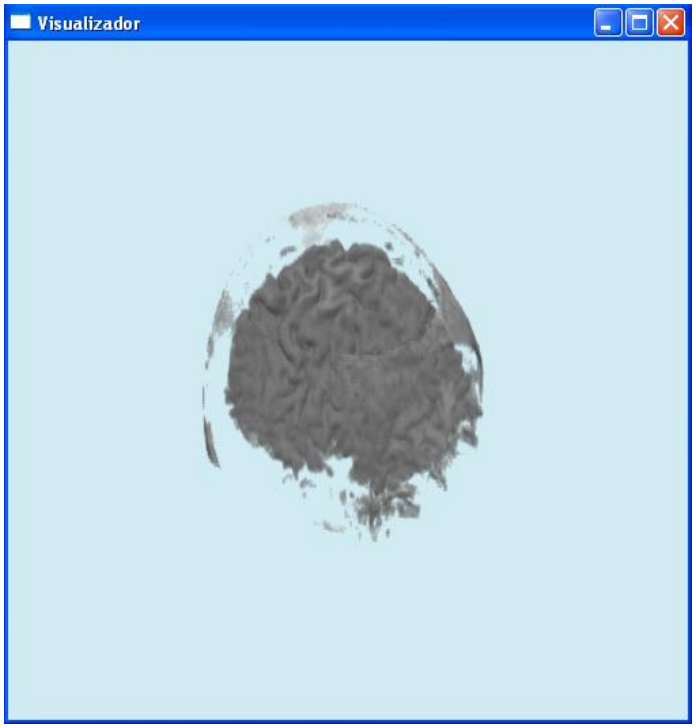


Figura 4.54. Materia Blanca

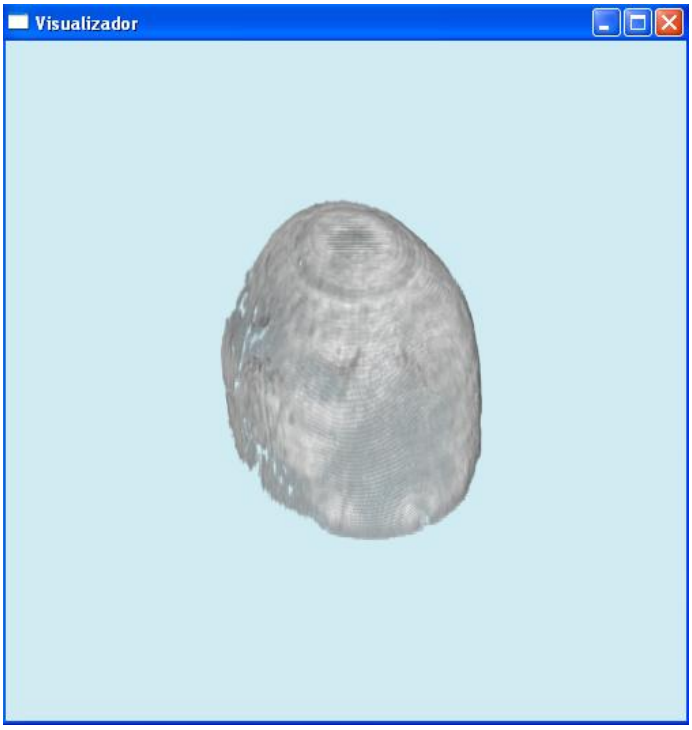


Figura 4.55. Tejido Óseo

5. CONCLUSIONES

5.1. Anotaciones finales

De acuerdo a los objetivos planteados al inicio de este proyecto, se puede decir que se cumplieron satisfactoriamente.

En primer lugar, se desarrolló una aplicación capaz de permitir la visualización, de manera clara y sencilla, de volúmenes a partir de secuencias de imágenes. De esta forma, el programa cumple con ser didáctico y, por lo tanto, ser utilizado no sólo para analizar volúmenes de datos, pues si se considera que se implementó el módulo que permite la visión estereoscópica, la aplicación puede resultar aún más interesante para el usuario si se ejecuta en equipos que permitan este tipo de visualización.

Respecto al punto de lograr un sistema multiplataforma: la aplicación fue implementada en el lenguaje de programación ANSI C y también se incluyeron librerías basadas en C++: SGI OpenGL Volumizer, OpenGL GLUT, DevIL y GLUI. Debido a lo anterior, se asegura la correcta compilación y ejecución de la aplicación en los sistemas operativos Windows y sistemas basados en UNIX (como IRIX).

También se programaron distintas funcionalidades destinadas a manipular los datos por parte del usuario con el fin de aprovechar las ventajas de observar las imágenes como un ‘todo’. Entre tales funciones se retoma (del proyecto previo a éste) la transformada discreta de Hermite y es mejorada en aspectos de visualización de los coeficientes.

En el tema de segmentación, se implementaron varias técnicas como son k-means, fuzzy c-means y fuzzy c-means con información espacial, siendo este último el más importante por el enfoque con el que fue ideado y por la posibilidad de modificar sus parámetros. Además, cualquiera de las técnicas disponibles no requiere datos de entrenamiento y sólo requiere ser ejecutado una vez debido a que los resultados son

almacenados en archivos, los cuales son leídos en las ejecuciones posteriores que presenten la misma combinación de parámetros.

Los resultados de las técnicas de segmentación fueron aceptables. En el caso de imágenes de CT con tres clases, analizado mediante el índice kappa, éste resultó con un valor muy alto demostrando el buen desempeño del modelo fuzzy c-means con información espacial. Por otra parte, la segmentación del volumen de RM en cinco clases resultó coherente de acuerdo a lo expuesto en documentos y demás fuentes especializadas; en este caso la evaluación se realizó de forma subjetiva ya que las imágenes que se usaron son bastante complejas para determinar con exactitud si los pixeles clasificados corresponden a los tipos de tejido en cuestión. Es claro al observar los resultados que la segmentación que proporciona la aplicación no es perfecta. En resumen, los elementos mal clasificados son pocos en comparación con el número de elementos clasificados correctamente.

Un aspecto que fue mejorado de la aplicación inicial fue la capacidad de aceptar más formatos de imágenes. Ahora, se pueden procesar los formatos JPG, BMP, PNG, TIFF, RAW (de 8 bits y de 12 bits), así como archivos de volúmenes TIF y RAW.

5.2. Mejoras propuestas

Existen varios aspectos que pueden ser mejorados en torno al desempeño de la aplicación como herramienta de visualización o, también, respecto a la segmentación. Las principales mejoras que se proponen son:

- La transformada discreta de Hermite puede emplearse como herramienta de clasificación. En específico, se puede aplicar enfocando su uso a texturas.
- Agregar la facultad de visualizar coeficientes de ordenes superiores para transformada de Hermite
- Implementar la función que permita leer archivos de formato DICOM, ya que éste es un formato muy usado para almacenar imágenes en el campo de la Medicina.

- Desarrollar el módulo que permita la carga de volúmenes a partir de imágenes de colores sin tener que mapear los colores a niveles de gris.
- Hacer mejoras en cuanto a la visualización de los resultados, por ejemplo, usar colores para mostrar las clases resultantes del proceso de segmentación.
- Actualizar la interfaz gráfica empleando herramientas más poderosas que GLUI, como pudiese ser QT de Trolltech o bien desarrollar el sistema de ventanas usando la API de cada sistema operativo; aunque esta sugerencia final podría constituir un proyecto por separado.

5.3. Posibles campos de aplicación

El programa desarrollado puede tener aplicaciones en sitios donde se requiera el análisis de volúmenes a partir de secuencias de imágenes. Aunque, un campo donde se puede usar con buenos resultados es en el de la Medicina. Específicamente, puede ser útil para los usuarios de este campo que deseen visualizar, en 3D, un conjunto de imágenes y poder segmentar el volumen resultante.

Sin embargo, esta aplicación no está restringida a ambientes médicos, sino que también puede utilizarse en el análisis de secuencias en movimiento.

Una aplicación más puede encontrarse en el área docente. En este campo, puede ser muy útil en la enseñanza del procesamiento digital de imágenes y también del desarrollo de ambientes virtuales.

ANEXO A.
NOTAS SOBRE LAS LIBRERÍAS.

A.1 Librería DevIL [23].

La librería de manipulación de imágenes para desarrolladores DevIL (por su acrónimo en la lengua anglosajona *Developer's Image Library*) es, en sí, un conjunto de librerías de código abierto que permite su inclusión en aplicaciones que requieran la carga y manipulación de imágenes en sus diversos formatos, además del uso de los metadatos implícitos de las imágenes. Esta librería es capaz de manejar imágenes de diversas maneras y tiene la facultad de pasar los datos obtenidos de la manipulación de las mismas a otras interfaces de programación de aplicaciones como lo son OpenGL y Direct3D.

Figura A.1. Logotipo de DevIL

DevIL se compone de tres sublibrerías que aportan aspectos diferentes de manipulación y son llamadas IL, ILU e ILUT:

- IL es la librería base que permite la carga, almacenamiento y conversión de las imágenes de diversas maneras dependiendo de lo que se espera obtener.
- ILU se refiere a una librería de nivel intermedio destinada a la manipulación de las imágenes y contiene funciones de filtrado, escalamiento, manejo de contraste, ecualización, corrección gamma, entre otros procesos aplicados comúnmente sobre las imágenes.
- ILUT es una librería de alto nivel para el despliegue de las imágenes y contiene diversas funciones entre las cuales se pueden encontrar funciones para la creación de animaciones y el mezclado de subimágenes.

Una vez construida la librería DevIL ésta puede usarse para lo que se necesite, sin embargo antes de emplear los métodos contenidos en ella, primero deben inicializarse las sublibrerías que serán requeridas tomando en cuenta la estructura piramidal de las mismas. En código, la manera de inicializar dichas librerías es la siguiente:

- IL: ilInit();
- ILU: iluInit();
- ILUT: Ésta debe inicializarse dependiendo del soporte que se requiera para una determinada API, por ejemplo, para soporte de OpenGL se llama al método ilutRenderer(ILUT_OPENGL);

A.2 Librería OpenGL.

OpenGL permite la creación de programas interactivos que produzcan imágenes en color del movimiento de objetos tridimensionales. Cabe señalar que el título de GL corresponde a la siglas de “librería de gráficos” en el idioma inglés.

Figura A.2. Logotipo de OpenGL

La interfaz de *OpenGL* está compuesta por alrededor de 150 comandos diferentes [24] que se emplean para la especificación de objetos; estos comandos también especifican las operaciones que son requeridas para la producción de las aplicaciones interactivas de gráficos tridimensionales

La imagen final que es rendereada está compuesta de píxeles que son dibujados en pantalla, siendo el píxel, el elemento visible más pequeño que el hardware de gráficos

puede poner en pantalla. Información sobre los píxeles como el color que deben tener es organizada en memoria en objetos denominados "*planos de BIT*" (*bitplanes* en la lengua inglesa). Los "*planos de BIT*" son, a su vez, organizados en un búfer llamado "*framebuffer*" que contiene toda la información necesaria para que el hardware de gráficos encargado del despliegue controle el color y la intensidad de cada uno de los píxeles en pantalla.

OpenGL puede describirse como una máquina de estados, pueden establecerse varios estados que permanecerán en efecto hasta que éstos sean cambiados, por ejemplo el color actual de uno o varios objetos es una variable de estado. Puede determinarse un color dado cualquiera y, entonces, todos los objetos que se dibujen en pantalla serán pintados con ese color hasta que se coloque otro color, por lo cual, el color actual es una de la diversas variables de estado que maneja *OpenGL*, otras variables por ejemplo controlan la vista actual y las transformaciones de proyección.

La estructura básica de un programa útil que use *OpenGL* es relativamente simple: la primera tarea consiste en inicializar ciertos estados que controlen cómo se manejarán y especificarán los diferentes objetos, seguidamente se tiene la etapa de elaboración de los modelos para entrar finalmente al proceso de *rendering*.

Muchas implementaciones de *OpenGL* tienen un orden similar de operaciones, una serie de etapas de procesamiento que es llamada el *pipeline* de *rendering* de *OpenGL* [24] presentado en la figura A.3.

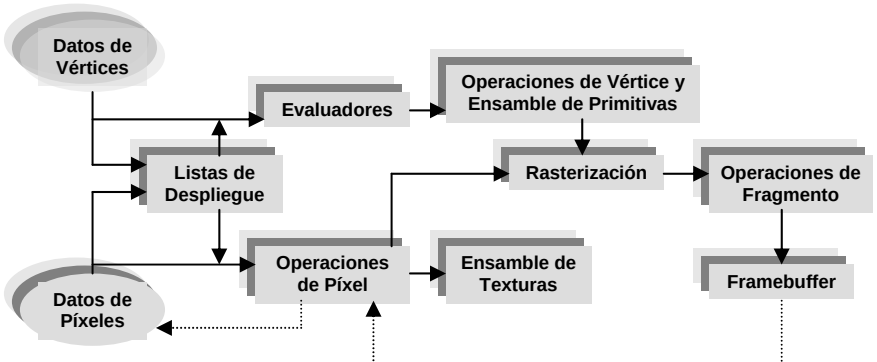


Figura A.3. El pipeline de OpenGL

A.3 Librería GLUT.

GLUT simplifica la implementación de programas que usan el rendering de OpenGL. La interfaz de programación de aplicaciones de GLUT requiere el uso de pocas rutinas para mostrar una escena de gráficos generada con OpenGL, así mismo esta interfaz es una máquina de estados.

GLUT se diseñó para su uso en aplicaciones que van desde simples a relativamente complejas que usen el proceso de *rendering* de *OpenGL*. GLUT implementa su propio ciclo o bucle de procesamiento de eventos, razón por la cual la acción de mezclar esta librería con otras herramientas que usen un ciclo de eventos diferente puede resultar complicada

Como se mencionó en el capítulo número cuatro, GLUT se organiza agrupando sus rutinas en sub-interfaces, las principales ya fueron mencionadas, sin embrago las restantes que complementan todas las funcionalidades de GLUT son las siguientes:

- *Inicialización:* Manejo de la línea de comandos, inicialización del sistema de ventanas.
- pueden ser registradas con GLUT para responder a un tipo específico de evento.
- *Manejo del mapa de colores indexado:* Estas rutinas permiten la manipulación del mapa de colores para las ventanas.
- *Recuperación de Estado:* Las rutinas de esta sub-interfaz permiten obtener el estado en el que está operando GLUT.
- *Rendereo de Fuentes:* Estas rutinas manejan el trazo en pantalla de fuentes de texto en las ventanas.

A.4 OpenGL Volumizer.

Presentado durante la conferencia SIGGRAPH en el año 2001 [29] *OpenGL Volumizer* consiste en un kit de desarrollo de software (SDK) y contiene una interfaz de

programación de aplicaciones (API) para entornos de desarrollo C++. El uso de esta librería produce beneficios directos como la reducción de los tiempos de desarrollo logrando también un incremento en el desempeño de la aplicación en mano, incluso, permite al programador concentrarse en las diferentes características de sus aplicaciones más que en la implementación de la mecánica de visualización del volumen.

OpenGL Volumizer es un producto de *Silicon Graphics* posicionado sobre *OpenGL*. *Volumizer* permite una integración simple y casi inmediata con las presentaciones gráficas a base de geometrías, por lo cual, diferentes aplicaciones y herramientas de software pueden reescribirse añadiéndoles capacidades de trazo de volúmenes de manera rápida.

El correcto funcionamiento de *Volumizer*, según lo reporta el fabricante [26], ha sido probado con éxito en las plataformas IRIX de *Silicon Graphics* que soportan mapeos de texturas en tres dimensiones. Las plataformas Linux de 32 bits SUSE Linux 9.1, Red Hat 8, Red Hat 9 y Red Hat Enterprise Linux con tarjetas gráficas ATI Radeon 9000, ATI FireGL o NVIDIA GeForce FX han probado ser capaces de ejecutar *Volumizer* siempre y cuando se tenga un compilador GCC 3.2 instalado; de igual manera funciona para las plataformas Linux de 64 bits. En los sistemas operativos Windows NT, Windows 2000 o Windows XP Volumizer ha sido probado con éxito siempre y cuando las tarjetas gráficas que se estén empleando soporten el mapeo de texturas en tres dimensiones y se tenga un ambiente de desarrollo de software de C++ [26].

Existen diversas maneras para visualizar volúmenes usando la computadora digital, muchas de ellas usan técnicas de análisis de datos con el objetivo de localizar superficies de contorno en el volumen de interés y entonces proceden al trazo de la geometría resultante. Un método común es el uso de la técnica denominada "*ray casting*" en la cual cada píxel de la imagen es construido rayo por rayo.

Supóngase que se tiene una función implícita de tres dimensiones:

$$g(x,y,z)=0$$

Si dicha función es también analítica [28], entonces *g* es una función miembro que permite probar si un punto particular pertenece o no a la superficie que la función describe; para saber esto, un rayo proyector parte de un punto de observación denominado centro de proyección buscando intersecar a la superficie descrita por *g*, un rayo proyector puede ser escrito como una función paramétrica:

$$\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{d}$$

O bien, también puede ser escrito en términos de sus componentes individuales:

$$x(t) = x_0 + td_X$$
$$y(t) = y_0 + td_Y$$
$$z(t) = z_0 + td_Z$$

Sustituyendo esto en la ecuación implícita, se obtiene una ecuación escalar en *t*:

$$g(x_0 + td_X, y_0 + td_Y, z_0 + td_Z) = u(t) = 0$$

La solución de está ecuación corresponde a los puntos donde el rayo proyector entra o sale de la superficie, por lo cual, si *g* es una función simple como una cuádrica, entonces u(t) se puede resolver directamente, así es como por medio de este método pueden trazarse superficies de tres dimensiones, ocurre lo mismo si se desea emplear la técnica directamente con volúmenes solo que además de determinar los puntos de entrada y salida se hace una interpolación para obtener un muestreo correcto y determinar los *voxeles*, mientras que, por otro lado, *Volumizer* usa una técnica distinta denominada “mapeo de texturas en tres dimensiones” que es equivalente a la técnica de *“ray casting”*.

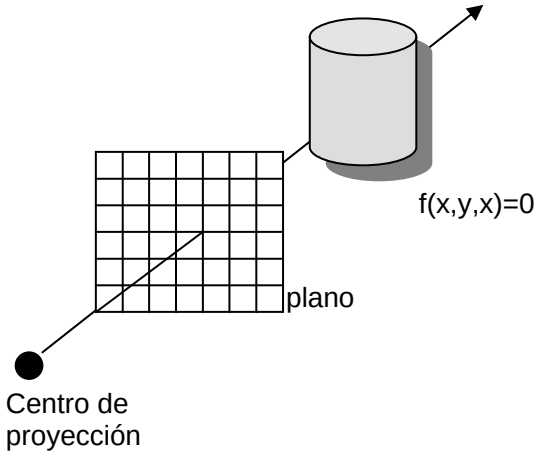


Figura A.4. Ray Casting

Volumizer separa el *rendering* de un volumen en dos partes: la geometría y la apariencia. La apariencia en sí consiste de una lista de parámetros que son específicos de una técnica particular de trazo que está siendo aplicada al nodo "*shape*". Los parámetros de apariencia trabajan como contenedores de datos para la acción de trazo (comúnmente denominada en esta API como "*render action*") y retienen los datos del volumen propiamente así como otros parámetros de sombreado como lo son la dirección de las luces o el contenido de las tablas de apareamiento denominadas "*look up tables*" (LUT)

Las acciones de trazo son implementadas como una clase separada derivada de la clase *vzRenderAction* y contienen todos los componentes necesarios para trazar la forma del volumen. La acción de trazo es responsable del manejo de los recursos que se necesitan para dibujar los nodos "*shape*" y elabora particiones poligonales de la geometría del volumen usando planos alineados con el puerto de vista, posteriormente aplica los parámetros de apariencia y de sombreado como las propias texturas en tres dimensiones o las tablas de post interpolación (LUT).

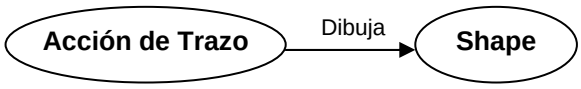


Figura A.5. La acción de trazado o "render action"

Las clases de los diferentes objetos usados por *Volumizer* son derivadas de la clase primaria de éstos denominada *vzMemory* que permite especificar la reservación y liberación de memoria para colocar y eliminar objetos en la memoria respectivamente, con lo cual se simplifica el manejo de la memoria; dado que todos los objetos en la escena gráfica cuentan con un índice de referencia de conteo, los objetos son borrados de la escena cuando dicho índice alcanza el valor de cero.

El motor encargado de realizar los trazos denominado *TMRenderAction* que se incluye en la distribución del producto implementa una técnica de trazo de planos semitransparentes y se basa en el uso de texturas en tres dimensiones. *Volumizer* incluye *shaders* que permiten generar los efectos visuales deseados empleando el mismo algoritmo de trazo. Cada *shader* implementa una técnica particular estableciendo a *OpenGL* en un estado específico. *TMRenderAction* soporta *shaders* que acepten parámetros para que sea aplicada la técnica en particular. En el presente proyecto fueron utilizados los *shaders vzTMSimpleShader* y *vzTMLUTShader* proporcionados con la distribución de *Volumizer*.

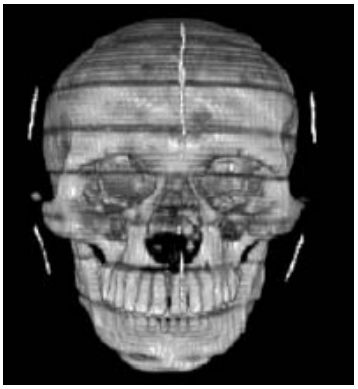


Figura A.6. Aplicación del *shader vzTMLUTShader* en la visualización de un cráneo [29]

TMRenderAction no realiza ningún ordenamiento de visibilidad de las formas dibujadas, mas bien, es responsabilidad de la aplicación ordenar la escena para su correcta visualización

A.5 Librería GLUI [30].

GLUI es una librería que provee una interfaz de usuario basada en GLUT para el lenguaje de programación de alto nivel C++, proporciona diversos controles como botones, casillas de verificación, botones radiales, listados y espacios de texto para aplicaciones que empleen *OpenGL*. EL proyecto GLUI fue creado originalmente por Paul Rademacher y fue continuado después por Nigel Stewart.

Escrita por completo sobre GLUT, una aplicación que contenga GLUI se comportará de la misma manera en sistemas de *Silicon Graphics*, en máquinas con el sistema operativo Windows, en computadoras Linux y en cualquier otro dispositivo capaz de manejar GLUT. Así pues las aplicaciones que solamente se encuentren empleando GLUT necesitan únicamente unas pocas modificaciones para que éstas comiencen a utilizar GLUI. La figura A.7 muestra una típica ventana de controles GLUI.

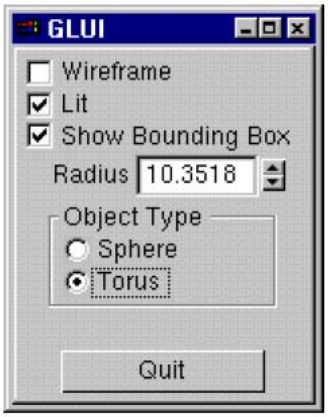


Figura A.7. Ejemplo de ventana GLUI

GLUI fue diseñada para obtener la máxima simplicidad de programación posible. Tanto las ventanas, controles y otros objetos GLUI pueden ser creados en una sola línea de código cada una. Los controles son automáticamente colocados en la ventana que los requiere, el programador no necesita dar coordenadas *xy* de pantalla para ubicar los controles, basta con especificar su alienación en la ventana y el orden se su disposición.

GLUI asocia a las llamadas “variables en vivo” o “variables vivas” (*live variables*) con la mayoría de los controles. Esta variables son variables comunes del lenguaje C como el resto pero son actualizadas automáticamente cuando se hace interacción con el control GLUI correspondiente, por ejemplo, una casilla de verificación tiene asociada una variable entera cuyo valor cambia entre cero y uno dependiendo si la casilla está chequeada o no, así también, por ejemplo, un campo de texto tiene asociado un arreglo de caracteres como *variable en vivo*. Cuando ocurre un cambio que debe ser reflejado en la pantalla de controles, como por ejemplo, un cambio en un campo de texto, se envía un mensaje de refresco de ventana por medio de GLUT para visualizar el cambio en pantalla.

REFERENCIAS.

[1] Gonzalez, R. C.; Woods Richard E. *"Digital Image Processing"*. Massachussets, Addison Wesley, 1987, pp. 34 – 39, 413 – 477.

[2] Apuntes del curso "*Visual Communications*", INSA, Francia.

[3] Laurent Deniau "*Image Processing I*" CERN Technical Training 2003, pp. 12, 13.

[4] [En línea]. Disponible en Internet: <http://www.uam.es/personal_pdi/psicologia/travieso/web_percepcion/sistemav.html>, [Julio 2007].

[5] Girod, Bernd *"Human visual perception topics"* [En línea] digital image processing, Universidad de Stanford, 2006, pp. 4, 6. Disponible en Internet: <http://www.stanford.edu/class/ee368b/Handouts/09-HumanPerception.pdf>, [Julio 2007].

[6] [En línea]. Disponible en Internet: <http://www.britannica.com/ebc/article-64928>, [Julio 2007].

[7] Escalante R. Boris *"Apuntes Procesamiento Digital de Imágenes Capítulo 4: Realce de la Imagen"* [En línea] Facultad de Ingeniería, UNAM, 2006, pp. 9 – 14. Disponible en Internet: <http://verona.fi-p.unam.mx/boris/teachingnotes/Capitulo4.pdf>, [Julio 2007].

[8] Malik, Jitendra; White, Ryan *"Human Visual System"* [En línea] Recognizing People, Objects and Actions, Universidad de California campus Berkeley, 2004, pp. 3**.** Disponible en Internet: <http://www.cs.berkeley.edu/~malik/cs294/lecture2-RW.pdf>, [Julio 2007].

[9] Imagen de Lena. Disponible en Internet: <http://sipi.usc.edu/database/database.cgi?volume=misc&image=12#top>, [Agosto 2007].

[10] Young, Richard; et al. *"The Gaussian derivative model for spatial – temporal vision Cortical Model"* Spatial Vision. Vol. 14 no. 3, 4, 2001, pp. 261 – 319.

[11] Martens, Jean B. *"The Hermite Transform - Theory"* IEEE Trans. Acoust., Speech, Signal Processing, vol. 38 no. 9, 1990, pp 1595 – 1606.

[12] Young Richard, et al. *"A Gaussian Derivative Based Version of JPEG for Image Compression and Decompression"* IEEE transactions on image processing, vol. 7, no. 9, 1998, pp 1311 – 1320.

[13] Martens Jean B. *"The Hermite Transform: A Survey"* EURASIP Journal on Applied Signal Processing, vol. 2006, artículo 26145, 2006, pp 1 – 20.

[14] Troncoso M. Miguel A. "*Sistema de codificación y transmisión de video sobre redes TCP/IP basado en la transformada polinomial"* Tesis de Licenciatura (Ingeniero en Computación). Director: Dr. Boris Escalante Ramírez. Universidad Nacional Autónoma de México, Facultad de Ingeniería, 2005, pp. 32, 33.

[15] Bezdek, J. *et. al. "Fuzzy models and algorithms for pattern recognition and image processing"*. Boston, Kluwer Academic, 1999.

[16] Rivero Moreno, C. J. y Bres, S. "*Conditions of Similarity between Hermite and Gabor Filters as Models of the Human Visual System*", Petkov, N. y Westenberg, M.A. (editores): *Computer Analysis of Images and Patterns, Lectures Notes in Computer Science*, vol. 2756. Springer-Verlag, Berlin Heidelberg, pp. 762-769, 2003.

[17] Pham, D. L., Xu, C. y Prince J. L. "*Current methods in medical image segmentation*". *Annual Review of Biomedical Engineering*, vol. 2, 2000, pp. 315-337.

[18] Liu, H., *et. al*. "*Multiresolution Medical Image Segmentation Based on Wavelet Transform*". Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the, Vol., Iss., 2005, pp. 3418-3421.

[19] Báez Rojas, J. J., et. al. "*Segmentación de imágenes de color*". Revista Mexicana de Física, Vol. 50, 2004, pp. 579-587.

[20] Ho, S. y Lee K. "*An efficient evolutionary image segmentation algorithm*". Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, Vol.2, Iss., 2001, pp. 1327-1334.

[21] Chuang K *et. al. "Fuzzy c-means clustering with spatial information for image segmentation*". Computerized Medical Imaging and Graphics, Vol. 30, Iss. 1, 2006, pp. 9-15.

[22] Kendall, K. E. y Kendall, J. E. *"Análisis y Diseño de Sistemas"*, Pearson Educación, 1997, pp. 229 – 237.

[23] Woods, D. *"Developer's Image Library Manual"*, [En línea]. Disponible en Internet <http://openil.sourceforge.net/download.php>, [Julio 2007].

[24] Shreiner, D. *et. al.* "*OpenGL Programming Guide: The Official Guide to Learning OpenGL"*, Versión 1.1, Addison Wesley Publishing Company, 1997, pp. 13 – 22.

[25] Kilgard, M. J. *"The OpenGL Utility Toolkit (GLUT): Programming Interface"*, [En línea] Versión 3, 1996. Disponible en Internet: <http://www.opengl.org/resources/libraries/glut/glut-3.spec.pdf>, [Julio 2007].

[26] Silicon Graphics, *"SGI OpenGL Volumizer 2 Programmer's Guide"*, [En línea] Versión 2.8. Disponible en Internet: <http://www.sgi.com/products/software/volumizer/vol-prog-guide.pdf>, [Julio 2007].

[27] GnuWin32 [En línea]. SourceForge, Inc. Disponible en internet: <http://gnuwin32.sourceforge.net/>, [Julio 2007].

[28] Angel E. *"Interactive computer graphics a top down approach with OpenGL"*, Addison Wesley, 2002, pp. 517, 572 -574, 583, 584.

[29] Silicon Graphics, *"OpenGL Volumizer 2.x white paper"* [En línea]. Disponible en Internet: <http://www.sgi.com/pdfs/3176.pdf>, [Julio 2007].

[30] Rademacher P. *"GLUI: A GLUT Based User Interface Library"*, [En línea] Versión 2.0, 1999. Disponible en Internet: <http://glui.sourceforge.net/#download>, [Julio 2007].

[31] Kohavi, R. y Provost, F., "*Glossary of Terms*" Machine Learning, Vol. 30 (2/3), pp. 271-274, 1998.

[32] "*Confusion Matrix*" [En línea]. Computer Science 831: Knowledge Discovery in Databases. Department of Computer Science. University of Regina. Disponible en Internet: <http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html>, [Junio 2007].

[33] Landis, J. R. Y Koch, G. G. "*The measurement of observer agreement for categorical data"*. Biometrics, Vol 33, No. 1, pp. 159 – 174, 1977.

[34] "*Kappa Statistic*" [En línea]. Statistical Glossary. Disponible en Internet: <http://www.statistics.com/resources/glossary/k/kappa.php>, [Junio 2007].

[35] Segura M., R. y Trincado V., G. "*Cartografía digital de la Reserva Nacional Valdivia a partir de imágenes satelitales Landsat TM. Bosque (Valdivia)"*. [En línea]. Agosto 2003, Vol. 24, No.2, pp. 43-52. Disponible en Internet: <http://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0717-92002003000200005&lng=es&nrm=iso>. [Junio 2007].

[36] Mohamed N. et. al. "Modified fuzzy c mean in medical image segmentation". Acoustics, Speech and Signal Processing ICASSP Marzo 1999, Vol. 6, pp 15 – 19.

[37] Hermoye, L., *et. al*., *White Matter Atlas* [En línea]. Université Catholique de Louvain (Bélgica); Imagilys (Bélgica); John Hopkins University (E.U.A); F.M. Kirby Research Center for Functional Brain Mapping, Kennedy Krieger Institute (E.U.A). Disponible en internet: <http://www.dtiatlas.org/>, [Julio 2007].

[38] Development and Validation of Techniques for Brain Morphometry (BIOMORPH) [En línea]. Disponible en Internet: <http://www.cs.unc.edu/~styner/biomorph/>, [Julio 2007].