UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FES Aragón

Kernel de visualización

TESIS QUE PARA OBTENER EL TÍTULO DE

Ingeniera en Computación

P r e s e n t a

Erika Elizabeth Rocha Cordova

DIRECTOR
José Luis Villarreal Benítez

San Juan de Aragón, Edo. de México – 2006

Todos los escritos introductorios, inclusive el prefacio, fueron escritos en 2002; al inicio de este proyecto.

Agradecimientos

Quisiera agradecer a todo la gente que me brindó su apoyo durante la realización de este trabajo; pero muy en especial a la persona que hizo posible que este momento llegara.- a mi madre, gracias por darme la vida y por siempre estar en ella - madre mía.

No tengo palabras para agradecerte todo lo que has apoyado no solo en mi carrera sino en mi vida, ya que has sido mi mejor amiga, compañera y ejemplo a seguir. Tu eres la persona a la que le debo todo lo que soy y lo que podré llegar a ser. Gracias mamá por confiar y creer en mí, espero nunca defraudarte.

A Ramón por siempre estar a mi lado impulsándome a seguir adelante y continuar con mis sueños - gracias peque. A todos mis amigos que siempre estuvieron a mi lado apoyándome y confiando en mí, así como a los que no estuvieron a mi lado ya que sus caminos los separaron de mí.

A mi hermana Jessica, a mi sobrina Sheridan y a mi cuñada Adenice, por haberme ayudado a comprender mis errores y por toda la ayuda que me brindaron durante mis noches de angustia.

Me gustaría hacer una mención especial a mi asesor de tesis y jefe, José Luis Villarreal Benítez por haber apoyado y esperado tanto tiempo para terminar la tesis, así como un gran agradecimiento por haberme tenido paciencia al enseñarme y guiarme para realizar la tesis - gracias.

EERC
San Juan de Aragón, Edo. de México - abril del 2006

Índice general

P r e f a c i o

Cómputo de alto rendimiento y visualización en la UNAM

El poder de cálculo que proporcionaron las grandes computadoras en la década pasada fue muy significativo para los avances científicos y tecnológicos; principalmente en áreas que siempre han demandado cálculos rápidos y complejos (química cuántica, astrofísica, mecánica de fluidos, etc.). En esta década veremos nuevas áreas (medicina, psicología, biología y economía) y un incremento en la dependencia de las tecnologías del cómputo de alto rendimiento. A lo largo de la última década hemos visto la consolidación del cómputo científico en muchos grupos de investigación y desarrollo dentro de nuestra comunidad universitaria y se ha extendido a otras universidades e instituciones del sector público y gobierno.

Esta experiencia deja claro que debemos continuar el esfuerzo y estar listos para una mayor incidencia en disciplinas de la ciencia y la ingeniería que inician en el uso de esta tecnología, así como en otros sectores de la sociedad; ya que los equipos de cómputo intensivo y otras tecnologías asociadas, están penetrando sectores más amplios, como son: la industria, finanzas, servicios, administración pública y la macro-infraestructura.

Este éxito del cómputo científico intensivo se ve reflejado en la diversificación de las herramientas en hardware y software que ahora están en la misma área. Por ello en la UNAM se ha realizado este gran esfuerzo y la FES Aragón está participando activamente para mantenerse en los distintos escenarios que siguen evolucionando. Por lo que nuestro proyecto de visualización incluye diversos aspectos descubiertos y redescubiertos con el ejercicio en estos proyectos.

La visualización se ha convertido en parte integral del proceso; apoyando un cómputo más efectivo y la interpretación de los datos de gran escala generados por estos equipos, los cuales son cada vez más grandes y complejos. La madurez del cómputo científico nos ha colocado en una etapa en la que los sistemas y herramientas usados en su ámbito son (o tienden a ser) verdaderos sistemas integrales para la solución de problemas; donde la visualización interviene ya no solo en la interpretación visual y el descubrimiento de patrones y tendencias, también interviene en el proceso mismo al controlar el sistema y permitir la interacción humano-máquina. Este reconocimiento conlleva una necesidad de plantear nuevas estrategias en el diseño de software científico.

El Kernel de Visualización es un esfuerzo para facilitar el desarrollo de aplicaciones y herramientas de software que exploten eficientemente el hardware de cálculo numérico y gráfico, que permitan la implementación de entornos gráficos para la interacción humano-máquina y se usen las técnicas de visualización con una metodología científica.

EERC
JLVB
México, 2005

R E S U M E N

El cómputo científico se ha consolidado y fortalezido por el contínuo incremento en la capacidad de cálculo de las computadoras. Además del incremento en la resolución y escala de los problemas computacionales que se han resuelto en las diciplinas tradicionales, se han sumado nuevos grupos con problemas complejos; como la biología, las neurociencias y la sociología. También se han consolidado en otros sectores: las finanzas, los servicios y la industria. Este aumento en complejidad y diversidad requiere de nuevos entornos de trabajo e interacción con los sistemas de cómputo; la visualización es una alternativa para crear entornos integrales para la solución de problemas, ocupándose de la interpretación visual, el descubrimiento de patrones y tendencias, del control del sistema y la interación humano−máquina. Esta tarea ha sido resuelta a través de bibliotecas especializadas en cada una de los tareas o componentes de un entorno de trabajo con visualización: *OpenGL* para el dibujado de geometrías; *Qt*, *motif*, *gtk*, para el desarrollo de interfaces gráficas; etc. La complejidad resultante del software construido es muy alta, el aprendizaje de todas las herramientas es lento, su mantenimiento y reuso son difíciles. Este estilo de trabajo en ciencias e ingeniería conlleva nuevas estrategías en el diseño de software científico. En el presente trabajo se plantea el desarrollo de una *API* o nucleo de bibliotecas (kernel) que se aproxime a la solución del problema planteado y permita la implementación de entornos gráficos con interacción humano−máquina y visualización fenológica y de información. La propuesta resultante del análisis y la implementación lograda es una *API* que toma a las distintas bibliotecas establecidas como aplicaciones especializadas y permite su interación. Se propone la integración en el *kernel*, de *OpenGL*, *vtk*, *Qt*, *OpenH323*, *OpenAl*, *OpenSceneGraph* y *Pwlib*; se logra la integración de las tres primeras. La integración se logró con la implementación de objetos específicos para cada una de las características especificadas, aprovechando el encapsulamiento. El kernel permite la implementación de prototipos y su mejora contínua, es una herramienta eficaz para la implementación rápida de nuevas técnicas o nuevos algoritmos y promueve el desarrollo y evaluación de nuevas ideas para el desarrollo de entornos de trabajo visual. La evaluación del kernel se llevó a cabo a través de la implementación de una aplicación de visualización científica para ciencias biomédicas. Esta aplicación requirío de 150 líneas de código, mientras que el desarrollo sin el kernel, implicó 2000. La arquitectura del kernel permite fácil mantenimiento y reuso, mejor depuración y optimización de códigos.

Palabras clave: *API*, *Qt*, *OpenH323*, *OpenAl*, *OpenGL*, *Pwlib*, *vtk*, *OpenSceneGraph*, *C++*, visualización científica, *isosuperficies*, *volume rendering*, biomedicina, kernel, *Motif* y *Gtk*.

Capítulo 1

Introducción

Los avances de la ciencia y de las tecnologías han permitido que los modelos y las escalas de los problemas que se abordan en estas disciplinas del conocimiento, sean cada día más complejos; sin embargo, el efecto antropocéntrico sobre nuestro entorno y las alteraciones al ambiente han complicado los escenarios de estudio y la urgencia de su comprensión. Ambos procesos han llevado a la búsqueda de nuevas aproximaciones y estilos para la solución de problemas, estos incluyen entornos de trabajo visuales, altamente interactivos y colaboratorios. Los profesionales de la visualización científica han sabido dar respuesta oportuna a esta demanda de herramientas.

Esta disciplina ha sufrido una serie de transformaciones en sus métodos y técnicas para cumplir las exigencias. Esta disciplina relativamente nueva, aún está alcanzando su madurez. Este proceso ha ido de la generación de imágenes para ilustrar un proceso ya comprendido por el especialista, hasta la incorporación de la visualización en el proceso de descubrimiento de conocimiento.

Actualmente la visualización no sólo incluye imágenes o elementos visuales, también incluye elementos sonoros y cualquier elemento de interacción con los sentidos. Así mismo, esta actividad interviene en los diferentes procesos, apoyando las tareas de la investigación a través de la interacción del humano con los dispositivos, instrumentos y computadoras enlazados en redes de telecomunicaciones. Este nuevo estilo de visualización conlleva retos tecnológicos importantes, principalmente su ejecución en tiempo real y el desarrollo de prototipos rápidos y aplicaciones complejas a corto plazo.

La visualización se apoya en una serie de técnicas propias y otras muchas surgidas en áreas afines, como la geometría computacional, la programación matemática, el análisis de señales, entre otras. Muchas de estas técnicas han sido implementadas como software a través de bibliotecas especializadas que están orientadas a tareas; como los gráficos, el control de dispositivos de audio, el procesamiento de imágenes a través de kerneles, etc. La integración de estas bibliotecas o el desarrollo de subconjuntos especializados a llevado a la creación de aplicaciones monolíticas, las exigencias actuales exigen una mejor estrategia para lograr aplicaciones flexibles en su mantenimiento, reuso y evolución.

Para los desarrolladores, su trabajo no es solo crear imágenes para ser mostradas en una pantalla de computadora; sino ahora se les pide nuevas características. Con las herramientas necesarias es posible lograrlo, pero esto ocasiona que los tiempos de desarrollo crezcan de manera alarmante, o que al intentar recortar el tiempo de desarrollo se logren sólo

visualizaciones mediocres, que no cumplen con las expectativas buscadas. También debemos tener en cuenta que el aumento de complejidad en las visualizaciones a requerido mayor desarrollo. Las técnicas de visualización han evolucionando día a día buscando una mejor manera de mostrar los datos para su difusión. Antes podíamos resumir perfectamente la información en imágenes, después se requirió que fuera una secuencia de ellas, después de la secuencia fue que tuviera mayor interacción con el usuario, después de la aplicación de técnicas más complejas de desarrollar, de esta manera fue evolucionando los requerimientos que eran solicitados o necesarios para crear una visualización de calidad. El desarrollo de las técnicas de visualización han crecido también de la misma manera que los requerimientos para las mismas, existen técnicas muy poderosas pero su implementación es de difícil aplicación o estas técnicas solo están pensadas para colaborar con la búsqueda de la mejor manera de realizar la visualización.

Los desarrolladores cuentan con técnicas poderosas de visualización, el manejo del sonido, de manejo de datos, de manejo de periféricos, etc. Pero ahora se enfrenta al dilema en el diseño de las aplicaciones que se vuelven muy complejas a pesar del gran número de herramientas a su alcance. Existen programas comerciales que intentan dar mayor apoyo al desarrollador pero sus costos de adquisición y el costo de licencias son demasiado altos (de 100 mil pesos a un millón, para un sistema completo) como para que todos los centros de investigaciones pequeños tengan acceso a ellos, o estos programas tampoco cumplen con todas las expectativas. Aún con desarrolladores trabajando en estos programas, no se logran realizar todas las visualizaciones con ellos, ya que sólo son especialistas en ciertas áreas pero dejan a las demás sin apoyo, para contar con un buen apoyo a través estos programas se necesitaría una enorme inversión, a demás de no ser una buena solución. Esta solución no es la óptima, ya que si se realizara dependerían los desarrolladores de estos programas y muchos de ellos no pueden trabajar juntos para lograr mejores visualizaciones, es decir se estancaría la creatividad de los desarrolladores a sólo lo que estos programas pueden lograr para las visualizaciones. Además de que la aparición de nuevas técnicas o elementos no se podrán ampliar o implementar en los proyectos por que esto tipo de programas no ofrecen soporte para realizar ampliaciones de los desarrolladores, ya que de requerirse se tendrán que esperar a que estas técnicas sean implementadas para el programa y pagar por conseguir las actualizaciones de estos.

Esto no es lo que pretendemos que los desarrolladores realicen, deseamos que además de implementar nuevas técnicas como la estereoscopia para las visualizaciones se realicen nuevas búsquedas de mejores técnicas no sólo ser usuarios de ellas sino también desarrollador. La técnica de la estereoscopia es una de las más poderosas que tenemos hoy en día ya que a través de ella se han desarrollado nuevas herramientas muy poderosas como son la realidad virtual, entre otras, estas técnicas dan paso a nuevos horizontes para las visualizaciones. Se abre una brecha muy importante en los terrenos de la visualización a través de ellas, ya que la realidad virtual ofrece no sólo el grado de entendimiento de los datos a través de imágenes 3D, sino el grado de interacción deseado para ellas, así como la posibilidad de ampliar el número de datos que no es posible tratar. La realidad aumentada nos permitirá combinar imágenes reales con las realizadas por computadora para lograr una mejor comparación entre ellas, de esta manera se podrá lograr una mejor difusión de la información, entre otras técnicas. No sólo estas técnicas podrían ser beneficiadas de contar

con mayor implementación de nuevas herramientas, aún las animaciones de los eventos serian más comprensibles si contaran con mayor interactividad o un tipo de sonidos para representar ciertas circunstancias.

Con el objeto de lograr implementar estas técnicas se ha buscado experimentar con las bibliotecas que actualmente se cuentan pero aún la interacción entre ellas no es muy explícita para lograr un avance óptimo en ese terreno. Los desarrolladores han invertido grandes cantidades de tiempo en actualizarse en todos los terrenos de la programación para lograr realizar su trabajo de una manera más sencilla sin sacrificar calidad. De esta manera desde el año 1990, se han utilizado las bibliotecas pero no de la manera más óptima, ya que solo algunos tienen el tiempo indispensable para poder desarrollar y actualizarse, pero gracias a estas bibliotecas tenemos una opción digna de tomarse en cuenta. Además de las nuevas funciones que nos proporcionaran las bibliotecas, nos proveen del soporte para obtener más beneficios al utilizar el hardware de una manera más precisa, la única inversión que se realiza en la puesta en práctica de estas bibliotecas es la inversión en tiempo ya que la mayoría de ellas están bajo la licencia *GNU*. Después de realizar un estudio de las funciones y características de las distintas bibliotecas se podría lograr la unión éstas para lograr que su uso se realice de manera más fácil y de esta manera lograr que el trabajo de los desarrolladores sea menos complicado sin perder la calidad del mismo.

Técnicas básicas para la visualización de datos.

La visualización informativa está basada en una combinación de múltiples técnicas, las cuales han sido desarrolladas para representar valores y campos escalares, vectoriales o tensoriales. Estas van desde la visualización de volúmenes, isocontornos e isosuperficies, líneas de flujo de escalares o vectores, topología de vectores o tensores, funciones sobre superficies y características de partes de un conjunto de datos a partir de analogías geométricas o representaciones iconográficas.

Esta multiplicidad de aproximaciones responde a los requerimientos del amplio rango de aplicaciones a áreas tales como dinámica de fluidos, química computacional, fracturas mecánicas, desarrollo de materiales, neurocirujía, ortopedia, diseño de drogas, ciencias de la Tierra, ciencias de la atmósfera, etc. A continuación se presentan las principales técnicas que se usan para generar elementos visuales a partir de datos, para su exploración, interpretación o para su difusión.

Visualizaciones volumétricas con volume_rendering.

El *volume rendering* es una técnica de proyección que produce imágenes de datos volumétricos con el aspecto tridimensional (Drebin *et al.*, 1988). Su principal característica es que produce una vista dependiente del punto de visión pero no extrae la información geométrica de los datos.

Esta técnica se basa en atravezar un rayo de luz por pixel, siguiendo el punto de vista de los datos y acumulando o extrayendo información de los valores de intensidad de los voxeles o celdas.

Isocontornos.

Esta técnica consiste en la extracción de contornos o superficies de valor constante de campos escalares 2D y 3D. (Lorensen *et al.*, 1987) Se usa principalmente para determinar la estructura general del campo escalar y su evolución en el tiempo.

La extracción de curvas isométricas se basa en la evaluación de cada celda de la malla para determinar las intersecciones con el isocontorno de interés y en la conectividad que tienen estos puntos de intersección para trazar la curva.

Capítulo 2

Reconstrucción geométrica de objetos tridimensionales

Extracción de contornos y fronteras en datos volumétricos.

En alguna de las faces de estas técnicas es común que una de las tareas básicas de la visualización sea encontrar y definir las fronteras de objetos o partes dentro de los espacios 2D y 3D. Estos objetos pueden estar definidos por puntos de igual valor (isovalores), formando contornos o superficies (lo cual se denomina isosuperficies), bordes o frontera de un objeto o región homogénea bajo algún criterio (segmentación).

Las tareas son entonces, i) encontrar dentro del espacio definido, todos los puntos con un valor dado y determinar las conexiones que tienen estos puntos para definir cuáles forman un solo objeto y tomar estos puntos como su frontera; ii) encontran la frontera de un objeto optimizando una función que lo describa, definidas algunas características; y iii) ajustar una función a la frontera conocida de un objeto.

Como puede verse, estas tareas ocurren sobre una malla (un conjunto de elementos con una geometría y una topología), la cual puede verse como una red o grafo. También, la tarea puede verse como una búsqueda de valores o funciones óptimos (inclusive, algunos problemas pueden modelarse como ruta más corta, problemas de cobertura, árbol de expansión mínima, etc.). Por lo que la Investigación de Operaciones puede ser aplicada a la visualización.

Por otro lado, diferentes faces de los algoritmos completos, pueden hacer uso de herramientas de la Investigación de Operaciones para eficientar tareas específicas. Por ejemplo, aunque la visualización de una isosuperficie no es *per se* una tarea de optimización, la secuencia de visita de los voxeles para el trazado de la isosuperficie, puede realizarse más rápidamente si se aplica una técnica de Covertura de conjuntos (o alguna otra relacionada con la Optimización). Además, cuando la técnica de isosuperficies se usa como una técnica de interrogación, se convierte en un problema de optimización; ya que se puede buscar la isosuperficie que maximiza su área o se pueden buscar valores extremos dentro de esta superficie.

Isocontornos e Isosuperficies.

Esta técnica consiste en la búsqueda de puntos con un valor dado o isovalor, y su representación geométrica como un contínuo; en el caso de espacios 2D, contornos; mientras que en el espacio 3D son isosuperficies.

De forma general, esta técnica conlleva tres pasos básicos y diferentes algoritmos varian en los detalles de estos tres pasos. Cada paso incluye diversos cálculos que se revisan a detalle.

a) Triangulación de celdas cálculos para determinar el comportamiento de un contorno o una isosuperficie que interseca a una celda dada.

b) Búsqueda de celdas búsqueda de todas las celdas por las que cruza el contorno o superficie.

c) Recorrido de celdas orden del recorrido de las celdas.

A su vez, la triangulación de celdas incluye tres pasos básicos.

Triangulación de celdas

- binarización de vértices
- interpolación para la localización de puntos
- triangulación en cada elemento

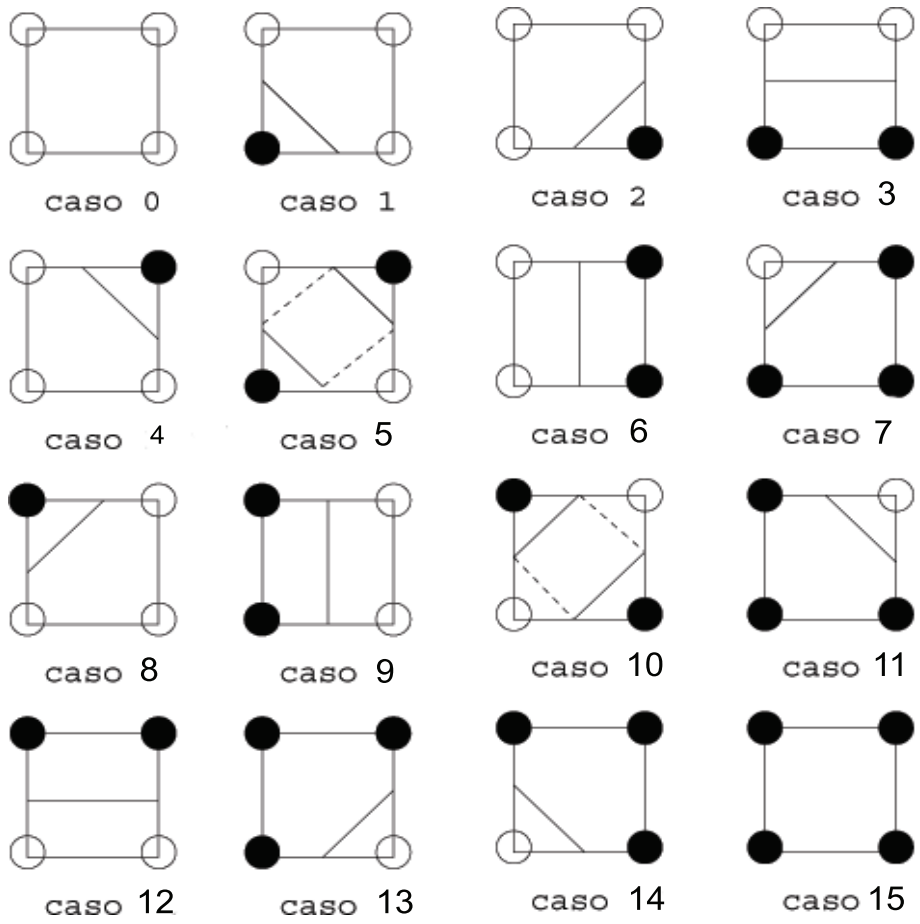


Figura 2.1 Consistencia y correctés; esto es, la configuracion mínima que permite llevar a cabo la tarea definida.

Búsqueda de celdas.

Los contornos pasan a través de sólo algunas celdas de la malla, por lo que una evaluación exhaustiva es ineficiente. Esta tarea es aún más ineficiente cuando se repite la acción de búsqueda del contorno para un nuevo isovalor, haciendo difícil la interactividad.

Por esta razón, es importante iniciar una sesión de exploración de datos con un preprocesamiento del campo escalar, que permita la construcción de una estructura de búsquedas para acelerar las acciones repetidas de isocontornos (o isosuperficies e isovolúmenes).

Búsquedas en el dominio.

Construir un [1] *octree* parcialmente descompuesto; en cada nivel del árbol se guardan los valores mínimo (mín) y máximo (máx) de cada celda.

Búsquedas de rango.

Esta búsqueda tiene la ventaja de ser independiente de la dimensión del dominio, ya que los valores de las celdas se asocian a una función contínua sobre el dominio.

[1]Un octree o árbol de ocho es una estructura no lineal formada con máximo de 8 nodos y en cada nivel se encontrara como máximo 8 ramas por nodo.

O sea, cada celda c está asociada con el conjunto de valores contínuos tomados por la función sobre el dominio:

$$R(c) = [\min_{x \in c} \mathcal{F}(x),\ \max_{x \in c} \mathcal{F}(x)]$$

Árbol de intervalo.

Un árbol de intervalo es implementado como un árbol binario sobre el conjunto de valores de intervalos mín \ máx. Cada nodo mantiene un valor de separación o divergencia s. Las inserciones se hacen al comparar el intervalo con el valor de separación; si el intervalo es menor, la inserción se hace en la izquierda, si es mayor, se hace a la derecha. Cada nodo mantiene dos listas de celdas de expansión. La primera lista es almacenada en orden creciente del valor mínimo, la segunda, en orden decreciente del valor máximo. Ya que los intervalos no se bifurcan en la inserción recursiva, cada intervalo es guardado únicamente dos veces y la complegidad de almacenamiento es $\mathcal{O}(n_s)$, como se muestra el la figura 2.2.

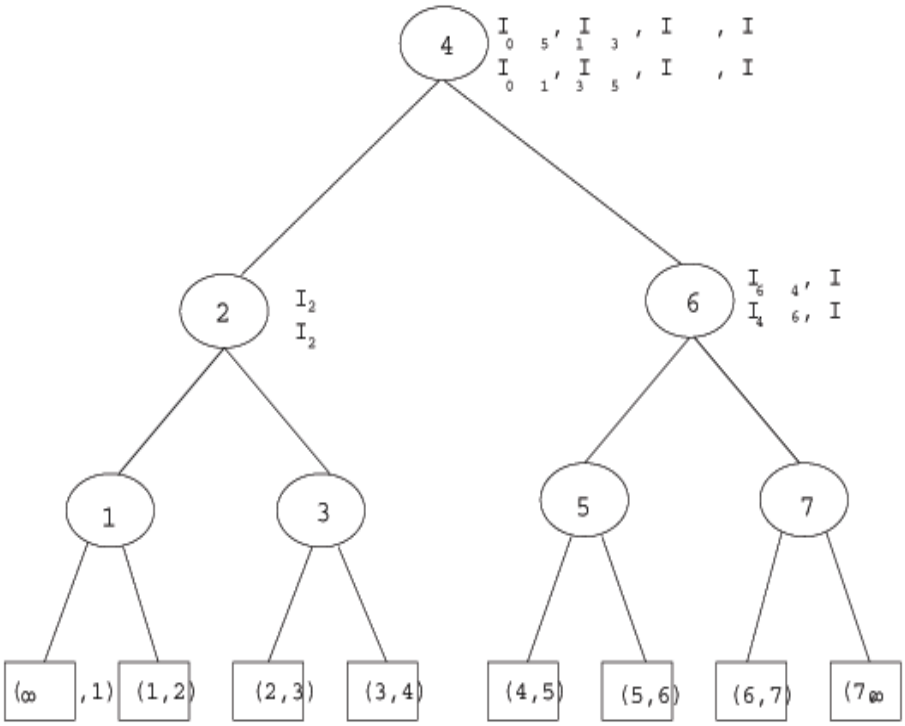


Figura 2.2. Árbol de intervalos, es la implementación de un árbol binario sobre el conjunto de valores de intervalos mín \ máx.

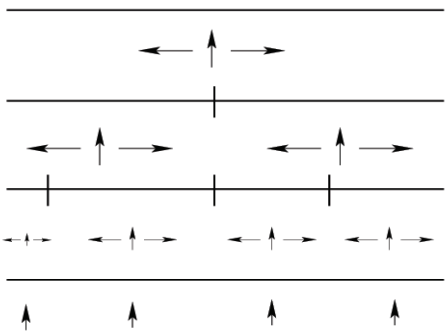


Figura 2.3. Estructura de intervalos,para las inserciones se comparar el intervalo con el valor de separación; si el intervalo es menor, se coloca en la izquierda, si es mayor, se hace a la derecha.

Árbol de segmentos.

Es un árbol de búsqueda binaria sobre el conjunto de valores mínimos y máximos de todas las celdas semillas. Este árbol guarda los segmentos como una multiresolución jerarquizada de intervalos, donde la raíz representa la línea infinita y cada nodo divide el intervalo padre en el valor de separación. Cuando se inserta un segmento en el árbol, es recursivamente partido y propagado hacia abajo en el árbol, para ser insertado en el grupo de nodos cuyos intervalos suman colectivamente el rango total del segmento. Cada identificador de segmento guarda almenos $\mathcal{O}(\log n_u)$ veces, donde $\log n_u$ es la altura o profundidad del árbol, teniendo como consecuencia una complejidad de almacenamiento de $\mathcal{O}(n_s \log n_u)$ para el peor de los casos. La complejidad de búsqueda para k celdas intersectadas por un isovalor dado, es $\mathcal{O}(k+\log n_u)$, como se muestra en la figura 2.4.

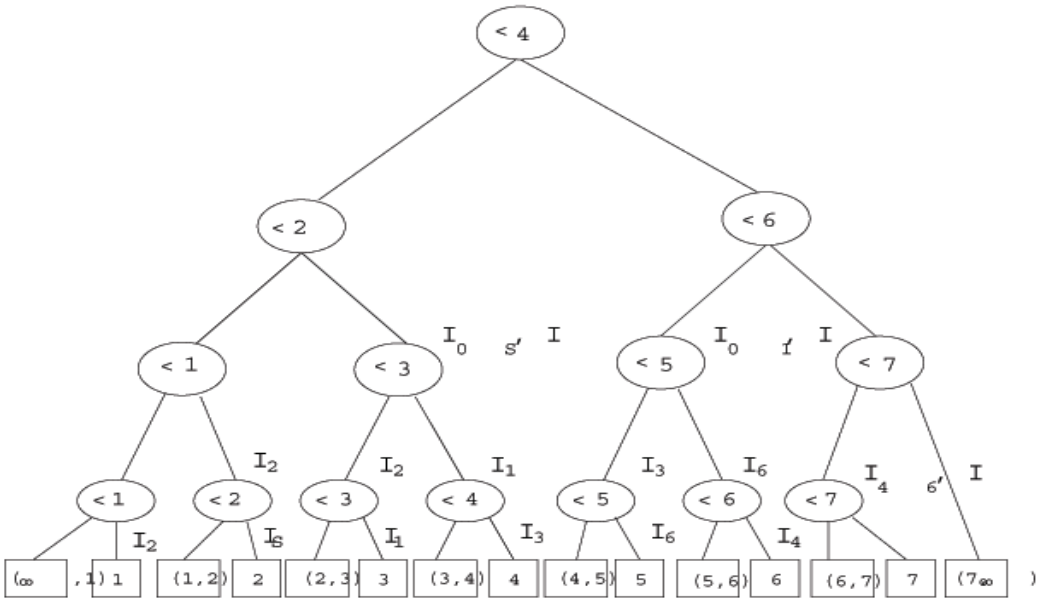


Figura 2.4. Árbol de segmentos.

Búsqueda de cubos.

Muchos de los datos científicos en imágenes 2D o 3D (p. ej. Tomografías computarizadas) son expresados en enteros con rangos cortos (10 u 8 bits); y representados en mallas regulares y mallas cartesianas. Estos formatos de datos permiten que una estrategia de búsqueda simple en cubos, sea eficiente. En esta estructura de búsqueda se tienen $n_u - 1$ cubos que representan intervalos unitarios $(h, h + 1)$. Se guarda un identificador para cada cubo, el cual es partido por la celda. El peor de los casos tiene una complejidad de almacenemiento de $\mathcal{O}(n_s n_u)$, la cual es inmanejable, pero si se tiene una estrategia para proporcionar un conjunto pequeño de celdas semilla, esta estructura es eficiente en almacenamiento y la búsqueda se logra en $\mathcal{O}(k)$; o sea, líneal en el número de celdas reportadas.

Estructura de búsqueda	Complejidad de almacenamiento	Complejidad de búsqueda
Árbol de intervalo	$\mathcal{O}\ (n_s)$	$\mathcal{O}\ (k + \log n_u)$
Árbol de segmento	$\mathcal{O}\ (n_s \log n_u)$	$\mathcal{O}\ (k + \log n_u)$
Cubos	$\mathcal{O}\ (n_s n_u)$	$\mathcal{O}\ (k)$

Cuadro 2.1. Cuadro comparativo de estructuras de búsqueda.

Recorrido de celdas

Este se refiere al orden de visita de las celdas o elementos para construir la superficie completa. Este orden impacta la eficiencia del algoritmo completo en varias formas.- evitando algunas evaluaciones de consistencia y correctés por el orden de visitas, evitando cálculos de intersecciones a lo largo de orillas compartidas por celdas, al guardar sólo celdas semillas para iniciar los recorridos. Estos múltiples aspectos han dado origen a diversos algoritmos para mejorar la eficiencia de los recorridos; cada cual con bondades y flaquezas en diferentes tipos de casos.

La propagación de contornos es un método de seguimiento de superficies que se basa en la continuidad del campo escalar. Este consiste en proporcionar una celda semilla componente de la superficie buscada y se encuentran todos los componentes de la superficie por "primero profundidad" a través de las caras adyacentes; el algoritmo para cuando se alcanza nuevamente a la celda semilla.

La segunda aproximación se basa en la construcción de un conjunto de semillas. Esta estrategia está muy relacionada con las técnicas de búsqueda de celdas, vistas anteriormente. Permitiendo que la búsqueda primaria sólo incluya a algunos elementos dentro del rango (semillas) y el recorrido encuentra al resto del subconjunto (búsqueda secundaria), por diversas técnicas, incluyendo la propagación de contornos.

El problema de esta estrategia es garantizar que cada componente conectado para cualquier isovalor arbitrario, interseca almenos una celda en el subconjunto, llamado un conjunto semilla. El problema se convierte en tener conjuntos de semillas óptimos o minimales, o sea, suficientes subconjuntos de celdas que garanticen semillas en todos los conjuntos de celdas desconectados.

El costo de implementación y cómputo de conjuntos de semillas optimales es restrictivo, por lo que es importante implementar heurísticas suficientes para satisfacer la exigencia y que sean "buenas" soluciones.

Desde el momento de la binarización, las celdas pueden ser etiquetadas con un atributo que indique a que subconjunto pertenecen o la conectividad con otras celdas.

ALGORITMO GLOTÓN.

Una estrategia glotona es cuando se selecciona una celda del rango (teniendo como banco de semillas a todas las celdas) y se van tomando las demás celdas por propagación de contorno; cuando se cubre el contorno, se toman las celdas semillas restantes (se puede usar un bit marcador) y se continua hasta cubrir todas las celdas del banco.

CHAIN CODE.

Una aproximación que puede ser costosa en uso de memoria y en cálculo en el pre-procesamiento, o preparación del análisis; aunque muy efectiva para proveer semillas de búsqueda, es el uso de cadenas que describen los recorridos de líneas envolventes de regiones homogéneas; o *chain codes*.

La representación con *Chain code* es una representación común para líneas dentro de imágenes de pixeles o raster. Esta representación usa el punto de inicio de la línea y una secuencia de números en el intervalo $[0 \ldots 7]$ la cual indica los puntos siguientes en la línea como se muestra en la figura 2.5.

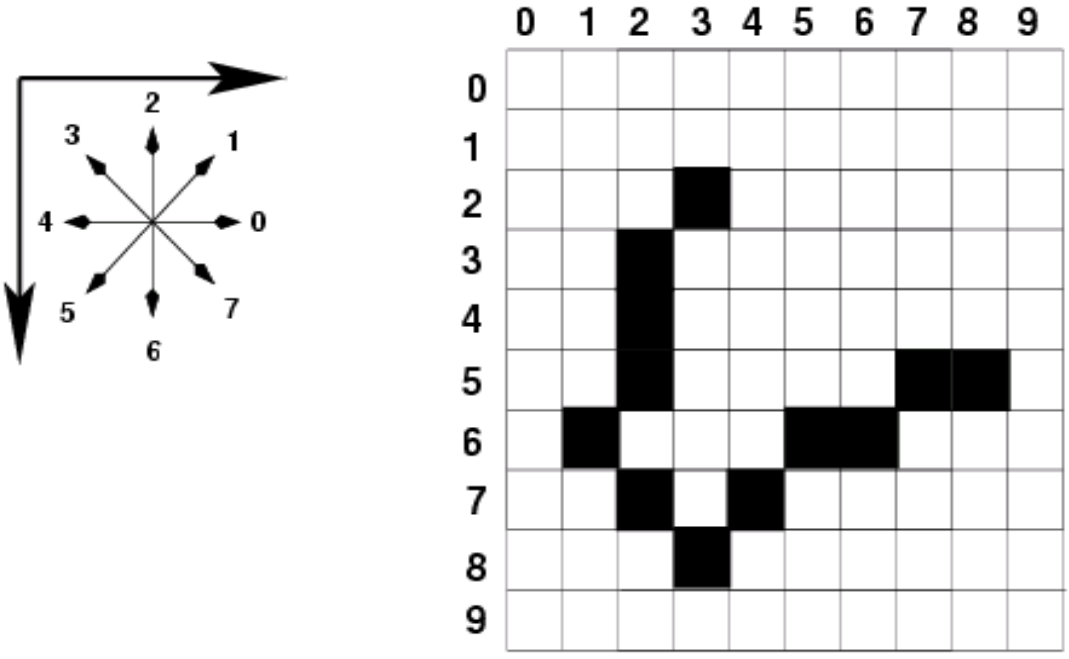


Figura 2.5. Representación de *Chain code*.
El *chain code* para la linea negra es (3,2),5,6,6,5,7,7,1,1,0,1,0

Selección automática de isovalores y posibles valores de frontera.

La mayoría de las técnicas de visualización exploratoria requieren que se les proporcionen datos iniciales de exploración y parámetros: un isovalor, una región de búsqueda, las características de un objeto o patrón a buscar dentro de un espacio, etc.

Estos datos y parámetros se introducen de forma interactiva hasta alcanzar el objetivo o una visualización satisfactoria. Por lo que esta fase de los cálculos es crítica y una aproximación que puede dar buenos resultados para mantener una interactividad rápida, es el seleccionar estos datos de forma automática.

Esta selección automática tiene un costo computacional alto, pero puede realizarse en preprocesamientos cuando inicia la sesión de visualización y servir (a través de un esquema de aprendizaje) para apoyar en la interactividad posterior; por ejemplo, encontrar valores iniciales para buscar varios contornos y el usuario afina un valor de búsqueda para un contorno, los demás valores de los otros contornos, pueden ajustarse considerando el *expertise* del usuario y el aprendizaje; así mismo, aprender de ambas entradas: algoritmo y usuario.

Estos cálculos deben ser rápidos (durante el preprocesamiento) y extraer suficiente información para que durante la face de interacción más intensiva, los cálculos sean más simples y efectivos. Por otro lado, se supone que aún no se conocen suficientemente los datos y las técnicas sólo pueden usar información general o global de los datos (como el histograma, el correlograma, tendencias centrales y disperción) y conforme se avanza en la visualización, las técnicas van incluyendo información de regiones y local; hasta llegar a un nivel en el que se base en hipótesis específicas y se puedan usar aproximaciones de visualización basada en hipótesis (físicas, geométricas, heurísticas, estadísticas, entre otras), las cuales generalmente pueden ser planteadas en términos estadísticos.

La selección automática de isovalores puede aprovechar la información del histograma de frecuencias de los valores de brillantés. Una imagen está formada por valores de intensidad sobre una malla cartesiana. Típicamente una imagen en tonos de grises tiene 256 niveles de intensidad, los cuales se colocan en un rango de 0 a 255 y pueden ser representados por ocho bits cada pixel.

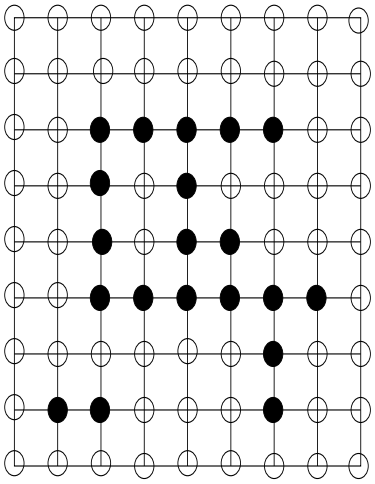
Si se obtiene un histograma de una imagen, es posible localizar varias características de la imagen, como son el valor modal (pueden encontrarse varias modas), el mínimo y el máximo, valles y crestas en la función de densidad, etc. Estos valores pueden ser usados para encontrar regiones envueltas por valores críticos o hacer rangos que representan regiones uniformes, y sus fronteras forman las curvas de interés. La selección de puntos críticos en el histograma puede realizarse siguiendo diferentes criterios y con muy diversas técnicas.

0	2	2	2	2	0	0	0
0	3	4	5	5	5	5	3
1	4	7	7	7	7	7	3
2	4	7	3	8	5	5	2
2	4	7	3	8	8	2	2
3	4	7	8	7	7	7	8
4	3	3	4	2	2	7	5
4	7	8	4	4	5	7	3

A)

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

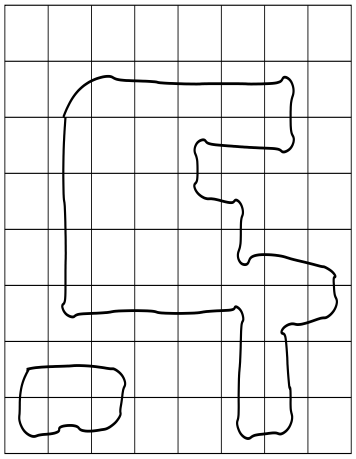
B)



C)

X	X	X	X	X	X	X	X
X							X
X							X
X						X	X
X							
X							
			X	X			X
			X	X			X

D)



E)

Figura 2.6 Ilustración de los pasos del algoritmo de*isosuperficie*.
A) Representación de una imagen por sus valores de pixeles.
B) Representación de una imagen por sus indices.
C) Representación de una imagen en cuadro binario.
D) Representación de la imagen en la elección de zonas a utilizar.
E) Representación del resultado en *isosuperficie*.

Volumen rendering

La adquisición de datos volumétricos consiste en un muestreo, generalmente para una malla cartesiana regular, de un campo escalar, vectorial o tensorial del espacio dentro del cual se encuentra un cuerpo sujeto a una fuente de radiación. Dicho registro representa la respuesta del cuerpo y revela información sobre diferentes propiedades del objeto. Sin embargo, para visualizar estas propiedades (por ejemplo, la estructura, la temperatura o el gradiente del movimiento *browniano* de las moléculas de agua en los tejidos de un ser vivo) es necesario procesar los datos y asociarles propiedades ópticas a los valores muestreados o encontrar geometrías y dibujarlas, a través de técnicas de *isosuperficies*.

Una alternativa para un dibujado informativo de un volumen de datos (esto es, la generación de una imagen tridimensional que revele las características que deseamos y oculte la información visual, los datos o ruido no deseado) es la técnica del *volume rendering* directo. Esta técnica consiste en una proyección del volumen de datos sobre una pantalla en dos dimensiones, que refleje la acumulación de los valores perpendiculares al plano (o en algún ángulo dado) del volumen (generalmente filtrados siguiendo una intensión de modificación de los valores; por ejemplo, obteniendo el gradiente para resaltar las fronteras de los objetos y atenuar el ruido o las secciones homogéneas). Estos valores de acumulación son transformados en propiedades ópticas y desplegados.

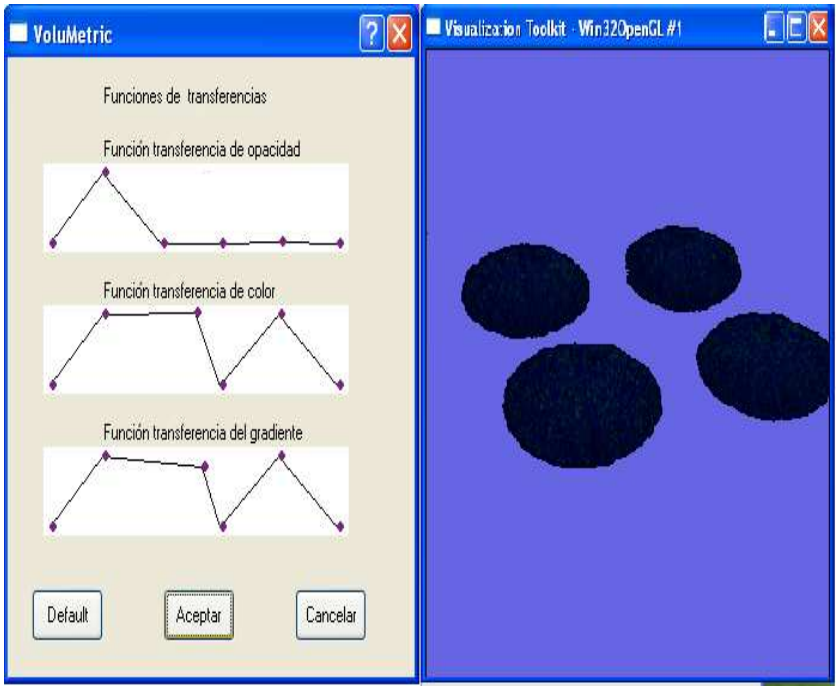


Figura 2.7. Ejemplo de función de transferencia.

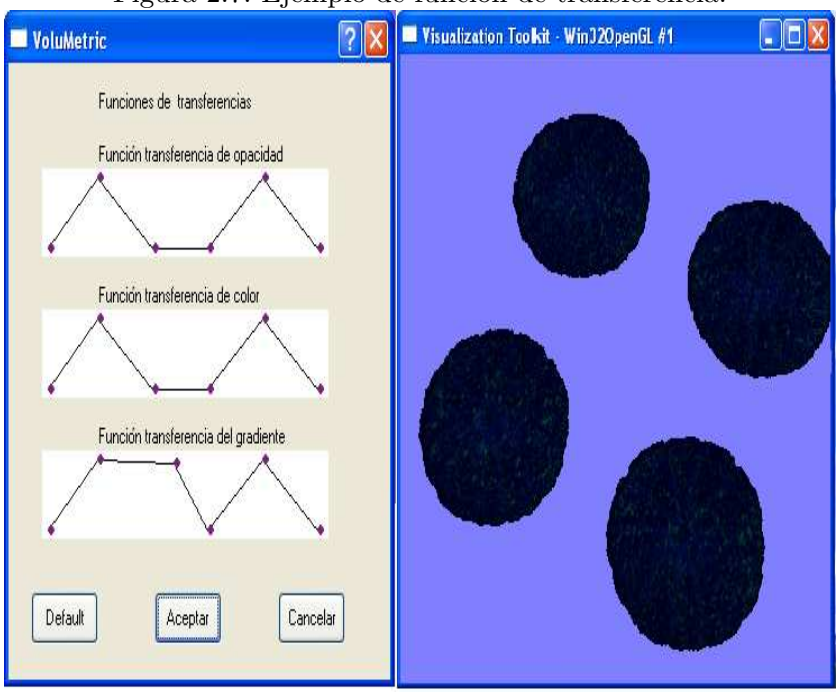


Figura 2.8. Ejemplo de función de transferencia.

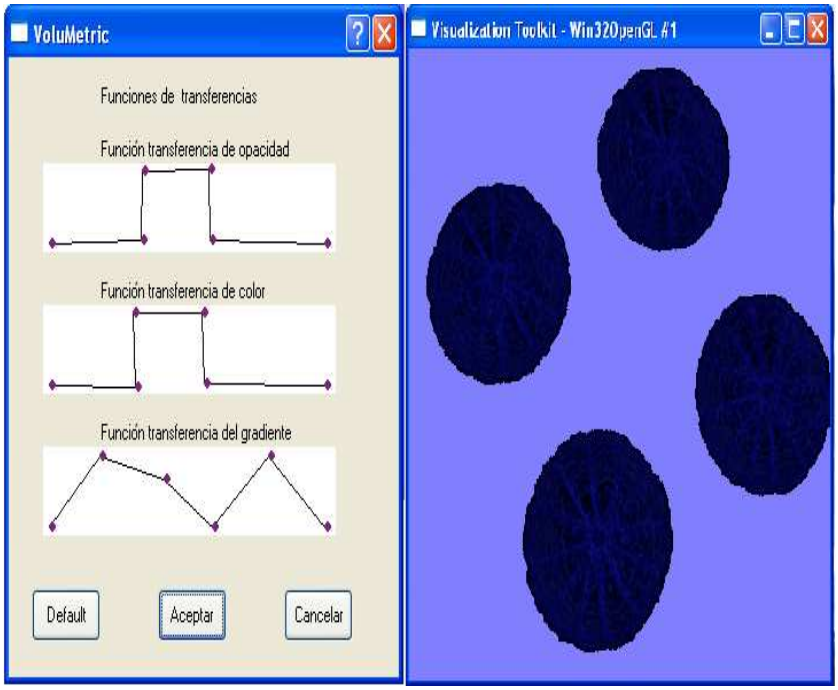


Figura 2.9. Ejemplo de función de transferencia.

Estas transformaciones se llevan a cabo a través de funciones de transferencia, las cuales son funciones que asignan propiedades ópticas a los valores numéricos del conjunto de datos; las propiedades ópticas básicas son opacidad, color y emitancia.

Los parámetros o la forma adecuados (aquellos que revelan la información deseada, un dibujado informativo) de las funciones de transferencia son dependientes del conjunto de datos; esto es, para cada conjunto de datos debemos ajustar los parámetros de las funciones de transferencia. Las características de la forma de la función de transferencia pueden ser modificadas de manera intuitiva, ya que su gráfica indica la intensidad de modificación contra los valores de los datos originales escalados en el eje de las ordenadas.

Encontrar los parámetros adecuados para cada conjunto de datos es una tarea muy laboriosa que requiere de experiencia. Por ello, se buscan técnicas semiautomáticas que obtengan valores iniciales para los parámetros, cercanos a los adecuados.

Momentos de orden mayor

Los momentos de orden mayor son estimadores estadísticos (independientes del modelo) de la tendencia central de una distribución de datos. Usualmente, el valor alrededor del cual se distribuye un conjunto de datos $(x_1, x_2, x_3, \ldots, x_N)$ y son medidos por la media M de la distribución, que está dada por

$$M = \frac{1}{N}\sum_{j=1}^{N} x_j$$

Los momentos de orden mayor de la misma distribución están definidos como

$$m_k = \frac{1}{N}\sum_{j=1}^{N}(x_j - M)^k$$

donde m_k es el momento de orden k. El segundo momento evalua la varianza, mientras que el tercero y cuarto evaluan el sesgo (*skewness*) y la kurtosis, respectivamente. Una muestra obtenida de la distribución Normal, tendrá kurtosis igual a cero. Uno puede caracterizar la forma de cualquier función conociendo los momentos centrales, pudiendo calcular un número infinito de estos.

Preservación de momentos

Los momentos son una propiedad de la imagen que refleja la forma del histograma y otras características de la función de densidad. Una estrategia es buscar que la imagen resultante de la segmentación basada en pixeles, preserve los mismos momentos que la imagen original (con tantas clases como valores de grises), con el histograma resultante de colapsar los rangos de los subconjuntos (el histograma resultante de las nuevas clases resultantes). Para lo cual es necesario estimar los momentos de la imagen original y encontrar las particiones (los valores que cortan el histograma en subconjuntos, formando rangos de valores de grises) que preservan los momentos, pero con las categorías o clases reducidas.

Esta aproximación puede resultar en una partición automática y determinista, sin iteraciones o búsqueda. Inclusive, para particiones en $N < 5$ subconjuntos, se pueden encontrar soluciones analíticas.

Dado una imagen f con n pixeles cuyos valores de grises en el pixel (x, y) se denotan como $f(x, y)$, se tiene como objetivo partir f en dos clases de pixeles, respetando el orden de los valores.

El $i-$ésimo momento m_i de f está definido como

$$m_i = \frac{1}{n} \sum_x \sum_y f^i(x, y), \quad i = 1, 2, 3, \ldots$$

Los momentos también pueden ser calculados del histograma de f como

$$m_i = \frac{1}{n} \sum_j n_j (z_j)^i = \sum_j p_j (z_j)^i$$

donde n_j es el número total de pixeles en f con el valor de gris z_j y $p_j = \frac{n_j}{n}$. El m_0 está definido como 1.

Si se considera que la imagen está formada por dos clases de objetos (un *background* y un objeto) y estos objetos generaron en la formación de la imagen, un rango de valores de grises Ω_1 y Ω_2, respectivamente; cuyos valores están ordenados ($f(x, y)_{\Omega_1} < f(x, y)_{\Omega_2}$). Por ejemplo, la imagen original digital es una versión borrosa de la imagen original física o real, se desea remplazar los valores $f(x, y)$ por solo dos valores (z_0 y z_1).

Entonces, los primeros tres momentos de la imagen f son preservados en la imagen binaria resultante g. Los p_0 y p_1 denotan las fracciones de los pixeles por debajo del punto de partición y por arriba de este umbral, respectivamente. Siendo los tres momentos de g

$$m_i = \sum_{j=0}^{1} p_j (z_j)^i \ , \quad i = 1, 2, 3$$

Para preservar los primeros tres momentos en g, se debe cumplir la siguiente igualdad

$$m'_i = m_i \ , \quad i = 1, 2, 3$$

Ya que $p_0 + p_1 = 1$, resultan cuatro igualdades de lo anterior

$$\begin{aligned} p_0 z_0^0 + p_1 z_1^0 &= m_0, \\ p_0 z_0^1 + p_1 z_1^1 &= m_1, \\ p_0 z_0^2 + p_1 z_1^2 &= m_2, \\ p_0 z_0^3 + p_1 z_1^3 &= m_3. \end{aligned}$$

donde m_i con $i = 1, 2, 3$ son calculados por cualquiera de las fórmulas dadas, mientras que $m_0 = 1$. Para encontrar el valor de t en el que se parte el conjunto, primero se debe resolver el sistema de ecuaciones para obtener p_0 y p_1; segundo, se escoge una t tal que la cota superior valga

$$p_0 = \frac{1}{n} \sum_{z_j \leq t} n_j$$

De forma equivalente, si el problema es partir el conjunto parcialmente ordenado en más de dos subconjuntos, se tiene un problema multiumbral. Generalizando, si se desea partir los valores de los pixeles en N clases, se requieren $N-1$ valores umbrales $(t_1, t_2, \ldots, t_{N-1})$.

Si z_i denota los valores de grises representativos de la clase de pixel $i-$ésima, y p_i denota la fracción de pixeles en la $i-$ésima clase; para preservar los primeros $2N-1$ momentos de f y usando el hecho de que la suma de todos los p_i es igual a 1, se tiene el siguiente conjunto de $2N$ ecuaciones:

$$\begin{aligned} p_0 z_0^0 + p_1 z_1^0 + \cdots + p_{N-1} z_{N-1}^0 &= m_0, \\ p_0 z_0^1 + p_1 z_1^1 + \cdots + p_{N-1} z_{N-1}^1 &= m_1, \\ &\vdots \\ p_0 z_0^{2N-1} + p_1 z_1^{2N-1} + \cdots + p_{N-1} z_{N-1}^{2N-1} &= m_{2N-1}. \end{aligned}$$

el cual puede resolverse para obtener todos los p_i y z_i, $i = 0, 1, \ldots, N-1$.

Una vez obtenidos todos los valores p_i, la estimación de los t_i puede encontrarse del histograma de f, al escoger t_1 según la cota superior de p_0, t_2 para $(p_0 + p_1)$; o sea, t_i según la cota superior de $(p_0 + p_1 + \cdots + p_{i-1})$, del histograma de f.

Un método de solución para este problema es la consecución de los tres siguientes cálculos (Tabatabai, 1981):

(1) resolver el siguiente sistema de ecuaciones para obtener un conjunto de valores auxiliares c_0, c_1, …, c_{N-1}

$$\begin{aligned} c_0 m_0 + c_1 m_1 + \cdots + c_{N-1} m_{N-1} &= -m_N \\ c_0 m_1 + c_1 m_2 + \cdots + c_{N-1} m_N &= -m_{N+1} \\ &\vdots \\ c_0 m_{N-1} + c_1 m_N + \cdots + c_{N-1} m_{2N-2} &= -m_{2N-1} \end{aligned}$$

(2) resolver la siguiente ecuación polinomial para obtener los valores de grises representativos $z_0, z_1, \ldots, z_{N-1}$

$$z^N + c_{N-1}z^{N-1} + \cdots + c_1 z + c_0 = 0$$

(3) sustituir todos los z_i valores obtenidos, en las primeras N ecuaciones de preservación de momentos ($p_0 z_0^0 + p_1 z_1^0 + \cdots + p_N z_N^1 = m_0$, etc.) y resolver el sistema de ecuaciones para encontrar $p_0, p_1, \ldots, p_{N-1}$.

Para garantizar que las N soluciones de la ecuación polinomial son todas distintas y de valores reales, deben existir almenos N valores de grises distintos en la imagen de entrada f. Para $N \geq 5$, se requieren métodos numéricos como el Método de *Newton*.

Soluciones exactas.

Solución para una binomial $(t_i, i = 1)$.

(1)

$$c_d = \begin{vmatrix} m_0 & m_1 \\ m_1 & m_2 \end{vmatrix} \; ; \quad c_0 = \frac{1}{c_d} \begin{vmatrix} -m_2 & m_1 \\ -m_3 & m_2 \end{vmatrix} \; ;$$

$$c_1 = \frac{1}{c_d} \begin{vmatrix} m_0 & -m_2 \\ m_1 & -m_3 \end{vmatrix} .$$

(2)

$$z_0 = \frac{1}{2}(-c_1 - (c_1^2 - 4c_0)^{\frac{1}{2}});$$

$$z_1 = \frac{1}{2}(-c_1 + (c_1^2 - 4c_0)^{\frac{1}{2}});$$

(3)

$$p_d = \begin{vmatrix} 1 & 1 \\ z_0 & z_1 \end{vmatrix} \; ; \quad p_0 = \frac{1}{p_d} \begin{vmatrix} 1 & 1 \\ m_1 & z_1 \end{vmatrix} \; ; p_1 = 1 - p_0$$

Solución para tres clases de pixeles $(t_i, i = 2)$.

(1)

$$c_d = \begin{vmatrix} m_0 & m_1 & m_2 \\ m_1 & m_2 & m_3 \\ m_2 & m_3 & m_4 \end{vmatrix} \; ; \quad c_0 = \frac{1}{c_d} \begin{vmatrix} -m_3 & m_1 & m_2 \\ -m_4 & m_2 & m_3 \\ -m_5 & m_3 & m_4 \end{vmatrix} \; ;$$

$$c_1 = \frac{1}{c_d} \begin{vmatrix} m_0 & -m_3 & m_2 \\ m_1 & -m_4 & m_3 \\ m_2 & -m_5 & m_4 \end{vmatrix} \; ; \quad c_2 = \frac{1}{c_d} \begin{vmatrix} m_0 & m_1 & -m_3 \\ m_1 & m_2 & -m_4 \\ m_2 & m_3 & -m_5 \end{vmatrix} .$$

(2)

$$\begin{aligned} z_0 &= -\frac{c_2}{3} - A - B \; ; \\ z_1 &= -\frac{c_2}{3} - w_1 A - w_2 B \; ; \\ z_2 &= -\frac{c_2}{3} - w_2 A - w_1 B \; , \end{aligned} \tag{2.1}$$

donde A, B, w_1 y w_2 son

$$A = \left\{ \left(\frac{c_0}{2} - \frac{c_1 c_2}{6} + \frac{c_2^3}{27} \right) - \left[\left(\frac{c_0}{2} - \frac{c_1 c_2}{6} + \frac{c_2^3}{27} \right)^2 + \left(\frac{c_1}{3} - \frac{c_2^2}{9} \right)^3 \right]^{\frac{1}{2}} \right\}^{\frac{1}{3}} \; ;$$

$$B = -\frac{(\frac{c_1}{3} - \frac{c_2^2}{9})}{A} \; ; \quad w_1 = -\frac{1}{2} + i \left(\frac{\sqrt{3}}{2} \right) \; ;$$

$$w_2 = -\frac{1}{2} - i \left(\frac{\sqrt{3}}{2} \right) \; ; \quad i = \sqrt{-1} \; .$$

(3)

$$p_d = \begin{vmatrix} 1 & 1 & 1 \\ z_0 & z_1 & z_2 \\ z_0^2 & z_0^2 & z_0^2 \end{vmatrix} \; ; \quad p_0 = \frac{1}{p_d} \begin{vmatrix} m_0 & 1 & 1 \\ m & z_1 & z_2 \\ m_2 & z_1^2 & z_2^2 \end{vmatrix} \; ;$$

$$p_1 = \frac{1}{p_d} \begin{vmatrix} 1 & m_0 & 1 \\ z_0 & m_1 & z_2 \\ z_0^2 & m_2 & z_2^2 \end{vmatrix} \; ; \quad p_2 = 1 - p_0 - p_1.$$

Momentos locales.

(Tenginakai *et al.*, 2001) propusieron un modelo de frontera basado en Momentos de orden mayor locales; para lo cual derivan las ecuaciones generales de los momentos. Si se supone una ventana W de $w \times w$ centrada en un punto P; si se hace W suficientemente pequeña para contener únicamente una orilla. Si está presente una orilla, se tiene que de la w^2 partículas (pixeles o voxeles), n pertenecen a una fase C_1 y m a la otra C_2. De lo anterior se tienen los siguientes resultados.

- Si $m = 0$ o $n = 0$, no hay una orilla en w
- Si $m > 0$ y $n > 0$, existe una orilla en W
  - si $n > m$, P se encuentra a la izquierda de la orilla
  - si $m > n$, P se encuentra a la derecha de la orilla
  - si $n = m$, P se encuentra sobre la orilla

Entonces, los momentos locales son calculados como

$$LM = \frac{1}{w^2} \sum x \ , \quad \forall x \in W$$

$$m_k = \frac{1}{w^2} \sum (x - LM)^k \ , \quad \forall x \in W$$

Poniendo estas ecuaciones para los momentos locales en términos del modelo de frontera, se reescriben como

$$LM = \frac{nC_1 + mC_2}{(n+m)}$$

donde C_1 y C_2 toman los valores de la muestra (pixeles o voxeles). Mientras que los momentos de orden mayor son derivados como

$$\begin{aligned} m_k &= \tfrac{1}{(n+m)} \left\{ n \left[C_1 - LM \right]^k + m \left[C_2 - LM \right]^k \right\} \\ &= \tfrac{1}{(n+m)} \left\{ n \left[\tfrac{m(C_1 - C_2)}{(n+m)} \right]^k + m \left[\tfrac{n(C_2 - C_1)}{(n+m)} \right]^k \right\} \\ &= \tfrac{(C_2 - C_1)^k}{(n+m)^{k+1}} (mn^k + (-1)^k nm^k) \end{aligned}$$

Haciendo $\Delta = C_2 - C_1$, y sustituyendo $n + m = w^2$, se tiene

$$m_k = \frac{\Delta^k}{(w^2)^{k+1}} (mn^k + (-1)^k nm^k)$$

Analizando la ecuación general de momentos, se tienen los siguientes resultados:

- Si $\Delta \neq$, w es una región con orilla.
- Una región con frontera u orilla implica la existencia de momentos de orden par, no cero.
- Un momento de orden no par será cero si está asociado con una región con frontera sobre P y será no cero si la región es una frontera izquierda o derecha.

- Los momentos de orden par o impar son cero en regiones sin orilla.

Para los diferentes órdenes se tienen los siguientes resultados.

- La ecuación de m_2 tiene un máximo en $m = n = \frac{w^2}{2}$, cuando la región W tiene una frontera que pasa por P. El valor de m_2 en el máximo es de $0{,}25\Delta^2$ (el cual es idependiente del tamaño de W y de la distribución presente en la región).

- La ecuación para m_3 tiene un máximo en $0{,}21w^2$ y un mínimo en $0{,}79w^2$, medidos desde el centro de la celda. El valor de m_3 sobre el máximo es $0{,}1\Delta^2$ y sobre el mínimo es $-0{,}1\Delta^2$. También presenta un cero para $m = n = \frac{w^2}{2}$, para el caso en el que la región paso sobre P.

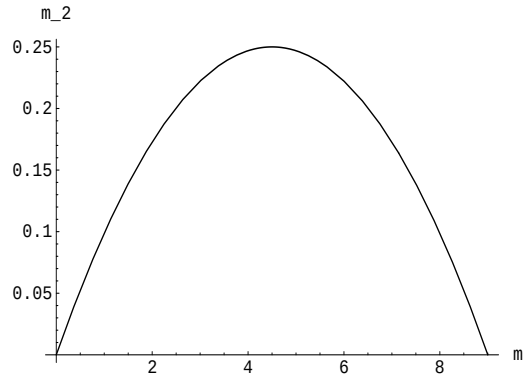
- La ecuación para m_4 tiene dos máximos en $0{,}21w^2$ y $0{,}79w^2$; los valores para m_4 sobre estos máximos son ambos $0{,}083\Delta^4$. También presenta un mínimo local en $m = n = \frac{w^2}{2}$, para una región en la que la frontera pasa sobre P; siendo el valor de m_4 en este punto mínimo, de $0{,}0625\Delta^4$.

- El sesgo s, caracteriza el grado de asimetría de la distribución alrededor de la media. El sesgo es monotónico decreciente alcanzando el cero en $m = n = \frac{w^2}{2}$, por lo que resulta que una región con frontera izquierda tiene un sesgo positivo; mientras que una con frontera derecha, lo tiene negativo.

- La kurtosis k, mide la picudez o aplanamiento de la distribución. La kurtosis tiene un mínimo en $m = n = \frac{w^2}{2}$ y el valor de k en este punto es de -2; teniendo que para regiones con frontera que pasa sobre P, la kurtosis es constante.

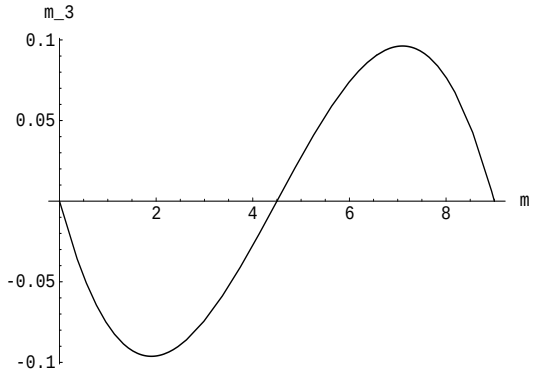
Usando algunos o todos los resultados, es fácil detectar todas las regiones en el conjunto de datos, que pasan por P; o sea, la frontera. Siendo que es posible detectar las regiones con frontera, es posible obtener indirectamente la localización espacial de las fronteras en el conjunto de datos, así como los valores de estas fronteras o valores sobresalientes.

De esta forma, la detección de isovalores sobresalientes en el conjunto de datos se realiza en el espacio muestreado, más que en el dominio espacial. Los momentos de orden mayor permiten detectar fronteras en el dominio espacial; si ahora se explora el comportamiento de los momentos graficando sus valores contra los valores de muestreo en sus puntos asociados, ya que se preserva la relación entre fronteras izquierdas, derechas y fronteras sobre P.



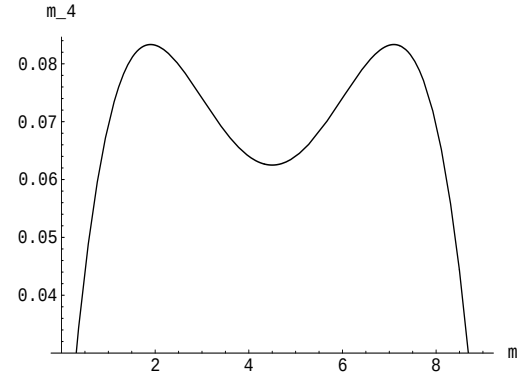
$$m_2 = \frac{\Delta^2}{(w^2)^2}(mn)$$

Figura 2.10. Gráfica y ecuación del momento 2.



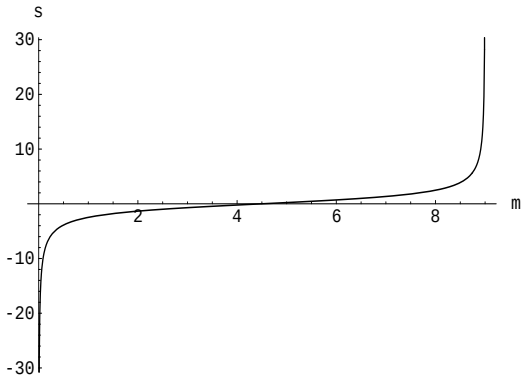
$$m_3 = \frac{\Delta^3}{(w^2)^3}(mn)(n-m)$$

Figura 2.11. Gráfica y ecuación del momento 3.



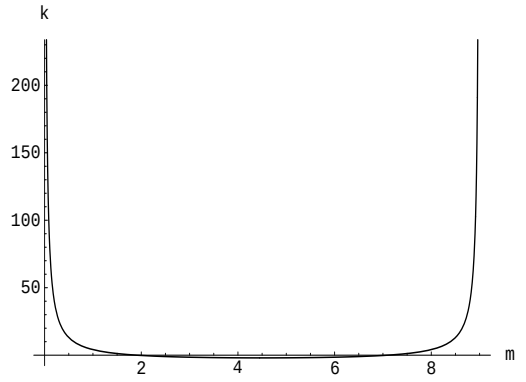
$$m_4 = \frac{\Delta^4}{(w^2)^4}(mn)(w^4 - 3mn)$$

Figura 2.12. Gráfica y ecuación del momento 4.



$$s = \frac{m_3}{(w_2)^{\frac{3}{2}}} = \frac{(n-m)}{\sqrt{nm}} \,|_w \text{sesgo}$$

Figura 2.13. Gráfica y ecuación del momento 5.



$$k = \frac{m_4 - 3m_2^2}{(m_2^2)} = \frac{w^4}{mn} - 6 \,|_w \text{kurtosis}$$

Figura 2.14. Gráfica y ecuación del momento 6.

Capítulo 4
Aplicación

La evaluación del kernel se llevó a cabo a través de la implementación de una aplicación de visualización científica. La especificación del sistema partió de los requerimientos de un grupo de usuarios que trabajan con conjuntos de datos tipo tomografía computada(campos escalares dados en una malla cartesiana regular). Sin embargo, el mayor grado de complejidad resulta de la diversidad de usos y de las habilidades técnicas en computación, de los usuarios finales.

Estos usuarios pertenecen al área biomédica y tienen diversos objetivos particulares para las imágenes tridimensionales generadas; por ejemplo, algunos tienen un interés en difusión, mientras otros las usan en docencia o investigación. Algunos publican sus resultados en sitios web en formatos tridimensionales, otros como imágenes digitales, mientras que algunos los requieren para publicaciones impresas en color.

La aplicación es un sistema de visualización de estructuras en campos escalares, cubriendo las tareas de detección de las estructuras, exploración y preparación para la difusión y docencia de los objetos gráficos generados. Esta aplicación permite la lectura de datos en diferentes formatos (jpeg, bmp, tiff, png, pgm, gif) que son soportados por *Qt*, además de los formatos crudos de 8 bits y 16 bits.

Las técnicas de visualización científica implementadas para satisfacer los requerimientos establecidos fueron *isosuperficies*, *volumen rendering*, cálculos de momentos de órden superior. La *isosuperficie* fue implementada aprovechando la biblioteca de *vtk* y se realiza para la reconstrucción geométrica de los objetos dentro del campo, el *volume rendering* también fue implementado con la biblioteca de *vtk*. La interfaz gráfica fue desarrollada completamente en *Qt*. Los códigos de control se implementaron en *C++*. El despliegue de la técnica es realizado con *OpenGL* en un *canvas* de *Qt*.

Lectura de datos

Los formatos

Los datos volumétricos obtenidos a través de diversos dispositivos (tomógrafos, microscopios confocales, etc.), en particular las imágenes biomédicas son almacenados en forma de una pila de imágenes digitales en dos dimensiones, donde cada una representa un corte con muestras discretas en 2D, del campo tridimensional. Otra forma común es el almacenamiento de un arreglo tridimensional (o alguna estructura de acomodo de los datos)

para guardar los datos de representación del campo volumétrico discretizado a través de una malla cartesiana regular.

Dicho campo puede ser una discretización de un espacio escalar, vectorial o tensorial, cuya digitalización consiste en la discretización espacial o muestreo (la imagen analógica es convertida en muestras dentro de una rejilla cartesiana; o sea, en píxeles) y una discretización numérica o cuantización (una representación numérica discreta o profundidad de pixel; o sea, cuántos valores diferentes podemos usar dentro de la escala, lo que se traduce en una cantidad de bits requeridos por pixel).

Además de estas características fundamentales de los datos, los diferentes sistemas de cómputo guardan los valores arriba mencionados, de muy diversas maneras o formatos; con el objetivo de reducir el espacio ocupado en disco, de proteger los datos poniéndolos en un formato propietario o simplemente para seguir reglas o convenciones para el intercambio de información.

De esta forma, la diversidad de tipos de imágenes desde el punto de vista meramente informático, es muy amplia; las imágenes son proporcionadas con diferente resolución espacial (los típicos son 640×480, 512×512, 800×600, 1024×768, 1024×1024 y en general resoluciones con base a potencia de dos) y representación numérica (8, 11, 16, 24 o 32 bits); así como diferentes acomodos de los datos que resultan en formatos de imagen (gif, tiff, pgn, png, jpeg, tga, dicom, etc.).

Cualquiera de las dos opciones arriba mencionadas (datos acomodados en una malla tridimensional o datos proporcionados como una pila de imágenes en dos dimensiones) requiere que los datos sean cargados en una malla tridimensional para realizar las operaciones necesarias para la generación de objetos gráficos. Por ello es necesario contar con decodificadores de los diferentes formatos o métodos de lectura para los diferentes formatos de los datos originales.

La estrategia adoptada, a nivel de kernel, fue separar la tarea de la carga de datos en dos partes, la lectura de datos y la lectura de imágenes con formatos; para luego pasarlos a un contenedor de *vtk* o uno propietario (con *OpenGL*).

La carga de datos

La función de lectura de datos incluye formatos de mallas y formatos de imágenes digitales, inclusive con formatos no comunes 9, 7 o 5 bits; usados en algunas imágenes biomédicas. Para la lectura de datos en arreglos cartesianos tridimensionales, se desarrolló la función KvVTKCarga, mientras que para la lectura de imágenes digitales se desarrolló la función KvVTKImage. Este diseño de separación considera cambios futuros en los módulos, sin afectar la función completa.

El módulo de lectura de datos dentro de KvCarga cuenta con una función virtual que se encarga de recibir datos de mallas o de componentes equivalentes en un arreglo unidimensional, considerando sus características de alto, ancho y profundidad; y los prepara para que sean colocados en un contenedor de *vtk*, considerando dichas características para que puedan ser usados dentro del *pipeline* de *vtk*.

El módulo de lectura de imágenes dentro de KvCarga se encarga de leer los archivos de imágenes con formatos especiales en los que no se cuenta con un decodificador (*CODEC*) y tras de algunos cambios necesarios, los asigna a un contenedor de *vtk* para que los datos estén disponibles en su *pipeline*. Esta operación la ejecuta a través de una función virtual llamada `lee()`, la cual reconoce imágenes de 8 y 16 bits; sin embargo, para permitir otros formatos solo se requiere recargar la función.

La lectura de imágenes y su manejo se implementó aprovechando una selección de bibliotecas, la cual está integrada por *vtk* y *Qt* en unión con *C++*; se seleccionó la biblioteca de *Qt* ya que contiene las especificaciones de los formatos de imágenes, los cuales se utilizarón para leer las imágenes requeridas. *vtk* fue seleccionada para ser utilizada como contenedor de la información de las imágenes; mientras que para la lectura de datos, tanto en el módulo de lectura de datos tipo malla, como para la lectura de imágenes, se desarrollaron bibliotecas específicas (código específico que invoca a bibliotecas estándares del lenguaje junto con las bibliotecas seleccionadas) en *C++*, ya que el lenguaje cuenta con las herramientas suficientes para dicha función; estas tareas se detallan a continuación.

Se implementaron funciones para la lectura de imágenes con formatos comunes (tiff, bmp, jpg, png, etc.), las cuales están dentro del objeto KvVTKImage, que maneja a los contenedores de *vtk*. Estas funciones utilizan la biblioteca de *Qt* como lector de imágenes; o sea, la clase cuenta con una función de lectura, la cual *Qt* se encarga de obtener los parámetros necesarios de las imágenes para su colocación en el contenedor de *vtk*, así como los valores de pixel de la imagen, los cuales son pasados a una función que llena los campos necesarios en el contenedor para que puedan ser usados en el *pipeline* de *vtk*.

Esto implica que será capaz de leer los formatos que tiene definidos *Qt*, los cuales son los más comunes y útiles. Sin embargo, la aplicación y el kernel solo soportan imágenes con tonos de grises; el manejo de imágenes en color o de más bandas no lo soporta el kernel por un conflicto en la carga de bandas en el contenedor de *vtk*. Este es un conflicto en *vtk* y se está buscando una solución.

Cómo extraer datos

vtk maneja su propio despliegue basado en *OpenGL*, lo que permitió una implementación sencilla en comparación con el acceso a datos de los filtros o de los procedimientos. No obstante, este despliegue no permite la interacción con elementos de *Qt* ni con el manejo de interactividad programadas a través de funciones propias o específicas; así mismo, este sistema propietario de *vtk* es lento en aplicaciones en tiempo real con datos complejos o de gran escala.

Estas limitaciones pueden ser superadas implementando el despliegue directamente sobre *OpenGL*, lo cual requiere de la recuperación de las geometrías y objetos gráficos generados por *vtk*.

Para la obtención de los datos, se estudió la manera de trabajo de *vtk*, la cual le asigna toda la responsabilidad del manejo de los datos a la clase vtkRender, dicha clase funciona como contenedor y controla el intercambio de la información que sea necesaria para el

despliege de las geometrías al realizar el render. En el estudio de la clase se notó que *vtk* maneja para el control de los datos, tres listas: la primera encargada de guardar las coordenadas del punto, la segunda se encarga de guardar las definiciones del orden de los vértices para formar los polígonos, mientras que la tercera lista es la encargada de guardar el tipo de polígonos que se están guardando. Por esto que requieren usar métodos de *vtk* para dibujar cada elemento por separado.

El arreglo de las listas es un factor fundamental para lograr la obtención de las geometrías, ya que *vtk* es muy estricto en el orden para la lectura de los datos. Primero se debe leer la lista del tipo de polígonos, ya que se obtuvo el tipo de polígono, se lee la lista de datos de los puntos, al obtener la lista de los puntos que conforman al polígono, se procede a leer la última lista, que es la de puntos; siguiendo este orden, *vtk* permite obtener las geometrías de forma gráfica, tal como se ilustra en la figura 4.1.

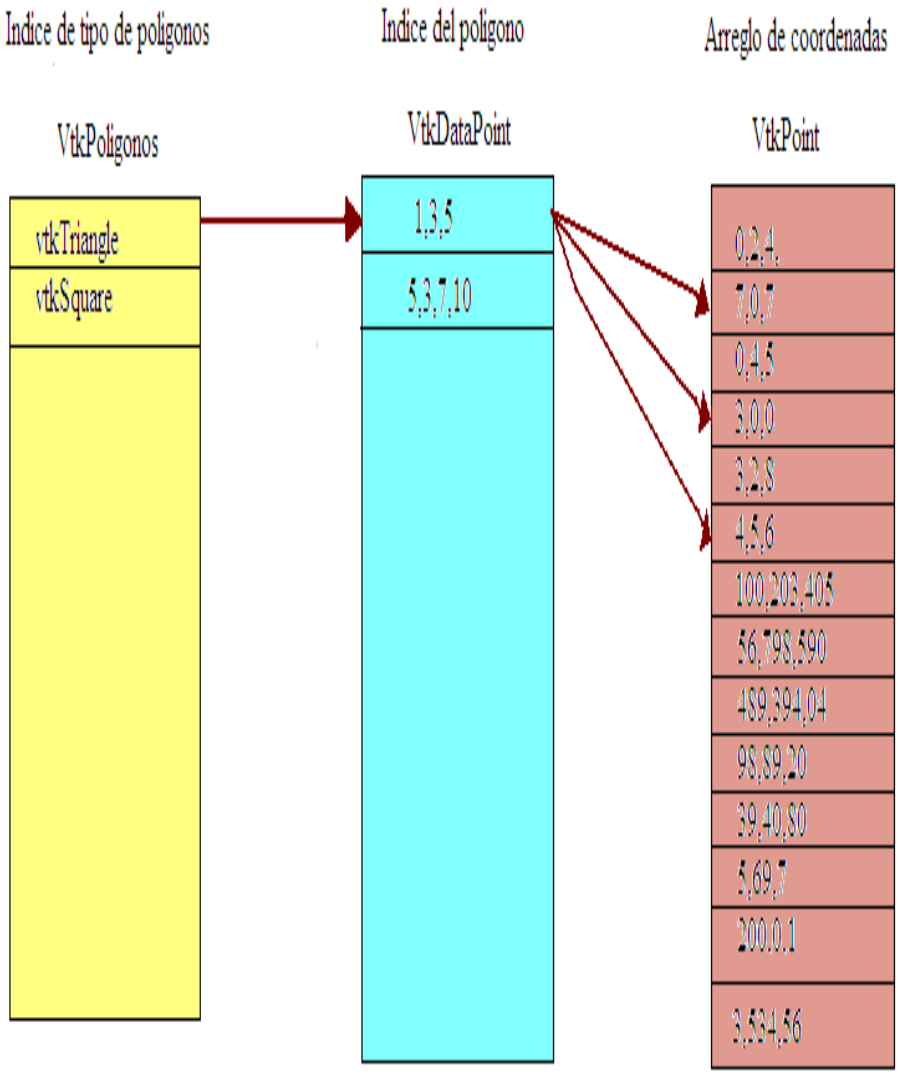


Figura 4.1 Esquema de arreglo de listas.

Para la obtención de las lista fue necesario revisar el código de la clase *vtkVrmlExport*; para saber la localización exacta de dichas listas, así como los métodos para acceder a ellas (ver figura 4.2). Al contar con la localización fue necesario crear una clase dedicada a la obtención de los datos; la clase KvVTKExport, la cual contiene una función que recibe un elemento vtkRender y guarda los puntos y el orden de los vértices para la creación de los polígonos en archivos.

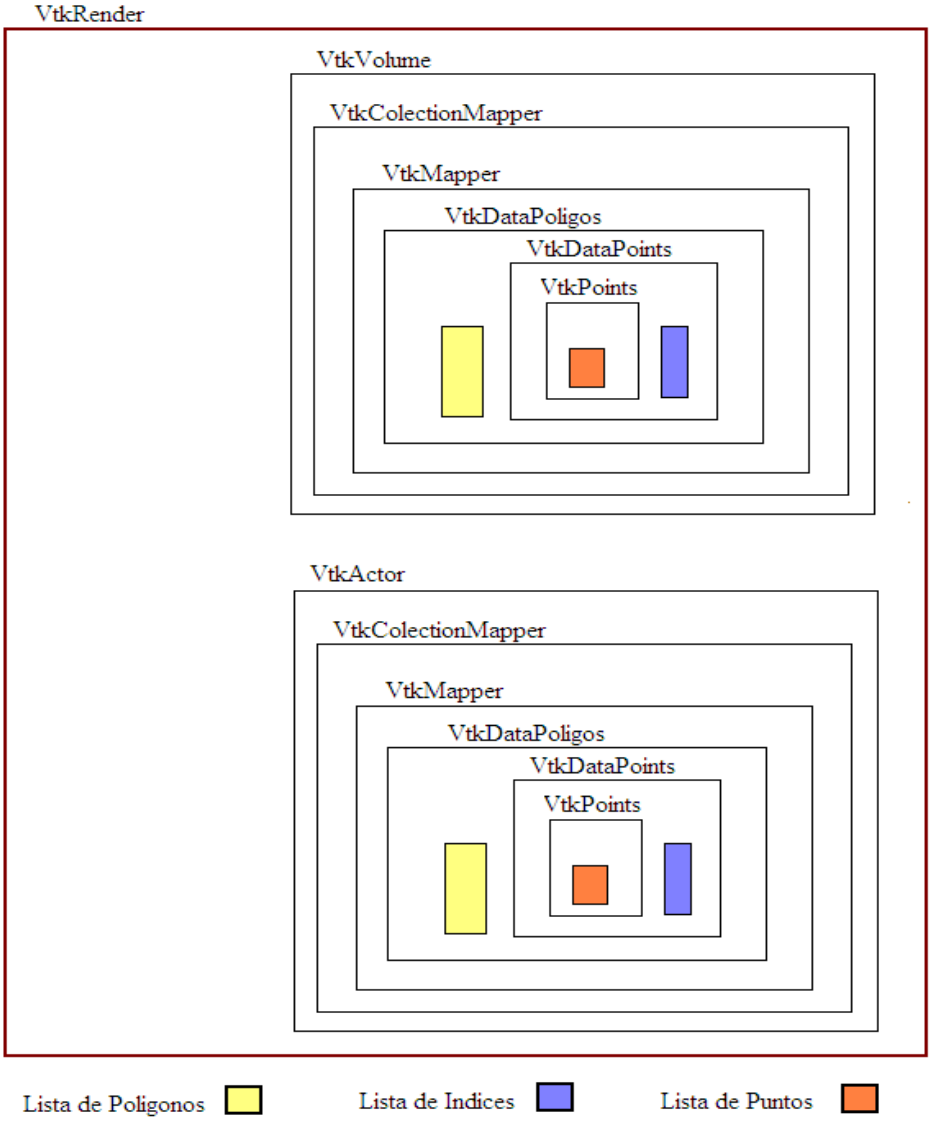


Figura 4.2. Esquema de arreglo de elemento de *vtk*. Los cuadros de colores muestran la localizació dentro de las clase de las diferentes listas.

Aún con la obtención de los datos de los métodos realizados en *vtk*, los datos de entrada siguen guardados en la clase KvVTKImage, en un contenedor de *vtk*; esto es por si nuevamente son requeridos por otros métodos de *vtk*. La razón por la que se guardan los datos en contenedores de *vtk* es por la especialización de ellos en el tipo de datos para la graficación y visualización.

Los contenedores

El uso de *vtk* conlleva el manejo de contenedores definidos en el *ToolKit*, tales como *vtkStructurePoints*, *vtkPolyData*, *vtkStructuredGrid* y *vtkUnstructuredGrid*. El kernel usa el contenedor *vtkStructuredPoints*, ya que se encontró que su uso es sencillo y se ajusta bien al tipo de datos que se contemplan en las aplicaciones derivadas del kernel. Este contenedor permite el acceso a los filtros y métodos que se utilizan el las aplicaciones de casos de estudio presentadas en este trabajo; también es posible acceder a todas las funcionalidades que proporciona *vtk*, simplemente a través de un filtro intermediario.

La conformación de un campo escalar volumétrico a partir de una secuencia de imágenes digitales, como son la mayoría de los conjuntos de datos que utiliza la aplicación desarrollada, se da de forma eficiente a través de una malla cartesiana regular o de sus coordenadas X, Y, Z, correspondientes a una regla de generación simple para ordenar los datos en ancho, alto y secuencia de imagen o profundidad. A cada nodo de la malla le corresponde un valor escalar tomado del valor de pixel correspondiente, de las imágenes.

La clase KvVTKImage es la encargada de controlar al contenedor, ya que ésta tiene sus métodos de acceso al contenedor, así como las funciones que reciben la salida de las funciones de lectura de datos y lectura de imágenes en formatos no estándares. Esta clase cuenta con los métodos necesarios para brindar el soporte a los métodos que se realicen posteriormente o a las modificaciones.

Además de contar con la información sobre el conjunto de datos, este contenedor cuenta con las características que se le asignan; como son los valores de las funciones para realizar el *volume rendering*, los isovalores para un *Marching Cube* (bordeado de cubos), los metadatos que se usarán como parámetros de las técnicas, etc. Esta información es almacenada es esta clase para evitar pérdidas en las modificaciones futuras.

La implementación del contenedor se llevó a cabo en *C++*, el manejo de datos a través de las funciones requeridas y aprovechando *vtk*, ya que éste proporciona su *vtkStructurePoints*.

Preparación de los datos

Los datos adquiridos a través de aparatos o simulaciones pueden tomar diferentes representaciones; una muy común es la que se ha venido exponiendo como estructura de dato genérica: un campo volumétrico de valores escalares discretizado en una malla cartesiana regular. Ya sea que estos datos son proporcionados en un formato particular o como una secuencia de imágenes digitales, solo contienen los valores registrados de los objetos y algunas de sus propiedades, dentro de ese espacio o campo escalar.

Para poder visualizar o revelar a los objetos y el comportamiento o configuración del medio circundante y sus propiedades, es necesario aplicar técnicas de visualización que consisten en procesar los datos y asignarles propiedades ópticas(propiedades de *Phong*), por ejemplo a las celdas de la malla o a los datos derivados; una *isosuperficie*, por ejemplo.

Estas técnicas requieren de parámetros o de condiciones iniciales que generalmente son propiedades numéricas o estadísticas de los datos crudos. Siendo así, buenas propuestas de estos parámetros y condiciones iniciales son encontrados a través del resumen de los datos

con análisis estadísticos y pueden ser encontrados sobre la primera corrida o previo al proceso mismo de la visualización; y almacenados en encabezados de metadatos y descripciones.

Estos metadatos son obtenidos con técnicas que procesen los datos para generar valores sobresalientes; por ejemplo, momentos, valores característicos, entre otros. Así mismo, las características de dimensionalidad del campo y de la malla, son datos necesarios para iniciar una técnica de visualización. La siguiente lista enumera los metadatos obtenidos en un preprocesamiento, éste puede realizarse justo antes de iniciar el proceso de visualización o la aplicación de una técnica.

Alto	Ancho	Profundidad
Color	Formato	Momento 1
Momento 2	Momento 3	Máximo momento 1
Máximo momento 2	Máximo momento 3	Mínimo momento 1
Mínimo momento 2	Mínimo momento 3	Media
Mediana	Modas	IsoValor∗

∗ Un valor por omisión asignado al momento de cargar los metadatos, en nuestro caso es la media.

Cuadro 4.1. Metadatos obtenidos de las imagenes.

Estos metadatos son requeridos por la aplicación en dos etapas, las cuales usan diferentes metadatos; por lo que se configuraron dos bloques obtenidos en cada etapa respectiva. El bloque formado por los datos del 1 al 5 de la lista, se obtienen de manera inmediata al cargar los datos; el bloque constituido por el resto de elementos, son necesario para la interacción.

El primer bloque se obtiene por medio de las clases KvVTKImage o KvVTKCarga. Esta última clase tiene implementada la obtención de los datos cuando es instanciada para la carga de datos, mientras que la clase KvVTKimage funciona como contenedor, tanto de datos como de metadatos; ya que la clase KvVTKCarga le entrega los datos.

El segundo bloque de metadatos está compuesto por elementos que requieren de al menos un parámetro proporcionado por el usuario para obtenerse; así que son calculados hasta que el usuario realiza una petición, el cálculo de momentos. Estos cálculos se realizan a través de la clase KvVTKMomento, que entrega el resultado en una dirección de memoria a la clase KvVTKImage, ésta los guarda y cada vez que son requeridos por la clase contenedora se le proporcionan, evitando ser recalculados.

Interactividad

La interactividad se refiere a toda comunicación que se tiene con la aplicación, ya sea a través de la interfaz gráfica, vía comandos o eventos del ratón; cada acción que se toma en estos elementos, corresponde una reacción por la aplicación.

El kernel considera dos tipos de interacción, las cuales están separadas en dos fases; la referente a la interfaz gráfica (botones, *sliders*, menús, etc.) y la interacción con el sistema de despliegue gráfico la manipulación de los objetos gráficos creados por las técnicas de visualización. La primera es proporcionada a través de secuencias de *Qt* directamente, ya

que *Qt* cuenta con el nivel de abstracción y simplicidad que no es conveniente encapsular más.

El despliegue gráfico se ejecuta a través de un componente especial de *Qt*, el cual recibe los elementos de *OpenGL*, mientras que su interactividad está a cargo de una clase de control de interactividad, esta interacción se realiza a través de teclas y ratón dentro del *canvas* (zona de dibujado) de la escena.

Esquema cliente−servidor

Los sistemas de visualización son aplicaciones de software muy particulares en cuanto a las tareas que realizan las computadoras en donde se ejecutan los programas correspon− dientes. Tradicionalmente, la carga de cálculos gráficos se realiza en la máquina local donde se está desplegando la aplicación; o sea, se considera la máquina local como el servidor de despliegue. Otros cálculos pueden realizarse en equipos remotos; por ejemplo, donde está residiendo la aplicación.

Actualmente se tienen diversas alternativas y también han cambiado las necesidades y estilos de trabajo con las aplicaciones de visualización. Ahora es común que se realicen los cálculos gráficos en una máquina remota en la que se encuentra la aplicación (típicamente una máquina poderosa o especializada para la tarea, un servidor de visualización) y se transmita a través de la red, el video tomado del *frame-buffer* de la tarjeta gráfica (*GPU*), hasta la máquina desde donde estamos trabajando.

Otra modalidad de trabajo que responde a los estilos y necesidades de los visualizadores, es el trabajo colaboratorio y distribuido. Este consiste en compartir objetos gráficos y la interactividad con ellos, para que diferentes personas en sitios remotos lleven a cabo visualizaciones complejas, compartiendo y repartiendo las tareas y resultados. Estos estilos se han implementado aprovechando las mejoras en las tecnologías de redes y han incorporado diferentes aspectos de dicha tecnología, llegando a diversos esquemas de trabajo.

Para incluir estos estilos de visualización, se diseñó un esquema cliente−servidor para el kernel, el cual es suficientemente flexible como para incluir esquemas tradicionales de servidor de despliegue y esquemas flexibles de manejo de *frame-buffer* y transmisión de video. La primera versión incluye dos clases básicas y la estrategia es desarrollar una aplicación genérica que soporte estos estilos para las aplicaciones derivadas del kernel (ver figura 4.3).

Esta extensión del kernel fue diseñada pero no implementada, ya que requiere un tratamiento más extenso y la ejecución de pruebas sobre los diferentes escenarios. Por ejemplo, evaluar la eficiencia sobre redes con diferentes anchos de banda, así como las ventajas de dichos esquemas con equipos gráficos y en ambientes heterogéneos.

El diseño contempla dos clases (KvCliente y KvServidor) programadas en *C++*, mientras que *Qt* proporciona la comunicación vía red y las operaciones para mantenerla. La clase KvCliente tiene la capacidad de recibir elementos de *Qt*; por ejemplo, una ventana o un control de *Qt*. Por lo tanto el despliegue se encuentra encapsulado en un *canvas* de *Qt*.

Una variante del diseñó es incluir una tercera clase (KvEmpaquetar), la cual se encargaría de solicitar los datos a la clase de control de interactividad, en forma de despliegue gráfico

y colocarlo correctamente en el *canvas* de *Qt*; entregándolo correctamente tanto al servidor como al cliente. Cabe mencionar que en el diseño básico de dos clases, es el cliente quien se encarga de realizar esta tarea.

La implementación de estas clases facilitará la programación de aplicaciones de visualización que compartan botoneras y despliegue o que aprovechan equipos remotos o sistemas de despliegue especializados.

Técnicas de reconstrucción

Los datos de entrada de un sistema de visualización son los valores de un campo discretizado y la geometría de discretización (una malla con sus posiciones y conexiones entre ellas y sus diferentes campos escalares, vectoriales o tensoriales). La visualización, a través de sus diferentes técnicas es responsable de las transformaciones a estas estructuras y la asociación con propiedades geométricas, ópticas y de iluminación de las resultantes de dichas transformaciones. O sea, una reconstrucción de objetos gráficos (dibujables de forma inteligible al ojo humano, con propiedades geométricas, espaciales como posición, propiedades lumínicas, entre otras) a partir del campo de radiación del objeto dentro del espacio muestreado.

El kernel permite el enlace con los diferentes métodos de *vtk* para llevar a cabo diversas técnicas. La aplicación desarrollada a partir del kernel incluye un par de técnicas, las cuales son generalmente las primeras en probarse sobre un conjunto de datos. Estas técnicas son adecuadas para un campo escalar, aunque si se tiene otro tipo de campo, siempre es posible aprovechar cualidades escalares y aplicar la técnica (por ejemplo, tomar la magnitud del gradiente en un campo vectorial).

Las técnicas disponibles en la aplicación son *isosuperficie*s y *volume rendering*. Esta última técnica está implementada a nivel de aplicación, mientras que la *isosuperficie* está a nivel de kernel; aunque existe la posibilidad de implementarse a nivel de aplicación, ya que la clase proporciona la entrega de datos para su realización por *vtk* o por una implementación particular nueva. La técnica de *isosuperficie* está implementada directamente en la clase contenedora KvVTKImage, de tal forma que se puede utilizar inmediatamente después de la carga de dados, solo proporcionando un valor de búsqueda dentro del rango de valores aparecidos en el campo; esto es un isovalor para reconstruir una superficie dentro del campo escalar.

La técnica de *volume rendering* fue implementada a nivel de aplicación ya que es una técnica más compleja que requiere de muchos parámetros, los cuales se desea tener bajo control del usuario; por otro lado, existen muchas aplicaciones y códigos que implementan variantes y nuevas aproximaciones.

Manejo de la escena

La escena es un conjunto de objetos dentro de un espacio tridimensional capaz de lograr una representación gráfica. La escena está compuesta por un subconjunto de objetos gráfico pertenecientes a una base de datos de objetos, un mundo tridimensional donde son colocados estos objetos y toman un espacio, una iluminación que interactúa con materiales y texturas

asignadas a los objetos y una cámara que contiene un punto de vista y un plano de proyección para generar una imagen en dos dimensiones de la proyección del mundo en tres dimensiones.

La cámara es un elemento muy importante de la escena y es posible asignar varias cámaras a un mundo virtual (el mundo 3D de la escena) e instanciar variantes de escena. Cada cámara cuenta con parámetros de configuración; los parámetros mínimos son la posición y el punto de vista. El kernel cuenta con una clase para la declaración de las cámaras, basada en los modelos que proporciona *OpenGL*; aunque el control de la cámara lo tiene *OpenGL*, la clase KvVTKCamara del kernel incorpora una interfaz gráfica para solicitar los parámetros al usuario del kernel, en las aplicaciones que se realicen.

El objeto instanciado de KvVTKCamara permite elegir entre dos tipos de cámara, las cuales cuentan con los componentes para generar una interfaz gráfica para solicitar al usuario de la aplicación, los parámetros básicos. Aunque es posible que el programador proporcione las instrucciones para control la cámara sin la intervención directa del usuario, a través de las funciones proporcionadas. La clase KvVTKCamara, define el tipo de cámara y los parámetros necesarios para declararla en *OpenGL*.

Otro elemento de la escena que puede producir variantes o variaciones requeridas por el usuario, es la luz. Las propiedades o características de la fuente de luz son el color, las coordenadas de localización y la intensidad. Los parámetros que controlan estas características pueden ser modificados a través de una interfaz gráfica o modificados por medio de las funciones para programadores, ambas manejadas por la clase KvLuz.

Manejo de las propiedades gráficas de los objetos

Los volúmenes y las superficies creados a través de las técnicas *volumen rendering* e *isosuperficie*s son objetos gráficos con propiedades geométricas (forma, posición, entre otras) y lumínicas (transparencia, color, etc.), las cuales se pueden modificar para conseguir las representaciones adecuadas a partir del proceso de *render* o dibujado de la escena. La clase KvCarac fue implementada para modificar estas características y lo realiza a través de una interfaz gráfica; los parámetros de cada volumen o cada superficie se manejan en otra clase, la que controla los datos (KvVTKImage).

La separación de estas tareas en dos clases responde a un diseño que evita la sobrecarga en la clase KvVTKImage; dejando la tarea del manejo de la comunicación con el usuario (con la interfaz gráfica) solo para cuando la clase KvVTKImage solicita datos de las características. De esta forma se conserva la especialización y la abstracción de la clase.

No es necesario utilizar la clase para la comunicación entre usuario y aplicación, ya que la clase KvVTKImage permite el manejo de las características sin el uso de la clase KvCarac; sin embargo, su manejo es responsabilidad del programador, así como los resultados que obtenga del kernel. Éste debe brindar las posibilidades de utilizar más funciones, ya sea programadas u obtenidas de otros programas; por eso siempre brinda las formas de manejar directamente los datos. No obstante, la función KvCarac se creó pensado que las características que se solicitan nunca cambian, por esto se creó su interfaz gráfica para ahorrar el trabajo de la programación una y otra vez, de las misma ventana y de los controles.

Manejo de las funciones de transferencia

Para la realización del *volume rendering* es necesario contar con las tres funciones de transferencia: la función de transferencia de opacidad, de color y la del gradiente de opacidad, de las cuales depende el resultado. La forma o configuración que toma cada una de dichas funciones, puede ser proporcionada por el usuario de forma interactiva para conseguir los resultados visuales deseados.

La interacción sigue una analogía gráfica, proporcionando la facilidad de interactuar sobre la forma gráfica de las funciones. Para lograr esta interacción se utilizó la clase KvFunciones, la cual proporciona un *canvas* de dibujo de *Qt* que dibuja gráficas editables a través de desplazamientos de puntos de control sobre la curva; requiriendo tres instancias de la clase KvFunciones. De esta manera gráfica se logró un mejor entendimiento de las funciones y del manejo de las mismas, además de ser más interactivo con el usuario.

El manejo de las funciones de transferencia no se limita al manejo de la interactividad, sino también al manejo de la interacción con los otros objetos de la aplicación. La clase KvFunciones no solo proporciona la interfaz gráfica para el manejo de las funciones de transferencia, también proporciona funciones para el manejo y control de las mismas a nivel de programación; ya que con la obtención de las funciones se proporciona al objeto que se encarga de la realización del *volume rendering*.

La clase KvFunciones maneja la versatilidad necesaria para el manejo de las funciones, así como de la sencillez requerida por los usuarios, además de proporcionar comunicación entre clases a nivel de programador como de usuario de la aplicación. De esta manera KvFunciones es la herramienta que brinda en su totalidad las opciones para satisfacer las necesidades requeridas para la aplicación.

Diseño de la aplicación

La aplicación ejemplo que manejamos es un sistema de reconstrucción tridimensional de organismos vivos, tomografiados o registrados con cortes ópticos o hitológicos; ese es el principal objetivo de la aplicación. Para lograr el mejor resultado fue necesario utilizar diferentes técnicas para lograr tal objetivo, es decir se utilizaron las funciones de momentos de alto orden para obtener el mejor isovalor; la reconstrucción se llevó acabo con base en la función de *volume rendering* o *isosuperficie*.

En este momento podemos separar en dos aspectos a la aplicación. Métodos que se aplicarán y conjunto de datos que intervendrán. En el primer aspecto, el kernel brinda el soporte suficiente para aplicar las técnicas, ya que son técnicas que las tiene integradas. En el aspecto de los datos, se hizo una gran recopilación de información acerca de las características de los datos que manejaríamos y de la forma que se nos proporcionaría. Esta parte es la más importante de toda la aplicación y por lo mismo, la más delicada.

En la recolección de información de los datos notamos que las clase Carga y KvVTKImage, serían de mucha ayuda, gracias a los métodos con los que cuentan, harían de manera sencilla la carga de datos, pero tendríamos que definir de una manera muy detallada la manera de cómo se los entregaríamos a la clase. Los datos que podríamos recibir son archivos de

imágenes en 8 bits, y 16, así como las secuencias de imágenes jpg, tiff, bmp, pnm, png. Teniendo esto en cuenta logramos crear la definición de las características que deberían cumplir los datos. Como primera característica tendrían que ser imágenes a tonos de grises, datos que podíamos recibir son secuencias de imágenes de formatos jpg, tiff, pnm, png y bmp, pero además debían de cumplir con la parte mas importante que fueran del mismo tamaño las imágenes, esto es muy delicado de no cumplir con esta característica podría causar resultados poco deseables en la aplicación de la técnica. Una de las consecuencias puede ser marca error en el programa en el momento de las pruebas, o causar un volcado de pila ya que intentara llenar espacio reservado y no encontrará datos con que hacerlo. Otra puede ser que en el momento de aplicar los métodos, nos den imágenes con líneas entre el volumen, causadas por la basura que se introdujo para rellenar los espacios vacíos.

La segunda parte fué la elección de los métodos de reconstrucción que serían a través de los métodos de *volume rendering* e *isosuperficies*, las cuales serían aplicadas por medio del *pipeline* de *vtk*. Se realizó una evaluación de cual sería la mejor manera de aplicar los metodos a nuestros datos; se tenían dos opciones, las cuales eran la de aplicarlos por medio de nuestro contenedor del kernel (KvVTKImage) o la aplicación por medio del *pipeline* de *vtk* de manera independiente.

La primer opción no se utilizarían las funciones que están dentro del contenedor del kernel (KvVTKImage), para tener un mayor control del proceso y tener la facilidad de introducir a futuro más técnicas; ya sean de *vtk*, de otras aplicaciones o programadas por nosotros. Esta opción, además brinda la oportunidad de utilizar *vtk* como la parte que brinde apoyo en los métodos, pero que *OpenGL* y las demás herramientas se encarguen del despliegue, para tener un *render* de mayor calidad para las aplicaciones de modo real. Además que de esta manera nos daría control sobre las características de las escenas, de otra manera no podríamos tener acceso a ellas, ya que *vtk* manejaría los valores por omisión.

La segunda opción al utilizar la función de *vtk* para *volume rendering* e *isosuperficies*,ofrece mayor simplicidad en el código y en la programación. Pero no tendríamos el control acerca del despliegue, ni del manejo de los datos de salida de nuestro método y sería demasiado complicado en un futuro introducir más métodos a nuestra salida del *volume rendering*; es decir que se podría hacer pero se haría más confuso si se aumenta el número de técnicas que introduzcamos.

Por último estudiamos el modo de dár las opciones de manejo de las características de la escena, utilizando los objetos del kernel que tiene las funciones para el manejo de ellas y que brinda además ya una interfaz predefinida. Como son KvCamara, KvMaterial, KvFunciones y KvCamMon, estos objetos del kernel servirían como interfaces entre las características de la escena y el programa principal, aunque su función no es simplemente una interfaz pero esta vez conviene utilizarlo de esa manera. Así decidimos qué clases del kernel serían útiles y de qué manera los utilizaríamos, de esta manera se continuó el diseño.

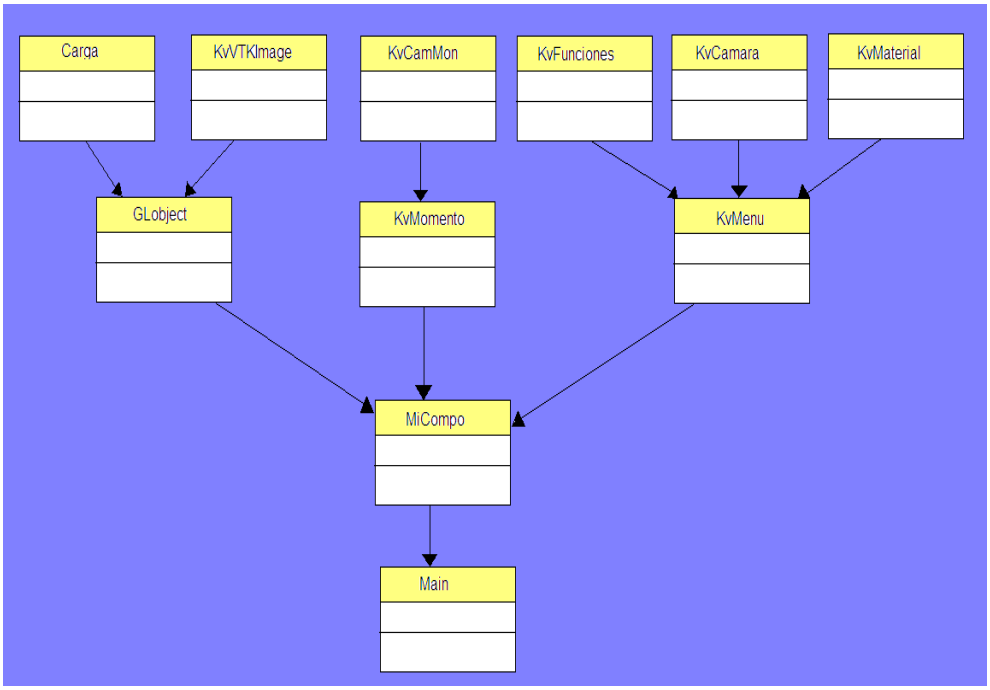


Figura 4.4. Esquema de programación de la aplicación.

En el diseño de la aplicación se tomarón en cuenta las ventajas que proporciona el kernel, como son la encapsulación en objetos y el llamado de funciones. Tomando en cuenta las características de la aplicación y encontramos la mejor solución era unirlo a través de encapsular todos en un objeto de *Qt*; pero al unir todos los elementos del kernel en un solo objeto, no era buena idea ya que el código sería muy difícil de entender así como muy propenso a cometer errores (por la gran cantidad de líneas del código), por lo tanto se decidió subdividirlo en tres objetos.

Para elegir qué componentes contendría cada eslabón, se tomó en cuenta el criterio de separar lo por funciones, es decir tomando en cuenta qué funciones realizan cada uno de los componentes del kernel y en qué parte irían especificados en la aplicación; de esta manera quedaron separadas por sus funciones, como se muestra en la figura 4.2.

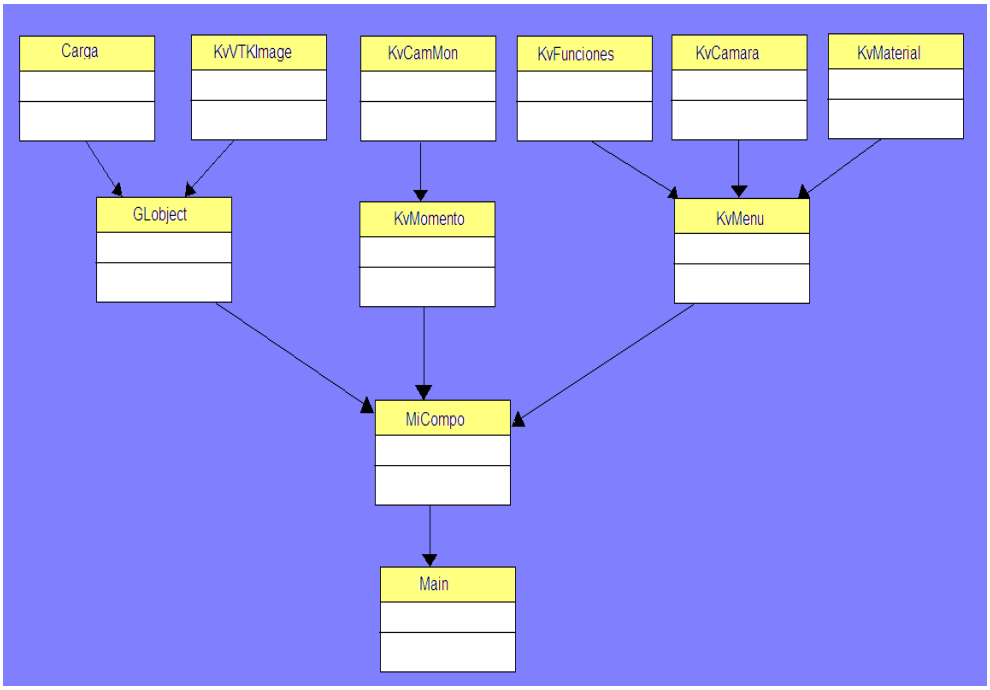


Figura 4.5. Esquema de clases marcando a la función que tienen encomendada.

Siguiendo la definición del esquema de trabajo con base en las funciones se puede ver que utilizaremos dos objetos que hereden de un *Qt*. Estos objetos son los siguientes: GLObject y miCompo, los cuales serán las uniones de nuestros objetos del kernel. Primero revisaremos a miCompo que tiene la función de unir a los objetos que tienen asignadas las funciones principales.

Clase miCompo

miCompo	+ int Iso:int
+ int cuentaPor:int	+ char cIsovalor [20]:char
− Etiqueta: char *	− Dialog: QFileDialog *
− opcion:int	−borra:int
− Ruta[120]:char + obj : GLObject	− transparencia: QListViewItem *
− Isovalor: QListViewItem *	− Material: QListViewItem *
− bOpcEsco:bool	− iOpEsco:int
− itemp:int	+ menu: KvMenu *
− Mom1: QListViewItem *	− Mom2: QListViewItem *
− Mom3: QListViewItem *	− Mom4: QListViewItem *
− Mom5: QListViewItem *	− Color: QListViewItem *
− aux:int	− Recorta(char*):char *
+ miCompo(QWidget *, const char *)	+ CuentaY(int);
+ AcepPorcen(void);	+ BorraArch(int)
+ GuardarComo(void);	+ BorraSelec(void);
+ Cuenta(int);	+ void etique(char *eti)
+ bBanOpcVol:bool	+ Momento(void);
+ Trans(int);	+ IsoVal(int);
+ Escoge(int);	+ Ini();
+ Arbol(QListViewItem *aca);	+ ArbolCarac(QListViewItem);
+ Ver(void);	+ selecc(int);
+ Borra(void);	− f: QFrame *

Cuadro 4.2. Funciones y elementos de la clase de la clase miCompo.

¿Qué es miCompo?

miCompo es una clase diseñada para unir a las clases encargadas de hacer las funciones de caracteristicas de la escena, cálculos de los datos y despliegue y carga de datos. Esto lo logra a través de tener declarados los objetos que realizaran estas funciones en ella. miCompo fué diseñada para que fuera la única clase que tuviera comunicación directa con el programa principal. Los objetos que tiene declarados son KvMomento, KvMenu y GLObject; de los cuales solo GLObject es una función diseñada al igual que miCompo, especialmente para esta aplicación, ya que las dos clases son del kernel y tienes sus funciones predefinidas y fueron explicadas con anterioridad.

¿Qué función realiza miCompo?

En miCompo se tienen definidos a los objetos de las clases principales ya, que de esta manera tendrá en control de las diferentes actividades que realicen los objetos hijos, además de que ella lleva a través de sus funciones el control del flujo de la información, de la misma manera lleva un respaldo de los datos más importantes de cada una de las funciones que tienen asignadas sus clases hijos.

El control lo mantiene a través de una variable que es el número de volúmenes que contiene la aplicación, así como el número del volumen que está siendo utilizado y el número de imágenes que se utilizarán para los cálculos, etc. Las clases hijas solo tienen comunicación a través de esta clase padre, hubiéramos podido ceder todo el control a las clases hijas, pero para evitar los errores se decidió que sería mejor que solo tuvieran este tipo de comunicación a través del padre. Además, esta función es la que manda a llamar a las funciones principales de cada una de ellas, ya que las funciones que contienen a las interfaces gráficas necesitan un padre para ser colocadas, esta función es el padre de todas las interfaces de la aplicación para que aparezcan sobre las coordenadas de la pantalla que tiene definida esta clase.

¿Cómo se utiliza?

En el programa principal se instancia de esta manera, se crean los objetos de las demás clases y las variables se inicializan con valores predeterminado para evitar errores. En su constructor es donde se encuentran las declaraciones de las distintas clase, así como el llamado a las funciones de las clases conforme el orden necesario.

Nombre de las funciones	Especificación
CuentaY(int);	Es una clase que lleva la cuenta de una secuencia para manejar a los *widget* que lo necesiten.
AcepPorcen(void);	Esta función se encarga de asignar los valores a las variables de la clase que se encargan de guarda el valor del porcentaje que se utilizará en los momentos. Además es el que se encarga de mandar a llamar a las funciones de KvMomento.
BorraArch(int);	Se encarga de tomar el número del volumen que se va a borrar lo utilizamos para el *widget* es un *spot*.
GuardarComo(void) BorraSelec(void);	Guarda en *VRML* el volumen que se muestra en Se encarga de dibujar un cuadro de díalogo donde se muestran los nombres de los volúmenes que están disponibles para borrarlos
Cuenta(int);	Es un *spot* que sirve para llevar la cuenta de algún *widget* de la clase.
etique(char *eti)	Recibe la ruta en la cadena y devuelve la ruta pero sin extención.
bBanOpcVol:bool	Nos dice si hay habido cambios en los volúmenes.
Momento(void);	Manda a llamar a la función que calcula los momentos.
Trans(int);	Manda a llamar a la función que manejará la transparencia.
IsoVal(int);	Es un *spot* que sirve para llevar la cuenta del *widget* que se encarga de dar el isovalor.
Escoge(int);	Es un *spot* que sirve para llevar la cuenta de algún *widget* de la clase.

Cuadro 4.3. Especificación de las funciones de la clase miCompo

Nombre de las funciones	Especificación
Ini()	Esta función es la que manda a llamar, en orden a las funciones para hacer el desplegado de los datos ya convertidos en volúmenes.
Arbol(QListViewItem *aca)	Maneja el árbol donde se muestra los volúmenes.
ArbolCarac(QListViewItem *aca)	Maneja el árbol de las características de los volúmenes.
Ver(void);	Manda a llamar al repintado de la pantalla de los volúmenes.
selecc(int);	Es un *spot* que sirve para llevar la cuenta de algún *widget* de la clase.
Borra(void)	Manda a llamar a la función encargada de borrar los volúmenes, así como pinta la interfaz para elegir el volumen a borrar.

Cuadro 4.3. Especificación de las funciones de la clase miCompo.

Clase GLObject

GLObjetc.h	+ ResVol: KvVTKImage
+ cargador: Carga	+ bBanVo: bool
+ iBanImgVol: int	+ iNumVolVis: int
+ NumVol: int	+ Alto: int
+ iNumABorrar : int	+ Ancho : int
+ Profu : int	+ bBanOpcVol : bool
+ Image: KvVTKImage	+ renWin: vtkRenderWindow
− arvo[120]:char	− formato: int
− int limInf: int	− limSup: int
− ban: int	− ival1: int
− val2: int	− x: int
− IsoValor: int	− opc: int
− y: int	− z: int
− BanCam : int	− Sepas: float
− EspaY: float	− EspaZ: float
− Ambiental: float	− Difuso: float
− Specular: float	− PoderEspe: float
− aRenderer: vtkRenderer	+ GLObject (QWidget* parent, const char* name)
+ GLObject()	+ Inicia(int): void
+ DameRango(int, int): void	+DameProfundidad(int)
+ DameAlto(int): void	+ DameAncho(int): void
+ DameArchivo(char*): void	+ isoValor1(int v, int): void
+ rePinta():void	+ Cambio(int): void
+ DameObservador(int, int,int): void	+ DamePosicion(int,int,int): void
+ DamePuntoFocal(int, int, int): void	+DameMaterial(float,float,float,float)
+ MasVolum(int): void	+ RespaIma(int): void
+ ResImaVoluBo(int): void	+ InicializaVol(int): void
+ isoValor2(int v): void	+ rotax(int r): void
+ rotay(int r): void	+ rotaz(int r): void
+ Opcion(int): void	initializeGL ():void
paintGL ():void	resizeGL (int w, int h): void
− int Volu(int);	

Cuadro 4.4. Funciones y elementos de la clase GLObject.

¿Qué es GLObject?

Esta clase es la encargada del despliegue y carga de datos. Esta clase está diseñada para dar el soporte necesario, así como proporcionar los datos necesarios a sus clases hijas que son las encargadas de hacer el despliegue de los datos y su carga.

¿Cómo funciona GLObject?

La clase hereda del objeto de *QGLWidget* de *Qt*, el que se encarga de hacer el ciclo del despliegue de *OpenGL*; con este ciclo se mandará el despliegue de los datos resultantes del

proceso de reconstrucción. Esta clase tomará el control sobre los objetos hijo que son Carga y KvVTKImage. Los objetos de la clase de carga se encargaran de recivir los datos de las imágenes de 16 y 8 bits, ya que tienen sus métodos especializados en esa labor, después la asignarán a un objeto KvVTKImage que servirá de contenedor a las imágenes para su posterior uso. Para las secuencias de imágenes de los formatos jpg, tiff, bmp, pnm y png, se utilizan los métodos del contenedor.

Al tener los datos cargados, manda una bandera a su clase padre para decirle si tuvo éxito en la carga o en caso de tener error, para que avise al usuario que no se logró para que este enterado. Cuando la clase padre se da por enterada del éxito de la carga de datos, continúa con el proceso; llegado el momento de pedir los datos para realizar las distintas funciones con ellos, la clase madre es la que los solicita y se los da a las funciones de las siguiente clases para que realicen los cálculos necesarios con ellos. Llegado el momento del despliegue, los resultados de los cálculos realizados (el isovalor principalmente) son regresados a esta función para realizar el proceso de despliegue.

La clase madre tiene que proporcionarle los datos de qué tipo, número, ancho, etc; de los datos que se van a cargar; por eso tiene definidas las funciones que se encargarán de recopilar datos proporcionados a las demás clases por el usuario y que la clase padre le hace llegar a través de ellas.

¿Cómo se utiliza?

Se declara un objeto, el cual estará contenido en la función miCompo. En el momento de la carga de datos, la clase miCompo se encarga de recopilar todos los datos que son necesarios para la creación del contenedor y se le asignan al objeto GLObjetc, el que se encargará de sincronizar a sus contenedores para al carga, dependiendo del tipo de datos.

Cuando están colocados en los contenedores, el objeto solo espera que se le mande a llamar para desplegarlos, así como para la asignación de los diferentes campos que son necesarios, que deben de ser proporcionado por la clase padre. Al contar con todos los campos, se manda a llamar a la función de despliegue y se regresa el control a la clase padre para que continue con el ciclo.

Nombre de las funciones	Especificación
isoValor1(int v, int)	Esta función es la que recibe los límites que tendrá el isovalor.
GLObject (QWidget* , const char*)	El constructor.
GLObject()	El destructor.
Inicia(int):	Inicia el proceso de despliegue de los datos.
DameRango(int, int)	Recibe el rango de las imágenes que tendrá los datos a cargar.
DameProfundidad(int)	Recibe el número de las imágenes, o el número de profundidad que tendrán los datos de 8 y 16 bits.
DameAlto(int)	Recibe el alto de las imágenes.
DameAncho(int)	Recibe el ancho de las imágenes.
DameArchivo(char*)	Recibe la ruta de las imágenes.
rePinta():	Manda a llamar a la función de despliegue.
Cambio(int)	Cambia el número de volumen elegido para su despliegue.
DameObservador(int, int,int):	Recibe las coordenadas de la localización del observador.
DamePosicion(int,int,int)	Recibe las coordenadas de la localización del la cámara.
DamePuntoFocal(int, int, int)	Recibe las coordenadas de la localización de dónde está viendo la cámara.
DameMaterial(float,float,float,float)	Recibe los parámetros de las características del material.
MasVolum(int)	Se encarga de crear más de un volumen y es llamada cada vez que se aumente un volumen a la pila de volúmenes.
RespaIma(int)	Hace un respaldo de los objetos KvVTKImage.
ResImaVoluBo(int)	Hace un respaldo de los objetos KvVTKImage.
InicializaVol(int)	Aplica el método de *volume rendering* a los datos.
isoValor2(int v)	Recibe el isovalor.
rotax(int r)	Recibe el ángulo de rotación en X.
rotay(int r)	Recibe el ángulo de rotación en Y.
rotaz(int r)	Recibe el ángulo de rotación en Z.
Opcion(int)	Recibe el número de opción de tipos de datos que recibirán los contenedores.
initializeGL ()	Inicializa la pantalla de *OpenGL*
resizeGL (int w, int h)	En caso de hacerse grande la pantalla, esta función redimensionará las coordenadas de la pantalla del despliegue.
Volu(int)	Manda a llamar a las funciones encargadas de los volúmenes, que estarán seleccionados.
paintGL ():	La misma función que en *OpenGL*.

Cuadro 4.5. Especificación de las funciones de la clase GLObject.

Vista de la aplicación.

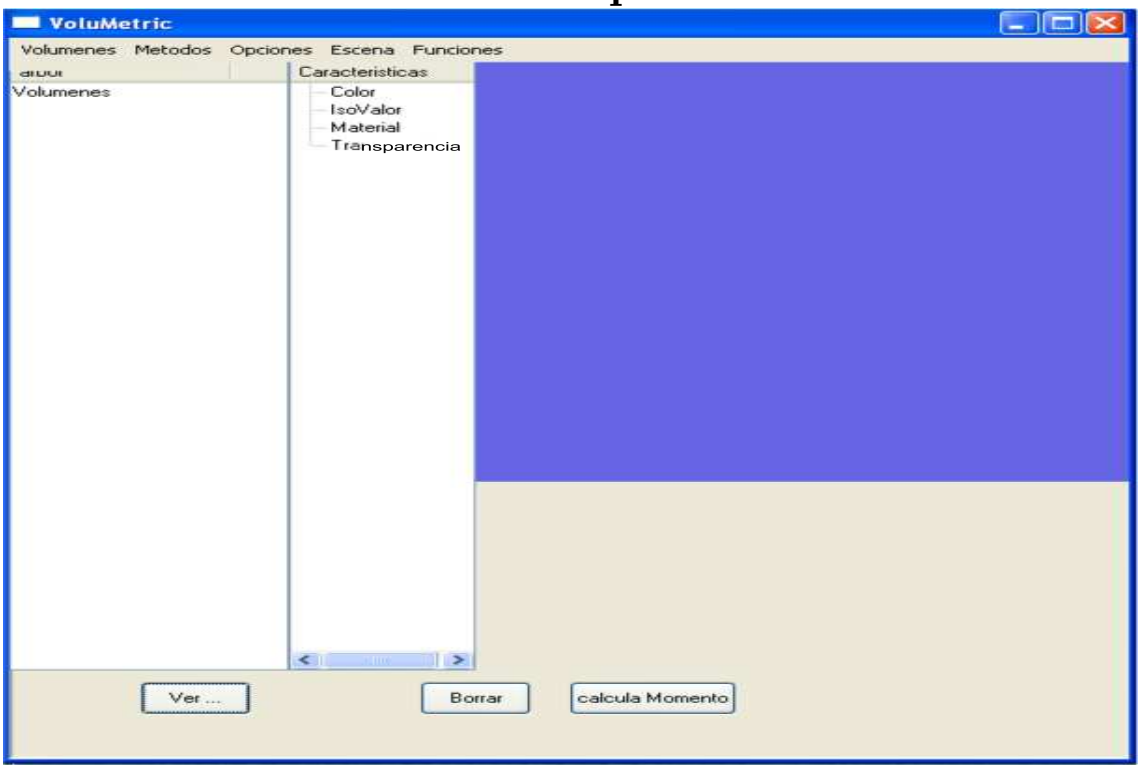


Figura 4.6. Pantalla Principal.

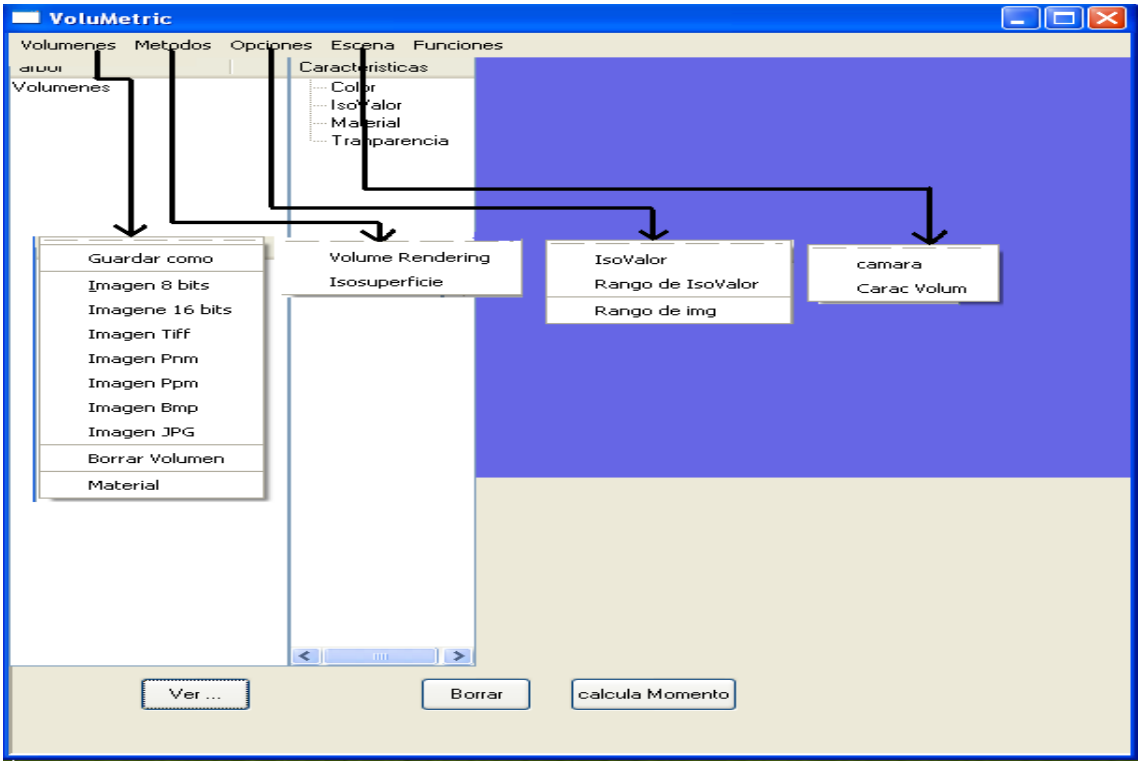


Figura 4.7. Opciones de los menú.

Ejemplo de la realización de una *isosuperficie*.

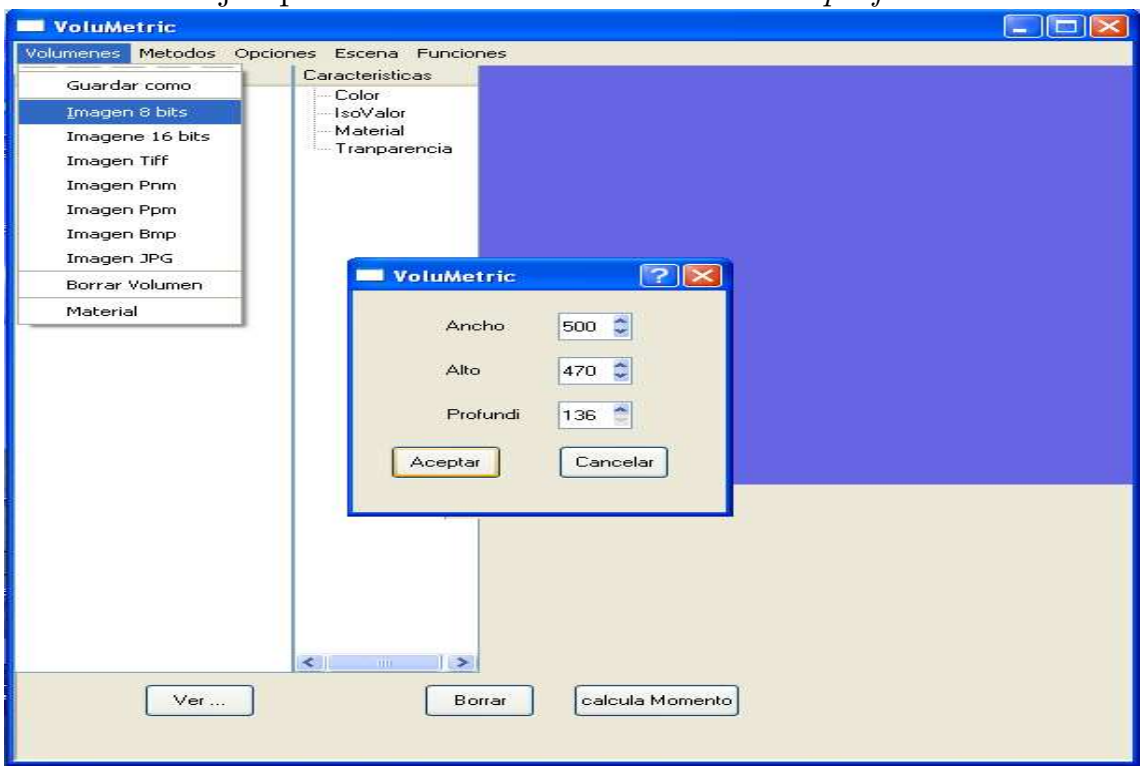


Figura 4.8. Elección de un tipo de archivo e Introducir los parámetros del archivo.

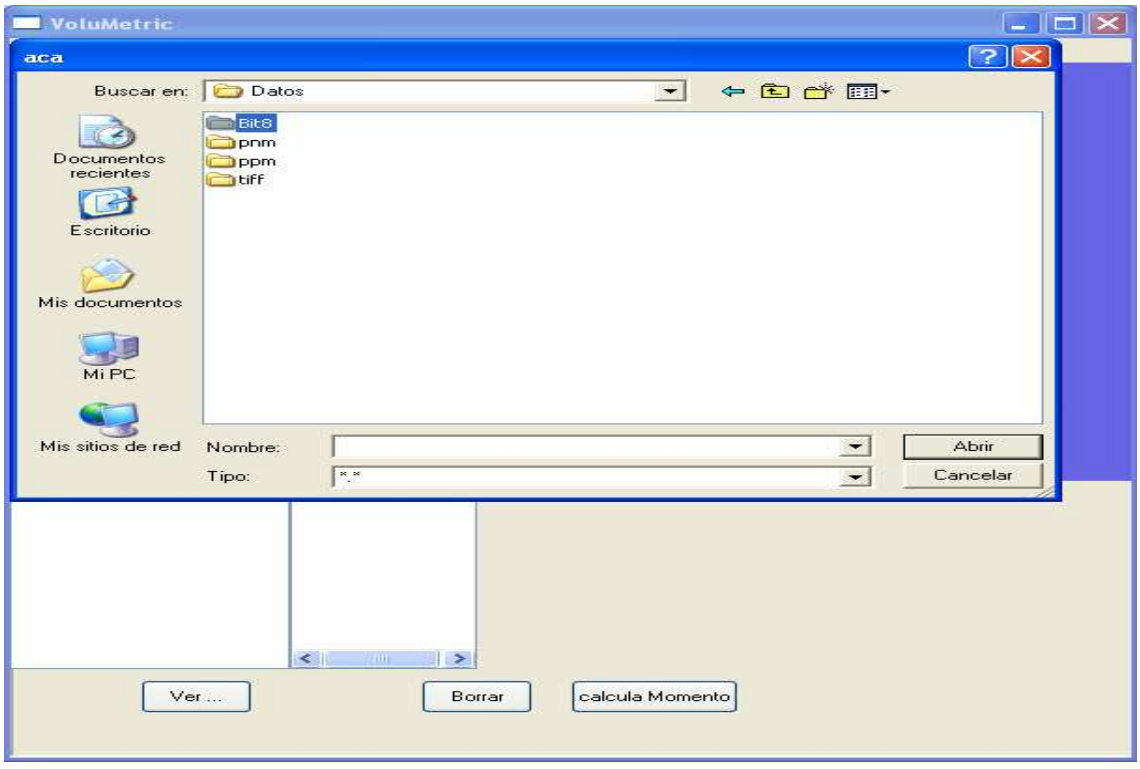


Figura 4.9. Elección del archivo a utilizar.

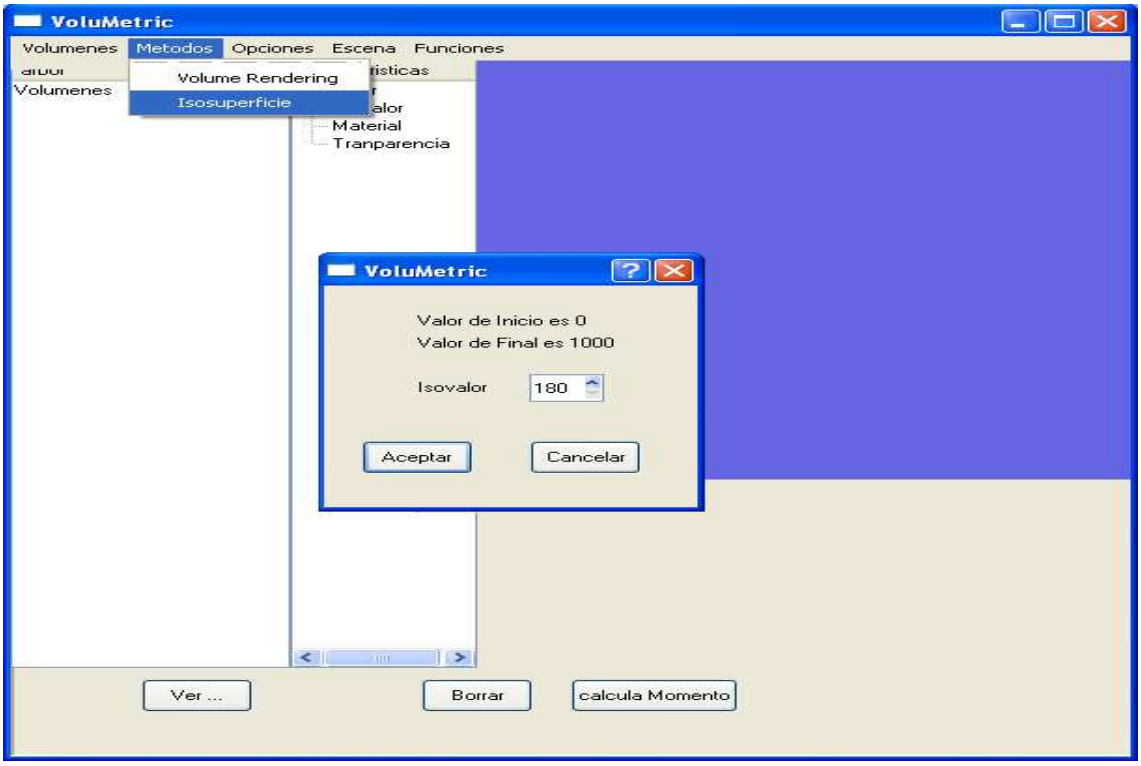


Figura 4.10. Elección de la técnica de *isosuperficie* e isovalor.

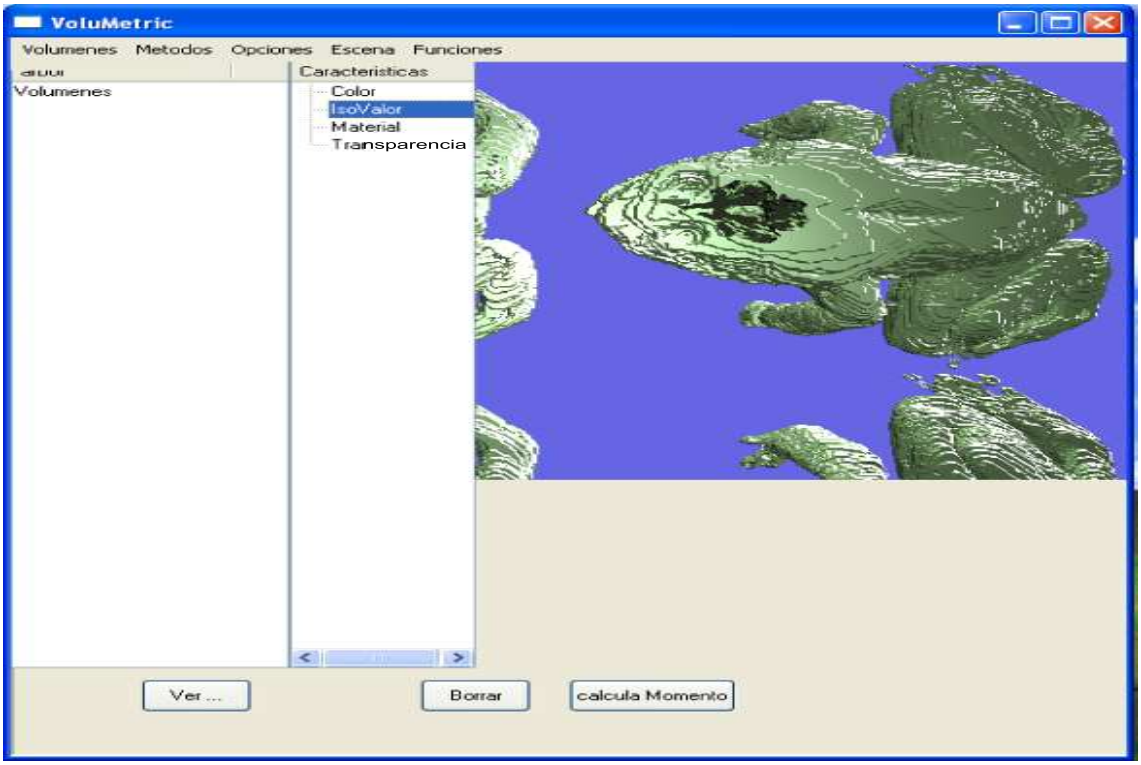


Figura 4.11. *Isosuperficie* resultante.

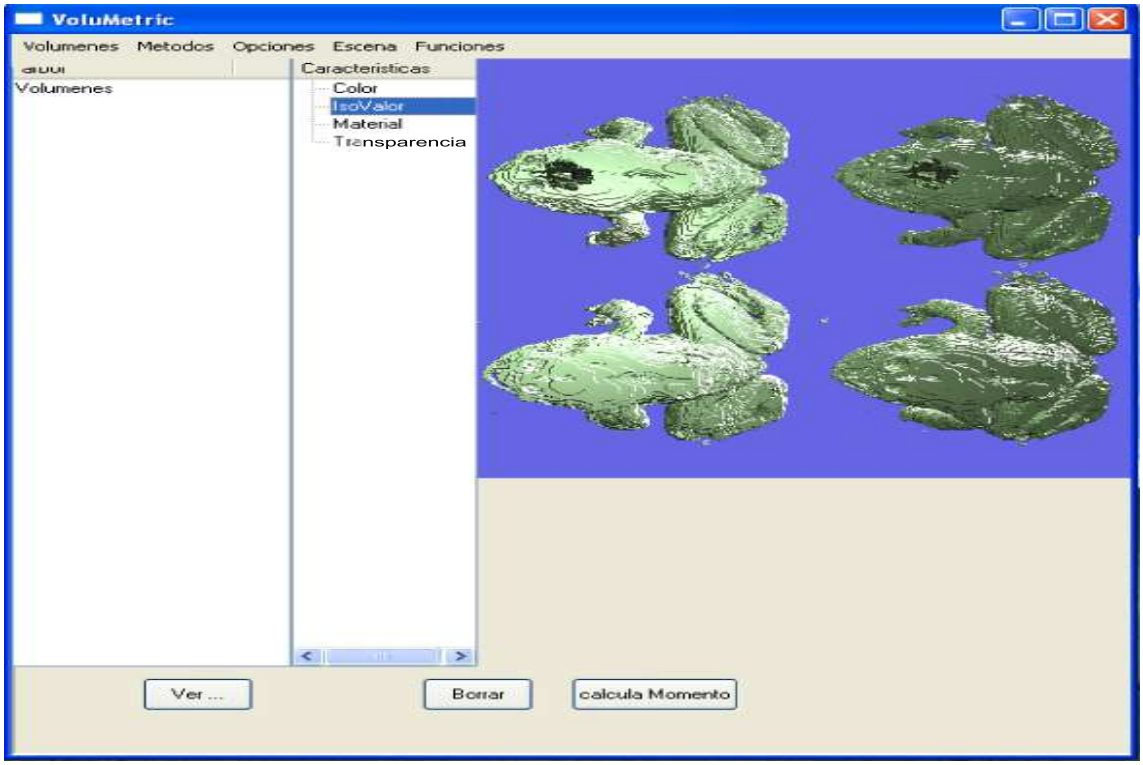


Figura 4.12. Otra vista de la *isosuperficie* resultante.

Ejemplo de la realización de un *Volume rendering.*

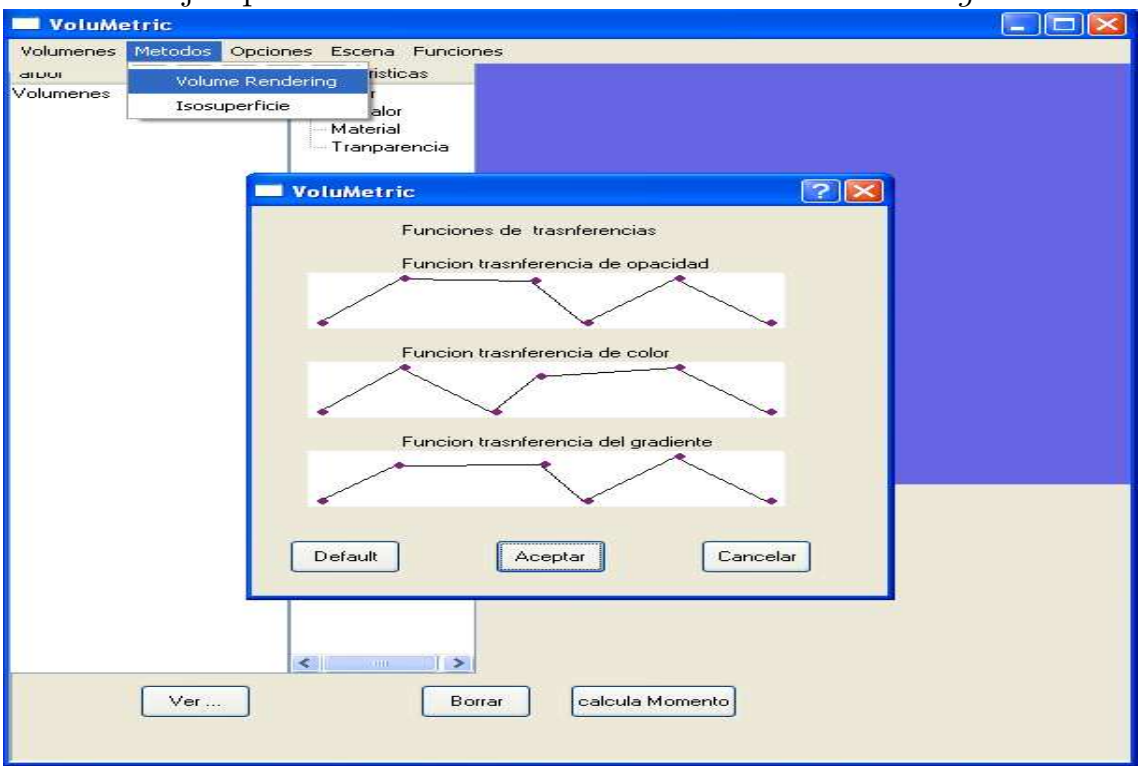


Figura 4.13. Elección del método *volume rendering* y Realización de las funciones de transferencia.

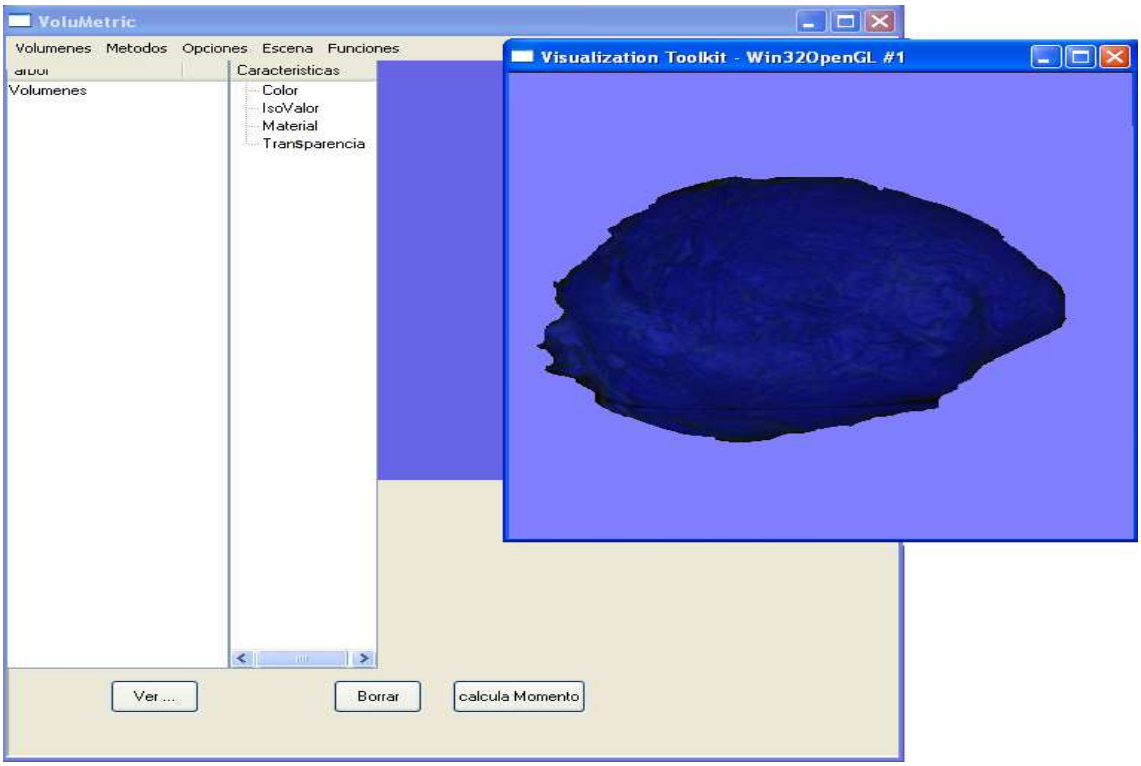


Figura 4.14. Resultado del *volume rendering.*

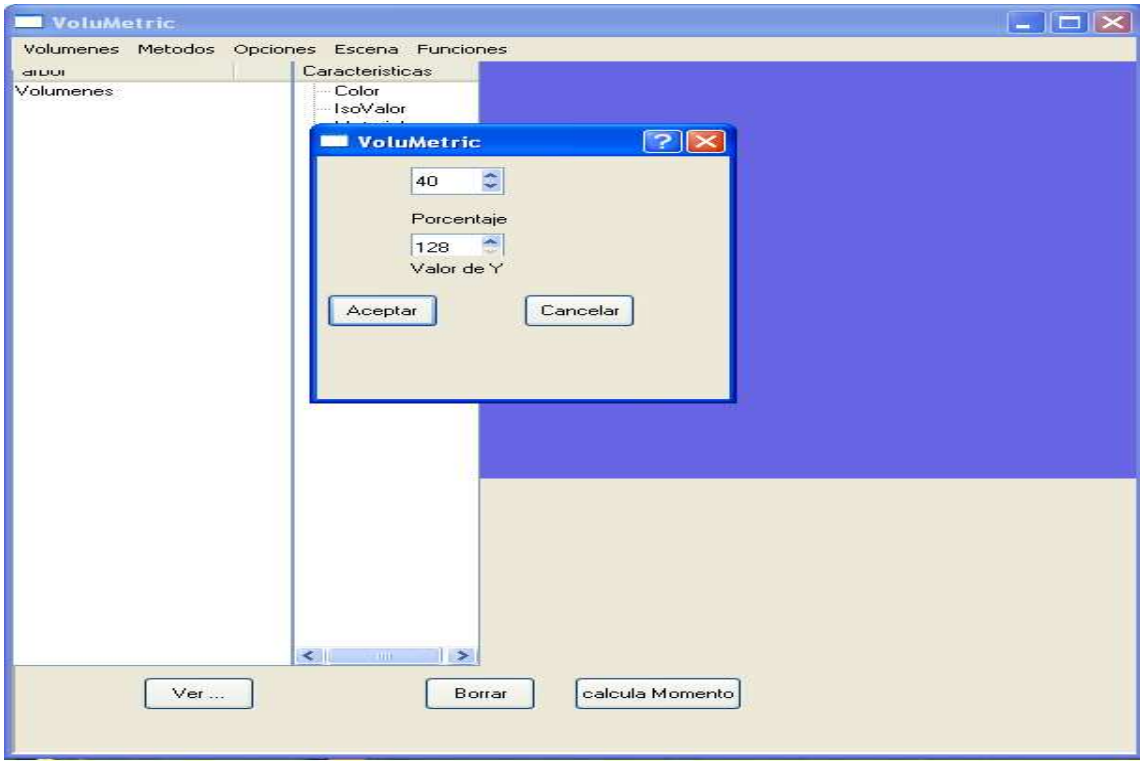


Figura 4.15. Opciones para calcular los momentos de orden mayor.

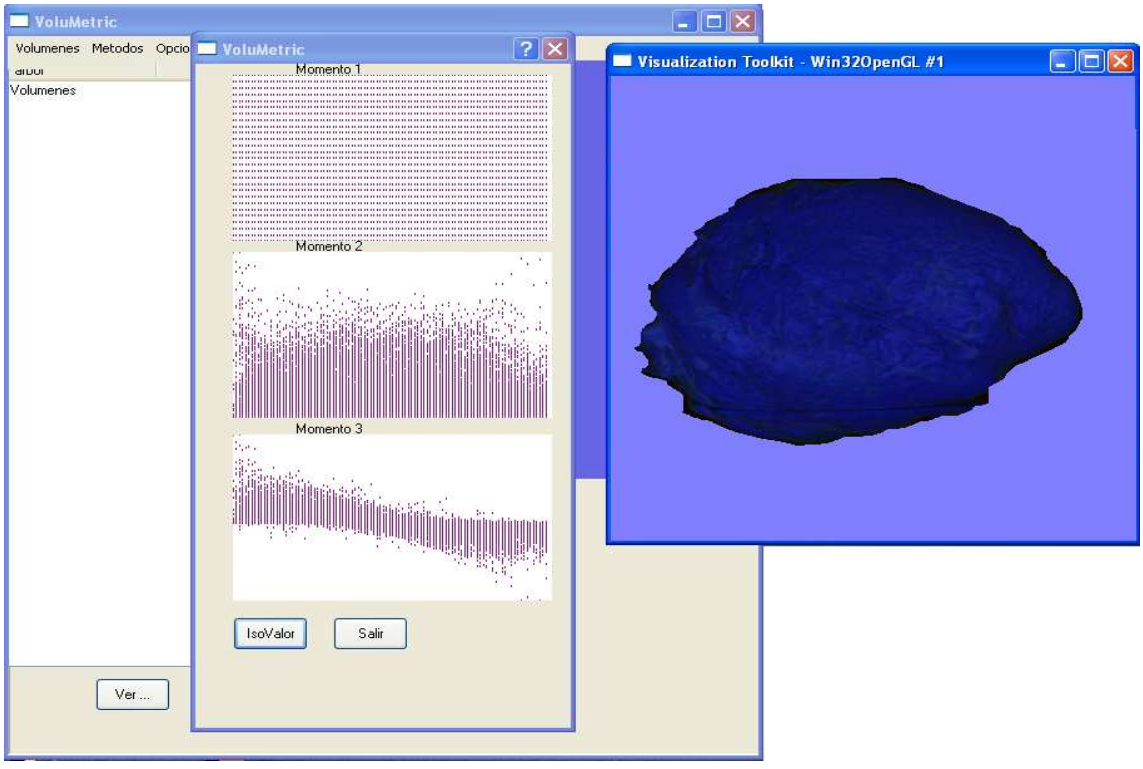


Figura 4.16. Gráficas resultantes de los momentos.

Capítulo 5
Conclusiones

Las metas que se propusierón en el planteamiento del kernel son ambiciosas, por lo mismo el tamaño del proyecto es grande por lo que no se puede realizar en una sola fase de desarrollo, por esta causa solo se concluyó una parte del desarrollo. El desarrollo del kernel se planteó como distintos objetos, los cuales atacarían problemas específicos, logrando de esta manera los resultados especificados. La parte inicial del desarrollo se ha concluido satisfactoriamente; la creación de los contenedores de datos, el planteamiento del *pipeline* en el kernel, las clases necesarias para las interacción con los datos, son solo algunos aspectos que han sido desarrollados.

Los distintos aspectos que se platearón para el kernel están siendo desarrollos en orden descendente, del más importante al menos importante; aunque todos los componentes son importantes, la forma de obtener el grado de importancia radica en las neceidades básicas para la creación de la visualización. Los aspectos del desarollo pendientes son: la creación de trabajo colaboratorio, la integración de sonido, entre otros aspectos; los cuales aún siguen en la etapa de estudio del diseño y prueba de las herramientas, razón por la cual no se concluyó su integración dentro del kernel. Aún se sigue estudiando la mejor manera de integrar dichos aspectos, donde se cuenta con algunas propuesta de solución (ver Capítulo 6, Trabajo futuro).

Las especificaciones de los parámetros de comparación que se tomaron en cuenta para el desarrollo del kernel son los siguientes i) el tiempo de desarrollo de las aplicaciones, ii) la complejidad del diseño en las aplicaciones, y la iii) funcionalidad de la aplicación. La realización de la aplicación de evaluación del kernel, que se llevó acabo (ver Cap´4. Aplicación) sirvió de referencia para verificar los parámetros de verificación mencionados, entre los cuales se pudo revisar tanto el diseño como la funcionalidad del mismo.

En las comparaciones se tomaron en cuenta sistemas que se utilizan para realizar las mismas aplicaciones o que brindan las mismas funcionalidades que el kernel, se tomamos en cuenta las posibilidades que brinda el kernel, así como sus desventajas. La aplicación que se desarrolló satisface las expectativas planeadas en su diseño, así como su tiempo de desarrollo.

Los criterios de evaluación son la satisfacción cualitativa de los objetivos establecidos para el kernel, por lo que se revisó a manera de *check-list*, el cumplimiento satisfactorio de estos objetivos, tomado como mejor, equivalente o peor; con respecto a las alternativas mencionadas.

Cuadro Comparativo

Propiedad	*vtk*	Kernel vs *OpenGL*	*OpenDx*
Portabilidad	equivalente	equivalente	equivalente
Manejo de sonido	mejor	mejor	mejor
Manejo de técnicas de visualización	equivalente	mejor	equivalente
Especialización en tipo de datos	equivalente	mejor	mejor
Mayor interactividad	mejor	mejor	mejor
Extensibilidad	mejor	mejor	mejor
Multidespliegue	mejor	mejor	equivalente
Reusabilidad	mejor	mejor	mejor
Bajo costo	equivalente	equivalente	equivalente
Flexibilidad	mejor	mejor	mejor
Rendimiento	mejor	mejor	mejor
Sencillez	mejor	mejor	peor
	Desventajas		
lenguajes en que se programa kernel	peor solo se programa en C++	peor	

Cuadro C.1. Cuadro de la evaluación cualitativa.

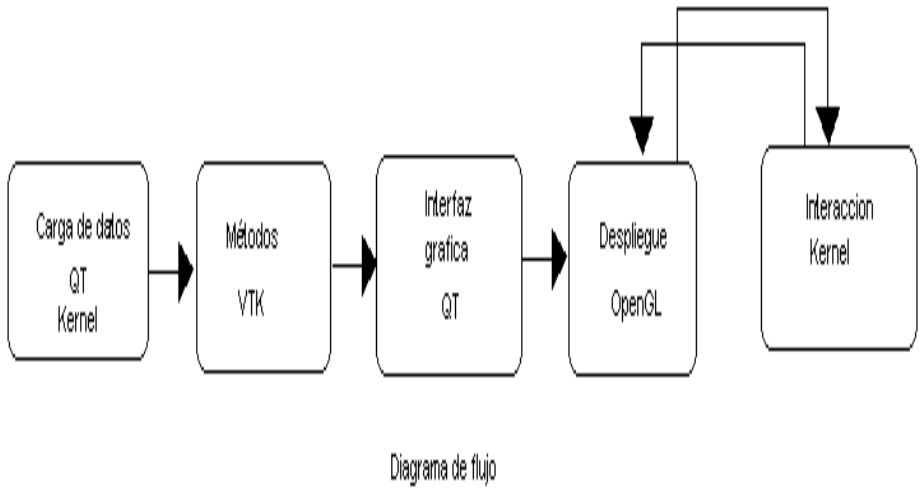


Figura C.1. Diagrama de flujo de trabajo de las implementaciones.

El diseño del kernel y su modularidad ofrecen una gran ventaja sobre la opción de utilizar las bibliotecas por separado, ya que se reduce significativamente la cantidad de código fuente a optimizar y a dar mantenimiento. El número de líneas de la aplicación en promedio es de 250 líneas de código. Pero si se realizara la aplicación sin la ayuda del kernel, las líneas del código aumentarían de manera alarmante. Enseguida se muestra un pequeño ejemplo; si deseamos realizar manualmente la función de cargar datos de 8 bits en un contenedor de *vtk*, sin interfaz gráfica o manejo de escena; simplemente la función que lea datos de 8 bits de un archivo y los preparé para ser asignados a un contenedor de *vtk*.

```
//Declaraciones de variables

        char archivo[120];
        bool color;
        bool bProfu;
        int iAncho;
        int iAlto;
        int Profun;
        unsigned short *iArre;
        int contador;
        int IniRan;
        int FinRan;

//Funcion para solicitar alto ancho

Alto_ancho()
{
 printf("Dame el ancho de la imagen \n");
scanf("%i",&iAncho);

printf("Dame el nombre del archivo a leer");
scanf("%s",arch01);
}
```

Función para reservar memoria para los arreglos necesarios, para las variables de *vtk*. Note que aún no serán asignados al contenedor.

```
//Funcion de Lectura
FILE *arch01;
 char cad[10];
 unsigned char pal[3];
 unsigned int num;
 if(((arch01=fopen(archivo,"rb"))!=NULL))
   {
      contador=0;
      while (!feof(arch01)&&(contador<(iAncho*iAlto*Profun)))
       {
         fread(pal,2,1,arch01);
 pal[2]='\0';
 num=pal[0]<<8;
 num|=pal[1];
         iArre[contador]=num;
         contador++;
       }
```

```
 fclose(arch01);
   }
 else
  {
    printf("No se pudo abrir \n");
  }
..
```

Necesitarían programar alrededor de otras 6 funciones más para la verificación y preparación de la integración a un contenedor de *vtk*, que aproximadamente son 143 líneas más de código a las mostradas arriba. Pero con la utilización del kernel quedaría solamente el siguiente código.

```
Carga *carga= new Carga();
Carga-> AsignaArchivo(arch01);
Carga->AsignaAncho(256);
Carga->AsignaAlto(256);
Carga->AsignaProfun(64);
Carga->ReservaMemoria();
Carga->lee();
```

Con estas 5 líneas de código ya están listos los datos del archivo de código para que sean recibidos por un contenedor de *vtk* tipo imagen. En esta simple función pudimos darnos cuenta del enorme ahorro de líneas de código en la aplicación. Si se deseara realizar toda la aplicación, se necesitarían alrededor de 8256 líneas de código para realizar una aplicación funcional, contra 250 líneas con el kernel.

Otro ejemplo de la utilidad del kernel es la aplicación de los métodos como el *volume rendering*. El siguiente ejemplo es una aplicación sencilla, la realización del *volume rendering* de un conjunto de datos ya cargados, solo mostrarlos y tener interación con el resultado. El código sencillo de vtk se llevaría acabo con 64 líneas; que son las siguiente:

```
  vtkColorTransferFunction *cTFun = vtkColorTransferFunction::New();
  cTFun->AddRGBPoint(   0, 1.0, 0.0, 0.0 );
  cTFun->AddRGBPoint(  64, 1.0, 1.0, 0.0 );
  cTFun->AddRGBPoint( 128, 0.0, 1.0, 0.0 );
  cTFun->AddRGBPoint( 192, 0.0, 1.0, 1.0 );
  cTFun->AddRGBPoint( 255, 0.0, 0.0, 1.0 );

  vtkPiecewiseFunction *goTFun = vtkPiecewiseFunction::New();
  goTFun->AddPoint(   0, 0.0 );
  goTFun->AddPoint(  30, 0.0 );
  goTFun->AddPoint(  40, 1.0 );
  goTFun->AddPoint( 255, 1.0 );

  vtkVolumeProperty *prop;
```

```
int index = 0;

        prop = vtkVolumeProperty::New();
        prop->SetShade(k);
        prop->SetAmbient(0.3);
        prop->SetDiffuse(1.0);
        prop->SetSpecular(0.2);
        prop->SetSpecularPower(50.0);
        prop->SetScalarOpacity(oTFun);
        prop->SetGradientOpacity( goTFun );
        prop->SetColor( cTFun );
        prop->SetColor( gTFun );
        prop->SetInterpolationTypeToNearest();

vtkVolumeProperty *mipprop;

    mipprop= vtkVolumeProperty::New();
    mipprop->SetScalarOpacity(oTFun2);
    mipprop->SetColor( cTFun );
    mipprop->SetColor( gTFun );
    mipprop->SetInterpolationTypeToNearest();

 vtkVolumeRayCastIsosurfaceFunction *isosurfaceFunction =
  vtkVolumeRayCastIsosurfaceFunction::New();
isosurfaceFunction->SetIsoValue(80);

vtkFiniteDifferenceGradientEstimator *gradest =
  vtkFiniteDifferenceGradientEstimator::New();


vtkVolume *volume;

    volume = vtkVolume::New();
    volume->AddPosition( 0, 0, 0 );
    ren->AddProp(volume);

vtkVolumeRayCastMapper *raycastMapper;

  raycastMapper = vtkVolumeRayCastMapper::New();
  raycastMapper->SetInput(reader->GetOutput());
  raycastMapper->SetGradientEstimator(gradest);
  volume->SetMapper( raycastMapper );


    volume->SetProperty( prop );
```

```
        raycastMapper->SetVolumeRayCastFunction( isosurfaceFunction );
        volume->SetProperty( prop );

  renWin->SetSize(400,350);

  ren->GetActiveCamera()->Zoom(2.0);

  renWin->Render();


    iren->Start();
```

En el caso de utilizar el kernel, se reduciría 4 líneas de código, sin contar la carga de los datos; como se muestra las siguientes líneas de código.

```
int  main(int argc,char *argv[])
{
  KvVTKImage *image= new KvVTKImage();
  image->AsignaArchivo(argv[1]);
  image->AsignaAlto(356);
  image->AsignaAncho(356);
  image-> AsignaRango (0,256);
  image->AsignaFormato(5);
  image->AsignaEspaciado(1.5,1.5,1.5);
 //inicio del Metodo Volume rendering
         itemp[0]=5;
         itemp[1]=128;
         itemp[2]=245;
         ftemp[0]=5.0;
         ftemp[1]=45.0;
         ftemp[2]=5.0;
   image->AsignaPuntoOp(3,itemp,ftemp);// 1 linea
   image->AsignaPuntoCo(3,itemp,ftemp);// 2 linea
   image->AsignaPuntoGr(3,itemp,ftemp);// 3 linea
 // rendering del volume rendering resultado.
   image->HazVolume(); // 4 linea
```

Para la realización de una *isocuperficie* con el kernel, si solo tomamos en cuenta la realización del método, sería necesario una línea de código.

```
int  main(int argc,char *argv[])
{
  KvVTKImage *image= new KvVTKImage();
  image->AsignaArchivo(argv[1]);
  image->AsignaAlto(356);
  image->AsignaAncho(356);
  image-> AsignaRango (0,256);
  image->AsignaFormato(5);
  image->AsignaEspaciado(1.5,1.5,1.5);
 //esta linea es la encargada de todo el proceso de la isosuperfice.
  image->HasIso(120);

}
```

En caso de no utilizar el kernel, las líneas de código aumentarían a 31 líneas, como a continuación se muestra.

```
int main (int argc, char **argv)
{

  vtkRenderer *aRenderer = vtkRenderer::New();
  vtkRenderWindow *renWin = vtkRenderWindow::New();
    renWin->AddRenderer(aRenderer);
  vtkRenderWindowInteractor *iren = vtkRenderWindowInteractor::New();
    iren->SetRenderWindow(renWin);

  vtkVolume16Reader *v16 = vtkVolume16Reader::New();
    v16->SetDataDimensions(64,64);
    v16->SetDataByteOrderToLittleEndian();
    v16->SetFilePrefix (argv[1]);
    v16->SetImageRange(1, 93);
    v16->SetDataSpacing (3.2, 3.2, 1.5);

  vtkContourFilter *skinExtractor = vtkContourFilter::New();
    skinExtractor->SetInput((vtkDataSet *) v16->GetOutput());
    skinExtractor->SetValue(0, 500);
  vtkPolyDataNormals *skinNormals = vtkPolyDataNormals::New();
    skinNormals->SetInput(skinExtractor->GetOutput());
    skinNormals->SetFeatureAngle(60.0);
  vtkStripper *skinStripper = vtkStripper::New();
    skinStripper->SetInput(skinNormals->GetOutput());
  vtkPolyDataMapper *skinMapper = vtkPolyDataMapper::New();
```

```
    skinMapper->SetInput(skinStripper->GetOutput());
    skinMapper->ScalarVisibilityOff();
  vtkActor *skin = vtkActor::New();
    skin->SetMapper(skinMapper);
    skin->GetProperty()->SetDiffuseColor(1, .49, .25);
    skin->GetProperty()->SetSpecular(.3);
    skin->GetProperty()->SetSpecularPower(20);
    skin->GetProperty()->SetOpacity(1.0);

  vtkCamera *aCamera = vtkCamera::New();
    aCamera->SetViewUp (0, 0, -1);
    aCamera->SetPosition (0, 1, 0);
    aCamera->SetFocalPoint (0, 0, 0);
    aCamera->ComputeViewPlaneNormal();


  aRenderer->AddActor(outline);
  aRenderer->SetActiveCamera(aCamera);
  aRenderer->ResetCamera ();
  aCamera->Dolly(1.5);


  aRenderer->SetBackground(1,1,1);
  renWin->SetSize(640, 480);

  aRenderer->ResetCameraClippingRange ();

  iren->Initialize();
  iren->Start();

  return 0;
}
```

La visualización científica es una herramienta muy eficaz para i) la exploración y el análisis en los procesos de investigación y desarrollo, ii) la comunicación especializada entre colegas de una disciplina, y ii) la docencia y la difusión de los conocimientos descubiertos en el proceso y conocimientos antes establecidos. Pero la accesibilidad a estas herramientas solo puede lograrse a través de aplicaciones sencillas en cuanto a uso, pero poderosas y rigurosas en cuanto a metodologías y técnicas especializadas.

El desarrollo de dichas aplicaciones es costoso, pero un kernel permite el desarrollo de prototipos y su mejora continua; potenciando y extendiendo el uso de la visualización científica en diferentes disciplinas y ámbitos diversos.

Con el kernel es posible desarrollar aplicaciones *ad hoc* rápidas y que cubran las exigencias en rendimiento y flexibilidad del especialista, así como las exigencias en facilidad de uso para los no expertos. Esto se logra al contar con una plataforma capaz de implementar las

técnicas más complejas y las más novedosas, pero con la especificación concreta y cerrada, hasta donde se requiera. De esta forma contamos con un sistema no monolítico que permite el diseño de componentes para lograr prototipos sólidos, los cuales eventualmente pueden conformar aplicaciones finales.

El kernel constituye una herramienta eficaz para la implementación rápida de nuevas técnicas o nuevos algoritmos en la literatura especializada o nuevas ideas en un grupo de desarrollo de entornos de trabajo de visualización.

Como concluciones técnicas puntualizamos las siguientes:

1. Las propiedades y ventajas analizadas de cada biblioteca (ver Capítulo 3. Desarrollo) no se perdieron en el proceso de integración

   a) La portabilidad entre plataformas continuó siendo transparente, siempre que se respeten las versiones de las bibliotecas; inclusive los cuidados en implementación son menores.

   b) El bajo costo no fue afectado en absoluto por la razón de que el kernel está desarrollado bajo la licencia de GNU.

   c) La flexibilidad en las aplicaciones no disminuyó, al contrario aumentó debido a la modularidad del kernel y a las opciones que éste proporciona para la creación de más módulos e integración de nuevos programas.

   d) La reusabilidad fué la parte donde hubo mayor contribución por parte del kernel, debido a que todo código que fuera desarrollado en él, puede volver a ser reimplementado en cualquier aplicación que se desee.

   e) De la misma manera que la extensibilidad de los códigos, aumentó la reusabilidad por la sencillez de hacer crecer las aplicaciones sin mayores problemas que el diseño de los nuevos componentes.

2. La integración fue más simple al contar con un diseño muy modularizado del kernel; teniendo como ganancia varios aspectos:

   a) La reducción en tiempos de diseño.

   b) Se redujerón los tiempos de desarrollo.

   c) La redución en líneas de código.

   d) La reducción en tiempo de mantenimiento del sistema.

   e) El aumento en el ciclo de vida de las aplicaciones realizadas con el kernel.

3. El flujo de trabajo con que el se puede manejar en el desarrollo de las aplicaciones les permite la reducción de los tiempos de desarrollo.

   a) ya que solo hay que conocer los métodos de las nuevas técnicas y no todo el proceso de carga de información.

4. Al contar con una sola biblioteca, se unificó el estilo de notación.

   a) logrando una reducción de líneas de código.

5. Se logró que la programación de las aplicaciones fuera más simple por las simplicidad de las instrucciones del kernel tanto en su uso como en su sintaxis.

   a) evitando de esta manera la saturación de tiempos de aprendizaje de los desarro−lladores.

   b) logrando una curva de aprendizaje mucho menor.

6. El proceso de depuración fue más simple, por la modulación del kernel.

   a) se lograrón establecer de manera más sencilla, rápida y precisa de los lugares donde se formaban cuellos de botella o los posibles cuellos de botella.

   b) al contar con la optimización del kernel, se tiene la seguridad de que sus instrucciones eran óptimas y no era necesario revisar toda la aplicación en sí para la optimización.

7. El mantenimiento de las aplicaciones que fueron desarrolladas por el kernel, es más sencillo por las caractericticas del kernel.

   a) la modularidad permite de manera sencilla cambiar módulos solamente cuando se actualice el kernel sin tener que cambiar todo.

8. El crecimiento del kernel también significa el crecimiento de las aplicaciones realizadas con él.

   a) de manera sencilla, las aplicaciones desarrolladas con el kernel pueden contar con las nuevas características de las versiones siguientes del kernel.

   b) el crecimiento de las aplicaciones no solo está dado por el kernel, ya que permite que nuevas herramientas sean programadas por los usuarios o que se incorporen herramientas localizadas por el usuario.

9. La modularidad del kernel permite el uso selectivo de módulos.

   a) no es necesario utilizar todas las herramientas (módulos) del kernel; se pueden elegir sólo las que se requieran para la implementación específica.

Capítulo 6

Trabajo futuro

En el esquema presentado al inicio de este trabajo se expusierón aspectos de gran importancia para una aplicación moderna de visualización, como son el esquema cliente-servidor, sonido y la oportunidad de trabajar en un entorno colaboratorio. El desarrollo del kernel de visualización no ha terminado, ya que aún no se han cumplido todas las expectativas del diseño original; satisfactoriamente podemos decir que la primera etapa del desarrollo del kernel se ha cumplido, pero aún falta continuar con el trabajo, ya que la utilidad del caso de estudio indica que vale la pena. Las partes del trabajo faltantes están en etapa de diseño y sigue la discusión de la mejor manera de integrarlas al kernel, debido a la complejidad que ellas representan; sobre todo a nivel de diseño.

Sonido

El diseño que se planea seguir para el control del sonido es la creación de una clase programada en *C++*, con la unión de *OpenAL*. Esta biblioteca fue previamente elegida y está siendo estudiada tanto en su diseño como en su estructura de trabajo, para evitar dependencias en lo posible. La biblioteca cumple con las expectativas solicitadas, pero el diseño para la clase aún está en discusión, debido a que la biblioteca tiende a hacer que las demás clases del kernel dependan de ella. Es decir la futura clase puede estar integrada en algunas otras clases para su mejor manejo, como se muestra en la figura T.1.

Este diseño obligaría a los usuarios a cargar la biblioteca de *OpenAL* al utilizar algunas clases del kernel. Además de sobrecargar a la clase con más funciones, que según descubrimos no son necesarias en su totalidad. El otro esquema de trabajo que es el deseado es el que se muestra en la figura T.2, este esquema pretende que funciones de la clase KvSonido obtengan los parámetros necesarios de otras clases que ya los tengan, para evitar que la clase KvSonido se sobrecarge con información que no utilice. A pesar de que este es el esquema deseado en la forma de trabajar de la futura clase, se encuentra un gran obstáculo para implantarlo, ya que en ocasiones se dependerá que ciertas clases del kernel sean llamadas para que KvSonido funcione. Hasta este momento se han estudiado cuidadosamente los siguientes puntos: cuales son los parámetros, de qué clases se podrían obtener y cuál sería la mejor manera de evitar la utilización innecesaria de clases.

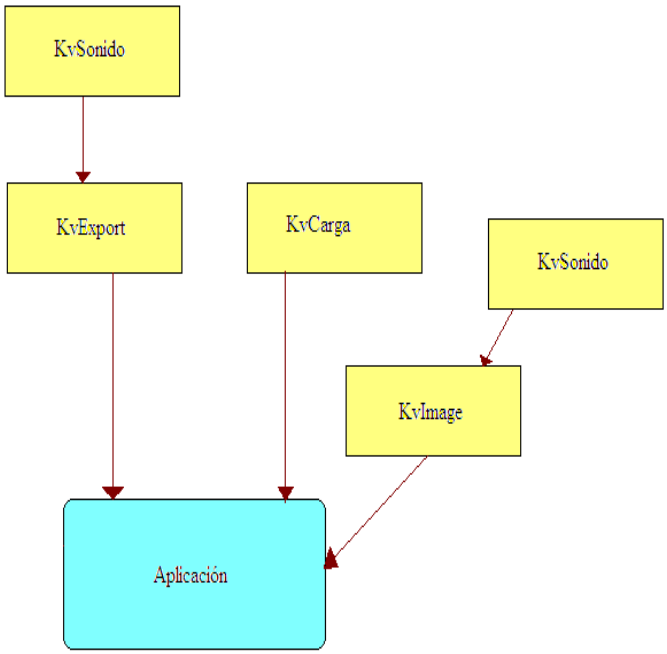


Figura T.1 Esquema de diseño con sobrecarga.

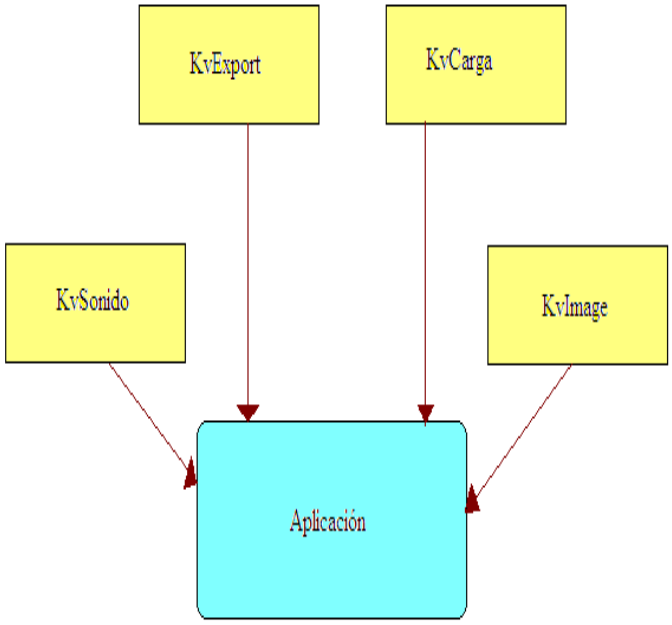


Figura T.2. Esquema de diseño de aplicaciones deseado.

Colaboratorio

Para implementar esta parte del kernel se cuenta con la biblioteca de *OpenH323*, la cual cumple con las expectativas pedidas. El avance alcanzado es el estudio de las bibliotecas, tanto de su diseño como de su estructura de trabajo, así como las respectivas pruebas de la misma. Se tiene planeado la elaboración de un diseño especial para esta clase, ya que al

comienzo del estudio se pensaba que el esquema y el diseño que utilizáramos en la clase KvSonido, se podría reimplantar en esta clase. Este idea fue benéfica, ya que al tenerla en mente se dedicó todo el trabajo a la realización del esquema de la clase KvSonido; aunque con el avance en el proyecto nos percatamos de que no sería posible realizarlo, ya que la futura clase de *OpenH323* necesita aún más parámetros para su correcto funcionamiento; además de que el diseño de la clase que se implantará para el colaboratorio será muy diferente al implantado en KvSonido. Entre otros detalles que se descubrieron es que la clase que implante las funciones de colaboratorio, será más compleja para lograr en correcto manejo de video o sonido y despliegues gráficos remotos, por lo tanto se requiere un mayor tiempo de estudio y planteamiento.

Esquema Cliente-Servidor

El esquema de cliente−servidor en las aplicaciones está siendo implementado y muy solicitado, por lo tanto no lo podemos dejar de tomar en cuenta para las futuras aplicaciones que se realicen con el kernel. Para la implementación del esquema se tienen pensado dos alternativas, la i) está en función de la realización de dos clases y la ii) es el desarrollo de tres clases; ambas propuestas aún se encuentran en estudio y no se ha llegado a una decisión final, como se ve en la figura T.3.

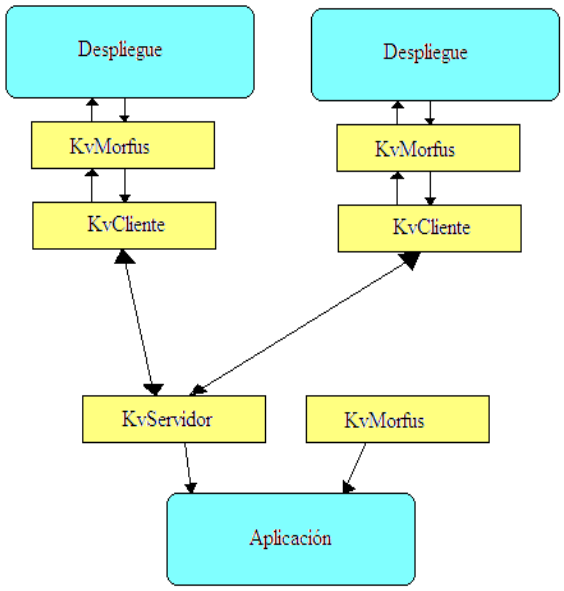


Figura T.3. Propuesta 1 para la implantación del esquema cliente-servidor.

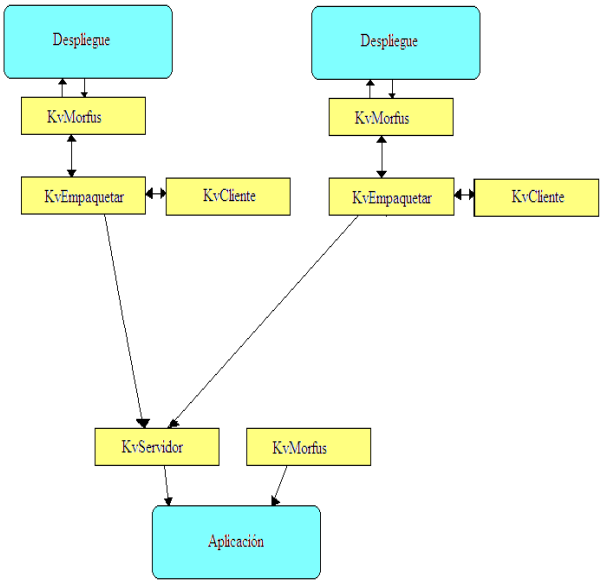


Figura T.4. Propuesta 2 para la implantación del esquema cliente-servidor.

Ambas propuestas cuentan con un factor en común, aunque el diseño no está en su totalidad terminado, se contempla para cualquiera de ellas contar con dos clases: KvCliente y KvServidor. Estas clases estarán programadas en *C++* y *Qt*; el último proporcionará la comunicación vía red y las operaciones necesarias para mantener la comunicación. La clase KvCliente tiene la capacidad de recibir un elemento de *Qt*, es decir una ventana o algún control de *Qt*; por esa causa, el despliegue se encuentra encapsulado en un canvas de *Qt*, para la correcta recepción y envió. La clase KvServidor tiene la capacidad de recibir y enviar un elemento, de esta manera trabajarán en conjuntos ambas clases.

En la segunda propuesta se está contemplando la posibilidad de crear una tercera clase que sería KvEmpaquedar, la cual se encargaría de solicitar los datos a la clase de KvMorfus en forma de despliegues gráfico y colocarla correctamente en el *canvas* de *Qt* y entregarlo de esta manera tanto al server como al cliente. La otra manera sería que el cliente se encargara de esta fase, además de las responsabilidades de recibir y mandar correctamente los mensajes entre su usuario y el servidor, como se muestra en la figura T.2.

La decisión debe contemplar ventajas y desventajas en ambas; por un lado la primera opción brindaría un mejor entendimiento de los mecanismos al momento de utilizarla y por otro lado sería necesario una inversión de tiempo un poco mayor para el estudio de cómo recibiría el cliente la información o como debería ser entregada. La segunda opción sobrecargaría el trabajo del cliente ya que además de tener que atender a los mensajes del servidor y realizar acciones pertinentes con el usuario, debería invertir tiempo en el proceso y acomodo de los datos, tanto para enviar como para recibir información.

Implementación de más casos de estudio y funcionalidad en la aplicación

La retroalimentación con los usuarios es fundamental para la evolución certera de este desarrollo; por esto es importante implementar más casos de estudio en el área biomédica

y en otras áreas, principalmente para la visualización de fenómenos naturales (dinámica de fluidos, por ejemplo), de ciencias sociales y del comportamiento, etc. Así mismo es necesario incorporar mayor funcionalidad, en respuesta a las exigencias de los diversos grupos de usuario. Dentro de estas funcionalidades se tienen detectadas las siguientes, según demanda de usuarios y capacidades incluidas en sistemas de visualización revisados.

Manejo de Viewports

Para lograr un mejor entendimiento y manejo de los datos es conveniente contar con varias ventanas de despliegue con la facilidad de manejar diferentes perspectivas, aspectos o técnicas del objeto. Esta manera de trabajo es clásica en el área de *CAD/CAM*, la cual proporciona una mejor apreciación de los objetos que se están desarrollando o analizando. Para lograrlo es necesario estudiar la mejor manera de comunicación de las ventanas y de repartir el despliegue entre ellas; el estudio debe realizarse también en una nueva clase que se encargue del *render* de los volúmenes, para que tenga el soporte necesario que le exijan las ventanas.

Planos de corte

Una de las herramientas más requeridas en los sistemas de visualización son los cortes en un objeto. Un corte es un plano que pasa a través de un objeto y se mapean los valores que toca en el espacio; generalmente estos valores los toma de algunos de los campos registrados. Por ejemplo, si se está visualizando la tomografía de una cabeza, la forma o estructura se obtiene a través de una *isosuperficie* o un *volume rendering*, mientras que a través de un corte podemos observar el comportamiento de alguna variable (la función) como la temperatura en una sección de la parte interna de la cabeza.

Estos cortes pueden colocarse en cualquier posición dentro de la escena y muestrear el campo; no obstante, se prefiere iniciar con posiciones centradas y perpendiculares a los ejes. La técnica básica requiere que se dibujen o borren a conveniencia, las secciones de los objetos gráficos que se ubiquen delante o detrás del plano de corte.

Mayor interactividad

Se pretende que la parte de interactividad con la escena y los objetos gráficos aumente, ya que por el momento está muy limitada, pero en este caso ya se comenzó a buscar la mejora en la clase de despliegue y de interactividad; aunque aún no se logra determinar la manera de corregir este punto. Por el momento ya se tienen pensadas las metas que se pedirán para la primera parte de dicha interacción.

Conversión de archivos (lectura y escritura en formatos estándares)

La representación de objetos gráficos tridimensionales es muy variada en sus formas matemáticas y en los formatos digitales en los que se almacena la información. Los modelos geométricos y volumétricos pueden representarse a través de polígonos, funciones implícitas o campos escalares, entre otras aproximaciones. Diferentes áreas o disciplinas y diferentes herramientas de software tienen preferencias sobre la representación y sobre los formatos de almacenamiento; estas particularidades responden a las propiedades de los datos y a sus

datos, así como a las conveniencias de los grupos de trabajo.

En el área biomédica existen diversos tipos de datos y formatos; sin embargo, *DICOM* (imágenes digitales en comunicación y medicina) es tomado como estándar para imágenes volumétricas. Este formato, por su gran uso es muy importante para esta área, por lo que es necesario contar con el manejo para su lectura, como para guardar los datos en dicho formato.

Tomando en cuenta estas consideraciones se decidió buscar la manera de implementarlo en el kernel con un diseño flexible, eficiente y robusto para su implementación, manteniendo los estándares con que cuenta el kernel. Se pensó en un objeto parecido al de la clase KvCarga, pero con algunas diferencias ya que el cargador estaría programado con *C++* y un cargador de *DICOM*, el cual aún no ha sido localizado. En la búsqueda del cargador, se dieron varios nombres, entre los que destacó *vtk*, porque ya forma parte del kernel de visualización; aunque no está cerrada la posibilidad de encontrar un cargador mejor que *vtk*. Se llegó a pensar en cargar estas imágenes con *vtk*, pero en el estudio realizado de compatibilidad se encontró que *vtk* es muy eficiente al cargar las imágenes, pero no permite el acceso a los datos de manera sencilla y eficiente, por lo que se descartó. Otras áreas a considerar para el desarrollo de clases de carga de datos específicas a formatos son *CAD/CAM/CAE*, Fluidos Computacionales, entre otras. En el área de *CAD/CAM/CAE* un formato muy útil es *IGES*.

La carga de datos en el kernel es una parte muy importante para el rendimiento y la accesibilidad. Este componente aún se debe mejorar para lograr un mejor desempeño en las futuras aplicaciones, ya que la manera de guardar los datos varía de manera inimaginable; por lo cual se debe buscar la forma de dar soporte a todos los actuales y futuros formatos.

Para el diseño de una clase de formatos que permitan un mejor acceso a los datos, se analizaron las estrategias para cárgarlos y utilizarlos. Una posible estrategia es crear convertidores de formatos a formatos o de los posibles formatos que saldrán en el futuro. Sin embargo, esta opción causa problemas para que el kernel tenga una manera de cargar los datos, aunque sean un formato nuevo. Así, se llegó a la conclusión de que era necesario dejar a un lado cualquier estereotipo de formato y crear una nueva manera de tener acceso a ellos.

Por ejemplo, cada formato de imágenes que se maneja, varía la manera de acomodar los datos, así como el número de bits que se le asignan a cada uno de los datos; en este caso de el número de bandas, etc. El kernel debe tener la flexibilidad de utilizar cualquiera de ellos, esa es nuestra problemática. Para resolverla se pensó en un mejor manejo de los datos en el kernel; la idea es crear una clase de formatos y métodos de acceso a datos con una sintaxis en *xml*. En el diseño de la clase encargada de realizar dicha actividad, se propone que ésta tenga la responsabilidad de interpretar la manera de leer los datos y acomodarlos como cada aplicación específica lo requiera y acorde al acomodo de los datos de entrada originales; recordemos que la manera de acomodar los datos, en muchas ocasiones determina un mejor desempeño de las aplicaciones.

La clase tendría la posibilidad no solo de leer archivos de imágenes sino de geometrías, los campos escalares, vectoriales y tensoriales (tanto sus posiciones, mallas, conexiones, valores, para los diversos tiempos de simulación), entre otros. El objetivo de esta clase sería que el usuario del kernel pueda utilizar una sintaxis en *xml* en un archivo de cabecera y

desde éste colocar las diferentes marcas que definieran la manera de acomodar y manejar los datos. De esta manera nos olvidaríamos de cómo se encuentran los datos en el archivo (o sea, sin importar el formato que utilice o la manera como están acomodados los datos), considerando sólo al archivo cabecera para leer cualquier tipo de archivo.

Buscando un mejor desempeño se exploró el desarrollo de una clase con tres componentes en su diseño; la localización de los datos, la carga de los datos y el acomodo de los datos en memoria. La localización de los datos estaría constituida por funciones que realicen un *parseo* de las marcas en *xml* para saber las características. Las marcas en *xml* serían para indicar la localización de los datos y la manera de acomodarlos en la memoria.

La carga de datos recibiría la manera de leer los datos y se encargaría de ir por el archivo o archivos para constituir todo el conjunto de datos y brindarlos a la función que se encargará de guardarlos en memoria, según se hallan pedido.

Las funciones parsearán el acomodo de los datos y recibirán los parámetros necesarios que tienen que tomar en cuenta para la respectiva reserva de memoria para los datos; dichos parámetros serán proporcionados por las funciones de parseo, las cuales se encargarán del acomodo de los datos que recibirán de las funciones de carga.

Marcas de *xml*

El *xml* es un lenguaje de marcas parecido a *html*, por eso las marcas que se proponen para las funciones que manejen en el parseo son las siguientes; con una breve explicación de cuáles serán sus funciones en el parseo.

La marca `Bit` indicará el número de bytes que sean necesarios para las funciones que tienen que hacer referencia al número de bytes que serán manejados.

```
<Bit>
   Numero
< \Bit>
```

La marca tamaño sirve para conocer el número de bytes que pertenece a cada elemento, tal como puede ser RGB o el tamaño de las variables.

```
<Tamano>
  <Bit>
     8
  <\Bit>
<\Tamano>
```

La marca `Tipo_Archivo` indicará si el archivo está en *little endian* o *big endian*.

```
 <Tipo_Archivo>
    LittleEndian | BigEndian
<\Tipo_Archivo>
```

La marca Limite proporcionará un número que será utilizado para limitar alguna función o como parámetro de funciones.

```
<Limite>
   Numero
<\Limite>
```

La marca TipoDato proporcionará el tipo de datos que manejaremos o estará esperando la función.

```
<TipoDato>
short | int | float | double | char
<\TipoDato>
```

Esta marca señalará el nombre de la referencia a algún método o carga de datos.

```
<Id>
 Nombre
<\Id>
```

La marca Archivo servirá para marcar el nombre del archivo de donde estarán los datos a leer.

```
<Archivo >
  Nombre
<\Archivo>
```

La marca Secuencia será creada como una solución a la necesidad de obtener los datos de una secuencia de archivos casi siempre de imágenes. Por esa causa la marca secuencia cuenta con tres parámetros

```
<Secuencia>
 <Archivo>
   Nombre de archivo
 <\Archivo>
 <Limite>
  Numero para senalar el limite inferior
 <\Limite>
 <Limite>
 Numero para senalar el limite superior
 <\Limite>
<\Secuencia>
```

La marca de ExpreRegular servirá para distinguir las expresiones regulares que se necesitarán para seleccionar los datos que desean trabajar.

```
<ExpreRegu>
  Expresin regular
<\ExpreRegu>
```

La marca de Valores servirá para el caso en el que se incluyan los valores en el mismo archivo de configuración. Aunque la idea de que en el archivo de control se tengan los datos, está fuera del contexto de lo que se desea, pero se les brindará esa opción a los usuarios.

```
<Valores>
 # # # #
 # # # #
# # # # # #
<\Valores>
```

La marca de Dimensión se utilizará para indicar el número de componentes del arreglo en cada una de las dimensiones. Se tomó en cuenta la utilización máxima de tres dimensiones que son X,Y y Z. Si no se desea utilizar ningún componente en la dimensión Z, simplemente no se deberá escribir ningún número en ese campo. En caso de no detectar algún parámetro, entonces se tomará el valor de omisión que es 256, 256, 0.

```
<Dimension>
 # # #
<\Dimension>
```

La marca Campo ayudará a especificar el campo de datos que deseamos manejar, es decir indicará de manera concreta en dónde están los datos y cómo los tomaremos para guardar en memoria. La marca podrá ser reconocida de varias maneras para especificar la lectura, en la función para convertir las marcas.

```
<Campo>
 <TipoDato>
 <Id>
<\Campo>

<Campo>
 <TipoDato>
 <Id>
 <Archivo> | <Secuencia>
 <Limite>-- Nos dira el numero de componentes que tendra asignado el campo.
 <ExpreRegu>
 <\Campo>

<Campo>
 <TipoDato>
 <Id>
 <Archivo> | <Secuencia>
 <Limite>
<\Campo>
```

La marca de Zona es una marca que servirá para definir una zona de datos que deseamos utilizar. En ocasiones no deseamos utilizar todo el conjunto de datos, solo deseamos trabajar con un subconjunto, entonces se utilizará esta marca para definir dicho subconjunto.

```
<Zona>
 <Id>
 <Campo>
 <Limite>
 <Campo>
<\Zona>

<Zona>
 <Id>
 <Campo>    ---- Este campo indica el campo o espacio de donde tomaremos los datos
 <Campo>    --- Este campo indica las conexiones que deberan tener los datos
<\Zona>

<Zona>
 <Id>
 <Campo>
 <Dimension>
 <Limite>
<\Zona>

<Zona>
 <Id>
 <Campo>
 <Dimension>
 <Dimension>
 <Dimension>
<\Zona>
```

La marca de Variable sirve para crear instancias de zonas y unirlas en una sola, es decir podemos tomar diferentes zonas de nuestros datos y con base en ellas crear una sola.

```
<Variable>
 <Id>
 <Campo>
 <Campo>    --- tiene que ser un numero int o short
 <Zona>
<\Variable>

<Variable>
 <Id>
 <Campo>
 <Dimension>
```

```
 <Zona>
<\Variable>

<Variable>
 <Id>
 <Campo>
 <Limite>
 <Zona>
<\Variable>

<Variable>
 <Id>
 <Limite>
 <Campo>
   <Zona>
 <Limite>
 <Limite>
<\Variable>
```

La marca Cabecera se encargará de indicar el lugar donde se encuentran los datos que deseamos manejar, con base en las marcas anteriores. Las posibles formas que puede tomar esta marca son las siguientes.

```
<Cabecera>
   <Id>
   <Variable >
    .
   ..
<\Cabecera>
```

Conociendo las marcas que se manejaran en *xml*, podremos pasar a la definición de la función encargada de hacer el *parser* de las marcas y con base en ellas, crear una forma de leer los datos a partir de los posibles archivos o secuencia de archivos. Para dicha función será necesario revisar la diferente documentación para la creación de los compiladores o precompiladores, ya que las funciones que se crearán son parte de un compilador de marcas o un precompilador de *xml*; para cambiar las marcas por parámetros que las funciones de lectura comprendan.

Función de parsec

La función que se tiene contemplada, maneja dos tipo de archivos; uno donde estén escritas todas las marcas que se utilizarán y con base en ellas, la función realizará el cambio de sintaxis para las funciones de lecturas. Se tomó esta decisión para no determinar el tamaño de las marcas o restringirlas a solo unas cuantas; si alguien desea ampliar, las marcas solo tendrá que modificar un archivo. Este archivo manejará la comunicación con la clase, ésta al ser instanciada irá a cargar todas las palabras reservadas como un compilador; pero en este caso, serán las marcas que reconocerá. La salida entregará un archivo de con-

figuración para las funciones, el cual se encargará de leer los datos de sus respectivos archivos.

La función de *parsec* será la parte central que deberá ser cambiada en caso de que se desee ampliar el lenguaje, aunque en ocasiones muy particulares, tal vez se tenga que sobrecargar a algunas funciones de lectura.

Función de lectura

Esta función o funciones, dependiendo de la manera como se implemente el manejo de la sintaxis de lectura, antes de ser llamada a leer, se deberá contar con un archivo de conocimiento o se tomará uno por omisión. Este archivo indicará qué tipo de datos va a leer y qué tipo de reservación tiene que hacer de la memoria; ya que con base en eso, realizará la reservación de memoria para los datos. Al contar con el archivo de conocimiento de la lectura, se mandarán a llamar a las funciones de carga de los datos.

Función de carga

La carga de datos se hará con base en los parámetros que tengan en la lectura, ya que le mandará el lugar donde deberán ser colocados y la manera de acomodarlos, ya que podrá recibir un conjunto de datos en un arreglo unidimensional y los deberán acomodar en el un arreglo bidimensional, o en su caso con posibles combinaciones, las cuales deberán ser capas de realizar sin problema.

Las funciones que se encarguen de esta parte, deberán estar diseñadas para que puedan ser sobrecargadas en caso de que en un futuro se encuentre una manera más adecuada de cargarlos. Aunque en realidad, esta parte no tiene tanta importancia, ya que donde se lleva más tiempo de proceso son las dos partes anteriores, por esa razón se debe buscar la mayor modularidad de las funciones entre ellas o en ellas, para que llegado el momento se puedan cambiar sin problema las funciones.

Epílogo

La ingeniería en computación es una disciplina muy amplia y su campo laboral basto; sin embargo, la principal pasión que despierta en sus estudiosos es también su principal fuente de desarrollo para sus practicantes.- la creación de herramientas que faciliten el trabajo de los demás a través de la creación de software.

El desarrollo de software científico es un reto y por lo tanto, una tarea muy entrenadora. Así que podemos considerar a esta área con un doble interés; primero constituye una actividad enriquecedora, ya que el software desarrollado es rápidamente evaluado por los demandantes científicos y una retroalimentación provechosa. En segundo lugar debemos considerar que la creación de este tipo de herramientas es muy satisfactorio, pues rápidamente se retribuye el esfuerzo al usarse en áreas que tienen pocos recursos para adquirir software, siendo tan útiles para la sociedad.

El kernel de visualización alcanzó un nivel de desarrollo que permite la implementación de aplicaciones profesionales en visualización científica; sin embargo, falta añadir bibliotecas y desarrollar módulos para programar prototipos rápidamente y probarlos para concretar aplicaciones distribuibles entre la comunidad de usuarios. En este sentido, vemos mucho potencial para el kernel y en general para proyectos de software científico.

Uno de los aspectos importantes en el desarrollo de sistemas es contar con la retro−alimentación por parte de los usuarios y poder monitorear el uso de la aplicación. Por ello el software científico representa una alternativa importante en la formación del ingeniero en computación. Por otro lado, es importante retribuirle a la UNAM, desarrollando herramientas para este sector de su comunidad.

EERC
San Juan de Aragón, Edo. de México.
abril, 2006.

Literatura Citada

Tenginakai S., Lee J. y Machiraja R. 2001.

Salient iso-surface detection with model-independent statistical signatures.

Proceedings of Visualization. pp. 231−238

Drebin R.A., Carpenter L. y Hanrahan P. 1988

Volume rendering.

Computer Graphics, Proceedings of SIGGRAPH. pp. 65−74.

Lorensen W. y Cline H. 1987

Marching cube: a high resolution 3D surface construction algorithm.

Computer Graphics, Proceedings of SIGGRAPH. pp. 163−169.

Chang Y.K., Rockwood A.P. y He Q. 1995

Direct rendering of freeform volumes.

Computer−aided Design. pp. 553−559.

Mason W., Neider J., Davis T. y Shreiner D. 2000

OpenGL Programming Guide.

Editorial ADDISON−Wesley. 3a edicón.

Richter J. 1997

Programación Avanzada en Windows.

Editorial McGraw−Hill. 3a edición.

Petzold C. 1995

Programación en Windows 95.

Editorial McGraw−Hill. 3a edición.

Schroeder W., Martin K. y Lorensen W. 1997

The Visualization Toolkit.

Editorial Prentice Hall. 3a edición.

Ceballos Sierra J. 1997
Encliclopedía del Lenguaje C.
Editorial Alfaomega.

URL

http://www.vtk.org

http://public.litware.com/VTK

http://www.OpenAl.org

http://www.opengl.org

http://www.OpenH323.org

http://www.OpenSceneGraph.org

Apéndices

Índice de Apéndices.

Instalación de *OpenGL* en *Windows*

Objetivo:

Efectuar de manera correcta la instalación de la biblioteca *OpenGL*, desde el correcto acomodo de los archivos fuentes de la misma, en una máquina con sistema operativo *Windows*.

Requisitos:

1. Contar previamente con el sistema operativo *Windows* instalado.
2. Contar previamente con *Visual* `C++` versión 6, *Visual Studio .NET* o un compilador de `C++` instalado.
3. Contar previamente con *Winzip* instalado.

Actividades:

1. Descargue el archivo `OpenGL.zip` que se encuentra en la siguiente dirección `http://www.opengl.org`.
2. Cree una carpeta temporal en sus documentos, con el nombre de temporal, en la cual guarde el archivo con el nombre `Opengl.zip`.
3. Con ayuda de winzip descomprima el archivo en la carpeta temporal que ha creado.

   Al descomprimirse el archivo verá que tiene tres tipos de archivos que son:

   - Los archivos de encabezados `*.h`
   - Las bibliotecas compiladas `*.lib`
   - Los archivos de vínculos dinámicos `*.dll`

4. Cree un directorio con el nombre de `GL` pero la ruta en la que se encontrará dependerá de que compilador tiene en su máquina y la ruta de este.

   Ya que todos los compiladores tienen un directorio include y el directorio `GL` debe estar dentro del directorio include, por eso antes de crearlo deberá ver donde se encuentra el directorio include de su compilador y posteriormente crearlo.

5. Al contar con el directorio `GL` en su lugar coloque en él todos los archivos de cabeceras `*.h`.
6. En el directorio donde se encontró el directorio include también se encuentra el directorio con el nombre `lib`, en ese directorio debe colocar todos los archivos de bibliotecas ya compiladas `*.lib`.
7. Coloque por último los archivos de vinculación `*.dll` en el directorio `C:/windows/system32/`.
8. Con esto terminó la instalación de *OpenGL* en el sistema.

Instalación de *vtk* en *Windows*

Objetivo:

Efectuar de manera correcta la instalación de la biblioteca *vtk*, desde la compilación de los archivos fuentes de la misma, en una máquina con sistema operativo *Windows*.

Requisitos:

1. Contar previamente con el sistema operativo *Windows* instalado.
2. Contar previamente con *Visual* `C++` versión 6, *Visual Studio. NET* o un copilador de `C++` instalado.
3. Contar previamente con *Winzip* instalado.

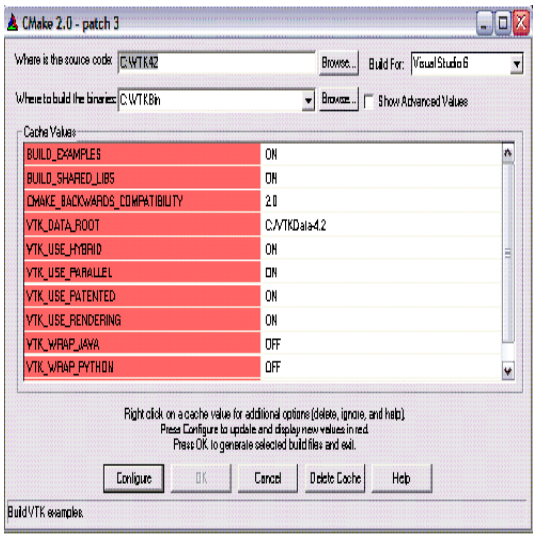
Actividades:

1. Descargue el archivo `VTK-4.2-latesrRelease.zip` el cual se encuentra en `http://public.litware.com/VTK/get-software`; el archivo se guardara en el disco duro.
2. Con la ayuda de *Winzip* se descomprimirá el archivo `VTK-4.2.latesrRelease.zip` en la ruta `C:\`.

   La ruta donde se descomprimirá el archivo `VTK-4.2.latesrRelease.zip`, será la ruta que usted usará para configurar cada vez que de use la biblioteca de *VTK*, si es muy larga o confusa no le será posible recordarla ni configurar correctamente *VTK*. Se sugiere que utilice la ruta más corta como el directorio raíz donde trabaje como `C:`.
3. En la unidad `C:\` deberá buscar una carpeta llamada *VTK*, la cual deberá ser renombrada como `VTK42`.
4. Descargar el archivo `VTKData-4.2.zip` que se encuentra en `http://public.litware.com/VTK/get-software`. El archivo se guardará en el disco duro.
5. Con la ayuda de [*Winzip descomprime el archivo* `VTKData-4.2.zip` *en la ruta* `C:\`.
6. *En la unidad* `C:\` *deberá buscar una carpeta llamada* `VTKData-Release-4` *la cual deberá ser renombrada como* `VTKData-4.2`.
7. *En la unidad* `C:\` *se crea una carpeta llamada* `VTKBin`.
8. *Descargue el programa* `CMSetup202.exe` *de la página* `http://www.cmake.org/HTML/Download.html`*; el archivo se guardar en el disco duro.*
9. *Localice el programa* `CMSetup202.exe` *y ejecutarlo.*

   Este programa se encarga de instalar todos los componentes de CMake para Windows, los cuales son necesarios para la compilación de los archivos fuentes de VTK.

10. *Al término de la instalación de CMake, el programa colocará un icono de acceso directo en escritorio, por el cual debe entrar al programa, al hacerlo le debe mostrar la siguiente pantalla.*



11. *En esta pantalla, la primera caja de diálogo es para que coloque la ruta del directorio donde están los códigos fuentes (en este caso es* `C:\VTK42`*).*

    En ocasiones los valores de cache no aparecen al inicio de la sesión de CMake, pero si es la primera vez que lo ejecuta no se preocupe y continué con la instalación, después del paso 14 aparecerán los valores.

12. *La segunda caja de diálogo es para la ruta del directorio donde serán colocados los archivos binarios (en este caso* `C:\VTKBin`*).*

13. *La última caja de dialogo, la que se encuentra en la extremo derecho mostrará las diferentes opciones de compiladores que tenga disponibles en su máquina.*

14. *Al terminar de llenar las cajas de diálogo oprima el botón de configure.*

    En este momento aparecerán los valores de cache.

15. *En los valores de cache busque la bandera* `VTK_DATAROOT`*, al localizarla cambie la ruta que tiene por omisión y coloque la ruta en donde se encuentran los archivos de la documentación; en este caso* `C:\VTKData-42`*.*

16. *Cambie las siguientes banderas:*

    - `BUILD_SHARED_LIBS` *ON*
    - `VTK_USE_HYBRID ON`
    - `VTK_USE_PARALLEL ON`
    - `VTK_USE_PATENTED ON`

      Si se quieren instalar los ejemplos se procede a cambiar la bandera `BUILD_EXAMPLES`*.*

- BUILD_EXAMPLES *ON*

17. *Apretar el botón de configuración nuevamente.*

    En ese momento debe activar el botón de OK.

18. *Oprima el botón de OK.*

19. *Vaya al directorio* C:\VTKBin42 *en el cual debe buscar el archivo* VTK.swd.

    Este proyecto es el que se necesita para compilar las bibliotecas de VTK; requiere de Visual C++, *con el cual deberá abrirlo.*

20. *Cuando tenga abierto el proyecto, del lado derecho aparecerán los archivos del proyecto, en estos busque el archivo* ALL_BUILD *el cual seleccionará con el ratón y teclear con el botón derecho, se desplegará un menú, en el cual elija la opción de build.*

    En ese momento la compilación dará inicio y tendrá que esperar a que está termine. Cuando termine el proceso de compilación, tendrá los archivos *.dll *de VTK en el directorio* C:\VTKBin42\bin\debug.

21. *Al terminar el proceso de compilación vaya a la carpeta de* C:\VTKBin42\bin\debug, *en la cual debe copiar todos los archivos* VTK*:dll, *hecho ésto debe moverse a la carpeta* C:\windows\system32, *en la cual deberá copiar los archivos* VTK*.dll.

22. *El siguiente paso será la configuración de Visual* C++ *para la correcta compilación de futuros proyectos. En la barra de herramientas vaya a la opción de herramientas () en la cual debe elegir la opción de opciones (option).*

    En este momento debe aparecer una ventana con las opciones de las características de los proyecto de Visual C++.

23. *Seleccione la opción* VC++, *la cual se encuentra en la lista en la parte derecha de la ventana, le debe de mostrar la lista de los archivos de las librerías a la cual en la parte izquierda de la pantalla, Agregue* C:\VTKBin42\bin\debug.

24. *Escoger la opción de ver los archivos que se incluyen (include) y se colocaran los siguientes:*

    - C:\Vtk\Common\
    - C:\Vtk\Filtering\
    - C:\Vtk\Graphics\
    - C:\Vtk\Hybrid\
    - C:\Vtk\Imaging\
    - C:\Vtk\IO\
    - C:\Vtk\Parallel\
    - C:\Vtk\Patented\

- `C:\Vtk\Rendering\`
- `C:\Vtkbin\`

25. *En seguida presione el botón de OK.*

26. *En la barra de herramientas, escoger la opción de proyecto, aparecerá un menú del cual debe seleccionar la opción de propiedades del proyecto; debe escoger la opción de* `C/C++`.

27. *Deberá agregar en los directorios de incluye los siguientes directorios*

    - `C:\Vtkbin\`
    - `C:\Vtk\Common\`
    - `C:\Vtk\Filtering\`
    - `C:\Vtk\Graphics\`
    - `C:\Vtk\Hybrid\`
    - `C:\Vtk\Imaging\`
    - `C:\Vtk\IO\`
    - `C:\Vtk\Parallel\`
    - `C:\Vtk\Patented\`
    - `C:\Vtk\Rendering\`

28. *En la opción de vinculación (link) en general se debe agregar el directorio de las bibliotecas que es* `C:\VTKBin\bin\debug`.

29. *En vinculación en la opción de entrada, se deben agregar siguientes dependencias, en la opción de dependencias adicionales.*

    - `Common.lib`
    - `VtkFiltering.lib`
    - `VtkGraphics\.lib`
    - `VtkHybrid.lib`
    - `VtkImaging.lib`
    - `VtkIO.lib`
    - `VtkParallel.lib`
    - `VtkPatented.lib`
    - `VtkRendering.lib`

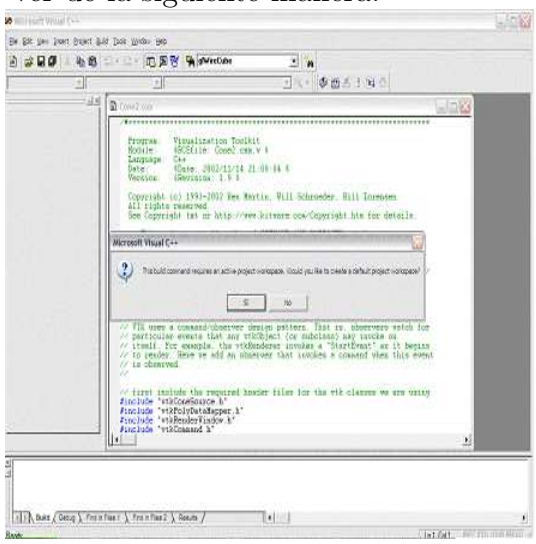
30. *En este momento la configuración ha terminado, solo resta que compile y recuerde que para todas sus aplicaciones de vtk es necesario que revise estas opciones para verificar que no se han modificado y pueda trabajar sin problema.*

Prueba de instalación:

1. Vaya al directorio donde está `VTK42`, en el directorio hay un subdirectorio llamado *Examples*, entre en éste y busque el directorio Tutorial ahora vaya al subdirectorio step1, vaya al subdirectorio `Cxx`. De manera gráfica sería.

   VTK42->Examples->Tutorial->Step1->Cxx

2. En este lugar debe estar un archivo con el nombre de cono `.cxx`. Deberá oprimir dos veces este archivo el cual abrirá el *Visual* `C++`, estando en el *Visual* `C++` de la opción de compilar, se debe ver de la siguiente manera.



3. Aparecerá una ventana con el letrero indicando si deseamos que *Visual* `C++` cree los archivos para el proyecto, oprima el botón de Si.

4. La compilación del ejemplo comenzará en ese momento, al terminar ejecutará el ejemplo y podrá ver un cono girando sobre un eje. Lo cual significará que *vtk* a quedado instalado.

En caso de errores vaya al tip número 1.

Tips o correcciones:

1. En la compilación puede haber tres tipos de errores los de *linker* y de bibliotecas, ambos son debido a que el compilador no esta encontrando las ligas o las bibliotecas .

   a) En caso de error por *linker*, estos tipos de errores se ven de la siguiente manera

```
Linking...
Cone2.obj : error LNK2001: unresolved external symbol "__declspec
(dllimport)public: void __thiscall vtkCamera::Azimuth(double)" (__
imp_?Azimuth@vtkCamera@@QAEXN@Z)
```

Son debido a que no están colocadas las ligas a los archivos *.lib, por tal motivo revise el paso 29, de estar bien las `*.lib`, entonces vaya a la barra de herramientas y escoja proyecto, vaya a las opciones de proyecto y revise el de linker; también en ese lugar deben estar declarados los archivos `*.lib`, de no estarlo, declarelos y compile de nuevo.

b) En caso de error por bibliotecas, lo que se debe hacer es revisar el paso 27, de estar correctamente colocado entonces vaya a las propiedades del proyecto y revíselo ya que también ahí debe estar declaradas en los archivos de incluyes o incluidos y compile nuevamente.

Instalación de *Qt* en *Windows*

Objetivo:

Efectuar de manera correcta la instalación de la biblioteca *Qt* desde la compilación de los archivos fuentes de la misma en una máquina con sistema operativo *Windows*.

Requisitos:

1. Contar previamente con sistema operativo Windows instalado.
2. Contar previamente con *Visual* `C++` versión 6, *Visual Studio .NET* o un copilador de `C++` instalado.
3. Contar previamente con *Winzip* instalado.
4. Consiga el *Qt* para *Windows*.

Actividades:

1. Con ayuda de *Winzip* se descomprimirá el archivo `Qt.zip` en su disco duro en la ruta `C:\` o en su unidad raíz.
2. Al descomprimirse el archivo le creará una carpeta `QT`, vaya a ella, en ella localice la carpeta comerciales en esa carpeta encontrara el archivo `QT-Win-Commercial-3.2.1`. Ejecute el archivo ya que es el archivo de instalación.
3. La primera pantalla será la de bienvenida a la cual le deberá apretar el botón de *next*.
4. Aparecerá la siguiente pantalla .

   Presione el botón de *Read from file* para seleccionar el archivo de licencia. Después deberá presionar el botón de *Next* .

5. Solicitará la confirmación de la aceptación de la licencia, la deberá aceptar y oprimir el botón de *next*.
6. Aparecerá enseguida una ventana como la siguiente.

En esta venta pedirá el directorio donde se instalará *Qt* y el compilador con el que se compilará.

Se recomienda que deje el mismo directorio que tiene por omisión; sino le agrada, cambie por el que desee.

7. En la siguiente pantalla le pedirá los siguientes:

   - El nombre del directorio inicial para todos los proyectos que se realizarán.
   - El directorio donde se encuentra el compilador .
   - Elija el grupo de trabajo que va a poder usar la biblioteca de *Qt*.
     Se recomienda que todos los usuarios la puedan utilizar,
   - Seleccione la opción de que cree la variable `QTDIR` como se muestra en la siguiente imagen se deberá ver en la pantalla.

Al terminar de dar los datos, presione el botón de *next*. La pantalla se debe ver de la siguiente manera.

En caso de posibles errores ver el tips 1.

8. La siguiente pantalla que aparece es la de configuración.

Tiene opciones identificadas por un cuadro amarillo enfrente de ellas, es en esas opciones las que tiene que verificar que estén de la siguiente manera:

- *Build*

  Debug
- *Library*

  Shared
- *Threading*

  Threaded

La opción de los módulos es con los que usted desee que cuente *Qt*. Al terminar de seleccionar los módulos presione el botón de *Next*, en ese momento comenzará la instalación de *Qt*.

9. Mostrará la siguiente pantalla.

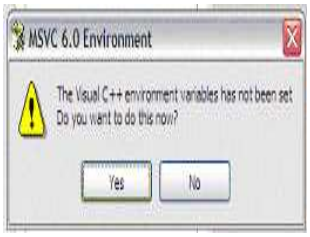
En la cual podrá apreciar el avance de la instalación. Cuando termine la instalación, oprima el botón de *Next*; si el programa no recibe respuesta en 30 segundos dará por

terminado el proceso y comenzará la construcción de *Qt*.

El proceso es un poco tardado dependiendo de la máquina en la que se esté instalando, por lo cual le recomendamos que tenga paciencia en una máquina *celeron* a 2.5 GHz se tardó alrededor de 25 min.

Tips o Correcciones:

1. Si al presionar **siguiente** en la ventana de configuración número 3 le aparece esta ventana



Se debe a que las variables del compilador no se encuentran, esta ventana pregunta si deseamos cerciorarnos de ello, conteste que si. Para que el programa arregle cualquier posible problema.

Instalación de *OpenGL* en *Linux*

Objetivo:

Efectuar de manera correcta la instalación de la biblioteca *OpenGL* desde el correcto acomodo de los archivos fuentes de la misma en una máquina con sistema operativo *Linux*.

Requisitos:

1. Contar previamente con el sistema operativo *Linux* instalado.
2. Contar previamente con un copilador de `C++` instalado.

Actividades:

1. Descargar el archivo `OpenGL.gzip` que se encuentra en la siguiente dirección `http://www.opengl.org`.
2. Cree una carpeta temporal en su directorio `/Home/` con el nombre de temporal, en la cual guardará el archivo con el nombre `Opengl.gzip`.
3. Con ayuda de `gzip` descomprima el archivo en la carpeta temporal que ha creado.

   Al descomprimir el archivo verá que tiene tres tipos de archivos que son:

   - Las cabeceras los archivos `*.h`
   - Las bibliotecas compiladas `*.lib`
   - Los archivos de vínculos dinámicos `*.so`
4. El siguiente paso es la creación de un directorio con el nombre de `GL` en la siguiente ruta `/usr/include`.
5. Al contar con el directorio `GL`, coloque en su lugar, todos los archivos de cabeceras `*.h`.
6. Los archivos de biblioteca `*.lib` se colocan en la siguiente ruta `/usr/lib` o `/usr/X11R6/lib`.
7. Coloque por último los archivos de vinculación `*.so` en la ruta `/usr/lib` o `/usr/X11R6/lib`.
8. Con esto terminó la instalación de *OpenGL* en el sistema.

Instalación de *vtk* en *Linux*

Objetivo:

Efectuar de manera correcta la instalación de la biblioteca *vtk* en todas las cuentas, desde la compilación de los archivos fuentes de la misma en una máquina con sistema operativo *Linux*.

Requisitos:

1. Contar previamente con el sistema operativo *Linux* instalado.
2. Contar previamente con un copilador de `C++` instalado.

Actividades:

1. Comenzar sesión como *root*.
2. Abrir una terminal y cambiese al directorio `/usr/local/`, estando en el directorio teclee.

```
cvs -d :pserver:anonymous@www.cmake.org:/cvsroot/CMake login
CVS password: cmake
cvs -d :pserver:anonymous@www.cmake.org:/cvsroot/CMake co CMake
```

3. En la pantalla aparecen los archivos conforme se van descargando, el tiempo de descarga puede variar dependiendo de la conexión de red que tenga. Al terminar de descargar teclee lo siguiente.

```
cvs -d:pserver:anonymous@public.kitware.com:/cvsroot/VTK login
CVS password: vtk
cvs -d:pserver:anonymous@public.kitware.com:/cvsroot/VTK checkout VTK
```

4. Al terminar la descarga, Teclee lo siguiente:

```
cvs -d:pserver:anonymous@public.kitware.com:/cvsroot/VTKData login
CVS password: vtk
cvs -d:pserver:anonymous@public.kitware.com:/cvsroot/VTKData co VTKData
```

5. En este momento tendrá dos nuevos directorios en el directorio `/usr/local/` los cuales son *vtk* y *CMake*. Lo siguiente es la configuración del `script` del `shell` para agregar los link necesarios para *vtk* y *CMake*. Esto es necesario para que se ejecuten de manera automática. Este proceso se realiza de la siguiente manera

6. En caso de utilizar el *Born shell* vaya al paso 10 en caso de utilizar el *C shell* continue con el paso 7.

7. Vaya al directorio `/etc/`.

8. Abra el archivo profile, busque, donde exportar el `PATH` y una línea antes coloque las siguientes lineas:

```
setenv PATH =$PATH:/Ruta_de_CMake/CMake/cmake/bin
setenv PATH =$PATH:/Ruta_de_VTK /VTK/bin
setenv LD_LIBRARY_PATH=$LD_LIBRARY_PATH /Ruta_de_VTK/VTK/bin
```

   Tenga cuidado con la sintaxis ya que en algunas máquinas varía de forma. Esto lo podrá saber observando la sintaxis que se está manejando en el archivo profile.

9. Guarde los cambios y salga del archivo.

10. Vaya al directorio `/etc/`.

11. Abra el archivo `profile`, busque dónde se exporta el `PATH` y una línea antes coloque las siguientes líneas:

```
export PATH=$PATH:/Ruta_de_CMake/CMake/cmake/bin
export PATH=$PATH :Ruta_de_VTK/VTK/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/Ruta_de_VTK/VTK/bin
```

   Tenga cuidado con la sintaxis ya que en algunas máquinas varía de forma. Esto lo podrá saber observando la sintaxis que se está manejando en el archivo profile.

12. Guarde los cambios y salga del archivo.

    Es de suma importancia escribir correctamente las rutas en donde se encuentran los directorios tanto de *vtk* como de *CMake*, ya que es el error más común. Cuando termine de editar el archivo, ejecute su *script* para configurar el *shell*.

13. Teclee `source`profile .

14. Abra una nueva terminal, vaya a la capeta de *CMake* que es `/usr/local/CMake`.

15. Teclee lo siguiente.

    - `./configure`
    - `make`

- `make`install

En este momento se está creando el makefile que usará para configurar el *CMake*.

16. Al terminar la compilación de *CMake* ya estará compilado e instalado, listo para continuar con la unión con *vtk*.

17. Vaya al directorio de *vtk* que es `/usr/local/VTK`.

    Para poder configurar el *CMake* con *vtk* de modo que exista interactividad entre ellos, se modificarán algunas banderas de *CMake*. En el siguiente paso aparecerán banderas, no todas serán modificadas, así que cuando la bandera no esté mencionada en la siguiente columna izquierda, solo le teclee enter, para no modifique, en caso de que sea la que buscamos escriba los datos y teclee enter.

```
cmake -i
Would you like to see advanced options? yes
```

Busque la siguiente secuencia de banderas que le aparecerán, recuerde que solo éstas debe cambiar y las que no estén en la lista, solo teclee enter para no modificarlas.

```
BUILD SHARED LIBS                  ON
CMAKE INSTALL PREFIX               /usr/local /VTK/
VTK USE HYBRID       ON
VTK USE PATENTED  ON
```

18. Teclee los siguiente:

```
make
make install
```

En este paso se compilará *vtk*.

19. Al terminar el proceso de compilación, la instalación ha terminado exitosamente; ya cuenta su sistema con las biblioteca *vtk* y *Cmake* para su uso.

Instalación de *Qt* en *Linux*.

Objetivo:

Efectuar de manera correcta la instalación de la biblioteca *Qt* desde la compilación de los archivos fuentes de la misma en una máquina con sistema operativo *Linux*.

Requisitos:

1. Contar previamente con el sistema operativo *Linux* instalado.
2. Contar previamente con un copilador de `C++` instalado.

Actividades:

1. Descargue el archivo `qt-X11-free-3.3.2.tar.gz` el cual se encuentra en `http://www.trolltech.com/download/qt/x11.html`, el archivo se guardará en el disco duro.

   Los últimos números del archivos le marcarán la versión de *Qt*.

2. Inicie la sesión como *root*.
3. Copie el archivo `qt-X11-free-3.3.2.tar.gz` al directorio `/usr/local/`.
4. Vaya al directorio `/usr/local/` en una terminal de modo texto, busque el archivo `qt-X11-free-3.3.2.tar.gz` y descomprímalo de la siguiente manera.

   `tar`zxf qt-X11-free-3.3.2.tar.gz

5. Se creará el directorio `qt-X11-free-3.3.2`, renómbrelo como `Qt`.

   El cambio de nombre es porque en la configuración de la Biblioteca se hace referencia a este directorio y si es demasiado largo aumentará el factor de riesgo de cometer errores.

6. Vaya al directorio `/etc/`, en el cual deberá abrir el archivo profile de la siguiente manera

   `vi` profile

7. Cuando lo tenga abierto, busque las líneas donde se exportan el `PATH` y una línea antes incluya las siguientes:

```
QTDIR=/usr/local/qt
PATH=$PATH:$QTDIR/bin
 ANPATH=$MANPATH:$QTDIR/doc/man
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$QTDIR/lib
export  QTDIR PATH MANPATH LD_LIBRARY_PATH
```

8. Guarde los cambios en el archivo y salga del archivo.

9. Teclee `source`profile

   Esta línea es para que los cambios que hemos realizado los tenga en cuenta el sistema y los respete; de no hacerlo, el sistema no sabrá que hemos cambiado la configuración de estos archivos y no realizará los respectivos cambios.

10. Vaya al directorio `/usr/local/qt`, en el cual deberá teclear.

    `./configure`-thread

    En la biblioteca se puede contar con diferentes opciones de soporte para diferentes herramientas, es necesario darlas de alta en este momento. Estas herramientas son dadas de alta por medio de diferentes banderas, en este caso solo necesitará dar de alta a *OpenGL*, por eso su bandera es `-thread`.

11. En ese momento pregunta que tipo de licencia, tendrá marcada por omisión la opción de *free*, la cual aceptará, tecleando *yes*.

    En ese momento comenzara la configuración y compilación de sus banderas tanto de sistema como en los archivos necesarios para continuar con la instalación, la cual puede tardar en terminar pero tenga paciencia.

12. Al finalizar la configuración teclee `make`.

    Este paso es el final, al terminar la compilación del sistema habrá terminado la instalación de *Qt*, pero este último paso es el que requiere mayor tiempo ya que la compilación será lenta y llevará un tiempo considerable.

Documentación de Clases

Las clases que conforman el kernel de visualización en su estado actual son las siguientes:

1. *Carga*
2. *KvVTKImage*
3. *KvCamara*
4. *KvMaterial*
5. *KvFunciones*
6. *KvCamMon*
7. *KvMomento*
8. *KvMenu*

En la figura A.1 se muestran las bibliotecas de las cuales depende la clase carga, de color rojo.

Clase Carga

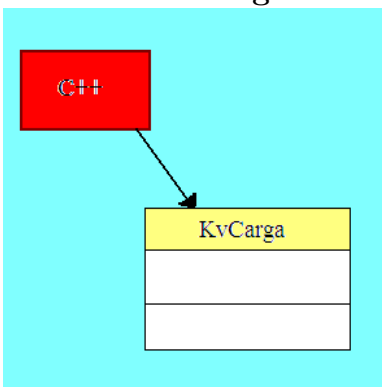


Figura A.1. Dependencias de la clase Carga.

	Carga	
iArre: unsigned short	archivo[120]: char	color:bool
IniRan: int	bProfu: bool	iAncho: int
iAlto: int	FinRan: int	Profun: int
iArre: unsigned short	contador: int	
+ Carga(void)	+ Carga()	+ Lee(void)
+ LeeVariosArchivos(void)	+ DameAlto(void): int	+AsignaDatos():int *
+ AsignaProfun(int)	+ ReservaMemoria(void)	+ DameArreglo(void): unsigned short
+ DameAncho(void): int	+ DameProfu(void): int	+ AsignaRango(int,int)
+ AsignaArchivo(char *)	+ AsignaAncho(int)	+ AsignaAlto(int)
		+Lectura(void)

Cuadro A.1. Funciones y elementos de la clase Carga.

¿Qué es Carga?

Carga es una clase dedicada a preparar los datos antes de colocarlos en los contenedores (KvVTKImage). Los datos no siempre son imágenes en formatos conocidos o se tienen los datos en otro tipo de estructura. *vtk* tiene contenedores que se utilizan para casi todos estos tipos de datos, solo es cuestión de revisar detenidamente la documentación, pero para evitar este tipo de problemas se creó esta clase.

Esta clase fue pensada para el tipo de formatos de imágenes que *vtk* no tiene forma de leerlos, como son los formatos de imágenes no comerciales (42 bits u 11 bits, por dar un par de ejemplos). Esta clase se encarga de leerlos y prepararlos para el contenedor de kvVTKImage, para sus procesos siguientes.

¿Cómo funciona?

La clase tiene dos funciones virtuales, una de ellas se llama Lectura; esta clase es la que debe implementar la persona que desee utilizar un objeto de este tipo. Carga sirve para poder leer cualquier tipo de datos; sin embargo, se dejó virtual para evitar un crecimiento exorbitante al programar todos los tipos de datos que se requiriesen; de esta forma, el desarrollo de cada aplicación incluirá los códigos para los tipos de datos que se usen en dicha aplicación.

La otra función virtual es AsignaDatos. Esta función es la que se encarga de asignar datos a diferentes contendores de *vtk*, ya se que no se quiere trabajar con el contenedor con el que trabaja KvVTKImage (*vtkImageData* es una de las clases de contenedores más generales y más usados por los métodos que maneja *vtk*). Al igual que con la función anterior, no es buena idea hacerla muy general; ya que en ocasiones solo utilizaríamos uno o dos contenedores y no tiene caso; por otro lado, casi todos los filtros reciben el tipo de datos que maneja KvVTKImage.

¿Cómo se utiliza?

Se declara el objeto y se asigna con sus funciones, el ancho, alto, rango (en caso de ser una secuencia de imágenes) y la ruta junto con nombre del archivo a leer. Al tener estos elementos declarados, se llama a la función de ReservaMemoria para reservar la memoria necesaria para guardar los datos antes de entregarlos al contenedor al cual serán asignados; al terminar se llama a la función de lectura (por ejemplo, a LeeVariosArchivo, Lee o Lectura), dependiendo de qué es lo que se requiera leer.

¿Dónde se utiliza?

Esta clase se utiliza como interfaz entre archivos de imágenes que no sean comerciales o conocidos y algún contenedor de *vtk* o un contenedor del kernel como KvVTKImage.

Nombre de las funciones	Especificación
Carga(void)	Constructor
Carga()	Destructor
Lee(void)	Lee los datos de un solo archivo
LeeVariosArchivos(void)	Lee los datos de una secuencia de archivos con el mismo nombre
int DameAlto(void)	Regresa el alto que tienen asignado las imágenes
Virtual AsignaDatos()	Función que sirve para entregar los datos a otros contenedores que no sean KvVTKImage
AsignaProfun(int)	Asignamos el valor de la profundidad que tendrá el arreglo
ReservaMemoria(void)	Reserva la memoria para crear el arreglo donde se guardaran los dato antes de entregarlos al contenedor
unsigned short DameArreglo()	Regresa los datos para entregarlos al contenedor de KvVTKImage
int DameAncho(void)	Regresa el ancho que tienen asignado las imágenes
int DameProfu(void)	Regresa la profundidad que tendrá el arreglo de datos

Cuadro A.2. Especificación de las funciones de la clase.

Nombre de las funciones	Especificación
AsignaRango(int,int)	Asigna los rangos de la secuencia de imágenes para obtener los datos
AsignaArchivo(char *)	Asigna el nombre del archivo con todo y la ruta para poder obtener los datos
AsignaAncho(int)	Asigna el ancho que tendrán las imágenes o imagen
AsignaAlto(int)	Asigna el alto que tendrán las imágenes o imagen
Lectura(void)	Esta función virtual es la que ayuda a leer cualquier tipo de datos, no importando la manera con que se deban leer

Cuadro A.2. Especificación de las funciones de la clase.

Ejemplos de la clase Carga

Enseguida se aprecia un ejemplo sencillo de cómo cargar y dar lectura a datos de 8 bits de un solo archivo, o de una secuencia de archivos. En el siguiente ejemplo, de una sola secuencia de archivos.

```
int  main(int argc,char *argv[]){
 Carga *car=new Carga();  // creamos un objetos de la clase.
  car->AsignaAncho(256);  // asignamos el ancho que tendran las imagenes
  car->AsignaAlto(256);   // asignamos el alto que tendran las imagenes
  car->AsignaProfun(64);  // asignamos la profundidad del arreglo de la
                          // secuencia de datos
  car->ReservaMemoria();  // se reserva la memoria para el arreglo para
                          // guardar los datos
  car->AsignaArchivo(argv[1]); // Tomamos el primero argumento como la ruta
                               // para leer los datos
  car->Lee();                  // Aqui  leemos el archivo donde obtenemos
                               // los datos requeridos
 }
```

Con esta simple secuencia de pasos ya estamos leyendo del archivo, el número de datos requeridos. Para leer una secuencia de imágenes se realiza de la siguiente manera.

```
int  main(int argc,char *argv[])
{
 Carga *car=new Carga();
       // creamos un objetos de la clase.
  car->AsignaAncho(256);
    //asignamos el ancho que tendran las imagenes
  car->AsignaAlto(256);
    //asignamos el alto que tendran las imagenes
  car-> AsignaRango(0,83);
  //asignamos la profundidad del arreglo de la secuencia
 // de datos
```

```
//en este caso es el numero de imagenes que tendremos.
 car->ReservaMemoria();
//se reserva la memoria para el arreglo para guardar los datos
 car->AsignaArchivo(argv[1]);
//Tomamos el primer argumento como la ruta para leer
// los datos
car-> LeeVariosArchivos(void);
//Aqui  leemos el archivo donde obtenemos los datos
//requeridos
}
```

De esta sencilla manera ya están acomodando los datos para ser entregados a un contenedor.

De igual manera se realiza la carga de datos de otros tipos; solo es necesario construir la función virtual de lectura y mandarla a llamar en lugar de las funciones de lectura que utilizamos en los ejemplos anteriores.

Clase KvVTKImage

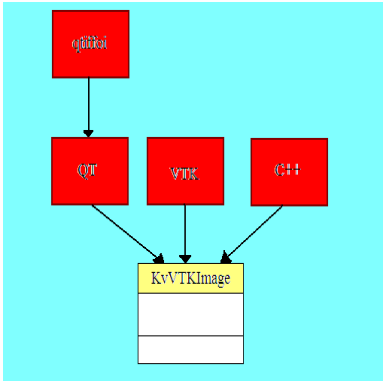


Figura A.2. Dependencias de la clase KvVTKImage.

	KvVTKImage
+ sombra:int	+ icolor:int
+ EspaX:float	+ EspaY:float
+ EspaZ:float	+ Specular:float
+ PoderEspe:float	+ iVol8:bool
+ bFunOpacid:bool	+ Ambiental:float
+ Specular:float	+ PoderEspe:float
+ iVol8:bool	+ bFunOpacid:bool
+ Difuso:float	− imagen:QImage
− Estructura: vtkStructuredPoints	− archivo[120]:char
− sFormato[10]:char	− bProfu:bool
− TipoDato:int	− Formato:int
− iAncho:int	− iAlto:int
− Profun:int	− IniRang:int
− FinRang:int	− Dimensiones[3]:int
− Contador:int	− NumPunOp:int
− NumPunCo:int	− NumPunRGB:int
− NumPunGr:int	−IsoValor:int
−iFunOpacid: int ∗	−iFunMapCol: int ∗
−iFunGraOpa: int ∗	−iFunMapRGB: int ∗
− Espaciado[3]:float	− fFunMapColR: float ∗
− fFunMapColG: float ∗	− fFunMapColB: float ∗
− fFunOpacid: float ∗	− fFunMapCol: float ∗
− fFunGraOpa: float ∗	+ KvVTKImage(void)
+ AsignaArchivo(char ∗)	+ AsignaRango(int, int)
+ AsignaTipo(int)	+ AsignaAlto(int)
+ AsignaAncho(int)	

Cuadro A.3. Funciones y elementos de la clase KvVTKImage.

+ AsignaProfu(int)	+AsignaFormato(int)
+AsignaEspaciado(float,float,float)	+AsignaDatos(unsigned short int *)
+AsignaDimensiones(int,int,int)	+ AsigRangIni(int)
+ AsigRangFin(int)	+CreaVolumen(void)
+ DameValGr(int):float	+ DameValOp(int):float
+DameValCo(int):float	+ DameValB(int):float
+ DameAncho(void):int	+ DameProfu(void):int
+ DameAlto(void):int	+DameFormato(void):char
+ DameArchivo(void):char	+ SacarAnchoAlto(void)
+ DameVolumen (void): vtkStructuredPoints	+ Destruye(void)
+ Otro(void)	+ AsignaEspa(float,float,float)
+ AsignaIso(int)	+ DameValIso(void):int
+ DamePuntoOp(int):int	+ DameNumOp(void):int
+ DamePuntoCo(int):int	+ DamePuntoRGB(int):int
+ DameValR(int):float	+ DameValG(int):float
− LLenaEstructuraGrises(void)	+ DamePuntoGr(int):int
+ DameNumCo(void):int	+ DameNumGr(void):int
+ DameNumRGB(void):int	+ AsignaPuntoOp(int, int*, float*)
+ AsignaPuntoCo(int, int*, float*)	+ AsignaPuntoGr(int, int*, float*)
+ AsignaPuntoRGB(int, int*, float*,float*,float*)	+ DatosVolu(char*,int,int,int)
+ DatVolAr(char*,int,int,int,int)	+ DameRangIni(void):int

Cuadro A.3. Funciones y elementos de la clase KvVTKImage.

¿Qué es KvVTKImage?

Esta clase es la más importante del kernel, ya que a través de ella podemos interactuar tanto con *vtk* como con las demás clases del kernel; además de que es la que contiene los datos de las imágenes o de los diferentes tipos de fuentes de datos. Esta clase es la que se encarga de resguardar los datos más importantes, como son los datos en sí, las funciones de transferencia para realizar el volume-rendering, el espaciado que se maneja en el despliegue, entre otras cosas. La clase no solo tiene para recibir los datos en el arreglo, sino tiene métodos internos para leer secuencia de imágenes como son en formatos jpg, tiff, pnm, png y bmp. También tiene funciones que permiten regresar los datos en caso de ser necesario. Esta clase devuelve un objeto de *vtk* para continuar con el pipeline de *vtk*, de ser requerido o simplemente para tener los datos en condiciones de ser usados.

¿Cómo funciona?

La clase tiene funciones para asignar datos como son el nombre, los rangos y el tipo de dato que se va a utilizar para distinguir entre las secuencias de imágenes y para indicar que va a recibir un arreglo con los datos; también tiene funciones que devuelven los datos requeridos. Este objeto tiene la ventaja de contar con una función directa para calcular una isosuperficie con solo una instrucción y la muestra, pero esta función no permite que otros componentes se hagan cargo de otros varios aspectos, como son las luces o características del material, etc.

¿Cómo se utiliza?

Se tienen que definir los elementos principales que son el ancho, alto y profundidad de los datos a leer y el formato, en caso de ser de los formatos ya mencionados; si no son con formato entonces no se utiliza esta función. Al tener los datos necesarios se utilizan las funciones de llenaEstructuraGrises, la cual se encarga de llenar el objeto de *vtk* con los datos proporcionados; de esta manera ya tenemos nuestros objetos de *vtk* listos para devolvernos el objeto o el despliegue de la función por default de isosuperficie.

¿Dónde se utiliza?

La clase se utiliza en cualquier parte que se quiera colocar datos dentro del contenedor de *vtk* de manera sencilla y tenerla disponible en cualquier lugar del programa. Ya sea para que tenga los datos seguros y de manera sencilla poder recuperarlos o para manejarlos de una manera que proporcione más fluidez que los mismo contenedores de *vtk*, que en ocasiones no se logra comprender donde están los datos que hemos introducido y que no los asignó en donde debería ser o de manera que no podemos recuperarlos o manipularlos de manera sencilla.

¿En quién se apoya?

Esta función se usa casi siempre acompañada de la clase de Carga para cargar diferentes tipos de datos. Pero no necesariamente tienen que estar juntas.

Nombre de las funciones	Especificación
int DameAncho(void)	Regresa el ancho que tienen asignado las imágenes
int DameProfu(void)	Regresa la profundidad que tendrá el arreglo de datos
AsignaRango(int,int)	Asigna los rangos de la secuencia de imágenes para obtener los datos
AsignaArchivo(char *)	Asigna el nombre del archivo con todo y la ruta para poder obtener los datos
AsignaAncho(int)	Asigna el ancho que tendrán las imágenes o imagen

Cuadro A.4. Especificación de las funciones de la clase KvImage.

Nombre de las funciones	Especificación
AsignaAlto(int)	Asigna el alto que tendrán las imágenes o imagen
int DameAlto(void)	Regresa el alto que tienen asignado las imágenes
AsignaProfun(int)	Asigna el valor de la profundidad que tendrá el arreglo
KvVTKImage(void)	Constructor
AsignaTipo(int)	Asigna el tipo de datos que va a recibir, ya que pueden ser 0=int,1=float 2=short,3=long 4=double 5=unsigned int, 6=unsigned short 7=unsigned long
AsignaFormato(int)	Asigna el formato con el que se va a trabajar
AsignaEspaciado(float,float,float)	Asigna el espaciado que habrá entre los puntos en la isosuperficie y se asigna a cada uno de los ejes X, Y, Z
AsignaDatos(unsigned short int *)	Aquí se recibe un arreglo unidimensional con los datos a guardar
AsignaDimensiones(int,int,int)	Se asignan las dimensiones que tendrá el objeto en los ejes X, Y, Z
AsigRangIni(int)	Asigna el rango del cual iniciará a buscar a los archivos al momento de recolectar los datos
AsigRangFin(int)	Asigna el rango del cual terminará de buscar los archivos al momento de recolectar los datos
CreaVolumen(void)	Crea el volumen con los datos que han sido dados
float DameValGr(int)	Devuelve el valor que tiene asignado en el número proporcionado de casilla de la función del gradiente
int DameProfu(void)	Devuelve la profundidad asignada a los datos
int DameAlto(void)	Devuelve el alto asignado a los datos
char DameFormato(void)	Devuelve el formato asignada a los datos
char DameArchivo(void)	Devuelve la ruta del archivo que está asignada a los datos
SacarAnchoAlto(void)	En el caso de las secuencias de las imágenes, se utiliza para obtener el ancho y el alto de las imágenes para realizar los cálculos necesarios para reservar memoria

Cuadro A.4. Especificación de las funciones de la clase KvImage.

Nombre de las funciones	Especificación
vtkStructuredPoints DameVolumen()	Devuelve el objeto de *vkt* cuando es requerido
Destruye(void)	Destruye la estructura cuando sea requerido
AsignaEspa(float,float,float)	Asigna el espaciado que habrá entre los puntos en la isosuperficie y se asignan a cada uno de los ejes X,Y,Z
AsignaIso(int)	Asigna el isovalor
int DameValIso(void)	Devuelve el isovalor
int DamePuntoOp(int)	Devuelve el valor que tienen asignado en el número proporcionado de casilla, las coordenadas de la función de opacidad
int DameNumOp(void)	Devuelve el número de elementos que integran a la función de opacidad
int DamePuntoCo(int)	Devuelve el valor que tienen asignado en el número proporcionado de casilla, las coordenadas de la función de transferencia de color en tonos de grises
int DamePuntoRGB(int)	Devuelve el valor que tienen asignado en el número proporcionado de casilla, las coordenadas de la función de transferencia de color en imágenes de color
LLenaEstructuraGrises(void)	Llena la estructura con los valores proporcionados, pero solo en imágenes de tonos de grises
int DamePuntoGr(int)	Devuelve el valor que tiene asignado en el número proporcionado de casilla, las coordenadas de la función del gradiente
int DameNumCo(void)	Devuelve el número de elementos que integran a la función de transferencia de color en imágenes de grises
int DameNumGr(void)	Devuelve el número de elementos que integran a la función de gradiente
int DameNumRGB(void)	Devuelve el número de elementos que integran a la función de transferencia de color en imágenes de color

Cuadro A.4. Especificación de las funciones de la clase KvImage.

Nombre de las funciones	Especificación
AsignaPuntoOp (int, int*, float*)	Se asigna el número de elementos, las coordenadas y los valores a la función de opacidad, a través de los arreglos proporcionados
AsignaPuntoCo (int, int*, float*)	Se asigna el número de elementos, las coordenadas y los a la función de la transferencia del color, a través de los arreglos proporcionados en imágenes en grises
AsignaPuntoGr (int, int*, float*)	Se asigna el número de elementos, las coordenadas y los valores a la función del gradiente, a través de los arreglos proporcionados
AsignaPuntoRGB (int,int,float,float,float)	Se asigna el número de elementos, las coordenadas y los valores a la función de la transferencia del color, a través de los arreglos proporcionados en imágenes de color
DatosVolu (char*,int,int,int)	Asigna los datos de ruta,formato, rango de inicio y rango del final a un volumen dado, cuando es en imágenes de formatos
DatVolAr (char*,int,int,int,int)	Asigna los datos de ruta, formato, rango de inicio y rango del final a un volumen dado, cuando son imágenes de 8 o 16 bits
int DameRangIni(void)	Devuelve el rango del inicio asignado a las imágenes
int DameRangFin(void)	Devuelve el rango del final asignado a las imágenes
float DameValGr(int)	Devuelve el valor que tiene asignado en el número proporcionado de casilla de la función del gradiente
float DameValOp(int)	Devuelve el valor que tiene asignado en el número proporcionado de casilla de la función de opacidad
float DameValCo(int)	Devuelve el valor que tiene asignado en el número proporcionado de casilla de la función de transferencia de color en caso de ser en tonos de grises

Cuadro A.4. Especificación de las funciones de la clase KvImage.

Ejemplos del uso de la clase

El siguiente ejemplo devuelve una isosuperficie de un conjunto de imágenes a través de la función con la que cuenta la clase. En el ejemplo se lee una secuencia de archivos en formato tiff, con la cual se conforma el volumen de datos donde se encuentra una isosuperfie con el isovalor de 120.

```
int  main(int argc,char *argv[])
{
  KvVTKImage *image= new KvVTKImage();
 // Se instancia la clase
  image->AsignaArchivo(argv[1]);
//se asigna la ruta de la secuencia a traves
       //del segundo argumento del main
  image->AsignaAlto(356);
// se asigna el alto
  image->AsignaAncho(356);
 //se asigna el ancho
  image-> AsignaRango (0,256);
// se asigna la profundidad a
 //traves de los limites
  image->AsignaFormato(5);
  // se asigna el formato
// de las imagenes, en este caso tiff
  image->AsignaEspaciado(1.5,1.5,1.5);
         // se asigna el espaciado para el despliegue
  image->HasIso(120);
         //se asigna el isovalor, asi
        // como se manda a pintar la isosuperficie
}
```

En el ejemplo anterior se utiliza una secuencia de archivos, pero el siguiente ejemplo se utiliza en conjunto con la clase carga para cargar imágenes de 8 bits.

```
int  main(int argc,char *argv[])
{
  Carga *car=new Carga();
  car->AsignaAncho(256);
  car->AsignaAlto(256);
  car->AsignaProfun(64);
  car->ReservaMemoria();
  car->AsignaArchivo(argv[1]);
  car->Lee();

  KvVTKImage *image= new KvVTKImage();
image->AsignaArchivo(argv[1]);
    //se asigna la ruta de la secuencia a traves
   //del segundo argumento del main
image->AsignaDimensiones(car->DameAncho()
               ,car->DameAlto(),car->DameProfu());
    // se asignan las dimensiones directamente
  image->AsignaEspaciado(1.5,1.5,1.5);
     // se asigna el espaciado para el despliegue
image->AsignaDatos(car->DameArreglo());
    // se asignan los datos que le proporcionara
   // el objeto Carga en un arreglo unidimensional
  delete(car);
   //se borra el objeto carga ya que no se utilizara mas
image->HasIso(120);
   //se asigna el isovalor asi  como se manda
   //a pintar la isosuperficie
}
```

Clase KvCamara

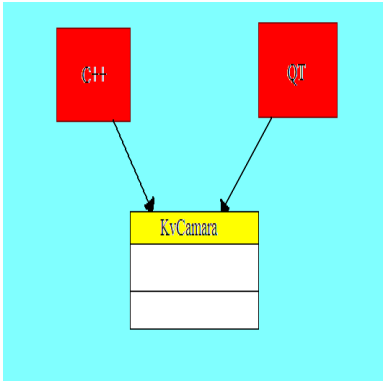


Figura A.3. Dependencias de la clase KvCamara.

	KvCamara
-Obser[4]	- Foco[4]
- Posi[4]	- BanOb
- BanPo	- BanFo
- tem[9]	
	+ DameObs(int)— : int
+ DamePos(int)— : int	+ DameFoc(int): int
+ DameBanOb(void): int	+ DameBanFo(void): int
+ DameBanPo(void): int	+ DameOb2(int)
+DameOb3(int)	+ DamePo1(int)
+ DamePo2(int)	+ DamePo3(int)
+ DameFo1(int)	+ DameFo2(int)
+ can(void)	+ DameFo3(int)
+ acepta(void)	+ DameOb1(int)
+ KvCamara(QWidget *parent=0, const char name=0)	+ DameObservador (void):int

Cuadro A.5. Funciones y elementos de la clase KvCamara.

¿Qué es KvCamara?

Es una clase que proporciona una interfaz gráfica para solicitar los datos acerca de las coordenadas del observador, la posición focal y de la cámara. Básicamente esta clase contiene una interfaz predeterminada para el control de estos datos, así como las funciones que tiene para devolver a su padre los valores al momento de sean tecleados.

¿Cómo funciona?

Funciona a través de *Qt*, ya que tiene los valores predefinidos de las coordenadas que se solicita; asícomo tres banderas, una para cada juego de coordenadas, las cuales le avisan al padre si este objeto ha modificado o no, los valores de las coordenadas. Al recibir el cambio de las banderas, el padre tendrá que solicitar los valores y proporcionarlos a las funciones correspondientes, si se requiere.

¿Cómo se utiliza?

Se utiliza como hijo de otra clase, o como un elemento más de *Qt*. Se manda a llamar con algún evento o cuando se desee y se deja hasta que el usuario cierre la venta; entonces se procede a revisar las banderas pasa saber si hubo cambios o no; de haberlos, la función regresa los nuevos valores de las coordenadas del observador, la posición focal y de la cámara.

¿Dónde se utiliza?

Se utiliza donde se quiera tener una interfaz para solicitar este tipo de datos.

Nombre de las funciones	Especificación
int DameObs(int)	Se devuelve el valor de la nueva coordenada del observador que se encuentra en posición del observador con el número que se proporciona en el arreglo
int DamePos(int)	Se devuelve el valor de la nueva coordenada del observador que se encuentra en posición de la cámara con el número que se proporciona en el arreglo
int DameFoc(int)	Se devuelve el valor de la nueva coordenada del observador que se encuentra en posición del foco con el número que se proporciona en el arreglo
int DameBanOb(void)	Devuelve la bandera de los cambios de las coordenadas del observador
int DameBanFo(void)	Devuelve la bandera de los cambios de las coordenadas del foco
int DameBanPo(void)	Devuelve la bandera de los cambios de las coordenadas de la posición de la cámara
DameOb1(int)	Asigna las coordenadas en el eje x, en el observador cada vez que se realice un cambio en ellas
DameOb2(int)	Asigna las coordenadas en el eje y, en el observador cada vez que se realice un cambio en ellas
DameOb3(int)	Asigna las coordenadas en el eje z, en el observador cada vez que se realice un cambio en ellas

Cuadro A.6. Especificación de las funciones de la clase.

Nombre de las funciones	Especificación
DamePo1(int)	Asigna las coordenadas en el eje x, en el observador cada vez que se realice un cambio en ellas
DamePo2(int)	Asigna las coordenadas en el eje y, en el observador cada vez que se realice un cambio en ellas
DamePo3(int)	Asigna las coordenadas en el eje z, en el observador cada vez que se realice un cambio en ellas
DameFo1(int)	Asigna las coordenadas en el eje x, en el observador cada vez que se realice un cambio en ellas
DameFo2(int)	Asigna las coordenadas en el eje y, en el observador cada vez que se realice un cambio en ellas
DameFo3(int)	Asigna las coordenadas en el eje z, en el observador cada vez que se realice un cambio en ellas
can(void)	En caso de cancelación, se regresa a los valores por omisión
acepta(void)	En caso de aceptar, se asigna a las variables que se devolverán, de ser requeridas
KvCamara (QWidget *, const char*)	Constructor
int DameObservador (void)	Devuelve la bandera de los cambios las coordenadas del observador

Cuadro A.6. Especificación de las funciones de la clase.

Ejemplo del uso de la clase

En el siguiente ejemplo se muestra el manejo de la clase.

```
int main (int argc, char **argv)
{
QApplication a(argc, argv);
  if (!QGLFormat::hasOpenGL()) {
  qWarning ("El sistema no soporta OpenGL!");
  return 1;
 }
QDialog *w=new QDialog(this,"Camara");
KvCamara *Camara= new KvCamara(&w,"Camara");
     //instanciamos la clase
Camara->DameObservador();
     //mandamos a llamar a la
    //funcion para que aparezcan los controles de la clase
 w.resize (500,650);
 a.setMainWidget(&w);
 w.show();
 return a.exec();
}
```

Clase KvMaterial

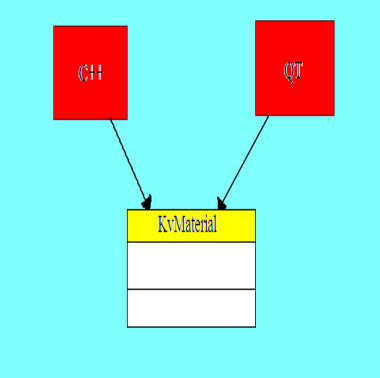


Figura A.5. Dependencias de la clase KvMaterial

KvMaterial	
+ BanMater: int	- ven:QDialog
- fSpe:float	- fPow:float
- fAmb:float	- fDif:float
+ DameSpecular(void): float	+ DameAmbiental(void): float
+ DameDifuso(void): float	+ DamePower(void): float
+ KvMaterial(QWidget*, const char*)	+ AsigSpecular(int)
+ AsigAmbiental(int)	+ AsigDifuso(int)
+ AsigPower(int)	+ DameMaterial()
+ can(void)	

Esquema A.7. Funciones y elementos de la clase KvMaterial.

¿Qué es KvMaterial?

Es una clase que proporciona una interfaz gráfica para solicitar los datos acerca de las características del material, como son el aspecto ambiental, especular, difuso y el grado intensidad tendrá. Básicamente esta clase contiene una interfaz predeterminada para el control de estos datos, así como las funciones que tiene los valores para devolver a su padre, al momento de sean tecleados.

¿Cómo funciona?

Funciona a través de *Qt*, ya que tiene los valores predefinidos de las características del material que se solicita; así como tres banderas, una para cada característica. Estas le avisan al padre que este objeto se ha modificado. Al recibir el cambio de las banderas, el padre tendrá que solicitar los valores y proporcionarlos a las funciones correspondientes, si se requiere. Los controles que se manejan en esta clase son *sliders*, para colocar el valor de las características requeridas.

¿Cómo se utiliza?

Se utiliza como hijo de otra clase, o como un elemento más de *Qt*, solo se manda a llamar con algún evento o cuando se desee y se deja hasta que el usuario cierre la venta, entonces se procede a revisar las banderas pasa saber si hubo cambios o no; de haberlos, la función regresa los nuevos valores de las características.

¿Dónde se utiliza?

Se utiliza donde se quiera tener una interfaz para solicitar este tipo de datos.

Nombre de las funciones	Especificación
float DameSpecular(void)	Devuelve el valor especular que se le asigna al material
float DameAmbiental(void)	Devuelve el valor ambiental que se le asigna al material
float DameDifuso(void)	Devuelve el valor difuso que se le asigna al material
float DamePower(void)	Devuelve el valor de la intensidad que se le asigna al material
KvMaterial(QWidget *, const char *)	Constructor
AsigSpecular(int)	Se asigna el valor especular al material
AsigAmbiental(int)	Se asigna el valor ambiental al material
AsigDifuso(int)	Se asigna el valor difuso al material
AsigPower(int)	Se asigna el valor de la intensidad al material
DameMaterial()	Es la función que manda a llamar al dibujado de los controles
can(void)	En caso de cancelación, coloca los valores por omisión en todos lo campos

Esquema A.8. Especificación de las funciones de la clase.

Ejemplo del uso de la clase

En el siguiente ejemplo se muestra el manejo de la clase.

```
int main (int argc, char **argv)
{
QApplication a(argc, argv);
  if (!QGLFormat::hasOpenGL()) {
  qWarning ("El sistema no soporta OpenGL!");
  return 1;
 }

QDialog *w=new QDialog(this,"Material");
KvMaterial *Material= new KvMaterial(&w,"Material");
 //instanciamos la clase
Material->DameMaterial();
//mandamos a llamar a la
//funcion para que aparezcan los controles de la clase
 w.resize (500,650);
 a.setMainWidget(&w);
 w.show();
 return a.exec();
}
```

Clase KvFunciones

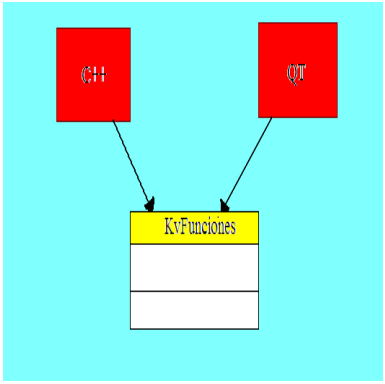


Figura A.5. Dependencias de la clase KvFunciones.

KvFunciones	
- cooX[3]: int	- cooY[3]: int
- bCooXY2: bool	- fValCooY[3]:float
- bCooXY1:bool	- fValCooY[3]: float
- CooXY1:bool	+ BFun:bool
- cooX[3]:int	- cooY[3]:int
- bCooXY0: boo	- bMovi: boo
- bCooXY2:bool	- bCooXY0:bool
- bMovi:bool	+ DameCooX(int): int
+ DameValY(int):float	+ AsingCooX(int *)
+ AsingCooY(float *)	paintEvent(QPaintEvent *mou)
mousePressEvent(QMouseEvent *mou)	mouseMoveEvent(QMouseEvent *mou)
KvFunciones(QWidget *, const char *)	

Esquema A.9. Funciones y Elemento de la clase KvFunciones.

¿Qué es KvFunciones?

Es una clase que proporciona una interfaz gráfica para solicitar los datos acerca de las funciones para realizar *volume rendering*.- la función de transferencia del color, la del gradiente y la de opacidad. Básicamente esta clase contiene una interfaz predeterminada para el control de estos datos, así como las funciones que tiene para devolver los valores a su padre, al momento de sean se
'onalados.

¿Cómo funciona?

Funciona a través de *Qt*, ya que tiene los valores predefinidos de las características del material que se solicita; así como tres banderas, una para cada característica, las cuales le avisan al padre si este objeto ha sido modificado. Al recibir el cambio de las banderas, el padre solicita los valores y los proporciona a las funciones correspondientes, si se requiere. La interfaz gráfica que se maneja es un conjunto de gráficas con puntos de control móviles, para que sean acomodadas para formar la gráfica que se desee, por el momento solo tiene seis puntos de control, pero se puede aumentar la cantidad de puntos de control, según se requiera.

¿Cómo se utiliza?

Se utiliza como hijo de otra clase o como un elemento más de *Qt*; solo se manda a llamar con algún evento o cuando se desee y se deja hasta que el usuario cierre la venta, entonces se procede a revisar las banderas pasa saber si hubo cambios o no y de haberlos, la función regresa los nuevos valores de las características.

¿Dónde se utiliza?

Se utiliza donde se quiera tener una interfaz para solicitar este tipo de datos. Principalmente en las aplicaciones donde se maneje *volume rendering*, ya que es más práctico manejar las funciones.

Nombre de las funciones	Especificación
int DameCooX(int)	Devuelve el valor de la coordenada que se encuentra en el lugar del arreglo que se proporciona
Float DameValY(int)	Devuelve el valor del elemento que se encuentra en el lugar del arreglo que se proporciona
AsingCooX(int *)	Asigna el valor de la coordenada que se encuentra en el lugar del arreglo que se proporciona
AsingCooY(float *)	Asigna el valor del elemento que se encuentra en el lugar del arreglo que se proporciona
paintEvent(QPaintEvent *mou)	Es el encargado de dibujar la zona de dibujo para las gráficas
mousePressEvent(QMouseEvent *mou)	Es el encargado de detectar dentro de la zona de dibujo para las gráficas, cuando con el mouse se oprime algún botón para obtener las nuevas coordenadas
mouseMoveEvent(QMouseEvent *mou)	Es el encargado de verificar el movimiento del mouse en la zona de dibujo de las gráficas y proporcionar las nuevas coordenadas
KvFunciones(QWidget, const char)	Constructor

Esquema A.10. Especificación de las funciones de la clase.

Ejemplo del uso de la clase

```
int main (int argc, char **argv)
{
QApplication a(argc, argv);
  if (!QGLFormat::hasOpenGL()) {
  qWarning ("El sistema no soporta OpenGL!");
  return 1;
 }
QDialog *w=new QDialog(this,"Funcion");
KvFuncion *Camara= new KvFuncion(&w,"Funcion");
  //instanciamos la clase
  // en el constructor se manda a llamar a la
 //funcion para que aparezca la zona de grafica de la clase
 w.resize (500,650);
 a.setMainWidget(&w);
 w.show();
 return a.exec();
}
```

Clase KvCamMon**.**

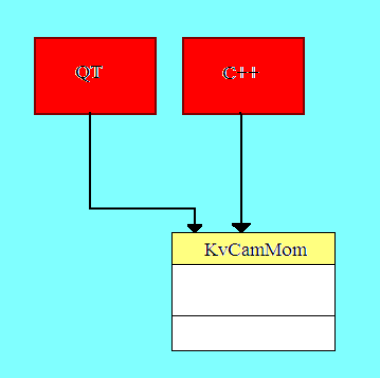


Figura A.6. Dependencias de la clase KvCamMom.

KvCamMom	
- cooX: int	- cooY: int
+ SelecX: int	+ SelecY: int
-total: int	- fMax:float
+ opcion: int	+ bMovi: bool
- fMin:float	- fValCooY:float
+ bCooXY: bool	+ Eleccion: bool
+ MinYMax(float,float)	+ AsignaMemo(int)
+ ConverDat(void)	+ Elegir(int)
+ DameMaxEsc(void):: float	+ DameMinEsc(void)::float
mousePressEvent(QMouseEvent *mou)	mouseMoveEvent(QMouseEvent *mou)
+ KvCamMom(QWidget *, const char *)	+ AsignaCoorY(float *)
+ AsignaCoorX(int *)	

Esquema A.11 Funciones y elementos de la clase KvCamMon.

¿Qué es KvCamMon?

Es una clase que proporciona una interfaz gráfica para dibujar gráficas de puntos. Básicamente esta clase contiene una interfaz predeterminada para el control de la gráfica, así como las funciones que tiene para devolver el valor del punto seleccionado a su padre, al momento de sean tecleados.

¿Cómo funciona?

Necesitará recibir dos conjuntos de valores, uno de ellos es con el valor en x y el otro con el valor en y, para dibujar los punto en la gráfica. Teniendo eso, los dibuja y permite seleccionar un punto en la gráfica. Al elegir un punto se guardan los valores de sus coordenadas, al ser requeridas serán devueltas a través de las funciones de su clase.

¿Cómo se utiliza?

Se utiliza como hijo de otra clase, o como un elemento mas de *Qt*, solo se manda a llamar con algún evento o cuando se desee y se deja hasta que el usuario cierre la venta; entonces se procede a revisar las banderas pasa saber si hubo cambios o no; de haberlos, la función regresa el nuevo valor seleccionado.

¿Dónde se utiliza?

Se utiliza donde se quiera tener una interfaz para solicitar este tipo de datos.

Nombre de las funciones	Especificación
MinYMax(float,float)	Asigna el máximo y el mínimo al conjunto de datos, para dimensionar la gráfica
AsignaMemo(int)	Asigna memoria para el número de puntos que recibirá para dibujar las gráficas
ConverDat(void)	Convierte los datos que recibe a las coordenadas de la pantalla para que grafiquen correctamente
Elegir(int)	Dice cual es el número *id* del punto que se seleccione
float DameMaxEsc(void)	Devuelve el número máximo asignado al conjunto de datos
float DameMinEsc(void)	Devuelve el número mínimo asignado al conjunto de datos
paintEvent(QPaintEvent *mou)	Es el encargado de dibujar la zona de dibujo para las gráficas
mousePressEvent(QMouseEvent *mou)	Es el encargado de detectar en la zona de dibujo de las graficas, cuando con el mouse se oprime algún botón para obtener las nuevas coordenadas
mouseMoveEvent(QMouseEvent *mou)	Es el encargado de verificar el movimiento del mouse en la zona de dibujo de las gráficas y proporcionar las nuevas coordenadas
KvCamMom(QWidget *, const char *)	Constructor
AsignaCoorY(float *)	Asigna las coordenadas en Y tomando los datos del arreglo que recibe
AsignaCoorX(int *)	Asigna las coordenadas en X tomando los datos del arreglo que recibe

Esquema A.12. Especificación de las funciones de la clase.

Clase KvMomento

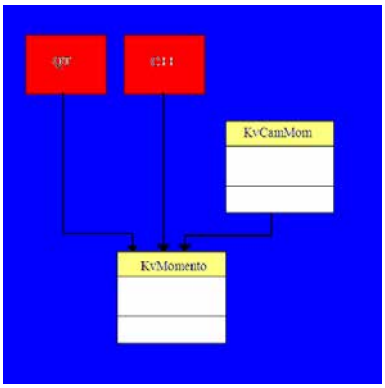


Figura A.7. Dependencias de la clase KvMomento.

KvMomento	
+KvMomento(QWidget *, const char *)	+ camvas1: KvCamMom *
+ camvas2: KvCamMom *	+ camvas3: KvCamMom *
- iMedias:int *	- iCs:int **
- iValY:int	- Momento1:float *
- Momento2:float *	+ Data: vtkImageData *
+ iRecta[5]:int	+ fRecta[5]:flota
- archivo[200]:char	- formato[20]:char
- iMaxM1:float	- iMaxM2:float
- iMaxM3:float	- Momento3:float*
- iMinM1:float	- iMinM2:float
- iMinM3:float	- iProfu:int
- iAncho:int	- iAlto:int
- iPorcen:int	- iauxi:int
- Valor:int	- iArre:int **
+ DameValor1(int): int	+ DameValor2(int): int
+ DameValor3(int): int	- HacerPorcen(void)
- LlenaArreglo(int,int,int,int)	+ DameRectaCoo(void): int *
+ DameRectaEsca(void): float*	+ princi(void)
+ eligeRe(void)	

Esquema A.13. Funciones y elementos de la clase KvMomento.

¿Qué es KvMomento?

Es la clase donde se realizan los cálculos para los momento de alto orden, así como su graficación. En esta clase están integrados las diferentes funciones necesarias para calcular los momentos de alto orden, además los componentes para graficarlos y permitir la interacción con las gráficas resultantes.

¿Cómo funciona?

A la clase se le deben asignar en sus variables, los valores de las coordenadas de las gráficas, así como indicarle cuántos valores se mandarán para que los grafique; cuando tenga

los datos, se procederá ha realizar los cálculos tal y como se indican en los procedimientos. Al terminar los cálculos se tendrán todos los elementos para graficar, esta función maneja el objeto del kernel de KvCamMon para graficar los resultados.

¿Cómo se utiliza?

La clase debe recibir un objeto *vtkStructuredData*, el porcentaje de la muestra que se tomará para realizar los datos y el valor de la Y que se utilizará como criterio para determinar el número de C1 o C2. Se procede a realizar los cálculos; a su término se mandan los resultados a un objeto de KvCamMon que mandará a graficar dichos resultados, así como iniciará la interacción del objeto.

¿Dónde se utiliza?

Se utiliza donde se requiera utilizar cálculos de momentos de alto orden.

¿En quién se apoya?

Esta clase se apoya en la clase KvCamMon para gráficar los puntos y lograr la interacción entre estos.

Nombre de las funciones	Especificación
int DameValor3(int)	La función devuelve el valor del momento 3, que corresponde a la posición que es dada por el valor que recibe
HacerPorcen(void)	La función calcula el número de datos correspondientes al porcentaje asignado. También obtiene del conjunto de datos ese porcentaje para asignarlo a la muestra que se utiliza para calcular los momentos
LlenaArreglo(int,int,int,int)	Esta función recibe las tres coordenadas x, y, z y el lugar en el arreglo de datos para el calculo de los momento. Esta función obtiene del arreglo de elementos que ocupa las coordenadas x, y, z proporcionadas y es asignado a la posición en el arreglo de los datos de los momentos
princi(void)	Es la función principal para mandar los datos a dibujar las graficas en los objetos KvCamMon
eligeRe(void)	La función se encarga de recibir el evento de seleccionado de algún elemento de las graficas de los momentos, y cambia los colores ha seleccionado

Esquema A.14. Especificación de la funciones de la clase KvMomento.

Nombre de las funciones	Especificación
int DameValor1(int)	La función devuelve el valor del momento 1, que corresponde a la posición que es dada por el valor que recibe
int DameValor2(int)	La función devuelve el valor del momento 2, que corresponde a la posición que es dada por el valor que recibe

Esquema A.14. Especificación de la funciones de la clase KvMomento.

Clase KvMenu

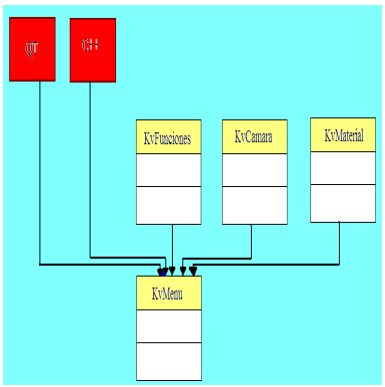


Figura A.8 Dependencias de la clase KvMenu.

KvMenu	
+ bFuncion: bool	+ bAgreVol: bool
+ iVol16: int	+ iVol: int
- Dat: QMenuBar	- volumenes: QPopupMenu
- opc: QPopupMenu	- carac: QPopupMenu
- mFunci: QPopupMenu	- label : Qlabel
- RevisaRangIso(void)	-CambiaRangIso(int ini, int fin)
- iTemp; int	+ iselec: int
+ Alto8; int	+ Ancho8; int
+ Profun8; int	+ Camara: KvCamara
- Ruta[120]: char	- inicio,final : int
- Iso: int	- temp1 : int
- temp2: int	- temp1A: int
- temp2A: int	- padre:Qwidget
+ Material: KvMaterial	+ Funcion1; KvFunciones
+ Funcion2: KvFunciones	+ Funcion3: KvFunciones
- temp3P: int	- OpcionVol: int
- banCamObs: int	- banCamPos: int
- banCamFoc: int	- RangIsoIni: int
- RangIsoFin: int	- BanIsoCam: int
+ AsignaIso(int)	+ DameOpcion(void):int
+ DameRuta(void):char	+DameNombre(void):char
+ DameInicio(void):int	+ DameFinal(void):int
+ DameIso(void):int	+ Archivo816(void)
+ Abrir()	+ Abrir8()
+ bit8(void)	+ bit16(void)
+ Profu(int)	+ tiff(void)
+ JPG(void)	

Cuadro A.15. funciones y elementos de la clase KvMenu.

KvMenu	
+ pnm(void)	+ IsoVal()
+ IsoValor(int)	+ RangoIma()
+ CambioIso(void)	+ Aplica()
+ AplicaIso()	+ Ancho(int)
+ ValorIni(int)	+ ValorFin(int)
+ Cambio(void)	+ defaul(void)
+ sFunci(void)	+ Ini(void)
+ IniArch(void)	+ AplicaFun(void)
+ fDefaul(void)	+ fCan(void)
+ sBorrar(void)	+ Alto(int)

Cuadro A.15. funciones y elementos de la clase KvMenu.

¿Qué es KvMenu?

Es la clase donde está definido el menú para abrir imágenes, borrarlas y el manejo de las características de la escena; así como el manejo de las funciones para realizar el *volume rendering*. Este menú no es muy completo en el aspecto general, esto es debido a que solo es la unión de las opciones que maneja el kernel con respeto a la escena y a los datos. Ya que si se requiere un menú más general, es mejor que se cree con *Qt*. El menú que maneja *Qt* es eficiente y se forma de manera sencilla, por eso no se contempló la realización de un menú muy general, sino uno que fuera acorde con las aplicaciones que se pretenden implementar con él. De la misma manera cuenta con las interfaces necesarias para pedir los datos para su carga.

¿Cómo funciona?

El menú se coloca en la clase principal, manda a llamar con base en las opciones que contiene, a diferentes funciones de los diferentes objetos que son parte de él. En dicho menú se integran cuatro funciones del kernel, que son referentes al manejo de la escena y de las funciones del *volume rendering*. A través de las funciones, el menú controla el flujo de la información, solamente a través de él tienen contacto unas clases con otras.

La clase menú está lista para el manejo de los eventos de selección de alguna de sus opciones, al recibir el evento de oprimido reacciona mandando el control a la función correspondiente para que sea procesada por algún objeto de los que está integrado. Ya que cada opción se maneja como un objeto externo de la clase KvMenu, la cual puede ser programada por el usuario o utilizar una por omisión.

¿Cómo se utiliza?

Se debe colocar en algún contenedor de *Qt*, ya sea el principal o alguna ventana hija, de esta manera la clase KvMenu se dibujará al momento de pintar al contenedor y comenzará a monitorear los eventos para saber si algún evento es de los que la clase espera. De ser así, reaccionará la clase y mandará a llamar a la función del objeto asignado para que se haga cargo del control. En caso de no desear todas las opciones que el menú presenta, será necesario heredar de esta clase para poder retirar los elementos no deseados, ya que en caso de no hacerlo de esta manera, es posible que se provoquen volcados de pila por las uniones que maneja la clase KvMenu.

¿Dónde se utiliza?

La clase KvMenu es útil en situaciones donde se requiera una interfaz que maneje las opciones de carga de datos, escena principalmente, ya que son las funciones más comunes y solicitadas. Otros casos donde puede ser utilizada son en situaciones de manejo de control del flujo de la información de procesos; ya que a través de esta clase se puede manejar ese tipo de situaciones, en donde se tiene procesos que solo se pueden utilizar uno a la vez; en este caso KvMenu proporcionará la interfaz gráfica con la flexibilidad de que solo se elija uno a la vez.

¿En quién se apoya?

La clase se apoya en tres clases del kernel que son KvCamara, KvFunciones y KvMaterial, las cuales se encargan del manejo de eventos. Las clases que apoyan a KvMenu son las que proporcionan el comportamiento de la aplicación según la opción seleccionada de la interfaz de KvMenu. También brinda el apoyo suficiente para controlar a las clases de Carga y KvVTKImage, de ser necesario pero son indispensables del menú.

La clase KvMenu es útil en situaciones donde se requiera una interfaz que maneje las opciones de carga de datos, escena principalmente, ya que son las funciones más comunes y solicitadas. Otros casos donde puede ser utilizada son en situaciones de manejo de control del flujo de la información de procesos; ya que a través de esta clase se puede manejar ese tipo de situaciones, en donde se tiene procesos que solo se pueden utilizar uno a la vez; en este caso KvMenu proporcionará la interfaz gráfica con la flexibilidad de que solo se elija uno a la vez.

¿En quién se apoya?

La clase se apoya en tres clases del kernel que son KvCamara, KvFunciones y KvMaterial, las cuales se encargan del manejo de eventos. Las clases que apoyan a KvMenu son las que proporcionan el comportamiento de la aplicación según la opción seleccionada de la interfaz de KvMenu. También brinda el apoyo suficiente para controlar a las clases de Carga y KvVTKImage, de ser necesario pero son indispensables del menú.

Nombre de las funciones	Especificación
AsignaIso(int)	Asigna el isovalor que devuelve su interfaz donde se solicita
int DameOpcion(void)	Devuelve la opción que fue seleccionada del menú
char DameRuta(void)	Devuelve la ruta del archivo
char DameNombre(void)	Devuelve el nombre que recibió de la interfaz donde el usuario la tecleo
int DameInicio(void)	Devuelve el valor del rango inicial
int DameFinal(void)	Devuelve el valor del rango final
int DameIso(void)	Devuelve el isovalor
Archivo816(void)	Interfaz para pedir la ruta y el nombre de los archivos de 16 bits

Cuadro A.16. Espeficicación de las funciones de la clase.

Nombre de las funciones	Especificación
Abrir()	Interfaz para pedir la ruta y el nombre de los archivos de imágenes
Abrir8()	Interfaz para pedir la ruta y el nombre de los archivos de 8 bits
bit8(void)	Marca la bandera de que los datos serán de 8 bits
bit16(void)	Marca la bandera de que los datos serán de 16 bits
Profu(int)	Asigna la profundidad
tiff(void)	Marca la bandera de que los datos serán de tiff
jpg(void)	Marca la bandera de que los datos serán de jpg
ppm(void)	Marca la bandera de que los datos serán de ppm
bmp(void)	Marca la bandera de que los datos serán de bmp
pnm(void)	Marca la bandera de que los datos serán de
IsoVal()	Interfaz para pedir el isovalor
IsoValor(int)	Asigna el isovalor que devuelve su interfaz donde se solicita
RangoIma()	Interfaz para pedir el rango de las imágenes
CambioIso(void)	Interfaz para pedir colocar el cambio de rango del isovalor
Aplica()	Interfaz para pedir aplicar el volume rendering
AplicaIso()	Interfaz para pedir aplicar la isoperficie
Ancho(int)	Asigna el ancho de las imagen
ValorIni(int)	Asigna el valor inicial de los rangos, (Slot)
ValorFin(int)	Asigna el valor final de los rangos
Cambio(void)	Interfaz para pedir el cambio de volúmenes
defaul(void)	Interfaz para colocar los valores default
sFunci(void)	Interfaz para mandar a llamar a la función de KvFunciones
AplicaFun(void)	Asigna los valores a las funciones de volume rendering

Cuadro A.16. Espeficicación de las funciones de la clase.

Nombre de las funciones	Especificacíon
fCan(void);	En proyecto a futuro
sBorrar(void)	Interfaz para borrar volúmenes
Alto(int)	Asigna el alto de las imágenes

Cuadro A.16 Espeficicación de las funciones de la clase.