UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SISTEMA DE VISUALIZACIÓN DE ISOSUPERFICIES

T E S I S

QUE PARA OBTENER EL TÍTULO DE :

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

MORALES ARREDONDO EFRÉN ARTURO

DIRECTOR DE TESIS: M. I. JUAN CARLOS ROA BEIZA

MÉXICO, D. F. MAYO DEL 2004

A mi familia y a mi novia

Agradecimientos:

Deseo dar las gracias a toda mi familia (Carmen, Enrique, Carlos, Iván, Gerardo y Daniel) ya que me han apoyado en toda mi vida, lo cual me ha ayudado a formar mi carácter y para mejorar en muchísimos aspectos, a mi futura esposa Yasmine por su comprensión y apoyo en el tiempo de realización del presente trabajo, a mi asesor por su paciencia y consejos para la realización de esta tesis, a mis amigos que fueron muy importantes a lo largo de los años, a quien fue una excelente amiga (Alaíde) por su inolvidable compañía y apoyo, también gracias al Dr. Barrios por facilitarme algunas imágenes tomográficas del Instituto de Neurobiología; Y por supuesto a mi querida Facultad como pieza clave de la UNAM, por su contribución a lo largo de varios años a mi formación y a la de muchos amigos y compañeros, en el aspecto académico y social dentro de sus muros a través de sus calificados profesores.

A todos gracias por formar parte de mi vida.

ÍNDICE

INTRODUCCIÓN

Con los avances tecnológicos, se registran de una forma en ocasiones cotidiana grandes volúmenes de datos proveniente de estudios y de sistemas complejos de exploración, estos suelen ser médicos o de investigación y análisis aunque resulta en muchos casos complejo analizar todo este conjunto de información debido a la forma en que se entrega esta por los sensores o cálculos realizados. Es por esto que en la actualidad es más conveniente presentar recursos que sirvan para poder analizar toda la información con menos tiempo, es por esto que la visualización presenta una importante herramienta para la muestra de datos, ya sea por medio de representación de patrones, tendencias o cantidades de una forma que un usuario sin conocimiento del campo pueda entender el comportamiento de la información; Pensando en estos antecedentes esta realizado el presente trabajo, en donde se presenta un análisis de estos datos para desplegarla, en este caso la representación visual es por medio de contornos tridimensionales que sirven para desplegar información que normalmente se visualiza como planos y que por medio de esta podrá ser analizada en un ambiente en tres dimensiones, existen diversos campos de aplicación, en el primer capítulo se pretende mostrar, de una forma general el método a usar para realizar el despliegue de esta información, se mostrarán algunos campos en los cuales son utilizadas estas representaciones.

En el capítulo II se hará un recuento de técnicas y herramientas usadas para lograr el objetivo, como son métodos estadísticos, forma de análisis de datos, el paradigma de la programación orientada a objetos el sistema operativo utilizado y además una herramienta de desarrollo, Qt; En el tercer capítulo se realizará el planteamiento del problema y la forma en que se plantea su solución, las perspectivas y sus requerimientos para un correcto funcionamiento.

En el último capítulo se mostrará el análisis del sistema y su desarrollo, tanto interna (back-end) así como su interfaz (front-end), se mostrarán la forma en que fue analizado el sistema haciendo uso de herramientas orientadas a objetos.

CAPÍTULO I

ENTORNO DE DESARROLLO

1.1 ¿Qué es y para qué es una Isosuperficie?

Def. Una Isosuperficie es un contorno en tres dimensiones, donde cada punto en la superficie tiene el mismo valor, estas isosuperficies son construidas a través de un volumen de datos [1]

En los últimos años, los avances en la tecnología han facilitado la obtención de una gran cantidad de información en una infinidad de lugares como pueden ser hospitales lectores ambientales o laboratorios de investigación, sin embargo, estos datos por si mismos no resultan útiles para la mayor parte de las personas, para hacer más accesible la presentación de esta información es necesario crear un método de visualización que permita un mayor entendimiento; esta representación permite crear una forma visible de algo, ya sea este un concepto, idea o un conjunto de datos. Visualizar es representar de una forma gráfica un fenómeno, ya sea estático, como puede ser una imagen, ver figura 1.1.1, una gráfica de barras ver figura 1.1.2, o dinámicamente, como puede ser una animación.

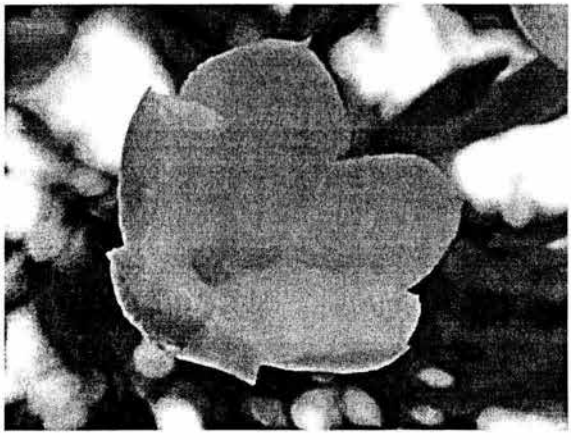


Figura 1.1.1 Imagen estática

[1] **Visualization**, using computer graphics to explore data en present information, Judith R. Brown pag. 69

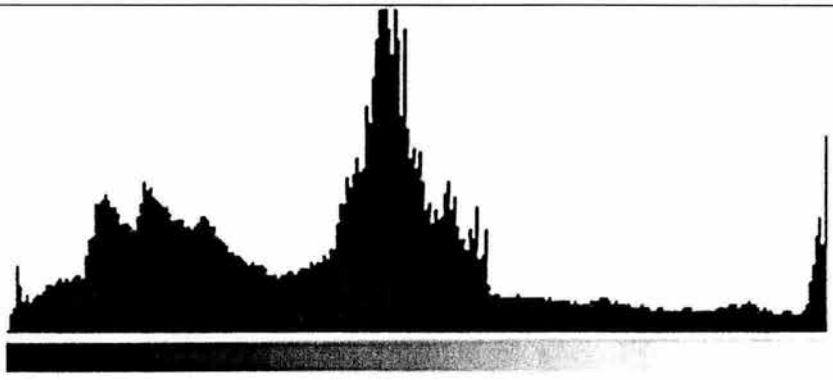


Figura 1.1.2 Grafica de barras de imagen anterior (histograma)

La visualización es utilizada debido a que una representación visual permite entender más fácilmente un conjunto de información, se estima que un 50 % de las neuronas se dedica a la visión, además, la densidad de información por unidad de área es notablemente mayor a la de un texto. Por otro lado la visualización nos permite reconocer lo que no resultaría fácil distinguir dentro de un gran conjunto de información, podemos reconocer patrones de comportamiento distinguir el agrupamiento de la información o definir tendencias.

Entre las ventajas que se tienen, es representar datos multidimensionales, lográndose por algunos métodos representar cuatro o más variables al mismo tiempo. Un ejemplo claro es el plano cartesiano, este puede mostrar dos variables, si agregamos otro eje, podremos ver 3 variables o dimensiones, si agregamos colores, tendremos 4 variables, si se hace alguna animación de la gráfica podremos apreciar una quinta variable, una ventaja extra es la independencia del lenguaje gráfico ya que si se intenta explicar a un japonés la manera en que se hacen estudios de flujos de contaminantes en la ciudad de México, probablemente no entienda una sola palabra, en cambio si se le muestra una simulación en video, hecho con datos reales tomados de estaciones atmosféricas en la Ciudad de México, ya solo será necesario explicarle que esa información fue recabada en cierto intervalo de tiempo.

Los modelos tridimensionales fueron iniciados como objetos compuestos de líneas o alambres, posteriormente se les dio volumen a través de una interpretación, la cual consiste en la creación de una superficie sólida por la interpolación de puntos existentes en la red. Posteriormente fueron diseñados métodos para la manipulación, modificación y animación de estos modelos; Algunos de estos métodos gráficos permiten (mediante la manipulación del plano), la representación de más de 2 variables en un solo plano, Los métodos de relación de datos crecen en complejidad conforme aumenta el número de variables a tomar en cuenta. Resulta común utilizar una computadora de gran capacidad y rendimiento para el proceso o generación de grandes cantidades de datos y posteriormente el uso de una estación de trabajo para la presentación gráfica de los resultados, de esta forma se pueden tomar las ventajas de ambos equipos. Posteriormente, las imágenes obtenidas pueden almacenarse en discos o grabarse en videos para facilitar su distribución y presentación.

Uno de los últimos avances en visualización es el uso de la realidad virtual. Por medio de ésta se puede generar "fácilmente" una representación tridimensional de objetos o lugares que no se podría lograr con una computadora y una pantalla de video normal. Por ejemplo se puede ver el funcionamiento de órganos o sistemas animales desde dentro del mismo, o se pueden tener representaciones en tres dimensiones de objetos de cuatro variables, sin que se vean reducidos a dos al presentarlos en una pantalla normal. Entre las aplicaciones más comunes de la visualización están el análisis exploratorio y confirmatorio (los cuales incluyen el análisis estadístico), la simulación y la educación. Dentro del área de análisis existen aplicaciones para control de calidad, proyecciones financieras, de esfuerzos, etc. Está última podría considerarse dentro del área de simulaciones, junto con los modelos atmosféricos. En cuanto a educación se tienen desde simulaciones matemáticas, hasta modelos de física cuántica y planetarios. El cálculo de isosuperficies es un método para presentar información recabada ya sea por medios electromecánicos o por simulaciones, este método de representación es por medio de superficies tridimensionales que son presentadas en una imagen generada por computadora.

La representación de una isosuperficie calcula y dibuja una imagen que contiene datos de campos volumétricos en 3D correspondiente a puntos que se basan en un valor escalar, se dice que es un análogo a los contornos, pero en 3D, en este caso se crean superficies donde alguna variable que no cambia su valor, de la misma forma los contornos de una isosuperficie puede dibujarse basada en variables vectoriales. Una isosuperficie puede cambiar su forma de acuerdo a su valor, estas se crean con elementos triangulares, cada uno de ellos dentro de los elementos tridimensionales,

Una isosuperficie genera una superficie o más dependiendo de los datos y del isovalor[2] asignado. Es decir se tiene un conjunto de datos los cuales son secuenciales tenemos un numero **x** que puede representar el ancho, un valor de **y** que puede representar el alto de los datos y un valor **z** que puede representar el numero de placas o imágenes a procesar, además de todo esto se debe tener un isovalor, el cual es un valor a partir del cual se debe generar la isosuperficie, a este método se le conoce como Marching Cubes (Cubos Continuos), un ejemplo de captura de imágenes secuenciales se encuentra en la figura 1.1.3

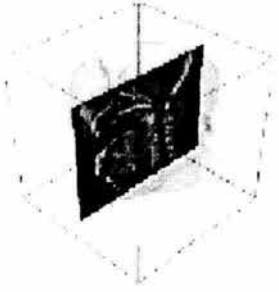


Figura 1.1.3 Placa de un conjunto de tomografías, juntas generan un volumen

El algoritmo marching cubes nos describirá la superficie en forma de polígonos descritos por las intersecciones de la superficie a evaluar con una composición de espacio en unas formas geométricas base. El algoritmo clásico comienza realizando

[2] Un isovalor es un valor que se elige para discriminar información dentro y fuera del contorno generado a partir de este

una descomposición del espacio afectado por la superficie en cubos de un tamaño determinado (geometrías base) a estos cubos se les denomina voxel por analogía a los pixeles bidimensionales; Dada la descomposición en voxeles, cada uno de estos puede encontrarse en una de las siguientes tres situaciones:

- Completamente fuera de la superficie
- Completamente dentro de la superficie
- Parcialmente dentro (intersecta la superficie).

El algoritmo después de realizar esta clasificación se dedicará a determinar la forma del polígono de intersección entre la ecuación de la superficie y el voxel, el número de posibles formas de intersección se encuentra limitado, el siguiente ejemplo es una analogía en 2D, ver figura 1.1.4, posteriormente se generalizará a 3D.

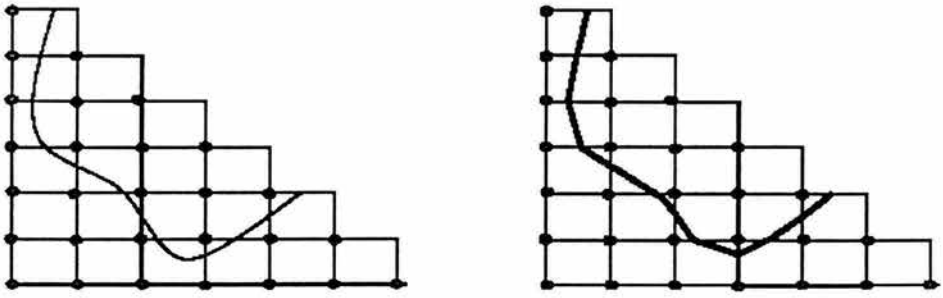


Figura 1.1.4 Comparación entre los datos reales y la imagen poligonal muestreada

En la imagen se observa una curva sobre un plano dividido por cuadrículas, esta curva limita un espacio dentro de cuadrados interiores otro de exteriores y un espacio de cuadrados intersectados, en este caso lo que se hace es segmentar la curva en tramos rectos los cuales definen los puntos de intesección de la curva con los

cuadrados, como se había mencionado este tipo de intersecciones se encuentra limitada a unas cuantas posibilidades en el caso 2D las formas de intersección se limitan a 16 posibilidades (dos de ellas no implican intersección) figura 1.1.5.

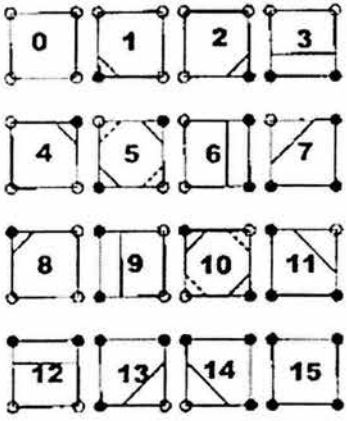


Figura 1.1.5 Posibilidades de intersección de la curva

En el caso 0 y 15 representan que no pasa la curva o esta completamente dentro respectivamente. La generalización en tres dimensiones es relativamente sencilla de comprender, aunque la implementación es bastante laboriosa, en este caso las intersecciones en lugar de ser segmentos rectos son polígonos y los casos de intersección se complican hasta 256 ver figura 1.1.6 diferentes casos, los mostrados son los principales casos.

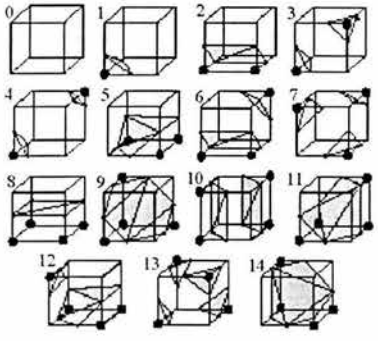


Figura 1.1.6 Posibilidades topológimente distintas para una superficie

Por lo que una superficie interpolada contenida en un conjunto de voxeles podría generarse como la mostrada en la figura 1.1.7

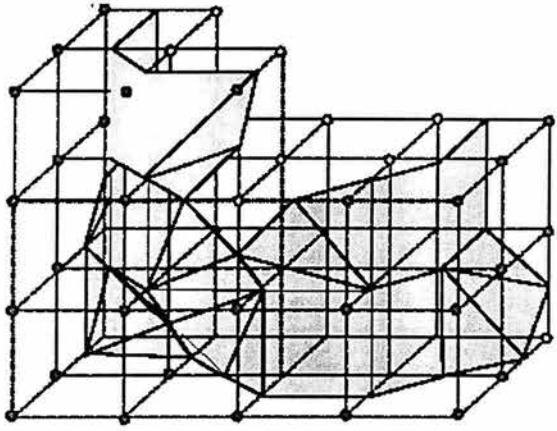


Figura 1.1.7 Superficie generada por el método de Marching Cube

Y posteriormente generando todos los polígonos de una superficie podríamos obtener una representación tridimensional más clara a la vista añadiendo texturas, colores, sombreado, etc. Como se muestra en la figura 1.1.8.

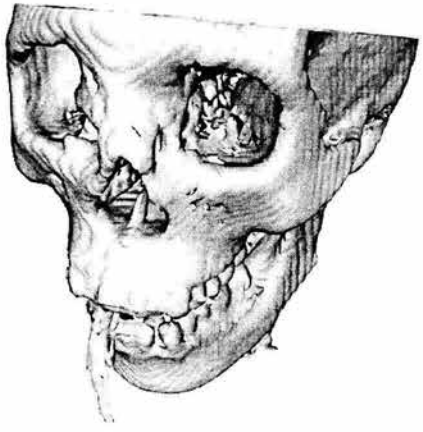


Figura 1.1.8 Isosuperficie final

1.2 Técnicas de representación de Isosuperficies.

No hay grandes variaciones en cuanto a la extracción de las isosuperficies, existen diversos métodos, pero en realidad se basan en un viejo algoritmo conocido como marching cube, el cual fue revisado anteriormente, y hay diversas variaciones para la extracción de isosuperficies, entre estas tenemos el método de marching tetrahedra (tetraedros continuos), este método es muy similar al método de marching cubes, pero en este caso son tetraedros las unidades básicas (estos tetraedros ordenados por seis forman lo que es un voxel, existen 8 diferentes casos ver figura 1.2.1:

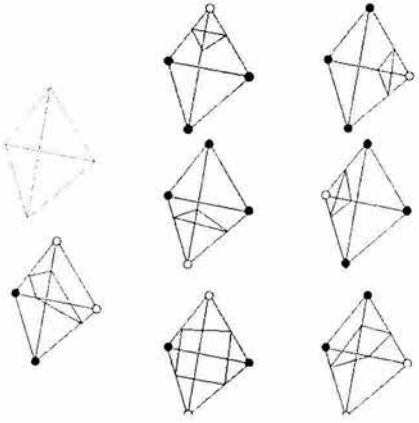


Figura 1.2.1 Diferentes casos para el algoritmo de tetraedros

Este método es básicamente el mismo del marching cube por lo cual no ha tenido tanta aceptación como el primero, en cuanto al algoritmo de Marching Cube este ha sido modificado para hacerlo mucho más eficiente, es por ello que a lo largo de los años este método ha sido fusionado con otras técnicas que enriquecen este método, aunque, varias de estas modificaciones son adaptaciones de acuerdo a las necesidades de cada proceso

Decimación de isosuperficies basada en octrees

Para entender primero que es un octree veamos que es un cuadtree, se sume que la imagen esta compuesta por pixeles blancos y negros, aquí el objetivo es encontrar regiones que contengan un objeto almacenándolas en un árbol y omitiendo el resto, para esto la imagen se divide primero en 4 en caso de encontrar en las cuatro regiones generadas un objeto se subdivide en otros cuatro cada cuarto y así sucesivamente hasta llegar a un nivel de pixeles, de esta forma se almacenan en el árbol solo los datos de la región ver figura 1.2.2.

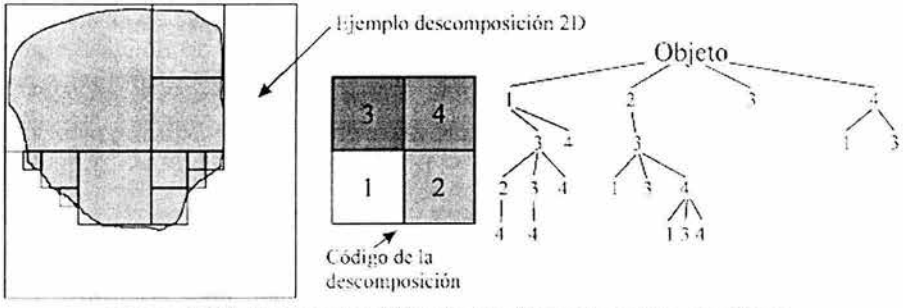


Figura 1.2.2 Descomposición de una imagen por un cuadtree

Un Octree es una representación jerárquica a través de la cual el espacio ahora se divide en ocho voxeles o nodos de igual tamaño, llamados octantes, el concepto anterior se utiliza para la representación de un octree, en la figura 1.2.3, se puede ver una representación de un objeto generado por un octree.

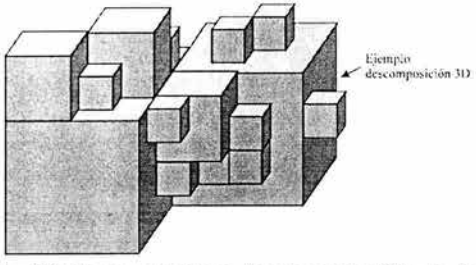


Figura 1.2.3 Analogía 3D de un cuadtree (representación de la descomposición por medio de un octree)

La extracción de isosuperficies utilizando el algoritmo de Marching Cube crea un gran número de polígonos los cuales dependiendo del tamaño genera un mayor número de triángulos lo cual hace mas tardado el tiempo de despliegue, es por esto que se presenta este método el cual equivale a eliminar una gran cantidad de elementos que normalmente no representan un gran conjunto dentro del total, estadísticamente se han analizado cuales son los casos sobre los cuales recae el mayor numero de probabilidad de ocurrencia, normalmente se tienen un total de 256 diferentes casos para la generación de isosuperficies, este método en especial consiste en sólo utilizar 6 casos diferentes ver figura 1.2.4 con sus respectivas variaciones de representaciones de voxeles.

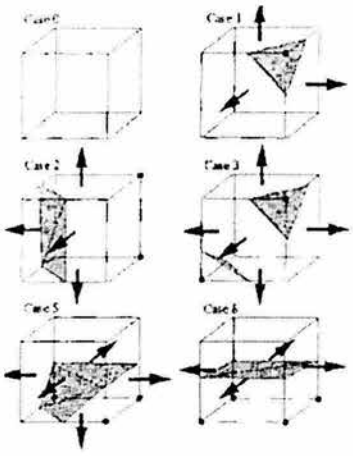


Figura 1.2.4 Diferentes casos para la decimación

Se calcula que aproximadamente un 90 % de las superficies utilizan estos seis casos, debido a lo cual permite ahorrar tiempo valioso de cómputo.

Los pasos que involucran este algoritmo de decimación son cuatro:

- Se elige el numero de celdas por las cuales pasa la isosuperficie

- El resultado se almacena en un octree
- Se procesa este octree, para que celdas simples sean remplazadas por celdas mayores si se permite por el criterio. (este método genera diferentes huecos en la superficie pero adelante se corrigen estos huecos)
- El paso final es realizar una triangulación para crear la superficie.

Este método tiene el inconveniente de que existe pérdida de información, por lo cual no es muy recomendable cuando se requiere una gran precisión de la isosuperficie.

Técnicas de extracción de isosuperficies para visualización de información volumétrica basada en Web.

En una aplicación Web los módulos pueden ser distribuidos en varias formas entre los clientes y servidores:

- Todos los módulos excepto para el módulo de despliegue se encuentran dentro del servidor, el cliente sirve para el despliegue solamente, a este únicamente se le envían las imágenes procesadas por el servidor, es decir la extracción de la isosuperficie se lleva a cabo dentro del servidor.

- En este escenario las coordenadas de pantalla de los vértices del triángulo con los colores desplegados se transmiten a la máquina cliente, mientras que el rendering[3] se lleva a cabo en la máquina servidor.

- En este escenario los valores de interpolación se trasfieren a través de la red
- El servidor es utilizado para filtrar las celdas del volumen de datos que interceptan a la isosuperficie, estas celdas son las que se transfieren al cliente.

- En este caso el servidor sólo se usa para almacenaje y manipulación del volumen de datos.

[3] Rendering - Render: Es la acción de procesar todos los datos de una escena gráfica, traduciendo sus parámetros y características a pixeles o imágenes.

Para el despliegue en este caso se ha hecho una comparación entre VRML y OpenGL para java (Java 3D), la independencia de la plataforma y el uso del render acelerado del lado del cliente son los objetivos importantes para elegir lenguajes de programación y API´s de render debido a la disponibilidad para los web browsers, Virtual Reality Modeling Language (VRML) es capaz de hacer uso de gráficos 3D para la visualización de isosuperficies, pero debido a sus limitaciones, OpenGL fue elegido para satisfacer los requerimientos de una isosuperficie basada en Web.

En una comparativa que se realizó se analizaron diferentes tipos de isosuperficies mostrando los resultados en VRML y OpenGL, el resultado obtenido: ambas librerías (VRML y OpenGL) presentaron una tendencia similar en el despliegue de los cero hasta los 85000 elementos en aproximadamente 5000 milisegundos, pero cuando se aumentan estos, en el caso del despliegue hecho por VRML presenta un incremento de tiempo representado por una gráfica exponencial a mayor número de triángulos mayor tiempo el requerido para el despliegue, mientras que utilizando las librerías de OpenGL la pendiente de la curva (triángulos/ms) es constante.

1.3 Procesos típicos para el análisis volumétrico de datos.

Los métodos de visualización permiten un eficiente análisis de datos, los cuales pueden guiar a investigadores a tener una visión más amplia. La extracción de isosuperficies es una técnica importante para visualización de campos 3D, este método proporciona un mecanismo para entender la estructura de los campos escalares, los contornos generados aíslan superficies de interés enfocando la atención en características importantes de los datos, pero este no es el único método de visualización de información tridimensional, existen varios métodos de representación para información volumétrica entre los cuales tenemos diferentes métodos interactivos que utilizan técnicas primitivas poco complejas que son conceptualmente fáciles de

implementar y rápidas, estas grafican dominios 3D utilizando técnicas convencionales de computación gráfica que generalmente trabajan con mallas de datos. Algunos métodos interactivos para estos tipos de datos son:

- Tyny Cubes (cubos o esferas de tamaño variable sin transparencia).

  Este modelo es utilizado para la visualización de la información espacial por medio de capas de color, y por separación de la información contenida en cubos, este método tiene como finalidad descomponer la información mostrada en pantalla para visualizar información interna, se descompone la superficie rendereada hasta obtener los resultados requeridos figura 1.3.1

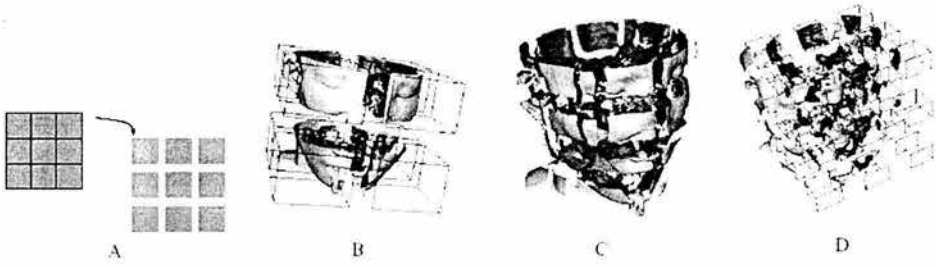


Figura 1.3.1 Descomposición de la imagen por el método de tiny cubes

En la figura 1.3.1 se puede ver la forma en que se hará la primera descomposición, mientras que cada área puede ser desplazada para incrementar el espaciado entre los datos. Las gráficas muestran diferentes isosuperficies que son descompuestos en cada sub-cubo con diferentes valores de separación.

- Vanishing cubes (lados de celdas proyectados con transparencia). Este método es muy similar al método de Tiny Cubes pero se hace uso de transparencias para el despliegue de voxeles.

- Métodos de Superficies, este método consiste en un valor de umbral a partir del cual se mapean los valores de datos en un conjunto de primitivas geométricas. Entre las características que tienen estos son que Transmiten menos información, se pierde una dimensión de información y no sirven para visualizar conjuntos de datos con formas complejas.

  Los pasos básicos de cualquier algoritmo de este tipo son:

  - Extraer la superficie a partir de una valor umbral.
  - Elegir parámetros de visión, de iluminación, tipo de proyección, etc.
  - Renderizar las primitivas geométricas.

  Algunos métodos de superficies destacados.

  • C*ubos opacos:* Renderiza un cubo opaco en cada celda cuyo valor está debajo del umbral. Es muy notoria la formación de bloques, ver figura 1.3.2

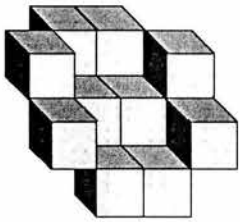


Figura 1.3.2 Cubos opacos

• *Dividing Cubes(cubos divididos)* : Renderizan puntos 3D basado en celdas. Divide adaptivamente las celdas hasta lograr proyecciones puntuales ver figura 1.3.3.

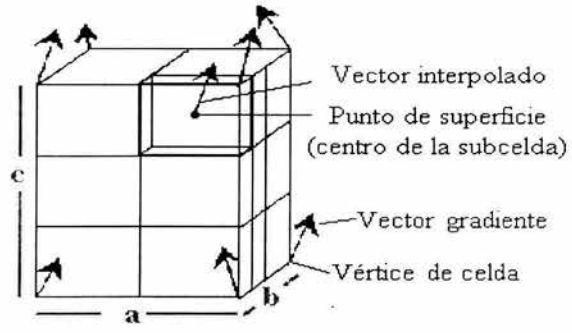


Figura 1.3.3 Método de dividing cubes

• Tracking de contornos: Conecta isocontornos en cortes transversales, produciendo un conjunto de triángulos que son renderizados por librería figura 1.3.4 .

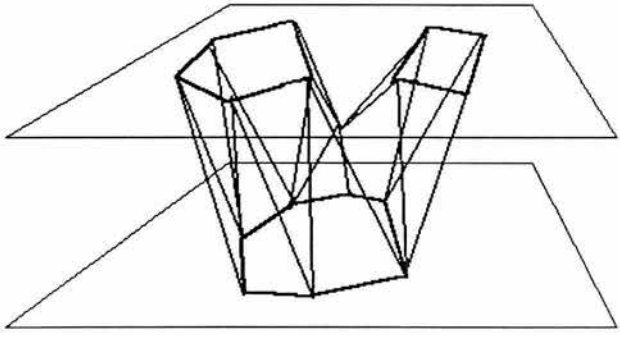


Figura 1.3.4 Método de tracking contornos

• Trayectorias de isosuperficies. Este método ya ha sido descrito anteriormente: Extrae una superficie umbral como conjunto de triángulos, que luego son renderizados por librería. Son muy utilizados porque se adaptan bien a datos estructurados en celdas o

a datos tetraedrizados (estructuras menos restringidas y más frecuentes).

1.4 Aplicaciones en Ciencias de la Tierra, Simulaciones de Fenómenos Físicos y Medicina.

El uso de isosuperficies abarca muchos campos, en tanto se maneje un gran volumen de información de la cual extraer datos para formarlas, se llevan a cabo simulaciones de diversos fenómenos meteorológicos a través de los cuales es más fácil entender el comportamiento de la naturaleza como son los fenómenos meteorológicos, para los científicos dedicados a este campo les resulta más fácil visualizar los comportamientos en el clima a través de información recopilada por medio de los satélites para poder hacer una predicción y poder llevar a cabo recomendaciones preventivas.

El campo de aplicaciones para la representación de datos a través de isosuperficies es muy amplio y actualmente se ha convertido en una herramienta muy útil para científicos médicos y personal involucrado en investigación para representar información de diferentes fuentes a través de una visualización 3D, existen muchos campos de aplicación de las isosuperficies como por ejemplo:

- **Química Computacional**

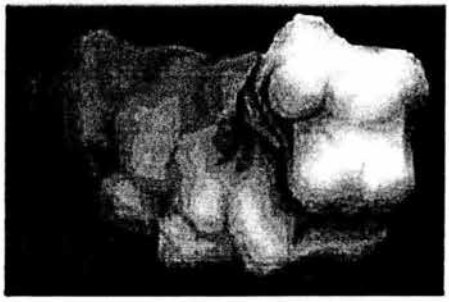
  Los modelos moleculares se han convertido en imprescindibles en el análisis e interpretación de los datos obtenidos (muchas veces excesivos) en los estudios de modelización molecular. Mediante la explotación de la percepción visual humana permiten identificar estructuras y reconocer formas y patrones, incrementando el entendimiento del problema.

  Una forma de visualizar el volumen molecular es mediante la representación de la superficie de la molécula. Los métodos para realizar esta operación son varios, y

las superficies que se obtienen son sensiblemente diferentes. La superficie se construye mediante la superposición de las esferas de los átomos. Su principal ventaja reside en la rapidez del cálculo. La superficie se construye haciendo "rodar' una sonda de radio pequeño determinado sobre la superficie. El tamaño de la sonda hace que existan regiones de espacio muerto que no son accesibles a la sonda, aumentando así el tamaño de la superficie. Normalmente se utiliza como sonda una molécula de agua, representada por una esfera entre 1,4 y 1,7 Å de radio La diferencia entre ambas superficies es muy pequeña en moléculas de bajo peso molecular, pero aumenta a medida que lo hace el tamaño molecular, y apreciándose una notable diferencia en estructuras proteicas, en las que pueden encontrarse regiones interiores que no contribuyen a la superficie accesible al solvente.

Este tipo de superficies proporcionan una buena representación de la forma y volumen que ocupa una molécula, los impedimentos esféricos y las colisiones entre átomos no enlazados, se utilizan ampliamente en la evaluación de interacciones intermoleculares. El uso de color ayuda en la representación de diferentes propiedades moleculares, como zonas hidrófilas, hidrófobas, capacidad de formación de enlaces de hidrógeno, acidez, carga parcial, densidad, potencial electrostático molecular, etc.

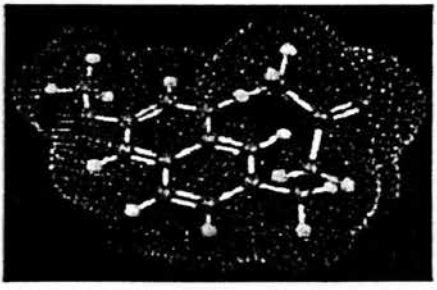
Las superficies suelen representarse mediante una serie de puntos uniformemente distribuidos (que pueden ser de distinto tamaño), mediante polígonos que forman una red (los triángulos son los más utilizados ya que son los que computacionalmente, menos recursos necesitan), o mediante una superficie sólida. En ocasiones se representan isocontornos o isosuperficies para simplificar, ver figura 1.4.1.



Superficie accesible Solvente

Superficie transparente

Superficie molecular



Superficie de puntos

Figura 1.4.1 Isosuperficies de representación de moléculas.

- **Análisis de datos meteorológicos**

Para el análisis de datos obtenidos a través de medidas por un radar (estos datos fueron obtenidos en mayo 3 en el norte de Alemania) se analizaron con isosuperficies, el objetivo de este proyecto es la cuantificación de la cantidad de lluvia basada en información del radar 3D, los datos proporcionados por el radar vienen de un sistema coordenado (cilíndrico o cónico), con estos datos el uso de técnicas como volume render, serían difícil de implementar, pero con múltiple extracción de isosuperficies son fácilmente aplicables y proporcionan diferentes imágenes y cortes, figura 1.4.2.

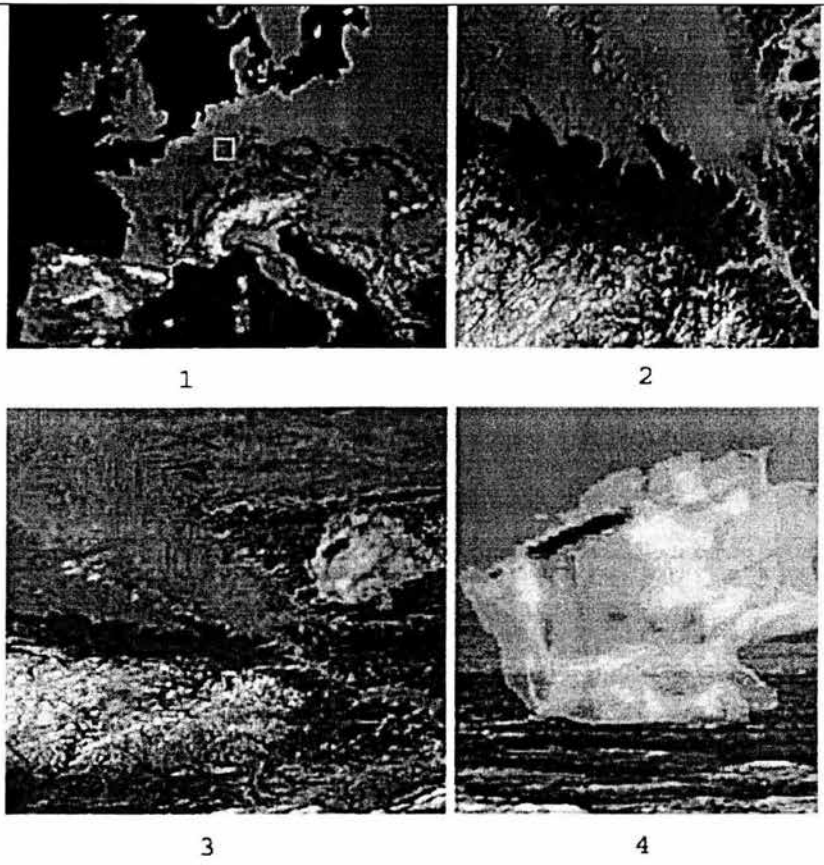


Figura 1.4.2 Generación de isosuperficies en información Meteorológica
1. Localización 2. Vista superior 3. Vista de pájaro 4. Formación amplificada

En esta figura pueden verse diferentes vistas de la posición de las nubes desde el norte de Europa hasta visualizar la formación de estas en la imagen d, como puede verse este tipo de información visual muestra mucha información con respecto a la posición de estas nubes.

La visualización de la información no es necesariamente plana, como bien se sabe existe movimiento del clima y por ello son creados campos vectoriales por medio

de los vientos, estos campos también son representados a través de isosuperficies, figura 1.4.3 Estas isosuperficies muestran a través de las flechas en la imagen el movimiento que tienen los vientos, de esta forma es más sencillo predecir la dirección de las formaciones nubosas.

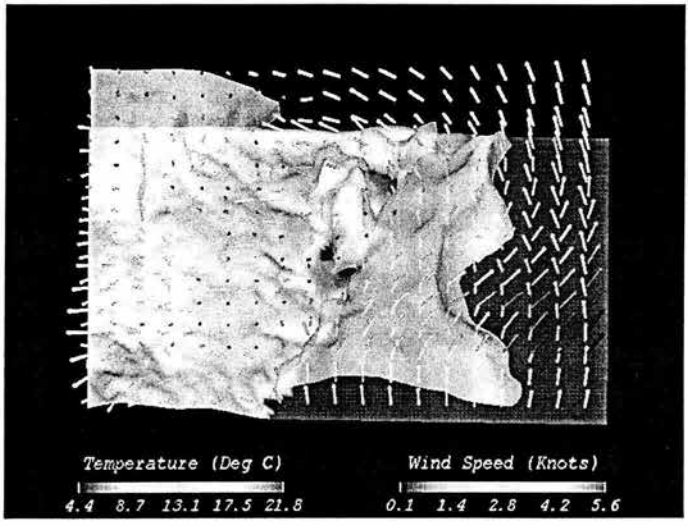


Figura 1.4.3 Representación de campos vectoriales

Cabe destacar que la información meteorológica global muestra informaciones de diferentes tipos como puede ser concentraciones de CO2, análisis de la capa de ozono entre otros.

- **Análisis vulcanológico**

  Otra aplicación es el análisis vulcanológico como ejemplo se manejara el caso de la unión de la placa del Pacifico y de la placa de Norteamérica y que desciende a través de la península de Kamchatka, se hicieron estudios debido a que en este lugar se presentaron dos episodios catastróficos en los 90´s, en información recabada y presentada como imágenes térmicas esta información es de una cadena de volcanes al este del Kamchatka que trazan 100 km de profundidad, con el modelo generado a través de las imágenes obtenidas ver figura 1.4.4.

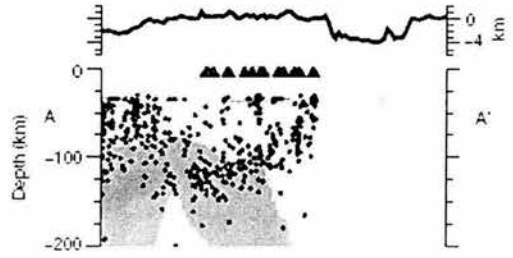


Figura 1.4.4 Imagen tomográfica

La representación tridimensional permite visualizar las concentraciones magmáticas de la falla y con ello entender de una forma más sencilla la distribución de diferentes elementos dentro del subsuelo, figura 1.4.5.

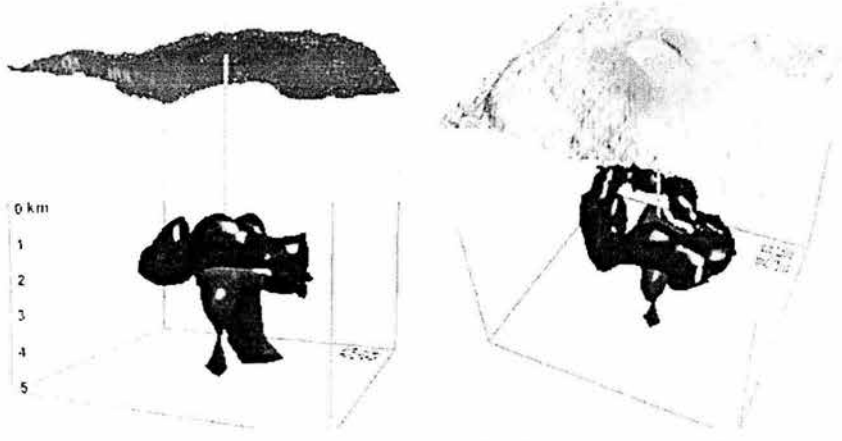


Figura 1.4.5 Representación de la distribución magmática.

- **Visualización de Estructura y Biología de Virus**

  En la biología moderna se hace énfasis en la relevancia de la estructura biológica de una proteína u otras macromoléculas, similarmente la estructura de la partícula

de un virus contiene muchísima información acerca de su biología y esta información puede ser utilizada para interpretar los datos genéticos para el diseño de drogas antivirales. La reciente tecnología ha permitido avances para determinar la estructura de muchos virus, para facilitar la extracción de información biológica de estas estructuras, se presentan de una forma más accesible para que puedan evaluarlas los especialistas en el campo, esta información puede ser animada o desplegada interactivamente.

En este caso la información se obtiene a través de micrografías que contienen una gran cantidad de detalles, las micrografías están en dos dimensiones y muestran como se encuentran esparcidos en una superficie de soporte en varias orientaciones y posiciones relativas, el nivel de ruido de las imágenes es bastante grande mientras que la calidad de la imagen es muy pobre ver figura 1.4.6.

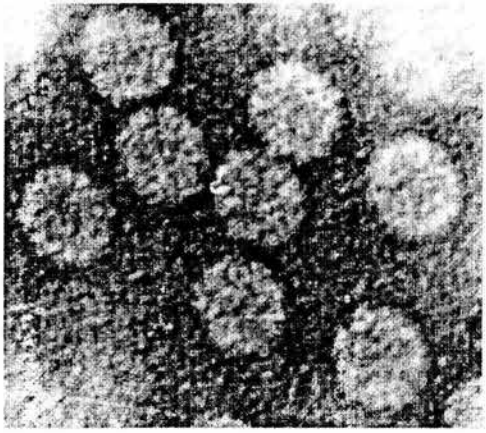


Figura 1.4.6 Micrografía de un virus

El despliegue de la estructura del virus es a través de superficies, comúnmente llamados isosuperficies, estas son usadas para mostrar una estructura tridimensional del virus, la información necesaria para llevar a cabo este despliegue es por medio de planos de corte bidimensionales que en conjunto forman la muestra de datos tridimensionales, generalmente este conjunto de datos viene en

un sistema cartesiano, en donde los cortes son planos ó son esféricos en estructuras radiales, ver figura 1.4.7.

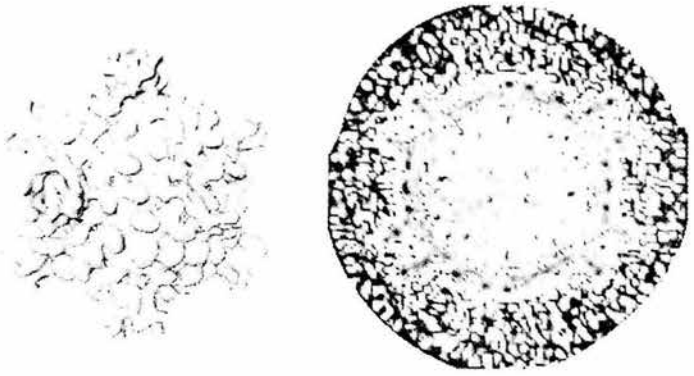


Figura 1.4.7 Isosuperficies generada a partir de cortes como la imagen derecha

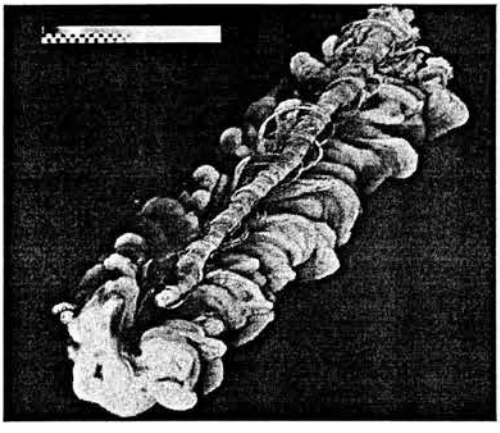
Una imagen es convertida de un subconjunto de valores de densidad tridimensional y para la representación de la isosuperficie se elige un isovalor para la aproximación de la superficie por medio de polígonos, los valores de la micrografía pueden tener diferentes tonos de acuerdo al valor de densidad, los planos agrupan los datos de densidad.

- **Utilización de Isosuperficies en Aeronáutica**

  La mezcla y la geometría de la concentración de fluidos en jets ocasionan una turbulencia transversal en jets de acuerdo a estudios hechos. La concentración de fluidos fueron medidos con una fluorescencia inducida por láser y algunas técnicas digitales. El campo escalar acertó en las medidas clásicas realizando estudios a través de las funciones de densidad de probabilidad. La geometría de las isosuperficies de campos escalares en turbulencia es importante para varias aplicaciones, en este caso para la combustión no premezclada. Un ejemplo de un corte de las imágenes del combustible se puede visualizar en la figura 1.4.8

Figura 1.4.8 Un corte transversal de la toma de imágenes del combustible

En este caso la isosuperficie generada, en la presentación tridimensional se pueden apreciar detalles que en una simple imagen no pueden apreciarse, pueden distinguirse patrones y elementos que a simple vista en un corte podría parecer ruido de la imagen, la isosuperficie generada por la imagen de arriba junto a su complemento puede apreciarse en la figura 1.4.9



1.4.9 Isosuperficie que muestra el movimiento del combustible en un jet

- **Análisis de Imágenes Médicas**

Para la visualización de datos volumétricos de alta resolución, fueron procesados agrupando puntos del conjunto de datos obtenido en un disco, dentro del campo médico el uso de las visualizaciones son muy diversas, para comprender más ampliamente una tomografía es muy útil una rutina visual en 3D, interactiva y de alta calidad, que aunque aun tiene altos requerimientos de hardware el equipo utilizado para estos fines, los datos generados de una investigación tomográfica requieren primero un análisis bidimensional y posteriormente continuar con las siguientes capas hasta terminar el volumen de imágenes, una vez que se tiene toda la información se extrae sólo lo que se requiere para llevar a cabo el despliegue de la isosuperficie, la elección correcta de una isosuperficie puede llevar a mostrar contornos del cerebro y quizás en el caso de existir algún tumor dentro de este mostrar que geometría presenta y la posición de este.

En las imágenes siguientes podemos observar diferentes isosuperficies generadas a partir de imágenes secuénciales obtenidas por diferentes lectores para poder hacer uso de los datos volumétricos. En la siguiente, figura 1.4.10 pueden verse isosuperficies generadas a partir de un conjunto de datos obtenidos de cortes del cerebro.

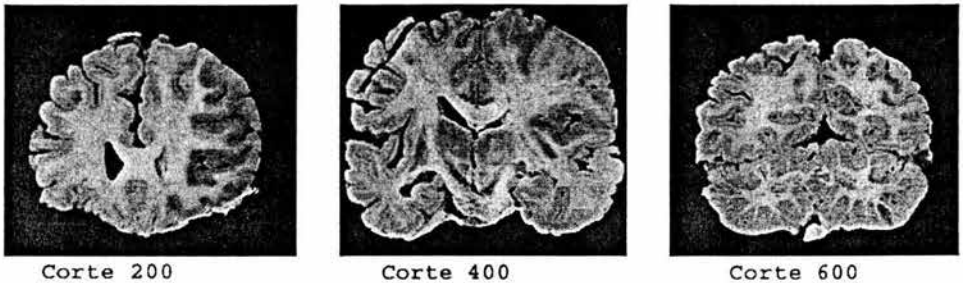


Figura 1.4.10 Cortes Tomográficos del Cerebro

mientras que las isosuperficies generadas por medio de estos cortes pueden ser apreciadas con diferentes características ver figura 1.4.11

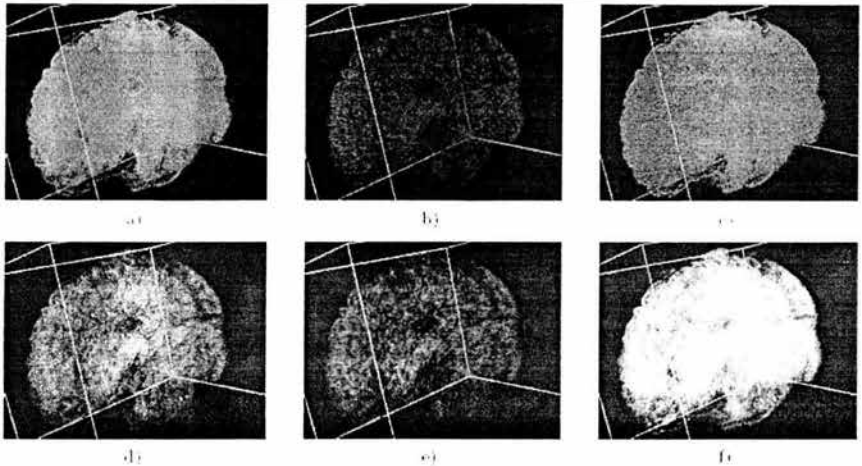


Figura 1.4.11 Isosuperficies generadas a partir de datos obtenidos de los cortes tomográficos a)Tamaño de píxel b)Con normales activadas c)El tamaño del píxel d)Blending activado e)Se cambió el valor gamma f)Modificación de Transparencia

Cabe destacar que este sólo es un ejemplo de lo útil que pueden ser las isosuperficies en el campo médico se ha utilizado esta herramienta en muchos campos médicos como puede ser la cardiología y la odontología entre otros ver figura 1.4.12.

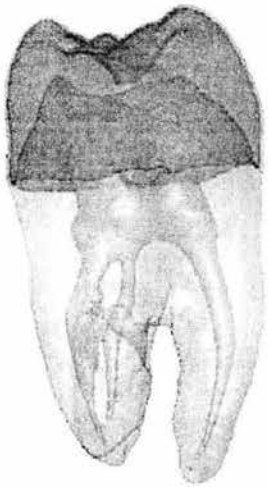


Figura 1.4.12 Isosuperficie de una muela

CAPÍTULO II

TEORÍA BÁSICA

2.1 Introducción al análisis estadístico de Isosuperficies

Para llevar a cabo el análisis de los datos a partir de los cuales se desarrollarán las isosuperficies es necesario hacer un estudio frecuencial de este para conocer características, el análisis que se lleva a cabo es por medio de cortes que unitariamente representan imágenes; una imagen está definida por un arreglo rectangular de elementos que la conforman, los cuales quedan representados mediante una matriz de datos, esta contiene un conjunto de datos que indican la brillantez de la imagen y el número es dependiente del tamaño de la imagen, si tenemos una imagen de 256x256 tendríamos un total de 65536 pixeles, en el caso de una imagen en tonos de grises, estos datos son valores que van desde el cero hasta el 255 en caso de que sea una imagen a color tenemos tres canales y no sólo uno, rgb que representan los colores rojo (red), verde (green) y azul (blue).Una imagen puede definirse matemáticamente como una señal bidimensional perteneciente a los reales $\Re$, pero al ser en dos dimensiones es un subconjunto en $\Re^2$.

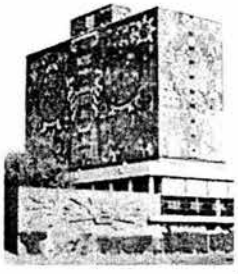
$$g = f(x, y)$$

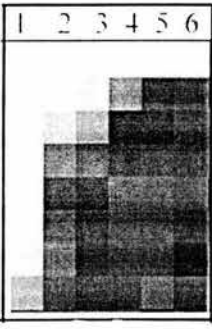
En donde f es una función que asocia el valor g al par (x, y). Esta es solamente una aplicación del espacio bidimensional en el espacio unidimensional, aunque es muy general esta definición cumple con el requisito de definir a una imagen a pesar de la diversidad de medios y características cromáticas, resulta poco adecuada, principalmente porque es una representación de dominios continuos con infinitos valores, lo cual es imposible de representar en la computadora. Es necesario entonces, lograr una definición que, aunque sea menos general sea más apropiada para la computadora. Por simplicidad, se considerará una imagen monocromática en que los elementos representan intensidades en forma de tonalidades de gris. Para ello se considerará el espacio discreto $\Re^2=\Re x\Re$ en el cual existen los pares (i , j) en el cual i y j son números naturales acotados, esto es no pueden tomar valores menores que

un valor mínimo ni sobrepasar un valor máximo. Considérese además, el valor discreto acotado g ϵ ℜ, con esto podemos definir una imagen como:

$$g = f(i,j); \quad \text{donde } i_{\min} \le i \le i_{\max} \text{ y } j_{\min} \le j \le j_{\max}$$

en que el par i, j corresponde a una posición en el espacio bidimensional, g un valor correspondiente al nivel de gris asociado a ese punto y f una función que asocia el valor g al par (i,j). Como en la práctica hay infinitos puntos, a cada par (i,j), le corresponderá, en general un promedio de las intensidades comprendidas en una zona finita de la subdivisión.





	1	2	3	4	5	6
1	102	166	117	090	129	083
2	136	132	055	081	094	069
3	139	124	066	122	025	082
4	117	157	049	075	010	081
5	118	092	072	**127**	038	048
6	120	045	068	095	088	084
7	087	042	065	085	111	122
8	065	084	045	081	140	187

Imagen Original en alta resolucion | **Imagen en Baja Resolucion** | **Valores asociados a los Elementos en baja resolucion**

Figura 2.1.1 representación de la imagen original en una matriz

La imagen en baja resolución queda reducida a la representación mostrada en la figura 2.1.1, en la cual cada recuadro corresponde a un valor de gris que representa al promedio de intensidades en el escenario original. Las zonas mas claras se representan por grises claros y las más oscuras por grises más oscuros.

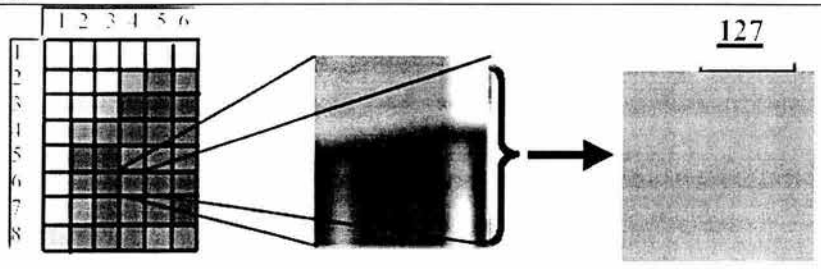


Figura 2.1.2 El promedio es el valor que queda plasmado en la matriz

En la figura 2.1.2 se muestra en la primera parte el escenario subdividido en 8x6=48 partes en que cada número vertical en la columna corresponde a un valor de j y cada número en el renglón corresponde a un valor de i. En la segunda parte se muestra un recuadro correspondiente a la zona (6,4), luego el valor de gris promedio de esa zona, en que 0 representa al negro y 255 representa al blanco.

Con lo anterior puede redefinirse la imagen, con orientación definida al análisis y procesamiento computacional como: Un arreglo de números en el cual la posición dentro del arreglo está asociada a una posición geométrica del escenario representado y cada valor en esta posición del arreglo es un número al que se asocia un tono de gris correspondiente al nivel de intensidad reflejada por el escenario original. En el manejo de datos relacionados a una imagen se usan los histogramas. El histograma es una representación gráfica en la que la frecuencia de la imagen está representada por un valor que representa un valor de píxel, en la figura 2.1.3 se muestra un histograma de ejemplo.

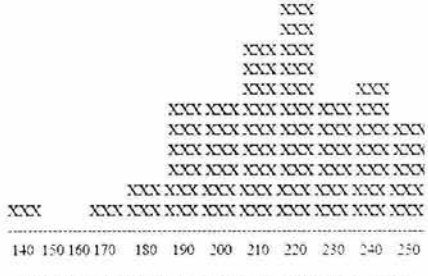


Figura 2.1.3 histograma de ejemplo

Para el análisis de datos es necesario obtener ciertos valores representativos de la distribución probabilística de los datos.

La media aritmética es la suma de los valores dividido por el número de valores. Si la media pertenece a una población se representa con la letra griega μ, si pertenece a una muestra con el símbolo de la variable con una barra encima x.

$$X = \frac{\sum_{i=1}^{n} X_i}{n} \quad \text{.....(1)}$$

$$X = \frac{\sum_{i=1}^{n} X_i}{n} \quad \text{.....(2)}$$

Xi representa todos los valores de una muestra o población dada

n numero de elementos muestrales o poblacionales

El rango se obtiene por medio de la diferencia entre el mayor y el menor de los valores. La varianza de una muestra 2 se define como se expresa en la ecuación 3:

$$s^2 = \frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{n} \quad \text{.....(3)}$$

Xi representa todos los valores de una muestra o población dada

n numero de elementos muestrales o poblacionales

x Representa la media

mientras que la varianza de una población finita, de N elementos (Ecuación 4):

$$\sigma^2 = \frac{\sum_{i=1}^{n}(X_i - \mu)^2}{N} \quad \text{.....(4)}$$

Xi representa todos los valores de una muestra o población dada

n numero de elementos muestrales o poblacionales

x Representa la media

μ media poblacional

El numerador de la varianza se le conoce como suma de cuadrados y se calcula generalmente como se expresa en la ecuación 5:

$$\sum(X - X)^2 = \sum x^2 = \sum_{i}^{n} X^2 - \frac{(\sum_{1}^{n} X)^2}{n} \quad \text{.....(5)}$$

A la cantidad (n-1) se le conoce como grados de libertad. En algunos casos se usa el valor (n-1) como denominador de la cuasi- varianza para no subestimar la varianza poblacional. La varianza como medida de dispersión nos sirve para determinar si desviaciones observadas son usuales o notorias.

Los momentos también son importantes en el análisis estadístico y estos son las esperanzas de las potencias de la variable. Por ejemplo el momento de orden v es la esperanza de la potencia v: $E[x^v] = \Sigma x^v \cdot p[x_i]$. Los momentos centrados con respecto a la media son las esperanzas de las potencias de los desvíos respecto a la media. Por ejemplo el momento de orden v con respecto a la media es la esperanza de la potencia v del desvío con respecto a la media Ecuación 6:

$$E[X-\mu]^v = \Sigma\,[X-\mu]^v \cdot p[X] \quad \text{.....(6)}$$

Para distinguir a los momentos centrales de los definidos previamente a éstos se les dice momentos ordinarios.

La media y la varianza como momentos, La esperanza es una idealización del concepto de media aritmética, de modo que se dice que la media es la esperanza de una variable. La varianza es el momento de segundo orden con respecto a la media, es decir es la esperanza del desvío cuadrático (Ecuación 7):

$$\sigma^2 = E(X - \mu)^2 \quad \text{.....(7)}$$

Función generadora de momentos. Si tenemos una función tal que su derivada n-ésima sea el n-ésimo momento de la variable se puede considerar que esa función "genera" los momentos de la variable. La función generadora de momentos no siempre existe, pero si existe es única. Por lo tanto se usa para identificar a la distribución. Por ejemplo, el Teorema del Límite Central comprueba que la función generadora de momentos de una suma de variables se aproxima a la de una normal.

Para la extracción de isosuperficies es necesario que tengamos un valor de referencia a partir del cual se llevara a cabo la eliminación de información que no es necesaria y la organización de los datos correspondientes a la isosuperficie, existen técnicas estadísticas para conocer valores sobresalientes dentro de un conjunto de datos, la técnica aquí abordada es una técnica de momentos, la cual consiste en realizar un análisis a partir de la distribución de los datos basada en el histograma,

2.2 Tratamiento matemático a través de matrices.

El estudio matricial se debe al matemático W.R. Hamilton en 1853. En 1858, A. Cayley introduce la notación matricial como una forma abreviada de escribir un sistema de *m* ecuaciones lineales con *n* incógnitas. Las matrices tienen un gran número de usos, dentro de las más comunes es la resolución de sistemas de ecuaciones lineales, diferenciales y parciales.

Una **matriz** de orden **m×n** representa conjunto rectangular de elementos a_{ij} dispuestos en **m** líneas horizontales (renglones) y **n** verticales (columnas) de la forma:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mj} & \cdots & a_{mn} \end{pmatrix}$$

Abreviadamente suele expresarse en la forma $A = (a_{ij})$, con i =1, 2, ..., m, j =1, 2, ..., n. Los subíndices indican la posición del elemento dentro de la matriz, el primero denota la fila (i) y el segundo la columna (j). Por ejemplo el elemento a_{25} será el elemento de la fila 2 y columna 5.

Para ingresar esta misma matriz en una computadora se debe tener un arreglo bidimensional de la forma A[i][j] en donde i=0,1,2,3,4...m-1 y j=0,1,2,3,....n-1, es decir el elemento a_{11} se introduciría en el elemento A[0][0], el a_{1n} en la posición A[0][n-1], el a_{mn} en la posición A[m-1][n-1] y así sucesivamente debe notarse que no cambia la dimensión de la matriz simplemente se inicia desde un valor cero en vez de 1.

La utilización de matrices (arreglos) constituye actualmente una parte esencial de los lenguajes de programación, ya que la mayoría de los datos se introducen en las computadoras como tablas organizadas en renglones y columnas por ejemplo en: hojas de cálculo, bases de datos simplemente al encender la computadora se lleva a cabo un refresh a los pixeles del monitor el cual tiene una organización matricial de dos dimensiones, ya dentro de la computadora en el sistema operativo todo el despliegue esta desarrollado a partir de un sistema de pares coordenados representado por posiciones, una imagen es en realidad una combinación bidimensional de números los cuales representan valores de pixeles lo más común es que esta representación tenga valores en el rango de (0-255) y en el caso de imágenes a color se tienen a parte otros canales que representan valores en rojo, verde y azul.

Como puede verse el álgebra lineal es fundamental para el análisis de imágenes como se mencionó anteriormente estas son matrices bidimensionales y tienen a la vez las mismas propiedades que estas, se pueden multiplicar, sumar, restar, obtener su inversa a partir de la cual podemos obtener la matriz identidad, etc. La manipulación de imágenes tienen una relevancia muy importante al manejar isosuperficies, como se ha explicado los datos a partir de donde se obtienen las isosuperficies son un conjunto de arreglos bidimensionales secuenciales y el orden debe ser respetado, es por esto que en el caso de tener un volumen de datos debe ser almacenado de la misma forma en la computadora, por ejemplo: si tenemos 64 tomografías, y cada una de estas son de un tamaño de 512 x 512 deberíamos almacenarlos en un arreglo data[512][512][64], con esto podemos darnos cuenta de la gran cantidad de memoria necesaria para su procesamiento.

2.3 Programación Orientada a Objetos.

Esta metodología es una de varias formas de programar.

- La programación procedural (C y Pascal) a través de una secuencia de instrucciones (normalmente organizadas de una forma estructuradas), utiliza funciones y corre los programas a través de la secuencia de instrucciones.

- Programación Funcional (por ejemplo Lisp) Un programa con estas características es un conjunto de funciones y el correr el programa involucra llamadas a funciones con algunos argumentos para obtener los resultados requeridos.

- Programación Orientada a Objetos. Un programa con estas características contiene un conjunto de objetos los cuales intercambian información unos con otros. En este caso el correr un programa involucra enviar mensajes a objetos.

La metodología orientada a objetos se basa en los siguientes puntos de vista:

- El mundo real consiste en objetos.
- Los objetos se comunican por intercambio de mensajes.

El enfoque de la programación orientada a objetos consiste en la identificación de objetos. Un objeto es una representación conceptual bien definida del mundo real, un objeto representa una cosa, ya sea tangible o intangible, en realidad todo lo que podemos imaginar, como puede ser un perro, un árbol, etc.

Un objeto se caracteriza por los siguientes componentes, *Identidad* cada objeto es único y puede ser distinguido de otros objetos, *Estado* es un conjunto de atributos caracterizados por el objeto. Funcionalidad es un conjunto de operaciones que un objeto posee y caracteriza su dinamismo, ver figura 2.3.1.

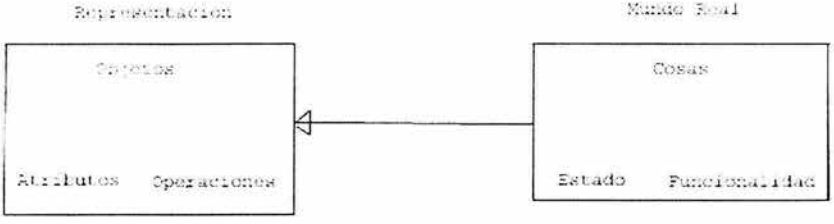


Figura 2.3.1 Abstracción de un objeto representación – mundo real

Los objetos son creados a partir de clases las cuales se utilizan en la programación orientada a objetos, una clase es un plan de construcción para todos los objetos de un tipo especifico y un conjunto de operaciones que son comunes a todos estos objetos. Después de definir una clase, pueden ser creados objetos llamados instancias o miembros de clase.

Otra característica importante de la programación orientada a objetos es la encapsulación, la cual tiene los siguientes beneficios:

- Modularidad. El código para un objeto puede ser escrito y mantenido independientemente del código fuente para otros objetos.
- Reusabilidad: Los objetos son entidades autónomas que contienen datos y operaciones para manipular estos, también pueden ser reutilizadas en diferentes situaciones.
- Ocultamiento de la Información: Un objeto tiene interfaz pública para que otros objetos puedan comunicarse. El objeto puede mantener información privada y detalles de la implantación que pueden ser cambiados en cualquier momento sin afectar otros objetos que pudieran ser dependientes.

Actualmente se ha convertido en un estándar el uso de UML para el modelado de la programación orientada a objetos

El lenguaje UML (en inglés, Unified Modeling Language) es un lenguaje para la especificación, visualización, construcción y documentación de las partes de un sistema de software. Este consiste en un conjunto de prácticas de ingeniería que mostraron ser exitosas en el modelamiento de sistemas complejos, a tal punto que muchas empresas están actualmente incorporando UML para el desarrollo de sus productos. UML es un lenguaje predominantemente visual, que consiste de varios diagramas, cada uno modelando un parte esencial del sistema a construir. Los diferentes elementos en UML se representan con un lenguaje gráfico Las clases son dibujadas con un rectángulo figura 2.3.2, dividido en tres partes: el nombre de la clase, los *atributos*, y las operaciones correspondientes. Puede agregarse también una división en donde se especifican las *responsabilidades* de esa clase.

Nombre
Atributos
Operaciones
Responsabilidades

Figura 2.3.2 Representación de la clase

Nombre: El nombre de la clase debe ser lo menos ambiguo posible, usualmente un sustantivo.

Atributos: Los atributos describen las características de los objetos. Poseen un *tipo*, que nos indica qué clase de atributo es. Si bien existen ciertos tipos primitivos, como *enteros*, *boléanos* y *reales*, cualquier tipo puede ser usado, incluso otras clases. La restricción más importante es que *los atributos son visibles únicamente por la clase que los contiene*. La sintaxis de declaración es la siguiente:

<nombre>:<tipo> < = valor_inicial >

El dato valor_inicial es opcional y permite inicializar los atributos directamente en la declaración.

Operaciones: Las operaciones son utilizadas para manipular los atributos o realizar consultas. La sintaxis para describir una operación es la siguiente:

<nombre_operación> (<parámetros>) : <tipo_resultado>

A diferencia de los atributos, las operaciones pueden tener diferente *visibilidad* hacia otras clases, la cual se denota entre llaves a la izquierda de la declaración. Todas las operaciones están agrupadas de acuerdo a los estereotipos <<comando>>, <<consulta>> o <<constructor>> de acuerdo a su función en la clase.

• *Responsabilidades:* Las responsabilidades son las obligaciones de una clase y son definidas por el usuario. Si bien existe un compartimiento dentro de la clase para la especificación de las responsabilidades, éstas son de carácter opcional.

Un ejemplo de lo anterior lo podemos ver figura 2.3.3:

Punto
x: entero; y: entero;
Punto(x,y:entero) mover(dist_x, dist_y:entero) distancia(P:Punto):real
Representa un punto en el plano

Figura 2.3.3 Ejemplo de una clase

A los gráficos se les pueden añadir notas o comentarios sobre algún aspecto interesante de la clase. Estas pueden incluir observaciones sobre la clase, alguna restricción de uso o pseudocódigo de algunas operaciones, estas se grafican por medio de un rectángulo con la esquina superior derecha plegada figura 2.3.4 y se une con una línea al elemento de la clase a la cual corresponde la nota.

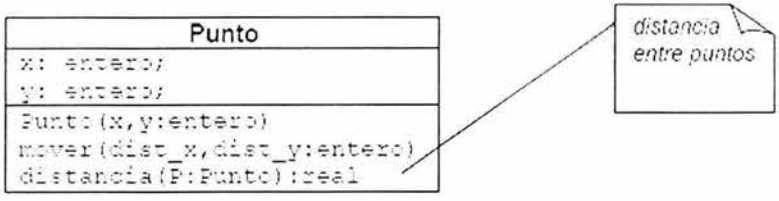


Figura 2.3.4 Representación de un comentario de una clase

Las relaciones entre clases más importantes son *asociaciones*, *agregaciones*, y *generalización*. Una asociación es una conexión entre clases. Significa que los objetos de dos clases tendrán un vínculo bidireccional en común, el cual puede interpretarse como "para cada X existe un Y". Se representa por medio de una línea continua entre dos clases figura 2.3.5.

Figura 2.3.5 Relación entre clases

En este caso cada programador utiliza una computadora, y cada computadora es utilizada por un solo programador (un objeto programador esta asociado a un objeto computadora) La asociación es una abstracción de la relación existente en los enlaces entre los objetos. Se utiliza *multiplicidad* cuando no necesariamente los vínculos entre objetos son de uno a uno. Se especifica en cada extremo del vínculo, a través de un rango, cuántos *objetos* pueden estar vinculados.

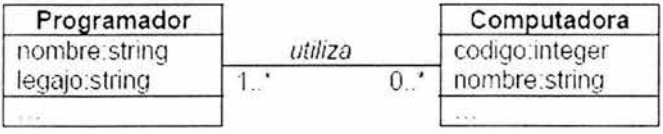


Figura 2.3.6 Relación de cantidades entre clases

La figura 2.3.6 indica que cada programador tendrá varias computadoras (posiblemente ninguna), y que cada computadora será usada por al menos un programador. Algunos ejemplos de rangos pueden verse en la figura 2.3.7

0..1	0..* (o sólo *)	1..*	2	5..11
cero o uno	*cero o muchos*	*uno o muchos*	*dos*	*cinco a once*

Figura 2.3.7 Rangos de ejemplo para relaciones

Cuando no se especifica ningún rango, se asume el valor uno (1) por defecto, pero, puede agregarse una flecha a la línea, figura 2.3.8, indicando la dirección del vínculo:

Rango 2.3.8 Manejo de herencia en UML

De esta manera, la clase A puede utilizar las operaciones de la clase B, pero no al revés. A esto se le llama herencia y en este caso A hereda los elementos de la clase B dependiendo del ocultamiento de la información.

Otro tipo de relaciones son las agregaciones, estas son un caso especial de asociación e indica que una o más clases *forman parte* de otra clase (clase *agregada*). Se denota con un rombo en uno de los extremos del vínculo, aquél que corresponde a la clase agregada. Puede interpretarse como "posee (o contiene) un" figura 2.3.9.

Figura 2.3.9 Agregaciones entre clases

Este diagrama indica que todo *auto* posee un *motor*, de una marca particular, y con su correspondiente número de identificación. En la agregación, la multiplicidad de la clase agregada debe ser cero o uno (0..1), pero en las clases que conforman las partes puede ser un rango cualquiera figura 2.3.10.

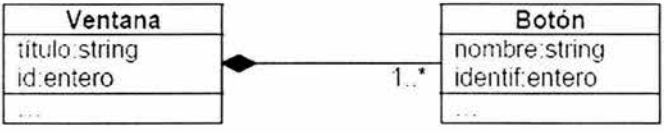


Figura 2.3.10 Ejemplo de agregación con multiplicidad

Aquí, el diagrama indica que una *ventana* de Windows puede contener uno o más *botones*. Otra posible forma de representar la *agregación* entre clases es incluir el gráfico de una clase (el agregado) dentro del gráfico de otra figura 2.3.11.

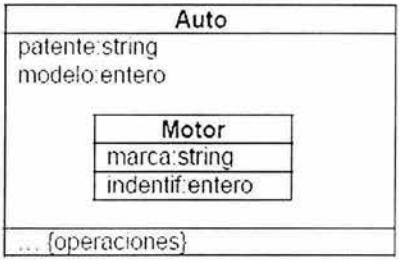


Figura 2.3.11 Otra forma de representar una agregación

Si el agregado requiere especificar multiplicidad, escribimos el rango correspondiente en la esquina superior derecha de la clase figura 2.3.12.

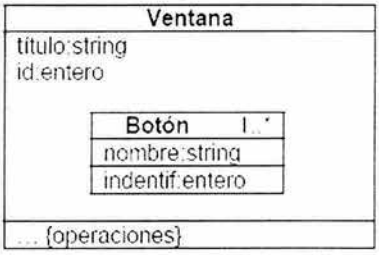


Figura 2.3.12 Agregando multiplicidad al ejemplo anterior

Otro tipo de relación entre una clase más general y un clase más específica es la generalización. En este caso la clase más especifica (denominada *subclase*) es consistente con la clase más general (denominada *superclase*), y contiene información adicional. La generalización permite la especialización de las clases. Se denota con un triángulo en el extremo del vínculo correspondiente al elemento más general. La relación puede interpretarse informalmente como "es un".

Figura 2.3.13 Ejemplo de herencia

La figura 2.3.13 indica que un *taxi* es una clase particular de *autos*, con una patente extra y un código utilizado por la empresa. Son *heredados* de la clase Auto todos los demás atributos, como la patente oficial, el modelo, el motor, etc. Cuando una clase posee más de una subclase, pueden organizarse en forma de árbol figura 2.3.14:

Figura 2.3.14 Organización arbórea de clases

Los objetos pueden modelarse también en un *diagrama de objetos*. Este diagrama utiliza la misma notación y relaciones que el *diagrama de clases*, puesto que los objetos son simplemente instancias de esas mismas clases. Mientras el diagrama de clases muestra las diferentes clases y sus relaciones, el diagrama de objetos muestra instancias específicas de estas clases, y los vínculos entre estas instancias en un momento dado. Los objetos son mostrados como las clases, pero con el nombre del objeto subrayado y precedido del nombre de la clase que representa. La sintaxis correspondiente es:

<nombre del objeto>:<clase>

Por ejemplo, utilizando el diagrama de las clases *Auto* y *Motor*, una instancia particular de estas clases puede ser por ejemplo la representada en la figura 2.3.15:

Figura 2.3.15 Representación de la relación Auto - Motor

A veces el nombre del objeto no necesariamente debe existir, por lo que en ese caso utilizaremos sólo los dos puntos (:) y el nombre de la clase, por ejemplo **:Avión**

La descripción de UML es mucho más extensa y poderosa para modelar sistemas orientados a objetos.

La programación orientada tiene principalmente las siguientes cualidades:

- Provee una clara estructura modular para programas, esto hace que sea bueno para definir tipos de datos abstractos, donde detalles de implantación son ocultos y las clases tienen una interface claramente definida.
- Proporciona unas buenas librerias, donde los componentes pueden ser fácilmente adaptados y modificados por el programador esto es particularmente útil para el desarrollo de interfaces gráficas.
- Hace que sea fácil de mantener y modificar código existente como nuevos objetos o crear nuevos con pequeñas diferencias de los otros objetos.

2.4 Características ventajas y desventajas de Unix y Linux.

A comienzos de la década de los 70´s nace el Sistema Operativo Unix, después de algunos años se codifica el núcleo de Unix en C dejando algunas rutinas en ensamblador, es este el inicio de este poderoso sistema operativo, con la implantación se crea un sistema operativo portable. Este SO introdujo una nueva idea en computación : Las aplicaciones son el conjunto de unas cuantas piezas simples, donde cada una de ellas realiza una única tarea, de tal manera que se pueden construir grandes aplicaciones a partir de una serie de secuencias simples, entre las principales características de este sistema operativo tenemos:

- *Portabilidad* Este sistema hoy día se encuentra en casi cualquier computadora y sus aplicaciones tienen el entorno adecuado para ser trasladadas.

- *Flexibilidad* El sistema se adapta a las más diversas aplicaciones, como es la automatización de fabricas, telefonía, juegos personales, bases de datos, etc.

- *Multiusuario y Multitarea* Una computadora puede trabajar con varios usuarios a la vez y desarrollar diferentes trabajos para cada usuario.

- *Orientado a Red* El sistema tiene el ambiente necesario para conectarse a otras máquinas por medio de la red.

- Su entorno es austero, lo cual lo hace consumir pocos recursos, aunque últimamente se ha mejorado el ambiente gráfico.

- Unix es un sistema de archivos jerárquico en el que todo se encuentra anclado a la raíz.

- El sistema de archivos esta basado en la idea de volúmenes de los cuales se pueden montar y desmontar para lo que se les asigna un nodo del árbol como punto de anclaje.

- Unix realiza un riguroso control de acceso a los archivos, cada uno de ellos se encuentra protegido por una secuencia de bits sólo se permite el acceso global al administrador (root), por lo tanto, el universo de usuarios Unix se encuentra dividido en dos grupos principales root el súper usuario quien no tiene restricciones de modificaciones al sistema y los demás usuarios que son controlados mediante las directivas de root.

- Una de las grandes ideas de Unix es la unificación y compatibilidad de todos los procesos de entrada y salida. De esta forma existe compatibilidad entre ficheros, dispositivos, procesos, pipes y sockets.

- El núcleo de Unix es relativamente compacto en comparación con otros sistemas de tiempo compartido. Introduce la idea de reducir el tamaño del

kernel y ceder ciertas funciones a programas externos al núcleo llamados demonios.

- El sistema presenta comandos de usuario (es decir, a nivel de shell) para iniciar y manipular procesos concurrentes *asíncronos*. Un usuario puede ejecutar varios procesos, intercambiarlos e interconectarlos a través de *pipes* o tuberías, simbolizados por el carácter ¦ (ASCII 124). En DOS, también existe la idea del pipe, sin embargo, al no existir concurrencia de procesos, no se trata de una comunicación en "tiempo real", sino de un paso de información a través de ficheros temporales.

Unix tiene muchísimas ventajas:

- *Sistema multitarea* - múltiples programas se ejecutan al mismo tiempo.
- *Multiusuario* - permite a mas de un usuario trabajar en la computadora al mismo tiempo compartiendo el tiempo de procesamiento entre distintos usuarios usando sistemas computacionales distribuidos.
- *Seguro* - previene al programa de accesar la memoria o el espacio en el disco ocupado por otro proceso, protección por un sistema de permisos de los usuarios a realizar ciertas funciones.
- Programación eficiente y poderosa del shell

El sistema operativo Linux tiene todo lo anterior y más, este sistema operativo es de libre distribución, debido a esto existen actualmente una gran cantidad de distribuciones con características propias mientras que el centro de este es básicamente el mismo en todos, entre las diferentes distribuciones encontramos:

- Red Hat
- Slackware
- Caldera

- Corel
- SuSe
- Mandrake

Es por esto que es tan llamativo este sistema operativo, quizás su principal desventaja es que este sistema operativo carece de soporte, esta es una parte importante por lo cual no tiene más presencia y aun así avanza día a día como principal sistema operativo, y debido al concepto de software libre crece día a día.

2.5 Características, ventajas y desventajas de C++.

C++ es un lenguaje de programación orientado a objetos y está diseñado partiendo del lenguaje C. Así se ha hablado mucho de él como un lenguaje orientado a objetos para los programadores de C (ya que casi todas las características y construcciones del C también están disponibles en el C++),

Entre las principales razones por las cuales utilizar C++ es su increíble versatibilidad, con este lenguaje podemos desarrollar pequeños programas y también realizar grandes sistemas como el caso de algunos sistemas operativos, también resulta importante destacar la portabilidad del código, es decir un programa escrito en C++, podrá ser compilado en cualquier sistema operativo sin la necesidad de modificar el código fuente, otra gran ventaja de C++ es que es un lenguaje multinivel, es decir puede ser utilizado para programar Hardware (esto depende del sistema operativo) y también desarrollar aplicaciones gráficas, si tiene desventajas el lenguaje C++ es que aun no existe un estándar como en el caso de C con el ANSI C, y el manejo de memoria aun presenta muchos casos en los cuales existen errores en el manejo de memoria.

El código de C++ normalmente era tratado por un compilador de C dado que es un supraconjunto de C. C++ ofrece todas las posibilidades de C y más aun. Debido a lo anterior podría pensarse que el cambio de C a C++ es muy fácil. Programadores que están familiarizados con C pueden comenzar "programando en C++" usando archivos

fuente con extensión .cc o .cpp en vez de .c y de esta forma utilizar todas las ventajas que C++ tiene. Sin embargo a lo programadores acostumbrados a los lenguajes estructurados (C, Fortran, Pascal, etc.) no hacen la transición a C++ de manera simple, pues se necesita cambiar la forma de atender los problemas para la programación orientada a objetos. En C++, además de haber más herramientas para la programación, también es posible programar utilizando diferentes paradigmas como programación modular, programación orientada a objetos y programación genérica. A pesar de que C++ no es un lenguaje totalmente orientado a objetos, es una herramienta muy útil para realizar proyectos grandes con esta metodología. En la POO existen conceptos que necesitan ser bien entendidos para tener éxito en la construcción de sistemas de software y esto se logra con la práctica.

La mejor forma de aprender C++ es enfocándose en los conceptos y dejar para después los detalles técnicos. El objetivo de saber programar en C++ es llegar a ser un mejor programador y diseñador de software, y no simplemente aprender una nueva sintaxis para hacer las cosas de diferente manera.

La sintaxis de C++ es muy similar a Java, pero normalmente los métodos de las clases creadas sólo colocan la firma de estos mientras que la implantación de la misma se colocan en un archivo externo con extensión .h o .hpp, la sintaxis general de una clase en C++ es la siguiente:

```
class Fecha {
  int d, m, a;   // datos privados por omisi'on

public:              // seccion publica
  Fecha() { ... };
  ~Fecha() { ... };

  void setFecha(int, int, int);

protected:           // seccion protegida
  ...
}
```

Como puede verse es la misma notación que se tiene en UML, claro que este ejemplo va más enfocado al compilador, cuando se crean métodos a través de los cuales se modificarán los atributos estas clases son almacenadas en archivos con extención .cpp, .C o .cc.

Una clase en C++ puede contener dos funciones especiales que están involucradas con el trabajo interno de la clase. Estas funciones son el constructor y el destructor.

El constructor es una función que se declara con el mismo nombre de la clase correspondiente. El constructor no tiene valor de regreso en su especificación, tampoco se puede usar void. En la ejecución, C++ hace que el constructor de la clase, si está definido, sea llamado al momento de crear un objeto. Es posible definir una clase sin constructor; en este caso el sistema no llama a ninguna función o llama a una función "virtual" que no realiza ninguna acción. Lo anterior depende del compilador. Cuando un objeto es una variable local en una función, el constructor es llamado cuando la función es ejecutada. Cuando el objeto es una variable global, static, el constructor es llamado al inicio del programa; antes de que main() sea ejecutada.

Normalmente en el contructor se inicializan valores y se crean ciclos que serán utilizados por default en un objeto dado.

Otra parte importante de la notación de C++ es el uso de la palabra reservada const, con la cual se indica que el valor del objeto no será modificado a través del programa. Cualquier intento de modificación de algún elemento declarado como const redituará a un error en tiempo de compilación.

Una de las principales características de C++ es el manejo de la memoria dinámica y para la creación y liberación de la memoria dinámica se utilizan los operadores new y delete, un claro ejemplo de su uso es:

```
float *apuntador;
apuntador = new float;
```

```
delete apuntador;
```

En este caso se declara una variable de tipo apuntador a un flotante y se utiliza para apuntar a la memoria que es asignada a través del operador new y posteriormente la memoria se libera a través de delete, es común que se olvide liberar la memoria solicitada cuando no se utiliza el objeto, y sólo es liberada esta cuando se termina la ejecución del programa, esto es muy útil cuando estamos utilizando grandes porciones de memoria y requerimos reutilizar las celdas de memoria para actualizar información. Normalmente se libera la memoria a través del destructor.

Otra característica de C++ es que permite hacer uso de Herencia múltiple, es posible heredar de varias clases a la vez. Con la herencia múltiple podemos añadir determinada funcionalidad a un conjunto de clases. Una clase que hereda de varias clases base obtiene la funcionalidad de más de una clase base al mismo tiempo. Sin embargo, existen algunos problemas con este tipo de herencia, estas pueden ser por coincidencia de nombres y herencia repetida, la primera puede ocurrir cuando tenemos dos clases con dos atributos con el mismo nombre y en una tercera clase se hereda de las dos primeras, cuando hacemos uso del atributo con el mismo nombre no se puede saber a que método se refiere pero para resolver esto se coloca como prefijo y dos puntos el nombre de la clase base, por ejemplo:

La clase B hereda de A1 y A2, las cuales tienen un método llamado f, la clase B puede hacer referencia y diferenciar los métodos de las diferentes clases por medio de <nombre-clase>::(método | atributo).

```
class B: public A1, public A2
{
  ...
  A1::f()    //Referencia a la f() de A1
  ...
  A2::f()    //Referencia a la f() de A2
  ...
};
```

el segundo conflicto que puede existir (Herencia repetida) la cual existe cuando las clases de las que se hace herencia múltiple, son a su vez derivadas de un ancestro común y se resuelve de la misma forma, teniendo una clase base A y unas subclases A1:

```
class A
{
public:
     char *comun;
     f(const char *c){ comun = c;};
};

class A1: public A
{
 // ...
};

class A2: public A
{
  // ...
};
```

Si ahora se construye una clase C que herede de A1 y A2 a la vez, tenemos que C hereda indirectamente dos veces de A. Esto se resuelve de la misma forma que fue resuelto el conflicto de coincidencia de nombres:

```
class C: public A1, public A2
{

  char* f() { return A1::comun; };

  char* h() { return A2::comun; };
};
```

Estas son algunas de las características de C++ que nos lleva a construir programas mejor organizados. Algunas de las ventajas de programar con C++ son:

- Programas nuevos se desarrollan en menos tiempo debido al rehúso de códigos anteriores.
- La creación y manejo de nuevos tipos de datos es más fácil que en C.
- El manejo de memoria es más fácil y transparente.

- Los programas son menos sensibles a errores dado que se usa una sintaxis y "chequeo" de tipos estrictos.
- El ocultamiento de la información, hace la implementación de los programas más simple.
- Tipos y funciones genéricas pueden construirse para reducir el tamaño del código.

La realidad es que los puntos de arriba podrían ser un poco exagerados si hablamos solamente de C++. En general, las ventajas antes mencionadas son ciertas para la POO.

El desarrollo de nuevos programas reutilizando código que ya existe puede también hacerse con el lenguaje C (y en otros lenguajes similares) usando funciones que se encuentran en diferentes bibliotecas. En C++ la reutilización del código también es posible usando conceptos de la POO, además de que también se puede hacer a través de las tradicionales bibliotecas.

Crear y usar nuevos tipos de datos también es posible en C a través de las estructuras (structs y typedefs). De los tipos creados es posible derivar nuevos tipos haciendo uso de estructuras que contengan estructuras, y así sucesivamente. Sin embargo en C++ existen mecanismos que permiten hacer lo anterior mucho más fácil.

El manejo de memoria es en principio tan fácil o tan difícil en C++ como en C, especialmente cuando se usan las funciones dedicadas de C para este efecto, como xmalloc() y xrealloc(). El manejo de memoria en C o en C++ puede ser codificado de manera "elegante' o "fea" (o algo intermedio), eso depende del programador y no del lenguaje. Otra vez, en C++ ya existen formas elegantes para el manejo de memoria que pueden aprovecharse.

Concerniente a los errores, C ofrece algunas herramientas. Por ejemplo, donde quiera que sea posible, variables locales o static pueden usarse y tipos de datos especiales tales como structs pueden ser manipulados por funciones especiales. Usando tales técnicas el ocultamiento de datos puede ser realizado aún en C; aunque es necesario

decir que C++ ofrece construcciones sintácticas especiales. En contraste, los programadores que prefieren usar variables globales muy probablemente no se verán beneficiados por el concepto de ocultamiento de la información, ya sea en C o en C++. Concluyendo, C++ en particular y la POO en general no son soluciones para todos los problemas de programación. C++, sin embargo, ofrece sintaxis elegante y muchas de sus herramientas continúan siendo estudiadas.

2.6 Características ventajas y desventajas de QT

QT es una aplicación multiplataforma que permite a los desarrolladores escribir una aplicación que correrá nativamente en Linux / Unix, windows, Mac OS X y se encuentra embebido en Linux con una simple recopilación, es una aplicación muy fácil de utilizar para cualquier programador de C++, Qt es soportada por los siguientes sistemas:

- AIX - 4.1 o superior
- BSD/OS - 2.0 o superior
- DG/UX
- FreeBSD - 2.1 o superior
- HP-UX - 10.20 o superior
- IRIX - 6.x
- Linux
- Mac OS X
- NetBSD
- OpenBSD
- OpenServer
- QNX
- Reliant Unix
- Solaris - 2.5.1 o superior
- Tru64 Unix - 4.0 o superior

- UnixWare
- Windows 95
- Windows 98 and Me
- Windows NT (4.0 or later), 2000, and XP

Como puede verse debido a que la programación de las interfaces es con C++ permite que el sistema sea multiplataforma, además de darle una gran potencia y velocidad. Qt tiene una gran ventaja para aplicaciones no lucrativas (por ejemplo para sistemas de investigación y desarrollo de la UNAM), esta ventaja consiste en el uso gratuito de la herramienta para sistemas Unix/X11 macintosh y Linux, esta versión gratuita es para desarrolladores open source (código fuente libre), mientras que existen dos versiones comerciales de este sistema.

Qt incluye varios Widgets (controles en terminología windows) que proporcionan una funcionalidad estándar para el desarrollo de Interfaces gráficas de Usuario, Qt introduce una alternativa que es innovadora para comunicación entre objetos, llamadas señales, las cuales remplazan las viejas callbacks. Qt tiene un modelo convencional de eventos para el manejo del mouse, teclas, etc.

Qt tiene un buen soporte para uso de gráficas 2D y 3D, también puede crear aplicaciones utilizando sistemas de bases de datos estándar e incluye drivers para:

- Oracle
- Microsoft SQL Server
- Sybase Adaptive Server
- PostgreSQL
- MySQL

Qt utiliza unicode y también tiene un importante soporte para la internacionalización incluye Qt Linguist y otras herramientas para soportar traducciones, las aplicaciones

pueden mezclar y usar texto en árabe, Chino, inglés, hebreo, japonés, ruso y español entre otros idiomas soportados por unicode.

Qt tiene una variedad de clases, por ejemplo tiene un módulo de XML que incluye parsers para SAX y DOM, los objetos pueden ser almacenados en memoria utilizando la colección de clases STL de Silicon Graphics. Clases para el manejo de archivos remotos y locales, así como para comunicaciones en red estándar.

La creación de interfaces gráficas dentro de KDevelop es un proceso sencillo gracias a QT Designer, figura 2.6.1 la herramienta creada por trolltech, Qt designer sirve como entorno de desarrollo de interfaces gráficas, Esta herramienta nos permite diseñar una interfaz de forma visual.

Figura 2.6.1 Qt designer

La creación de un nuevo widget o una nueva interfaz se realiza mediante un asistente que facilita la tarea. Se puede acceder a este asistente desde el menú *Archivo->Nuevo*. Disponemos de una paleta de widgets muy completa, que incluyen los widgets más comunes de las librerías QT figura 2.6.2. Si además hemos instalado las librerías para el desarrollo de aplicaciones KDE, tendremos widgets adicionales.

Figura 2.6.2 Tipos de proyectos y wizards

El funcionamiento es de estilo *Selecciona y dibuja*, es decir, basta con seleccionar un tipo de widget en la paleta y luego al colocarlo sobre el formulario se dibuja con la geometría que queramos. Las propiedades de un widget cualquiera se pueden cambiar fácilmente en tiempo de diseño con el panel de propiedades figura 2.6.3.

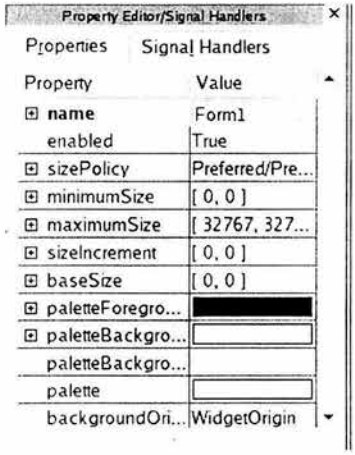


Figura 2.6.3 Barra de propiedades

A medida que guardamos los cambios de la interfaz, si editamos el fichero .ui vemos que va cambiando el contenido. La descripción de la interfaz se guarda en ese fichero en formato xml. Cuando regresemos al KDevelop a la hora de compilar, el propio KDevelop se encarga de generar a partir del *.ui, los ficheros correspondientes *.cpp y *.h. Así por ejemplo, si el fichero se llamaba MainWindow.ui se generan MainWindow.cpp y MainWindow.h

Para hacer que nuestra ventana sea la ventana principal de la aplicación basta con hacer que el widget que nos generó como ventana principal herede del nuevo que hemos creado con el QT Designer.

El desarrollo de interfaces gráficas con qt tiene las siguientes ventajas:

- Acceso al código fuente de las herramientas y bibliotecas con las que se trabaja.
- Gratuidad de estas mismas herramientas y bibliotecas.
- Ayuda por parte de la comunidad de desarrollo de software Libre.
- Disponibilidad de herramientas de tratamiento de ficheros de texto, como sed, y tar, entre otras. muy útiles en muchas ocasiones.

CAPÍTULO

III

PLANTEAMIENTO DEL PROBLEMA Y PROPUESTA DE SOLUCIÓN

3.1 Planteamiento del problema.

3.1.1 Identificación del problema

La extracción de isosuperficies es una importante herramienta para la visualización, han existido numerosas técnicas para mostrarlas, investigadores de muchas ciencias y campos de la ingeniería obtienen datos volumétricos ya sea a partir de sensores o también de simulaciones, estas representan información importante. Los métodos de presentación para un análisis de datos eficiente puede guiar a los investigadores a descubrir nuevas características contenidas en estos datos. Las isosuperficies proporcionan un mecanismo para entender la estructura de los campos escalares.

Los contornos de las isosuperficies nos permiten descubrir regiones de interés, figura 3.1.1.1, enfocando la atención en características importantes en la información tridimensional, como ya se ha visto muchas disciplinas como: la medicina, vulcanología, química, etc. han utilizado estas técnicas de una forma efectiva.

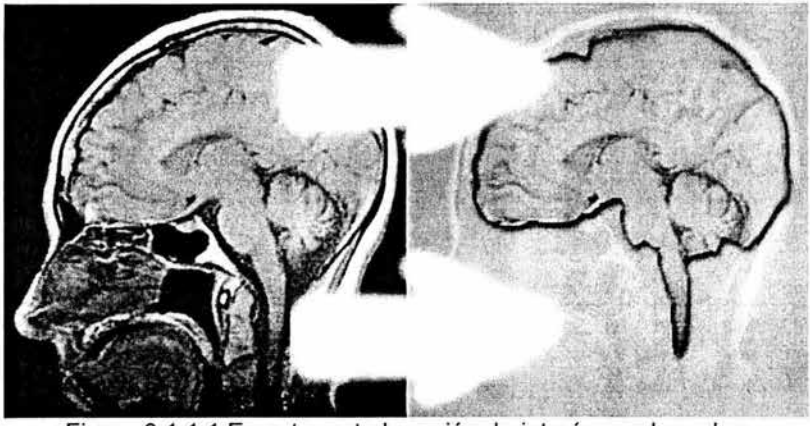


Figura 3.1.1.1 En este corte la región de interés es el cerebro

Los algoritmos que utilizan estructuras de datos para acelerar la extracción de isosuperficies, generalmente no varían mucho del algoritmo original de Marching Cube, entre más eficiente sea el método permiten una mejor interacción entre los investigadores y sus datos para un mejor entendimiento de estos.

Existen varias herramientas para visualizar datos volumétricos desarrolladas por diversas empresas, pero estas tienen un costo a nuestra universidad de varios miles de dólares en ocasiones, es por ello que se debe buscar sustituir estas herramientas por sistemas desarrollados en nuestra universidad, ya es común el uso y generación de software libre, en instituciones de enseñanza superior como la nuestra la aplicación de software libre, permite abatir importantes costos y con ello utilizar estos recursos en otro tipo de infraestructura; la compra de licencias de software no es el costo único que se tiene que cargar al presupuesto, también cabe destacar que el soporte técnico que podría generar el uso de las herramientas comerciales es considerable mientras que en el caso de que el desarrollo sea local, permite adaptar el sistema a las necesidades propias de quien las utiliza (investigadores, médicos, químicos, etc.). El desarrollo de este sistema pretende ser una contribución para el soporte de un importante sistema de visualización adaptado a las necesidades de nuestra institución.

3.1.2 Consideraciones generales

La principal base del presente proyecto viene del concepto de imagen digital, la cual es un dominio finito en dos dimensiones. El proceso para la reconstrucción podría dividirse en las siguientes etapas:

- Generación de las imágenes, estas pueden ser generadas de muchas formas, como puede ser a través del proceso de una investigación de algún científico, el cual al desarrollar esta puede generar simulaciones, y con ellas imágenes consecutivas, por otro lado existen sensores que permiten la

lectura de cuerpos tridimensionales en donde las lecturas de estos cuerpos se convierten en una secuencia de imágenes consecutivas (pueden ser construidas estas por cortes físicos continuos), La lectura de los datos anteriores y su almacenamiento en memoria, como ya se ha mencionado, permite el uso de estas herramientas de obtención de información que son muy importantes para la generación de las isosuperficies (representación tridimensional por computadora de los diferentes fenómenos físicos) la figura 3.1.2.1 puede ser un ejemplo claro de la generación de imágenes para la visualización de las isosuperficies.

Figura 3.1.2.1 Conjunto de cortes de una vértebra

- Identificación y clasificación de estructuras contenidas en el volumen de datos. Esta identificación es un estudio estadístico que se realiza a la isosuperficie en base a la distribución de los datos y pueden conocerse valores que permiten distinguir rasgos estructurales importantes para la generación de la representación tridimensional de la fuente de datos.

En el caso de la presente aplicación se lleva a cabo un estudio del histograma que nos permite por medio de preservación de momentos obtener algunos valores que pueden servir como base para realizar una discriminación de los datos leídos. Esta no será la única forma para que se genere la isosuperficie, el usuario podrá elegir el valor a partir del cual se llevará a cabo la selección y discriminación de la información figura 3.1.2.2.

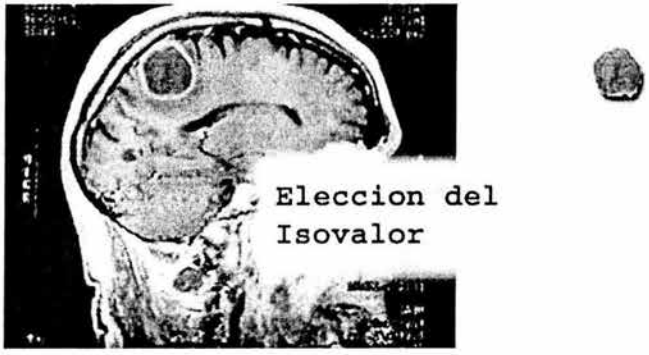


Figura 3.1.2.2 Con la elección de un isovalor adecuado pueden distinguirse rasgos importantes en las isosuperficie.

- Generación de los polígonos que conformarán la isosuperficie. Una vez hecha la elección del isovalor el siguiente paso es la eliminación de los datos que no importen y después comenzar a obtener los valores de los puntos en el espacio de interés para poder continuar con el siguiente paso.

- Rendering En este último paso, se genera el despliegue de la aplicación en el canvas[4] En este paso se lleva a cabo la generación de las primitivas que en conjunto generarán le isosuperficie, se formará triángulo a triángulo el despliegue de la representación tridimensional de la isosuperficie.

[4] Área de dibujo en una aplicación.

Estos son los pasos que serán la base para el desarrollo del sistema, aunque existen muchos factores que pueden hacer variar alguno u algunos de los pasos anteriores.

En la figura 3.1.2.3 puede verse el flujo de datos principal, esta es el diagrama raíz a partir del cual todo el procesamiento se realiza, en el puede verse a grandes rasgos el flujo de la información, pero esta gráfica es muy general. Más adelante se irán desglosando los diferentes niveles de procesamiento y transmisión.

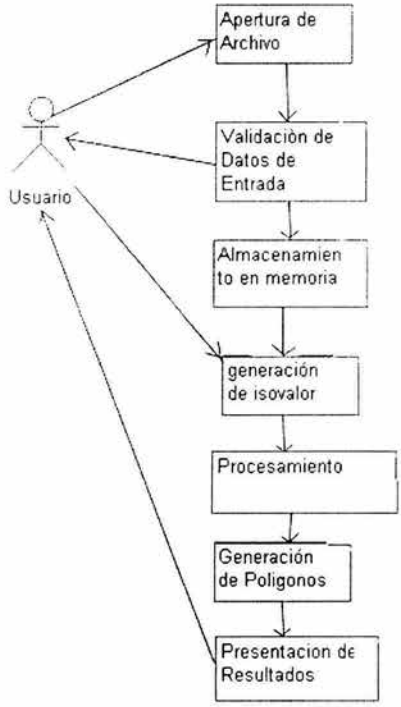


Figura 3.1.2.3 Flujo de datos en el sistema

En primer lugar se realiza la solicitud por parte del cliente, el usuario lleva a cabo una conexión remota, se configuran las variables de ambiente correspondiente y se

realiza el despliegue remoto de la aplicación hospedada en el servidor ver figura 3.1.2.4.

Figura 3.1.2.4 Solicitud al servidor para el despliegue

El paso siguiente es llevar a cabo las solicitudes de procesamiento requeridas, el usuario elige el conjunto de cortes a partir de los cuales se crearán las isosuperficies, una vez hecho esto se elige el isovalor del cual se generará el despliegue el usuario tiene la posibilidad de utilizar el isovalor generado por la aplicación o introducir su propio isovalor figura 3.1.2.5.

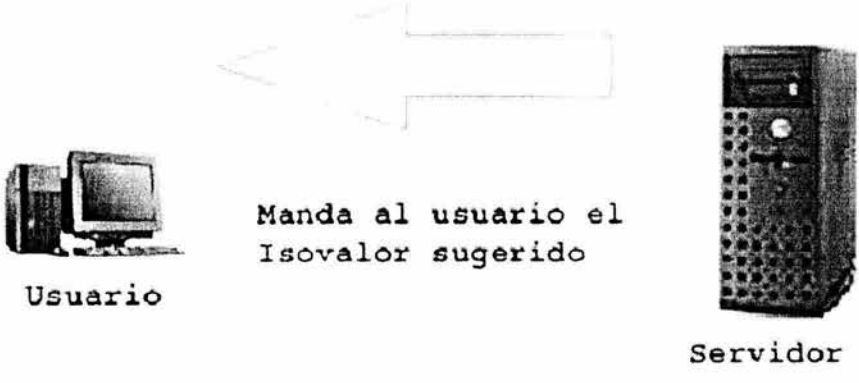


Figura 3.1.2.5 Selección del Isovalor

Una vez hecha la selección será procesada toda la información en este caso primero es llevar a cabo la lectura y el almacenamiento en la memoria, todos los cortes serán almacenados en un arreglo tridimensional (como se mencionó anteriormente) ver figura 3.1.2.6, al mismo tiempo que se genera el almacenamiento de los datos, en donde se obtendrá el histograma.

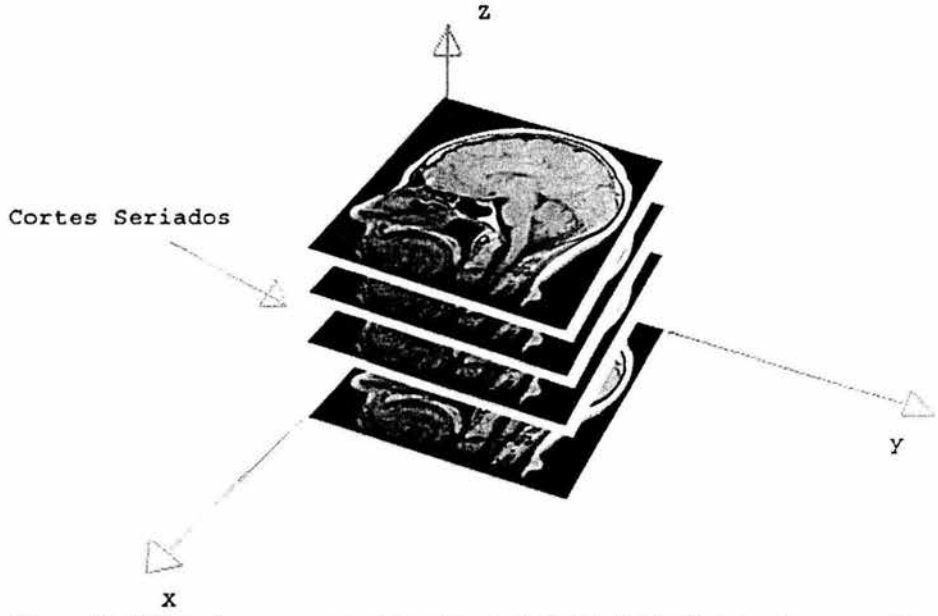


Figura 3.1.2.6 Imágenes que representan un "cubo" y las imágenes almacenadas en el arreglo tridimensional.

La idea de obtener el histograma es para generar una sugerencia de isovalor a partir de diferentes datos estadísticos de acuerdo a la distribución que presentan los cortes, este análisis se llevará a cabo siguiendo el método de momentos y a través de estructuras de datos para la resolución de los sistemas de ecuaciones y obtención de raíces de las mismas, el isovalor obtenido debe ser una buena sugerencia para la representación y generación de la isosuperficie.

Después de la elección del isovalor el siguiente paso es generar los puntos por donde pasa este, se hará uso de la estructura de datos para estos fines y al mismo tiempo se hace uso de las librerías gráficas, utilizadas principalmente para presentación de gráficos tridimensionales.

La librería gráfica OpenGL es por naturaleza una librería orientada al trabajo con modelos basados en polígonos, por tanto será fácil realizar una representación de esta naturaleza. Puntualizaremos que se trata de una librería de funciones orientada principalmente a modelos interactivos, por ello se premia la rapidez frente al espacio, el tipo de representación poligonal que empleara será por tanto explícita. Las definiciones de primitivas poligonales en OpenGL se encierran entre las llamadas a las funciones: ***glBegin***(GLEnum tipo_primitiva) y ***glEnd***(void). Entre dichas funciones deberemos especificar la lista de vértices que componen nuestro polígono. La función para pasar las coordenadas de cada vértice es ***glVertex3fv***(GLFloat *coor), donde 'coor' es un vector que contiene las tres coordenadas del vértice, tabla 3.1.1.

Los valores normales para el tipo de primitiva son las constantes:

Valor de la Cte GL	Tipo de Primitiva Poligonal
GL_POINTS	Puntos aislados
GL_LINES	Líneas de dos vértices
GL_LINE_STRIP	Línea de cualquier numero de vértices
GL_LINE_LOOP	Línea Cerrada.
GL_POLYGON	Polígono de Cualquier tipo
GL_TRIANGLES	Polígonos de tres lados
GL_TRIANGLE_STRIP	Tira de Triangulos
GL_QUADS	Polígonos de cuatro vertices
GL_QUAD_STRIP	Tira de Cuadrilateros.
GL_TRIANGLE_FAN	Abanico de triangulos.

Tabla 3.1.1

Por ejemplo la definición de un triangulo en OpenGL seria:

```
float mitrian[3][3]={{0,0,0},{1,1,0},{2,0,0}};
/* ....*/
        glBegin(GL_TRIANGLES);
                glVertex3fv(mitrian[0]);
                glVertex3fv(mitrian[1]);
                glVertex3fv(mitrian[2]);
        glEnd();
/* ....*/
```

en este caso se hará uso de vertex para generar los polígonos de la superficie, por ello los valores que maneja esta función son importantes en la figura 3.1.2.5

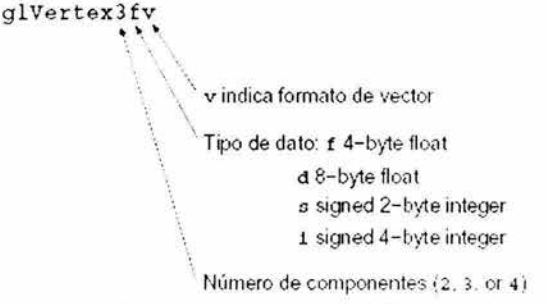


Figura 3.1.2.7 valores que genera la función glVertex

3.2 Descripción de la lectura de datos.

La creación de isosuperficies, inicia a partir de la lectura de datos, estos vienen en un formato determinado (imágenes secuenciales) para la generación del volumen y poder generar la representación de estas imágenes en formato tridimensional, el primer paso para el almacenamiento es la lectura de las imágenes que tendrán el mismo ancho y el mismo alto, el volumen lo determinará el numero de cortes, como ya se ha mencionado anteriormente una imagen es un arreglo de valores que representan los pixeles de esta; cada valor de píxel es en realidad uno o más números (depende de la profundidad) en este caso la lectura será basada en imágenes en tonos de grises, estos arreglos tendrán valores entre 0 y 255 . la matriz que contiene esta información serán los cortes ver figura 3.2.1.

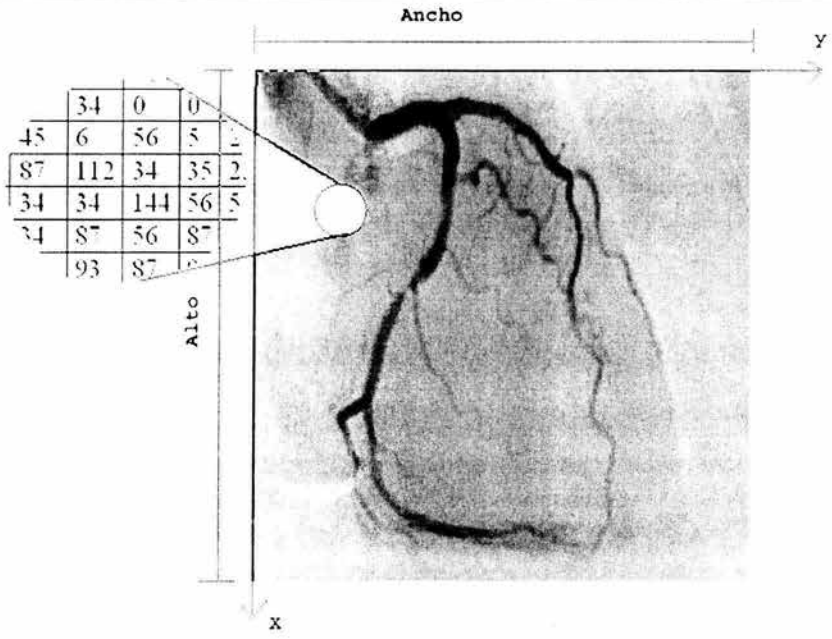


Figura 3.2.1 La imagen es una matriz de números entre 0 y 255

Existirá un archivo de cabecera que servirá para que el programa pueda leer las constantes que serán manipuladas a través del proceso, como lo es el "ancho", "alto" y profundidad (numero de cortes) una vez leído esto se procederá a leer un archivo que contiene todos los valores ordenados de los diferentes cortes. El primer paso para el almacenamiento es conocer las dimensiones de los datos, en este caso el ancho, alto y la cantidad de imágenes que representan los cortes nos proporcionarán ese dato.

Una vez que se tiene el tamaño de las imágenes y la cantidad de estas, podemos preparar el arreglo para almacenar los datos, se necesita reservar la memoria para guardar el volumen de datos, este requerirá que la memoria de reserva sea el producto de ancho x alto x profundidad, es decir:

$$mem = float(w)(h)(d)$$

$$w = ancho\ de\ las\ imagen$$

h = alto de las imágenes

d = profundidad (número de imágenes)

el algoritmo de lectura será el siguiente:

```
Hacer i = j = k = 0
    Mientras k < numero de imágenes, k++{
        Mientas j < alto, j++{
            Mientras i < ancho, i++{
                Leer y guardar  Data[ i ][ j ][ k ]
            }
        }
    }
```

al tener la información en un formato tridimensional almacenamos los datos en un arreglo dinámico tridimensional en donde, cada píxel tendrá una posición en el espacio computacional con coordenadas en 3D (x, y, z), estos valores serán representados por la triada (i, j, k).

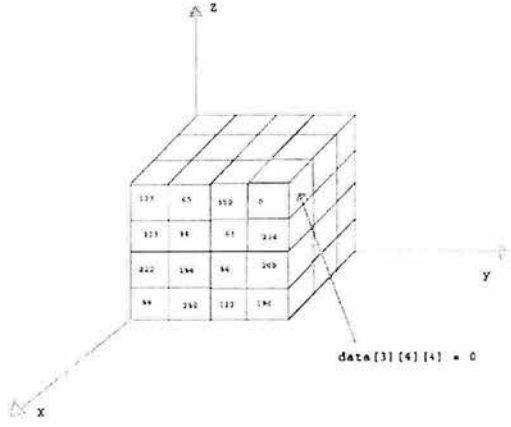


Figura 3.2.2 Representación de almacenamiento de los voxeles

En la figura 3.2.2 puede apreciarse que cada voxel tendrá su propia celda en el arreglo que se tendrá en memoria, esto permitirá que no se pierda la posición de cada uno de estos voxeles en el espacio.

A partir de estos datos se realizará la clasificación de los mismos para localizar la isosuperficie, el siguiente paso es la generación de los puntos a partir de los cuales se generarán los polígonos (triángulos). Para no saturar excesivamente la memoria estos puntos serán desplegados al momento de ser generados y no serán almacenados, en el caso de que se desee volver a generar la isosuperficie con dicho valor, deberán recalcularse los puntos.

3.3 Requerimientos del sistema.

3.3.1 Generales.

Software

Este sistema será desarrollado para sistemas Unix y similares (Linux), para que pueda ser desarrollada esta aplicación es necesario tener el entorno de desarrollo Qt designer, en este se creará la interfaz gráfica de usuario IGU o GUI por sus siglas en inglés (Graphic User Interface) para el desarrollo de esta se deben tener en cuenta algunos puntos para el diseño de una buena interfaz :

- Optimizar los factores humanos y la ergonometría, esto se refiere a hacerla confortable y atractiva.
- Maximizar la velocidad de uso.
- Minimizar la posibilidad de error.
- Tener un recall[5] rápido.
- Tener consistencia con lo que el usuario espera.

Cabe destacar que todos estos puntos no pueden ser incluidos necesariamente en el diseño de una interfaz para una aplicación, para esto debe ser necesario

[5] Este se refiere al recuerdo del uso del sistema por parte del usuario

determinar los objetivos más importantes para el usuario y el propósito de la aplicación.

Para el diseño de la interfaz se tomaran en cuenta dos tipos de lenguajes:

- Usuario – Computadora, este se expresa vía acciones aplicadas a los distintos dispositivos de interacción.

- Computadora – Usuario este lenguaje es expresado gráficamente por medio de líneas, cadenas de caracteres, áreas y colores combinados formando imágenes y mensajes.

Resulta recomendable hacer uso de los diagramas de estado para representar los eventos que se manejarán en el sistema, por ejemplo la figura 3.3.1.1 muestra una representación del manejo del evento de mouse.

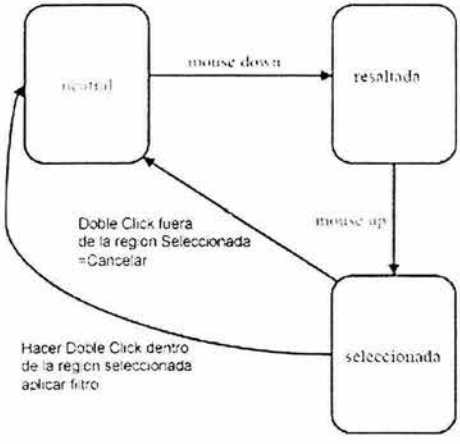


Figura 3.3.1.1 Representación de los diagramas de estado para el diseño de interfaces.

El sistema hará uso del estilo de manipulación Directa, este describe sistemas que poseen las siguientes características.

- Visibilidad de los objetos de interés
- Rapidez, reversibilidad, acciones incrementales
- Reemplazo de lenguajes de comandos de sintaxis complejas por manipulación directa de los objetos de interés.

También resulta importante tomar en cuenta la consistencia, se dice que un sistema es consistente si el modelo conceptual, la funcionalidad y la secuencia son acordes con lo que se desea realizar.
Como ejemplos de consistencia podemos tener los siguientes:

- La misma codificación de color se usa de la misma manera.
- Los ítems de menú se muestran siempre en la misma posición relativa.
- Determinados caracteres del teclado tienen siempre la misma función.
- Se proveen comandos genérico (Move, Copy, ...) y pueden ser aplicados, con resultados predecibles sobre cualquier tipo de objeto en el sistema.

Algo más que se debe tener en cuenta es minimizar en cuanto se pueda las posibilidades de error, no se debe permitir que el usuario cometa errores. Por ejemplo el usuario no debe introducir letras en donde se deben meter números y viceversa.

El uso de qt designer no es el lo único que será necesario para el desarrollo también es necesario el uso de un compilador de c++, por ejemplo el CC de silicon graphics o el g++ de Linux, como ya se ha señalado el proyecto será desarrollado en c++ completamente. Por otro lado para la presentación tridimensional se realizará por medio de las librerías gráficas de OpenGL, estas pueden ser descargadas

gratuitamente de la página http://www.mesa.org, el paso siguiente es configurar las variables de ambiente para la compilación y el desarrollo.

El sistema operativo manejará la transmisión del despliegue remoto a través de una conexión de secure shell, y el despliegue se configura por medio de la variable DISPLAY.

Hardware

Debido a la gran cantidad de polígonos que serán generados es necesario pensar en un sistema potente y con una tarjeta gráfica poderosa.

- **Servidor** este debe ser más potente que el cliente y se recomienda lo siguiente:
  - Pentium 4 a 2.6 Ghz. , o Superior
  - Memoria RAM 512 o Superior
  - Disco duro lo suficientemente grande para el almacenamiento de los volúmenes de datos.
  - Mouse
  - Teclado
  - Tarjeta de red ethernet 10/100

- **Cliente**
  - Procesador Pentium III o superior
  - Memoria RAM 256 o superior
  - Mouse
  - Teclado
  - Tarjeta de red ethernet 10/100

3.3.2 Requerimientos Particulares.

Como características particulares pueden ser señaladas las referentes a la forma en que será manipulado el despliegue o el control de la isosuperficie, como bien puede ser:

- Abrir gráficamente el archivo
- Abrir una pantalla para elección del isovalor
- Rotación
- Traslación
- Zoom
- Texturización
- Posibilidad de realizar un nuevo rendering con otro isovalor

3.4 Comparación con otras herramientas.

En el mercado de software existen diversas aplicaciones para la generación de isosuperficies, estas tienen funciones, arquitecturas, diferentes visiones entre estas tenemos:

- **VGStudio MAX**

Es un sistema para el análisis y visualización de voxeles, este tiene la característica de manejar conjuntos de datos de gran tamaño, por ejemplo de modernos

escáners, conjunto de información sísmica, de la exploración de gas y petroleo y de otros sistemas de producción de imágenes 3D. Esta es una herramienta interactiva de rendering y procesamiento en tiempo real de imágenes 3D de varios Gbytes de tamaño, entre las características de este sistema se encuentran:

- Render y manipulacion de voxeles de datos 3D.
- Importa series de cortes en formato 2D para crear un volumen.
- Soporta una gran cantidad de formatos como RAW, BMP, JPEG, TIF, DICOM, entre otros.
- Importa archivos de volumen.
- Crea render de datos volumétricos, proyecciones de máxima intensidad, rayos x.
- Permite ver cortes arbitrarios del conjunto de datos (axial, sagital, frontal).
- Render de varios volúmenes juntos.
- Variación de la opacidad y color en cortes y volúmenes.
- Permite guardar escenas y volúmenes de datos en un archivo binario.
- Almacena imágenes rendereadas y cortes.
- Un rápido render de datos volumétricos.
- Procesamiento de imágenes 3D.
- Procesa más de 1 GB de datos en una PC normal.
- Procesa varios GB con procesadores poderosos como el Procesador Itanium de Intel.
- Soporte para multiprocesadores

- **IBM Visualization Data Explorer**

Data explorer ver figura 3.4.1 es un sistema de herramientas e interfaces de visualización de datos. En términos generales la visualización de datos pueden ser consideradas dentro de tres etapas:

- Descripción e importación de datos.

- Procesamiento de datos a través de un programa de visualización.
- Presentación de resultados.

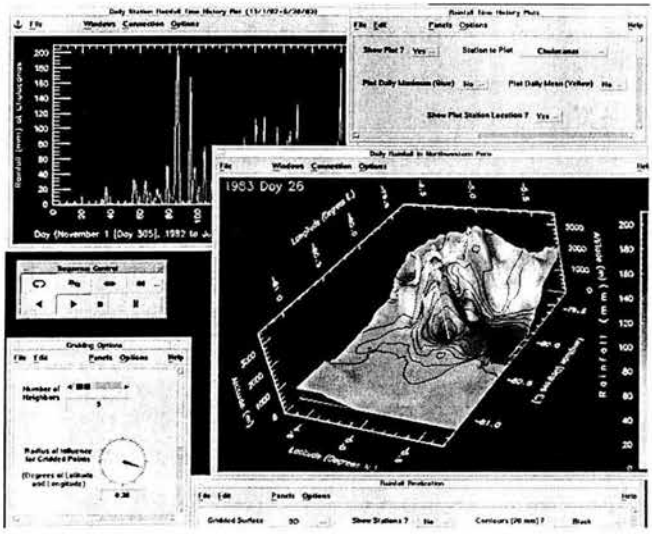


Figura 3.4.1 Data Explorer pantalla de muestra

Las herramientas e interfaces están posicionados para mostrar su proceso general de visualización. Para la generación de la visualización es necesario pasar por los siguientes pasos:

- Modelo de Datos. Es un conjunto de definiciones, reglas y convenciones usadas para describir entidades de data explorer (incluye campos de datos, objetos geométricos e imágenes).
- Datos Puntuales, Interfaz de usuario para la descripción de datos para ser importados al Data Explorer.
- Navegador de Datos. Interfaz para ver el archivo de datos. Determinando las capas y la organización de datos, transfiriendo su información al punto anterior.
- Un simple lenguaje para creación de programas de visualización, también puede ser usado en un modo de comando para ejecutar varias tareas.

- VPE Visual program editor (editor visual del programa). Interfaz gráfica para la creación y modificación de programas visuales. Los programas generados con este editor se traducen a un lenguaje de script por el data explorer y se almacenan de esa forma.
- Diagramas de Bloques que constituye un programa de red.
- Ventana de Imagen. Ventana interactiva para ver y modificar, la presentación de la imagen producida por un programa visual.

El sistema esta desarrollado para trabajar en los siguientes sistemas

- AIX
- IBM
- IBM Power Visualization System
- RISC System/6000
- IRIX

- **IsoSurf**

Este sistema genera triangulación de superficies se desarrolla en una resolución definida por el usuario, utilizando el método de marchig tetrahedra si su resolución es muy grande se interpolan los planos, para este sistema entre más rápido es el despliegue resulta menos realista.

Este sistema funciona en base a parámetros desde la línea de comandos, y genera información de la triangulación (los valores de los vértices de los triángulos), la figura 3.4.1, muestra una isosuperficie generada por este sistema.

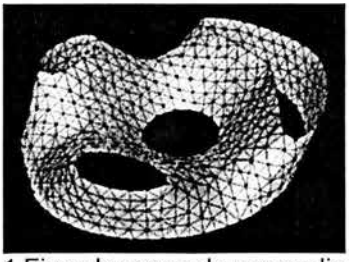


Figura 3.4.1 Ejemplo generado por medio de IsoSurf

- **TAn (Tetrahedra Analizer)** Analizador de tetraedros

Este es un sistema de visualización y modelado para datos volumétricos, ver figura 3.4.2, este sistema soporta modelado multiresolución y visualización de conjuntos de datos por medio de la descomposición por tetraedros, este tiene las siguientes herramientas:

- Crea una representación multiresolución basada en celdas simples.
- Define funciones de transferencia interactivas.
- Visualiza datos soportando múltiples técnicas de render, como ajuste de isosuperficie, volume render directo, así como un híbrido de Isosuperficies y volume rendering.
- Se pueden elegir dos niveles de detalle (alto y bajo).

El sistema esta hecho para ser utilizado en sistemas Unix y fue compilado en una estación IRIX.

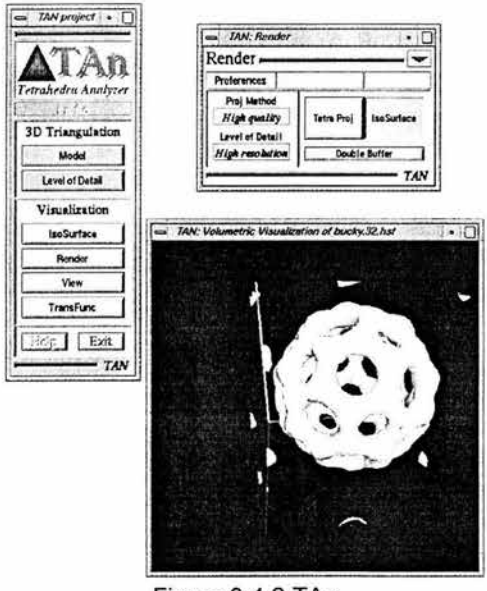


Figura 3.4.2 TAn

- **Visual MODFLOW 3D Explorer**

El sistema esta construido para desplegar y animar modelos, este tiene las siguientes características:

- Crea y salva animaciones en un formato estándar AVI
- Ajusta el ángulo y la fuente de Luz desde cualquier dirección.
- Permite colocar etiquetas en los contornos.

Este permite crear despliegue de múltiples isosuperficies para diferentes concentraciones de elementos, en cada caso, La isusuperficie tendrá las siguientes propiedades:

- definirá los nombres de la superficie que aparecerá en un modelo de árbol.
- Valor de isosuperficie.
- Color de la isosuperficie.
- Una paleta de color será colocada de acuerdo a un valor de isosuperficie.
- Semi transparencia que permite ver a través de la isosuperficie.

- **Isosurface Visualizer**

Es una herramienta de visualización 3D que permite a los usuarios observar datos de una forma simple, se tiene un conjunto de datos en un simple arreglo de datos de tipo flotante en 3D, los índices de este arreglo esta ordenado en x, y, z. La variable de cada nodo determina por donde pasa la isosuperficie.

Las características de este sistema son:

- Controlar la orientación de la geometría incluyendo la translación, rotación y escalamiento.

- Control de los parámetros de las fuentes de luz incluyendo arbitrariamente x, y, z y fuentes de luz de color RGB.
- Especificación de un material que incluye ambiente, material difuso y especular.
- No necesita un hardware caro para el render de datos 3D

Requerimientos del sistema:

Para correr las isosuperficies son necesarios:

- Una estación de trabajo Silicon Graphics 4D con al menos 24 bits de profundidad de color.
- Hardware con Z buffer de 8 MB o más.
- El sistema fue desarrollado en un sistema IRIX 4.5.

Los paquetes anteriores presentan varias características importantes para el diseño de una herramienta de visualización de datos volumétricos, en este caso muchas de estas tienen características para la visualización de isosuperficies, algunas de ellas provenientes de compañías tan importantes como IBM, presentan varios enfoques para la presentación de datos entre los dos mejores se encuentran el Data Explorer y el VGStudio MAX, el primero desarrollado para sistemas Unix, mientras que VGStudio esta desarrollado principalmente para funcionar en plataformas Windows, estas herramientas ofrecen una forma eficiente de manejar los datos volumétricos.

3.5 Elección de la óptima.

La herramienta tendrá muchas posibilidades de interacción con la aplicación sobre todo al momento del despliegue, con ello se permitirá poder mover la isosuperficie aplicándole algunas funciones básicas como puede ser la traslación, rotación, zoom

cambio de texturización, muchas de las herramientas presentadas permiten hacer esto, pero se tienen básicamente los siguientes problemas:

- Formato de datos de entrada
- Sistema Operativo en donde funcionará el sistema
- Un tercer punto es el Costo que tiene el sistema

El sistema esta pensado para que funcione bajo el sistema operativo Unix y Linux, por lo cual reduce la posibilidad de uso de herramientas de visualización implementadas para windows, como se mencionó en los puntos anteriores el precio es un factor importante debido a que las licencias son caras y no permiten hacer modificaciones para adaptar el sistema a nuestras necesidades sino que nosotros debemos adaptarnos a las necesidades del sistema adquirido.

Debido a que el volumen de datos se almacena de muy diversas formas se necesita adaptar la lectura de estos a nuestros requerimientos, es por ello que al tener acceso al código fuente de nuestro sistema podemos modificarlo al tipo de información que se tenga, pensando que en el futuro pueda manejarse un estándar para el sistema de visualización.

La presente herramienta tendrá estas características adaptivas para futuros trabajos que se lleven a cabo con el, además su forma modular permitirá agregar más herramientas que pudieran necesitarse o agregarse este a un sistema mayor

CAPÍTULO IV

DESARROLLO E IMPLANTACIÓN DEL SISTEMA

4.1 Diagramación de la POO

4.1.1 Diagrama de Casos de Uso

El Modelado de Contexto es un diagrama que modela la relación del sistema con elementos externos ya que son estos los que forman el contexto del sistema, en este sistema es importante identificar a los actores involucrados, organizar estos y especificar sus vías de comunicación con el sistema

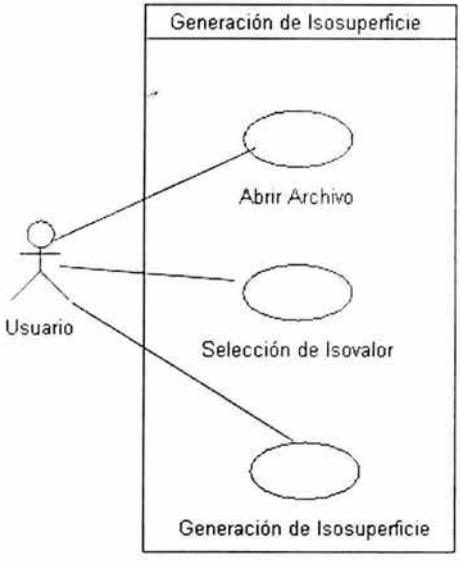


Figura 4.1.1.1 Diagrama de modelado de contexto

En el diagrama de contexto, ver figura 4.1.1.1 se puede ver que el usuario se relaciona básicamente en tres acciones con el sistema de visualización:

- Apertura del archivo que contiene la información volumétrica

- El usuario selecciona el isovalor para seleccionar el contorno
- EL usuario interactúa con la imagen generada

Modelado de requisitos. La función principal, o la mas conocida del diagrama de casos de uso es documentar los requisitos del sistema, o de una parte de el, Los requisitos establecen un contrato entre el sistema y su exterior, definen lo que se espera que realice el sistema, sin definir su funcionamiento interno. Es el paso siguiente al modelado del contexto, no indica relaciones entre autores, sólo indica cuales deben ser las funcionalidades (requisitos) del sistema. Aquí se incorporan los casos de usos necesarios que no son visibles desde los usuarios del sistema.

Para modelar los requisitos se recomienda establecer un contexto, para lo que también podemos usar un diagrama de casos de uso, es importante Identificar las necesidades de los elementos del contexto, identificar que casos de uso pueden ser especializaciones de otros, o buscar especializaciones comunes para los casos de uso ya encontrados.

En la figura 4.1.1.2 puede verse otro diagrama de caso de uso del presente sistema, el diagrama de modelado de requisitos, en este se indican las acciones que llevará a cabo el sistema, desde la lectura de datos, la validación de los mismos, el almacenamiento en memoria, procesamiento de datos, la generación de las isosuperficies y la presentación.

Como ya se ha indicado en este diagrama se representan acciones internas del sistema y no existe interacción del usuario.

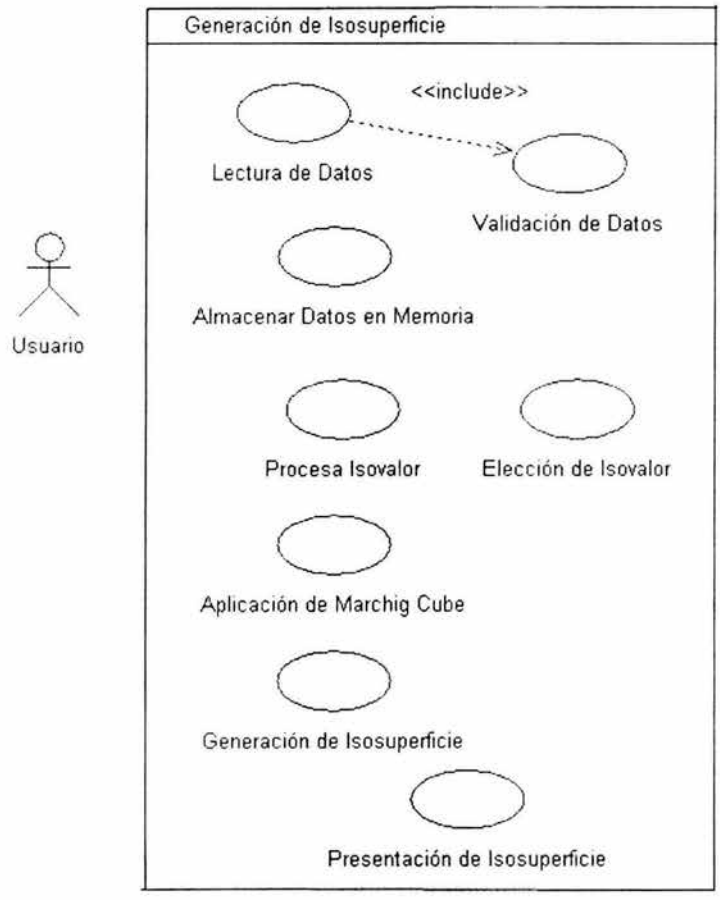


Figura 4.1.1.2 Diagrama de modelado de requisitos.

El flujo de este diagrama es top-down este es:

- Lectura de datos
- Validación de datos
- Almacenamiento en memoria
- Procesamiento del isovalor
- Obtención del isovalor

- Aplicación del algoritmo de Marching cubes
- Obtención de los puntos por los cuales existe el contorno
- Presentación visual de la isosuperficie.

Los puntos anteriores muestran los principales procesos que serán necesarios para el desarrollo del sistema en primer lugar en el diagrama de modelado de contexto en donde se indica la interacción del usuario con el sistema y posteriormente los procesos que serán necesarios para desarrollar la representación de las isosuperficies.

4.1.2 Diagrama de Clases

En el diagrama de clases se muestra la arquitectura general que se tendrá en el sistema, el cómo físicamente estará dividido todo el conjunto de clases, cómo interactuarán unos con otros para solucionar el problema planteado, en la figura 4.1.2.1 podemos ver las diferentes clases, la dependencia de estas y la clase raíz que llevara a cabo la sincronización de datos en el sistema.

La clase Interfaz se encargará de "abrir el archivo", en realidad este sólo obtendrá una cadena y esta posteriormente se transmitirá como parte del argumento a la clase MarchCubes, esta clase será la encargada de desplegar la ventana, mostrar la presentación del sistema a partir del cual se llevará a cabo toda la interacción con el sistema, si el usuario pretende abrir un archivo no válido por el sistema mostrará una caja de dialogo indicando el error. Una vez que ha sido validado el archivo de entrada se creará una instancia de la clase MarchCube.

Cuando se crea un objeto MarchCube lo primero es leer el encabezado del archivo de entrada, reconocer los datos necesarios para la extracción de los

datos volumétricos como son el ancho, alto y el número de cortes que contiene para inicializar las variables de widht, heigth y depth.

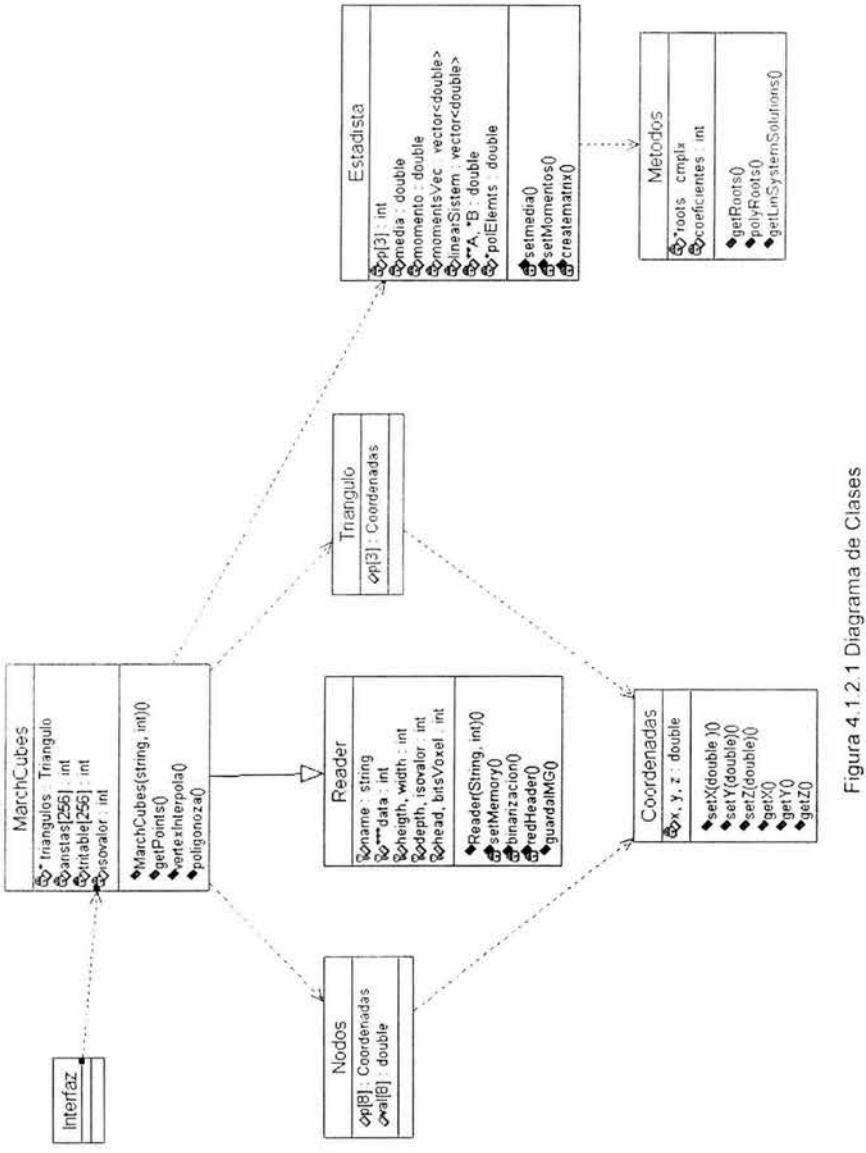


Figura 4.1.2.1 Diagrama de Clases

Una vez que se sabe el tamaño de las imágenes y el número de estas se prepara el arreglo para que se reserve la memoria dinámica en donde serán almacenados los datos, se crea una instancia de la clase Estadista, la cual será la encargada de generar un histograma a partir del cual se llevará a cabo un análisis de la distribución estadística de los datos, a través de esta instancia podremos obtener una sugerencia de isovalor que será mostrado al usuario como opción para la generación de la isosuperficie.

La instancia estadista hará uso de otra instancia que servirá para solucionar algunos problemas plateados por estadista, que consiste básicamente en la resolución de un par de sistemas de ecuaciones lineales y de obtención de las raíces de un polinomio generado por el método. Una vez que se obtiene el isovalor que se sugerirá se destruyen las instancias de los objetos creados (Estadista y Métodos), se regresa el control a Interfaz y este pasa el parámetro al objeto MarchCubes para comenzar a extraer la isosuperficie de los datos volumétricos almacenados en el arreglo tridimensional.

Cabe señalar que MarchCubes puede utilizar los métodos y atributos que tiene la clase Reader debido a la herencia que se señala en el diagrama.

En el momento que la instancia de MarchCubes obtiene los puntos de los triángulos que conformarán la isosuperficie también generará el rendering que será mostrada en pantalla.

Se tendrá la posibilidad de solicitar otro isovalor pero ya no será necesario solicitar nuevamente la apertura del archivo y tampoco será necesario sugerir el isovalor simplemente el usuario elegirá un valor diferente a partir del cual se limpiará la pantalla para mostrar otro isovalor a través de la ventana de despliegue de isosuperficie.

4.1.3 Diagrama de Componentes

Para el desarrollo del sistema serán necesarias librerías diferentes a las propias del lenguaje de C++ , estas son las librerías de Silicon Graphics: Stardard Template Library que consisten en un conjunto de estructuras de datos como pilas, colas, listas ligadas, etc. Para el almacenamiento de datos y objetos, también serán necesarias las librerías de Qt para el despliegue de la interfaz gráfica, esta contiene un gran número de widgets, y formas para la generación de interfaces y funcionalidades, para el rendering tridimensional serán utilizadas las librerías de OpenGL con las cuales en base a primitivas se podrán generar las isosuperficies tridimensionales, la figura 4.1.3.1 muestra el diagrama de componentes para el presente sistema.

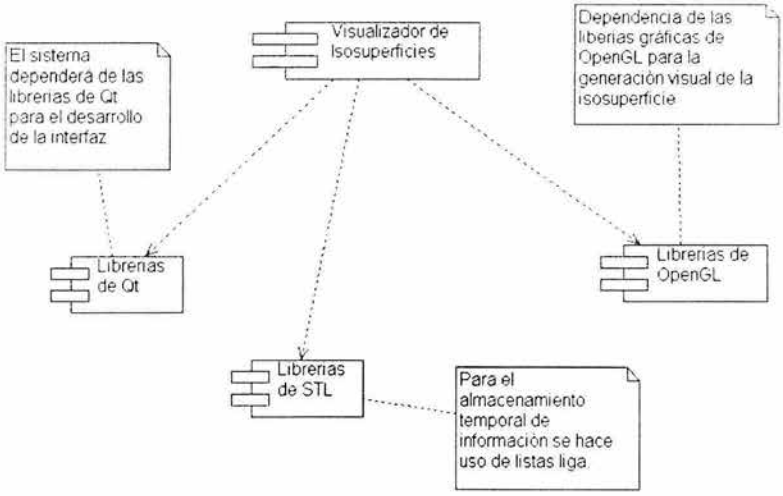


Figura 4.1.3.1 Diagrama de componentes

4.1.4 Diagrama de secuencia

El diagrama de secuencia, ver figura 4.1.4.1 forma parte del modelado dinámico

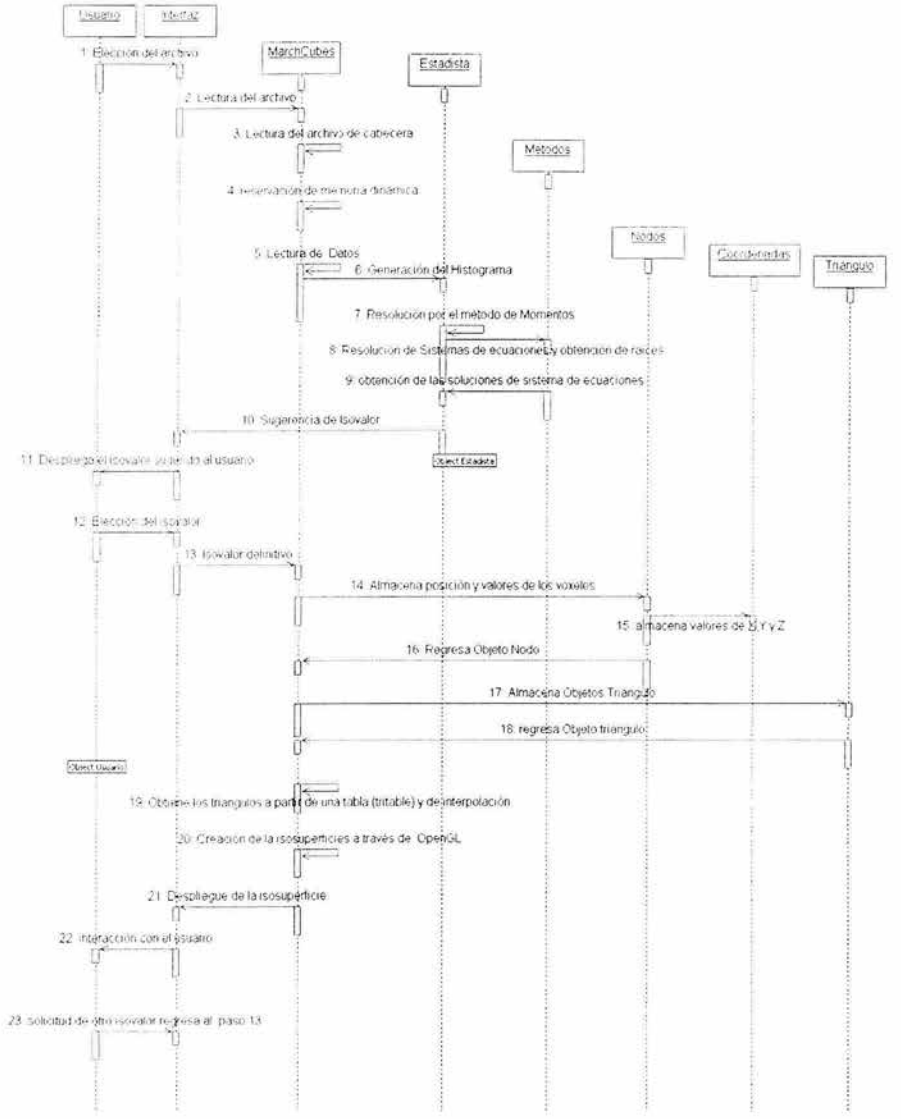


Figura 4.1.4.1 Diagrama de secuencia

del sistema, en este se modelan las llamadas entre clases desde un punto de vista concreto del sistema, este es útil para observar la vida de los objetos en el sistema, identificar llamadas a realizar o posibles errores del modelado estático, que pudieran imposibilitar el flujo de información o llamadas entre componentes del sistema.

El diagrama se forma con los objetos que forman parte de la secuencia, estos se colocan en la parte superior de la pantalla, normalmente en la izquierda se sitúa el objeto que inicia la acción de estos objetos sale una línea que indica su vida en el sistema.

En el diagrama de la figura 4.1.4.1 se puede apreciar que el objeto que inicia todo el proceso es el usuario el cual crea una instancia de la clase interfaz la cual a su vez desencadena el proceso que genera la obtención de resultados y termina con la obtención de resultados que serán desplegados en pantalla.

4.2 Desarrollo del Back-End

En esta sección se construirá la base del sistema que se encargará de la lectura de datos, generación del histograma, generación del isovalor, y generación de los puntos que conformarán las isosuperficies.

En los diagramas presentados anteriormente puede verse el diseño que conformará el sistema, en primer lugar se debe conocer como se encontrarán los datos y en donde serán obtenidos los valores que serán utilizados para todo el proceso.

Existirán dos archivos, uno con extensión hdr, el cual tendrá al menos la información mostrada en la tabla 4.3.1:

magic: 192837466
header length: 60
width: 256
height: 256
images: 109
bits/voxel: 8
Index bits: 0
scaleX: 1.000000
scaleY: 1.000000
scaleZ: 1.000000
rotX: 0.000000
rotY: 0.000000
rotZ: 0.000000

Figura 4.2.1 Posibles datos dentro del archivo hdr

en orden los datos presentados en la tabla anterior son:

- número mágico el cual es un número representativo del volumen de datos.
- header length que representará el numero de datos a ignorar al inicio de los datos volumétricos
- width representará el ancho de las imágenes
- height representará la altura de las imagines
- depth este dato nos proporcionará el número de cortes
- Bits/voxel representará los niveles de color (en el caso de que sea 8 quiere decir que serán 2^8 =256 niveles de grises)

El otro archivo que deberá encontrarse en el mismo directorio es el que tiene extensión vol, este precisamente contendrá los datos volumétricos a ser procesados.

La clase Reader es la que tiene los métodos necesarios para la lectura y almacenamiento en memoria, en primer lugar será elegido el archivo hdr, una vez hecho esto será almacenado el nombre del archivo en una variable propia de la clase, este almacenamiento será a través del método setName cuya firma es:

void setName(string)

al tener el nombre del archivo a abrir lo siguiente es leer y asignar a las variables las cantidades necesarias, dado que estamos utilizando c++ lo primero es abrir el archivo por medio de la creación de un objeto ifstream, una vez que se crea este objeto, se extraen los valores necesarios para guardarlos en las variables enteras headerLen, width, height, depth, bitsvoxel, una vez hecho esto ya se puede preparar la memoria para el almacenamiento, como se ha mencionado los datos serán almacenados en un arreglo dinámico tridimensional, la forma en que se reserva la memoria es la siguiente:

```
data= new int**[depth]

for(int i=0;i<depth;++i){
      data[i] = new int*[width];

      for(int j=0;j<height;++j)
            data[i][j]=new int[width];
}
```

Una vez que se haya reservado la memoria necesaria para el almacenamiento se puede continuar con la lectura del archivo que contiene los datos volumétricos (archivo con extensión vol)
este almacenamiento se realiza por medio del método setData de la clase Reader.

```
string datos=name+".vol";
ifstream volData(datos.c_str() ,ios::in);
volData.ignore(headerLen);

string tmp;
char tempo;
volData.unsetf(ios::skipws);
  for(int i=0;i<depth;++i){
    for(int j=0;j<height;++j){
      for(int k=0;k<width;++k){
          volData>>tempo;
         data[i][j][k]=static_cast<int>(tempo);
      }
    }
  }
  volData.setf(ios::skipws);
  volData.close();
```

con este método se almacenan en orden los datos dentro del arreglo tridimensional.

En la figura 4.1.2.1. puede verse que la clase Reader hereda a MarchCube sus propiedades, es por eso que esta última clase tiene acceso a los métodos protegidos y públicos de la clase Reader.

Para la generación de isovalores a sugerir de acuerdo a los datos y a la distribución de los mismos, para esto se utiliza el método de 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 (en nuestro caso es un conjunto de pixeles representados por números enteros de 0 a 255).

Los momentos tienen formas equivalentes al ser obtenidos a partir del muestreo de una señal, en este caso es una imagen y el valor de cada píxel, el primer paso para el cálculo de los momentos es obtener la media del conjunto de datos, la media esta definida por la sumatoria del conjunto de datos entre el número de elementos, como se describió en la ecuación 1 del capítulo 2, mientras que los momentos son obtenidos por

$$m_k = \frac{1}{n}\sum_{i=0}^{n}(x_i - m)^k$$

.....(8)

donde m_k representa al momento de orden k, el segundo momento evalúa la varianza, mientras que el tercero y el cuarto evalúan el sesgo y la curtosis respectivamente, también a partir del histograma de las imágenes una forma alternativa de obtener el histograma es por medio de la ecuación :

$$m_k = \sum_{j=\min}^{\max} p_j (z_j)^k$$

.....(9)

Donde $p_j = n_j / n$ (número de elementos con el valor j entre el número total de elementos), y z_j es el valor de gris del valor j actual, el valor del primer momento es 1 por definición.

Dentro del programa estos datos fueron calculados por medio de las siguientes líneas de código:

```
momentsVec.push_back(1);
for(int k=1;k<2*(numMoments+1)-1;++k){
    int j;
     for( j=min-1;j<=max;){
           momento=momento+((double)freqs[j]/N)*(pow(j-media,k));
           j++;
     }
    momentsVec.push_back(momento);
    momento=0;
}
```

previamente a esto fue calculada la media y las frecuencias del conjunto de datos, a la vez que se obtuvo el total de datos por medio del producto del (ancho)X (alto) X (numero de cortes), como se mencionó fueron utilizadas las librerías STL y por medio de estas se creo una lista ligada llamada moments, en donde fueron almacenados los momentos generados por medio de esta fracción del código.

Los momentos son una propiedad de las imágenes que reflejan la forma del histograma y otras características de la función de densidad. Una estrategia es buscar que la isosuperficie resultante de la segmentación basada en pixeles, preserve los mismos momentos que las imágenes originales, con el histograma resultante de colapsar los rangos de los subconjuntos (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.

Por ejemplo, la imagen original digital es una versión borrosa de la imagen original física o real, en donde se buscan los valores (z_0 y z_1). Los primeros tres momentos de la imagen f son preservados en la imagen binaria resultante g. P_0 y P_1 denotan las fracciones de los pixeles por debajo del punto de partición y por arriba de este umbral respectivamente.

Para obtener estos puntos de partición P_i el primer paso es resolver un sistema de ecuaciones una vez que se obtienen los momentos y la media, entonces se procede a crear el siguiente sistema de ecuaciones de acuerdo a los puntos solicitados, en el caso de que se requieran n puntos de la isosuperficie el sistema será el mostrado en la ecuación 10

$$
\begin{aligned}
&c_0 m_0 + c_1 m_1 + c_2 m_2 \cdots\cdots c_{n-1} m_{n-1} = -m_n \\
&c_0 m_1 + c_1 m_2 + c_2 m_3 \cdots\cdots c_{n-1} m_n = -m_{n+1} \\
&c_0 m_2 + c_1 m_3 + c_2 m_4 \cdots\cdots c_{n-1} m_{n+1} = -m_{n+2} \\
&\cdot \\
&\cdot \\
&\cdot \\
&c_0 m_{n-1} + c_1 m_n + c_2 m_2 \cdots\cdots c_{n-1} m_{2n-1} = -m_{2n-1}
\end{aligned}
$$

.....(10)

Una vez resuelto el sistema se obtienen los valores auxiliares c_i cuando tenemos esto se procede a crear un polinomio de orden n sustituyendo los valores c_i obtenidos en el sistema anterior como sigue:

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

.....(11)

Si z_i denota los valores de grises representativos de la clase del píxel i-ésima y P_i denota la fracción de los pixeles de la i-esima clase. Resolviendo este polinomio

tenemos los valores de grises representativos de la imagen $z_0, z_1, z_2, \ldots z_n$, los valores z_j encontrados son sustituidos en las siguientes ecuaciones

$$p_0 z_0^0 + p_1 z_1^0 + \cdots\cdots p_{n-1} z_{n-1}^0 = m0$$
$$p_0 z_0^1 + p_1 z_1^1 + \cdots\cdots p_{n-1} z_{n-1}^1 = m1$$
$$\vdots$$
$$p_0 z_0^{2n-1} + p_1 z_1^{2n-1} + \cdots\cdots p_{n-1} z_{n-1}^{2n-1} = m_{2n-1}$$

.....(11)

Cuando se obtienen los valores P_i se pueden obtener los valores que representarán a los p_i por medio de la ecuación 12.

$$p_i = \left(\frac{P_i}{\sum_{i=0}^{n-1} P_i} \right) x(numMax)$$

.....(12)

En la ecuación anterior los valores pi a obtener se basan en el producto máximo de la imagen con el cociente de los valores p_i's obtenidos en el último sistema de ecuaciones sobre la sumatoria de todos los valores Pi.

El siguiente código muestra como fue implementada esta parte:

```
int i,j,cont =0;
      for( i=0;i<numMoments;){
        for( j=0;j<numMoments;++j){
           A[i][j]=momentsVec[cont++];
```

```
        }
            B[i]=-momentsVec[cont];
        cont=0;
        ++i;
        cont+=i;
      }
c=1;
sgesv(N1, c, AT, N1, pivote, B, N1, info);

Metodos metodos;
i=0;
for(;i<numMoments;){
   polElements[i]=B[i];
   ++i;
}

 polElements[i]=1;

  int value =metodos.polyRoot( polElements,numMoments-1);

 double *raices=metodos.getRoots();
cont=0;
    for(int k=0;k<numMoments;++k){
       for(int i=0;i<numMoments;++i){
          A[k][i]=pow(raices[i],k);
       }
       B[k]=momentsVec[cont++];
    }
c=1;
sgesv(N1, c, AT, N1, pivote, B, N1, info);
```

```
double suma=0;

    for( i=0;i<numMoments;++i){
        suma=suma+B[i];
    }

double *porcentaje;
     porcentaje= new double[numMoments];
       for( i=0;i<numMoments;++i){
            porcentaje[i]=B[i]/suma;
       }
       for( i=0;i<numMoments;++i){
          int data=(int)floor(porcentaje[i]*max);
          if(data<min)
             data=0;
         else if(data>max)
            data=255;
         cout<<"punto["<<i<<"] "<<data <<endl;
        arrayPi[j]=data;
   }
  return arrayPi
```

en esta parte arrayPi tiene los puntos sugeridos de acuerdo a la distribución del conjunto de datos totales.

Una vez que se tienen los datos volumétricos y la sugerencia del isovalor el siguiente paso es obtener los puntos que generarán la isosuperficie, esto es desarrollado por medio de diversos métodos contenidos en la clase MarchCube:

Ahora se verá la forma en que se llevará a cabo el cálculo de los puntos que generarán la isosuperficie.

Para la generación de los índices se utilizará una convención para poder generar los puntos, la mayor dificultad del algoritmo es el gran número de posibilidades que existen para el cálculo de los puntos, además de que se necesita derivar una combinación consistente de facetas para cada solución y de esta forma conectar los puntos de una forma correcta sin mostrar inconsistencias, en la figura 4.2.1 puede verse la notación que se siguió para la obtención de los vértices que utiliza el método.

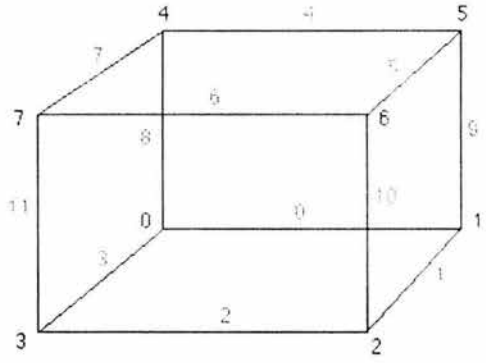


Figura 4.2.1 Notación para los vértices y aristas de los voxeles.

La primer parte del algoritmo hace uso de una tabla de aristas la cual mapea los vértices bajo los cuales la isosuperficie intersecta a estas., los índices se forman por valores de 8 bits ordenados de acuerdo a los vértices, en donde cada bit corresponde a un vértice.

```
cubeindex = 0;
if (cubo.val[0] < isovalor) cubeindex |= 1;
if (cubo.val[1] < isovalor) cubeindex |= 2;
if (cubo.val[2] < isovalor) cubeindex |= 4;
```

```
if (cubo.val[3] < isovalor) cubeindex |= 8;
if (cubo.val[4] < isovalor) cubeindex |= 16;
if (cubo.val[5] < isovalor) cubeindex |= 32;
if (cubo.val[6] < isovalor) cubeindex |= 64;
if (cubo.val[7] < isovalor) cubeindex |= 128;
```

la tabla de aristas regresa un número de 12 bits los cuales equivalen a las aristas de cada voxel, regresa 0 en cada bit por el cual no existe corte por parte de la isosuperficie dentro del voxel dado, en el caso de que todos los vértices se encuentren bajo la isosuperficie regresa un 0x00 o 0xff en el caso de que se encuentren todos los puntos sobre la superficie.

La intersección exacta de los puntos se calculan por interpolación lineal, si P_1 y P_2 son los vértices de corte de una arista y V_1 y V_2 son los valores escalares de cada vértice la intersección del punto P esta dado por:

$$P = P_1 + (isovalor - V_1)\,(P_2 - P_1) / (V_2 - V_1)$$

La última parte del algoritmo involucra la formación de las facetas correctas de las posiciones de las aristas que interceptan los voxeles, nuevamente la tabla a partir de la cual se obtienen los vértices que intersecan a la isosuperficie.

4.3 Desarrollo del Front-End

El diseño y construcción del front-end ha sido desarrollada utilizando QtDesigner de trolltech, ver figura 4.3.1.

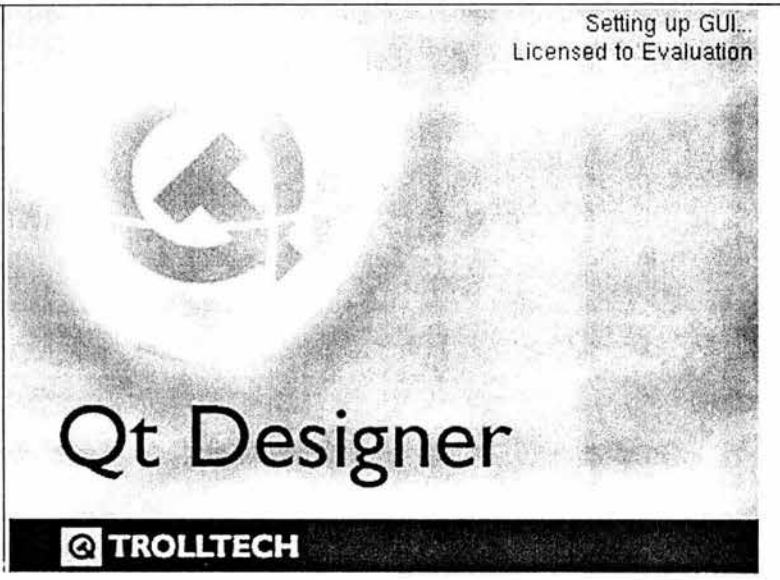


Figura 4.3.1 Pantalla de Inicio de Qt Designer

El diseño de QtDesigner es muy similar al de otros entornos de desarrollo de aplicaciones ver figura 4.3.2 en el lado derecho tenemos los diferentes widgets que podemos añadir a nuestra aplicación, en la versión gratuita se tienen un gran número de widgets, pero no se tienen todos aquellos que permite utilizar la aplicación, en esta versión tenemos widgets para botones, contenedores, bases de datos, de entrada y despliegue, en esta versión no se tienen algunos widgets que serán necesarios para la aplicación que se desarrollará, como es el caso del menú, submenú y el widget para OpenGL, al inicio de QTDesigner se abre un menú wizard a partir del cual se puede elegir el proyecto o la ventana a crear.

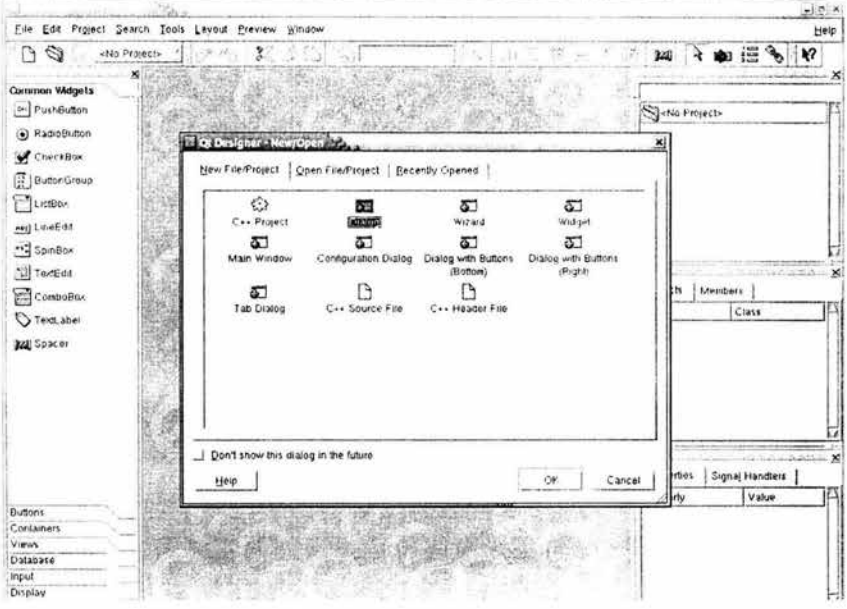


Figura 4.3.2 Pantalla de elección de proyecto

El primer paso para la creación de un proyecto es elegir el c++ project y una ventana de dialogo, en esta podemos ir pegando widgets ver figura 4.3.3 la aplicación generará código en C++, pero las conexiones y la funcionalidad de cada objeto debe hacerse introduciendo la funcionalidad en el código.

Para el diseño de este sistema se creó una clase en la cual fue introducida la funcionalidad necesaria y manejo del flujo, esta clase es GLObjectWindow, en esta se introdujo el menú, en donde se manejarán diferentes funcionalidades en estos objetos, directamente este código fue introducido en el constructor de la clase.

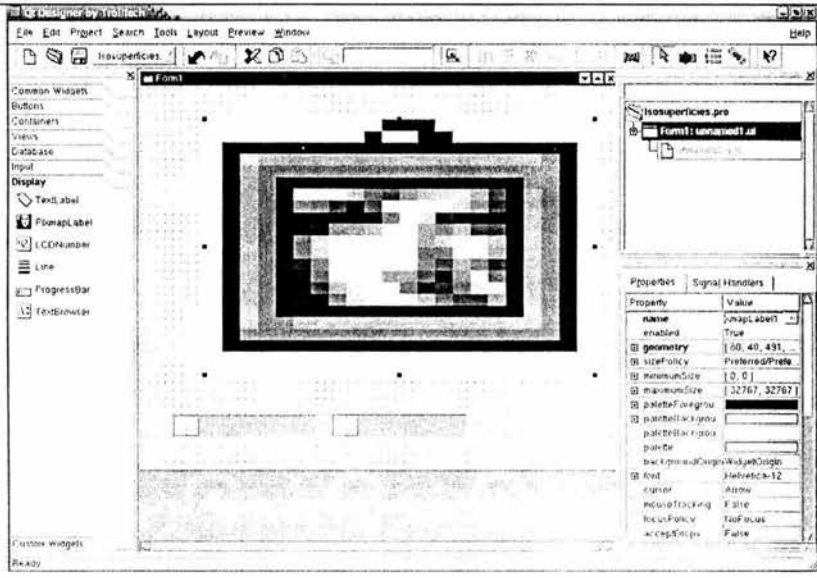


Figura 4.3.3 Los widgets se pegan directamente en la ventana recientemente creada

Este es el código a partir del cual se creó la barra de menú, el widget de OpenGL y los diferentes sliders para manipulación de la imagen.

```
mnuArchivo = new QPopupMenu( this );
mnuArchivo->insertItem( "Abrir Archivo",0,CTRL+Key_A );
mnuArchivo->insertItem( "Guardar Imagen",1, CTRL+Key_G );
mnuArchivo->insertItem( "Salir",  qApp, SLOT(quit()), CTRL+Key_S );
QObject::connect(mnuArchivo, SIGNAL(activated(int)),
                              this,SLOT(slotSetInputOutputFile(int)) );

mnuIsovalor = new QPopupMenu( this );
mnuIsovalor->insertItem("Inserta Nuevo Isovalor",0);
QObject::connect(mnuIsovalor, SIGNAL(activated(int)),
                              this,SLOT(slotIsovalorTxtBox()) );
```

```
    // Crea a frame widget OpenGL
    f = new QFrame( this, "frame" );
    f->setFrameStyle( QFrame::Sunken | QFrame::Panel );
    f->setLineWidth( 2 );

    // Crea el Widget de OpenGL
    c = new Despliegue( f, "glbox");

    mnuTextura =new QPopupMenu(this);
    mnuTextura->insertItem("Textura1",TEXTURA1);
    mnuTextura->insertItem("Textura2",TEXTURA2);
    mnuTextura->insertItem("Textura3",TEXTURA3);
    mnuTextura->insertItem("Textura4",TEXTURA4);
    mnuTextura->insertItem("Textura5",TEXTURA5);
    QObject::connect(mnuTextura, SIGNAL(activated(int)),
                                 c,SLOT(setTextura(int)) );
//                               this,SLOT(sendTextura(int)) );
    mnuAyuda = new QPopupMenu( this );
    mnuAyuda->insertItem("Acerca de", 0);
    QObject::connect(mnuAyuda, SIGNAL(activated(int)),
                               this,SLOT(slotVentanaAcerca()));

    m = new QMenuBar( this );
    m->setSeparator( QMenuBar::InWindowsStyle );
    m->insertItem("&Archivo", mnuArchivo );
    m->insertItem("&Isovalor", mnuIsovalor );
    m->insertItem("&Texturas",mnuTextura);
    m->insertItem("A&yuda", mnuAyuda );
```

```
x = new QSlider ( 0, 360, 60, 0, QSlider::Horizontal, this, "xsl" );
x->setTickmarks( QSlider::Left );
QObject::connect( x, SIGNAL(valueChanged(int)),c,SLOT(setXRotation(int)) );

y = new QSlider ( 0, 360, 60, 0, QSlider::Horizontal, this, "ysl" );
y->setTickmarks( QSlider::Left );
QObject::connect( y, SIGNAL(valueChanged(int)),c,SLOT(setYRotation(int)) );

z = new QSlider ( 0, 360, 60, 0, QSlider::Horizontal, this, "zsl" );
z->setTickmarks( QSlider::Left );
QObject::connect( z, SIGNAL(valueChanged(int)),c,SLOT(setZRotation(int)) );


zoom = new QSlider ( 0, 360, 60, 0, QSlider::Horizontal, this, "zoomsl" );
zoom->setTickmarks( QSlider::Left );
QObject::connect( zoom, SIGNAL(valueChanged(int)),c,
                                        SLOT(setZoom(int)) );

QLabel *labelx = new QLabel( this );
labelx->setText( "Rotacion en X" );
labelx->setAlignment( AlignBottom | AlignCenter );

QLabel *labely = new QLabel( this );
labely->setText( "Rotacion en Y" );
labely->setAlignment( AlignBottom | AlignCenter );

QLabel *labelz = new QLabel( this );
labelz->setText( "Rotacion en Z" );
labelz->setAlignment( AlignBottom | AlignCenter );
```

```
QLabel *labelzoom = new QLabel( this );
labelzoom->setText( "Zoom" );
labelzoom->setAlignment( AlignBottom | AlignCenter );

QVBoxLayout *xLayout = new QVBoxLayout(20,"xLayout");
xLayout->addWidget(labelx);
xLayout->addWidget(x);

QVBoxLayout *yLayout = new QVBoxLayout(20,"yLayout");
yLayout->addWidget(labely);
yLayout->addWidget(y);

QVBoxLayout *zLayout = new QVBoxLayout(20,"zLayout");
zLayout->addWidget(labelz);
zLayout->addWidget(z);

QVBoxLayout *zoomLayout = new QVBoxLayout(20,"zLayout");
zoomLayout->addWidget(labelzoom);
zoomLayout->addWidget(zoom);

QHBoxLayout* vlayout = new QHBoxLayout(20, "vlayout");
vlayout->addLayout( xLayout );
vlayout->addLayout( yLayout );
vlayout->addLayout( zLayout );
vlayout->addLayout( zoomLayout);

QHBoxLayout* flayout = new QHBoxLayout( f, 2, 2, "flayout");
flayout->addWidget( c, 1 );

QVBoxLayout* hlayout = new QVBoxLayout( this, 20, 20, "hlayout");
```

```
hlayout->setMenuBar( m );
hlayout->addWidget( f, 1 );
hlayout->addLayout( vlayout );
```

La interfaz final del sistema quedó como se muestra en la figura 4.3.4 en esta se puede apreciar la barra de menú en la parte superior, en el centro la ventana negra es el widget de OpenGL y abajo se tienen algunos sliders de manipulación del objeto presentado y creado con las librerías gráficas.

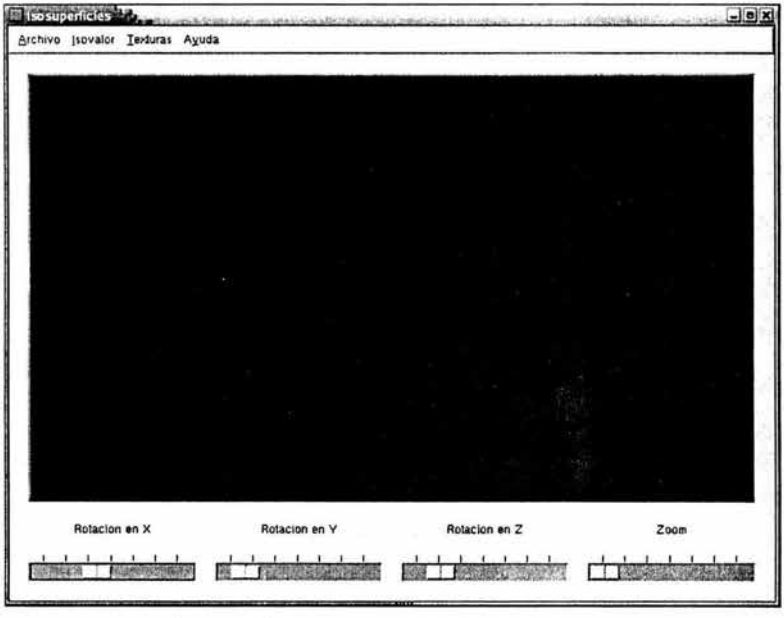


Figura 4.3.4 ventana de inicio del sistema

aunque esta es la ventana principal existen algunas opciones para abrir un archivo, figura 4.3.5, guardar imagen, figura 4.3.6, y elección del isovalor ver figura 4.3.7.

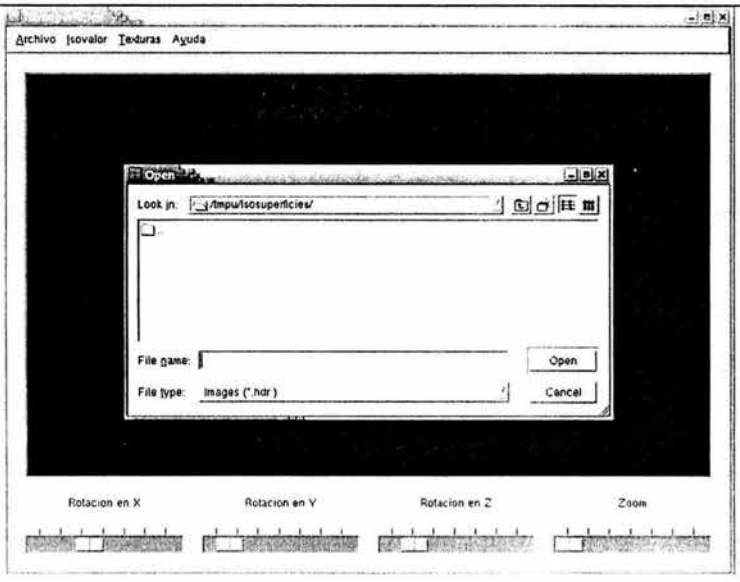


Figura 4.3.5 Ventana de apertura de archivo

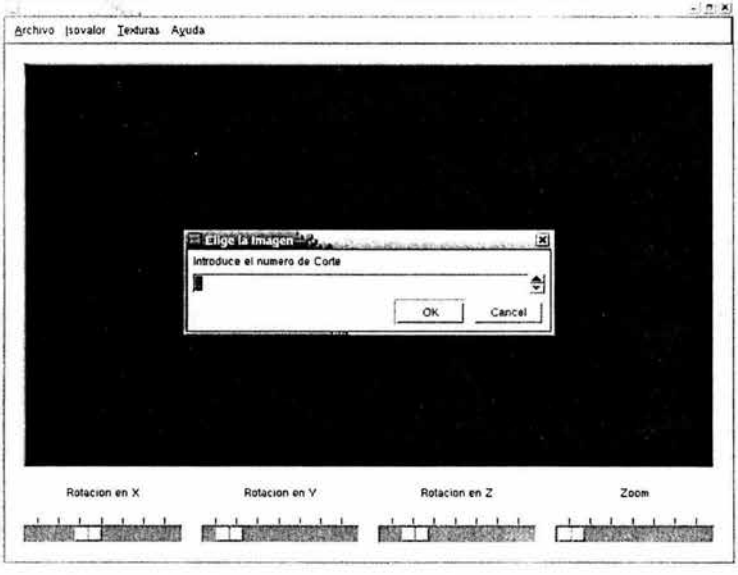


Figura 4.3.6 Ventana de elección de la imagen a salvar

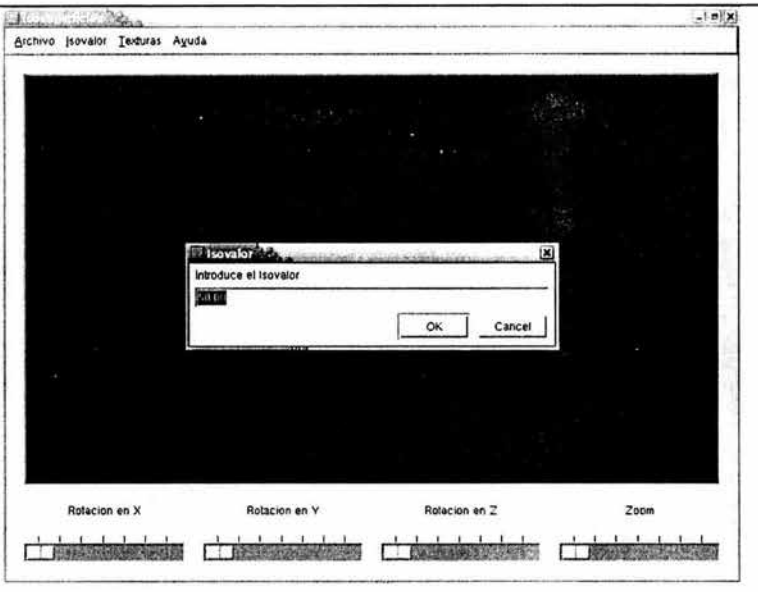


Figura 4.3.7 Ventana de solicitud del isovalor

4.4 Pruebas e Integración del Sistema

En esta sección se realizarán pruebas del sistema, se recomienda hacer estas pruebas para verificar y validar el software, estas son una actividad en la que el sistema o algunos de sus componentes se ejecutan en circunstancias previamente especificadas, los resultados se observan y se registran para realizar algún tipo de evaluación. Debemos tener en cuenta que un proceso de prueba será exitoso cuando sean encontrado errores y que estos no siempre son fruto de la negligencia del programador.

Entre las recomendaciones para evaluar eficientemente el sistema, en cada caso de prueba debe definir el resultado de salida esperado, el programador debe tratar de evitar probar sus propios programas, ya que desea (consciente o inconscientemente)

demostrar que funcionan sin problemas, el resultado de cada prueba debe ser inspeccionado a conciencia para poder detectar posibilidades de defectos.

Las siguientes son algunas recomendaciones para realizar pruebas:

- Al generar casos de prueba, deben incluirse tanto datos de entrada válidos y esperados como no válidos e inesperados.
- Las pruebas deben concentrarse en dos objetivos:
    - probar si el software no hace lo que debe hacer.
    - Probar si el software hace lo que no debe hacer, es decir si provoca efectos secundarios adversos.
- Deben ser evitados los casos desechables, es decir, los no documentados ni diseñados con cuidado.
- No deben hacerse planes de prueba suponiendo que, prácticamente, no hay defectos en los programas y, por lo tanto dedicando pocos recursos a las pruebas.
- La experiencia parece indicar que en donde hay un defecto existen otros, la probabilidad de descubrir nuevos defectos en una parte del software es proporcional al numero de defectos descubiertos.

En la figura 4.4.1 puede verse el proceso de análisis de pruebas a partir de algún desarrollo se lleva a cabo la planeación de pruebas, se diseñan las pruebas para su posterior ejecución, para el diseño es necesario información sobre software, como será la configuración de las pruebas (casos y procedimientos) y a partir de todo esto se obtienen resultados los cuales serán evaluados para la corrección o depuración del código y a partir del análisis de errores se puede hacer una predicción de fiabilidad.

Figura 4.4.1 Proceso de prueba

Para el diseño de casos de prueba se utilizan técnicas para conseguir una confianza aceptable en la que se detectarán los defectos existentes, equilibrio entre los recursos empleados y la confiabilidad de los casos de prueba, elegir los casos de prueba que puedan representar a los demás y elegir adecuadamente (no hacerlo aleatoriamente).

Existen tres enfoques para el diseño de casos de pruebas:

- El enfoque estructural o de la caja blanca figura 4.4.2.
- El enfoque funcional o de la caja negra figura 4.4.3.
- El enfoque aleatorio consisten en utilizar modelos (muchas veces estadísticos) que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba.

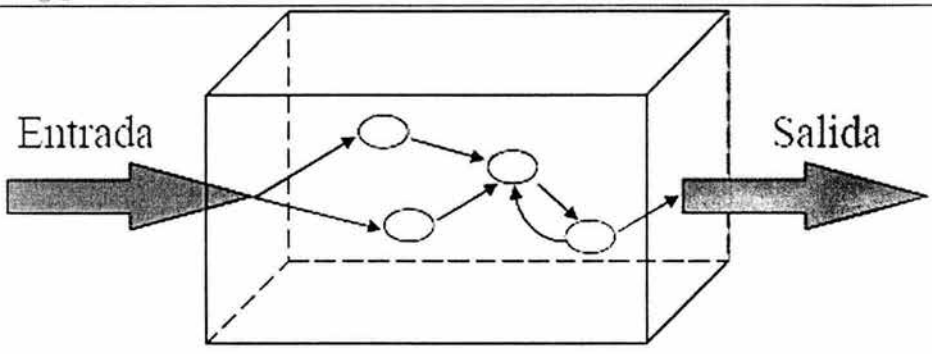


Figura 4.4.2 Caja blanca

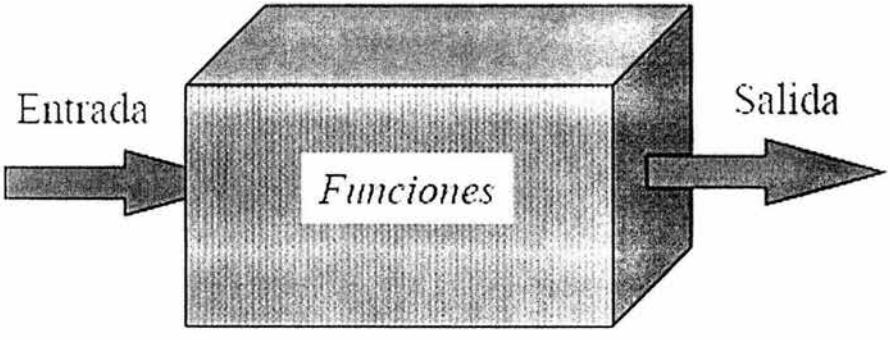


Figura 4.4.3 Caja negra

El diseño de las pruebas estructurales debe basarse en la elección de caminos importantes que ofrezcan una seguridad aceptable, deben ser utilizados criterios de cobertura lógica, a la vez que se recomienda realizar un diagrama de flujo de control (flowchart) a partir del cual se probará el proceso.

En el caso del sistema se generó el diagrama mostrado en la figura 4.4.4

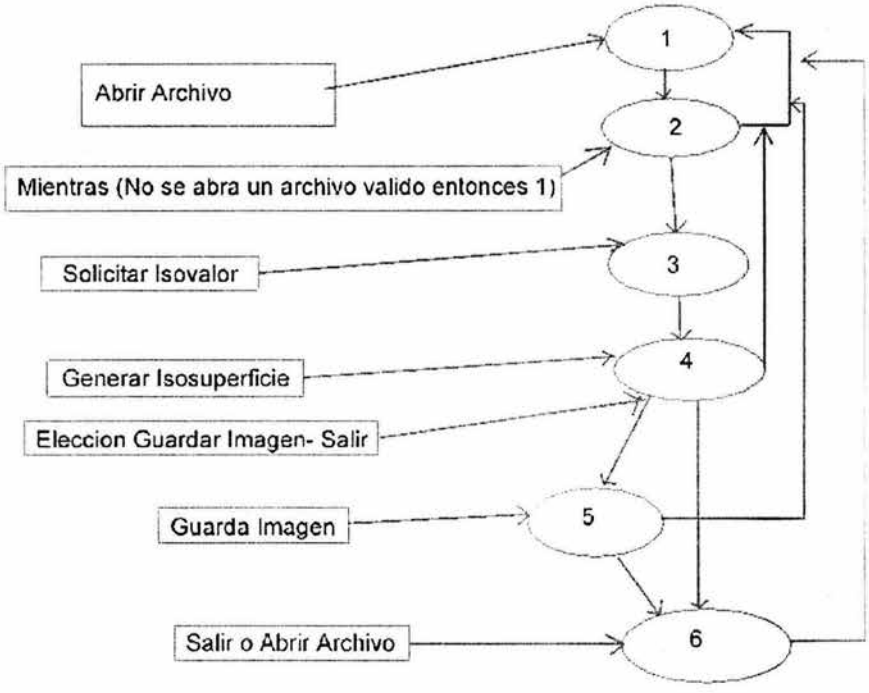


Figura 4.4.4 Diagrama de pruebas estructurales

En este caso siguiendo el flujo del sistema este presento un funcionamiento correcto, en todos los caminos posibles, para elegir los caminos independientes a probar se utiliza el método del "camino básico", el cual consiste en realizar variaciones sobre el camino de prueba típico llamado básico, los posibles caminos existentes son:

1-2-1

1-2-3-4-5-6

1-2-3-4-5-6

1-2-3-4-1

1-2-3-4-5-1

1-2-3-4-5-6-1

1-2-3-4-6

cabe destacar que en ninguna de las rutas anteriores se presento algún error.

Por otro lado las pruebas funcionales consisten en la identificación de clases de equivalencia y la creación de casos de prueba correspondientes a estas, se debe identificar las restricciones al formato y contenido los datos de las entradas, para la identificación de las clases de equivalencia se deben identificar las restricciones al formato y contenido de los datos de las entradas e identificar los datos válidos y los datos no válidos o erróneos.

Algunas reglas que ayudan a identificar clases:

- Si se especifica un rango de valores para los datos de entrada, se creará una clase válida y dos clases inválidas
- Si se especifica un numero de valores, se creará una clase válida y dos no válidas.
- Si se especifica una situación del tipo <<debe ser>> o booleana (por ejemplo: el primer carácter debe ser una letra), se identifican una clase válida (es una letra) y una clase inválida (no es una letra).
- Si se especifica un conjunto de valores admitidos y se sabe que el programa trata de una manera diferente cada uno de ellos, se identifica una clase válida por cada valor y una no válida.
- En cualquier caso, si se sospecha que ciertos elementos de una clase no se tratan igual que el resto de la misma, deben dividirse en clases menores.

Clases:

Archivo de Entrada:

- archivo de cabeceras con extensión "hdr" que contiene la información para leer los datos volumétricos, por omisión sólo abre archivos con estas características.

Elección Isovalor

- Isovalor numero double que no permite numeros mayores a 255 ni menores a 0.
- Posibles datos de entrada: Int, float y double (no permite caracteres no numéricos)

Selección de la imagen a guardar

- numero entero que no permite números mayores al numero de cortes ni menor a cero.
- Posibles datos de entrada: Int, float y double (no permite caracteres no numéricos y si no es entero lo redondea a entero)

Condición de entrada	Clases Válidas	Clases inválidas
Archivo de entrada	archivo HDR	Cualquier otro tipo de archivo
Elección del Isovalor	número mayor a cero y menor a 255	- Isovalor <0 - Isovalor >255 - Isovalor =[a-z|A-z]* - Isovalor = valores simbólicos
Selección de la Imagen a guardar	numero > 0 y numero < numero de cortes	- numero < 0 - numero># de cortes - numero=[a-z|A-z]*

En estos casos no existieron problemas para el manejo de estas reglas debido que es propio de las librerías de Qt manejar objetos de un tipo solamente, hay objetos para cada tipo de dato de entrada y no permite ingresar datos de otro tipo, por lo que estas pruebas fueron negativas (no se encontró error al ingresar los datos).

Para el caso de pruebas aleatorias entre las recomendaciones que se tienen es simular una entrada de datos habitual de un programa, se crean datos que sigan la frecuencia y la secuencia que tendrían en la práctica y se usan generadores de pruebas en donde se describen los datos, se alternan secuencias de entradas posibles y la posibilidad de ocurrencia.

En esta parte se modificó el orden en la que se ejecuta el programa, en vez de abrir el archivo se introduce el isovalor, en este caso el programa no ejecuta ninguna operación y no realiza el despliegue dentro del widget de OpenGL. Otro paso fue el de almacenar una imagen en donde tampoco se obtiene resultado, es decir con estos casos no existió falla alguna al cambiar este orden, si en vez de presionar el botón de Ok y se cancela, se manda un mensaje indicando la necesidad de introducir los datos para generar el despliegue.

Las pruebas de integración estan ligadas a la forma prevista de integrar los distintos componentes de software hasta contar con el producto global que debe entregarse, existen dos tipos de integración:

- Incremental (ascendente y descendente) Se combina el siguiente módulo que debe probar con el conjunto de módulos que están probados.
- No incremental En este caso se prueba cada módulo por separado y luego se integran todos a la vez y se prueba el programa completo.

Pruebas de Sistema

Las pruebas de sistema verifican que todos los requisitos funcionales, considerando el producto de software final por completo en un entorno de sistema, también verifica el funcionamiento y rendimiento en las interfaces de hardware, software , de usuario y de operador.

Como se ha mencionado en apartados anteriores para correr eficientemente el programa necesitamos hacer algunas configuraciones primero:

En el caso de Linux se deben configurar algunas librerías, primero se debe verificar que existen las librerías de Qt y en caso contrario se deben descargar de la página de trolltech mostrada en la bibliografía, y también deben ser instaladas las librerías libres de OpenGL (Mesa) que también se indican en la bibliografía (seguir información para instalación) no se debe tener mayor problema para su instalación siguiendo la documentación de instalación.
Lo siguiente es configurar las variables de ambiente para el correcto funcionamiento. En el directorio Home de cada usuario se tiene algún archivo oculto de configuración de shell por ejemplo el .tcshrc en el caso de tcshell en este archivo podemos configurar nuestras variables de ambiente entre muchas otras cosas:

La notación para esto en tcshell es la siguiente:

Setenv <variable> <valor>

Por ejemplo

setenv LD_LIBRARY_PATH /OpenGL/Mesa-5.0.2/lib

setenv QT_LIBRARY /usr/lib/qt-3.1

setenv MESADIR /OpenGL/Mesa-5.0.2

al introducir lo anterior debemos guardar el archivo y en la línea de comandos escribimos lo siguiente:

.

source .tcshrc

con esto reiniciamos la configuración del shell y así ya tenemos las variables solicitadas por Qt para compilar el sistema.

4.5 Puesta en Marcha

Con la configuración de las variables de ambiente debemos colocar el código fuente, (archivos cpp y h) en algún directorio esto lo podemos hacer descomprimiendo el archivo Isosuperficies.tar.gz por medio del comando:

tar zxvf Isosuperficies.tar.gz

con ello se generará un directorio llamado Isosuperficies en donde se encuentra todo el código fuente, el siguiente paso es compilar todos los archivos para generar el ejecutable del sistema de Isosuperficies, debemos ejecutar los siguientes comandos en el directorio en donde se encuentran todos los archivos:

qmake –project

esto creará un proyecto en caso de no existir uno.

qmake

este comando creará el Makefile

y por último escribimos:

make

con esto se procesan los archivos uno a uno hasta que se genera el archivo ejecutable Isosuperficies.

Con lo anterior ya se tiene configurado y listo el sistema para su funcionamiento y para que un usuario remotamente pueda accesar a través de la red al servidor y poder generar el despliegue de datos remotamente. Esto último se realiza conectándose a través de secure shell al servidor una vez que estamos dentro debemos configurar nuestra conexión para traer el despliegue de las aplicaciones que se requieran, esto se puede hacer cambiando el valor de la variable DISPLAY de la siguiente forma (en tcshell):

setenv DISPLAY <ip de la máquina cliente>:0.0

Una vez hecho esto ejecutamos la aplicación y traeremos el despliegue a la máquina cliente.

Conclusiones

- En la actualidad se debe hacer una mayor difusión y apoyar proyectos de investigación, tomando en cuenta el enfoque social que caracteriza a nuestra Universidad.

- Fomentar el desarrollo de software libre en aplicaciones científicas con fines tanto académicos como de apoyo a los sistemas de la nación.

- Modernizar las herramientas utilizadas en las diversas áreas para crear nuevas necesidades de acuerdo a requerimientos nacientes.

- El desarrollo de software científico es muy caro y muchas veces las universidades no tienen tantos recursos, por esto es necesario promover el uso de software libre y de desarrollo de sistemas para ahorrar y poder hacer uso de esos recursos en otros campos o para abrir nuevas investigaciones.

- El desarrollo de este sistema sirvió para aplicar una buena cantidad de conocimientos adquiridos durante la carrera, resultó muy ilustrativo el poder diseñarlo e implementado, sobre todo para poder corroborar conocimientos anteriores y aprender muchas otras cosas.

- He podido aprender lo dinámico y lo importante que es la ingeniería en computación, es un área muy abierta, debido a que tenemos la posibilidad de aprender muchas otras cosas, debemos adquirir conocimiento de acuerdo al problema a solucionar, en el caso de un problema de finanzas debemos tener conocimientos de contabilidad, si es el problema desarrollar un sistema nuclear debemos adquirir algunos de estos conocimientos.

Referencias

- C++ black book
  Tony I. Hansen
  addison Wesley
- C++ effective object oriented software construction
  Datrttrati Kassav
  prentice hall
- C++ for programmers
  Leedert Ammeraal
- Computer Graphics
  Edward Angel
  addison Wesley
- Estadística básica
  Vázquez Rodríguez Carmen
  Santillana
- Graphic Interface
  Wayne Davis
  Canadian Information society
- Graphic file formats
  David c. Kay
  mc graw hill
- Image processing analysis and machine vision
  Milan Sonja
  PWS
- Linux and UNIX shell programming
  Tansley David
  Pearlson Education

- Linux complete
  Grant Taylor
  Sybex
- OpenGL manual de referencia
  addison Wesley
- OpenGL programming guide
  Mason Woo
  addison Wesley
- UML in a nutshell
  Alhir Sinan
  prentice hall
- UNIX complete
  Dyson Peter John
  Sybex

- http://www.siggraph.org.mx/boletin/smbol14.html
- http://wissrech.iam.uni-bonn.de/research/projects/gerstner/meteo/meteo.html
- http://www.research.ibm.com/weather/SC97/obs/SC97_obs.html
- http://www.vislab.usyd.edu.au/education/sv3/2000/luke/report.html
- http://www.mcc.unam.mx/~lmd/Documentos/cplusplus/notas/
- http://www.augcyl.org/glol/old/N_2/Kdevelop/node4.html
- http://cs.uns.edu.ar/~prf/DocPOO/poo2002.html
- http://www.opengl.org
- http://www.trolltech.org
- http://www.depvis.unam.mx
- http://www.sgi.com.mx
- http://www.ieeexplore.ieee.org

Proceso de mantenimiento de software

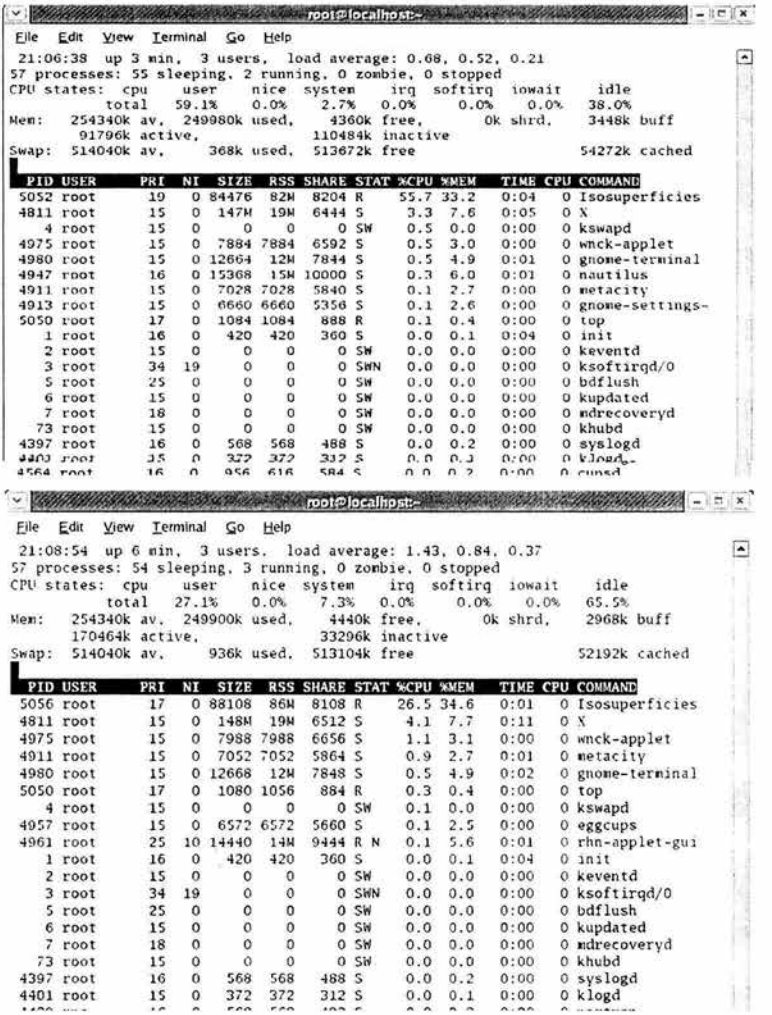
El mantenimiento del sw. es la modificación de un producto sw. después de su entrega al cliente o usuario para corregir defectos, para mejorar el rendimiento u otras propiedades deseables, o para adaptarlo a un cambio de entorno". En este proceso se realizan las modificaciones luego de haber concluido su desarrollo original y en proceso operativo.

Este proceso puede ser de diferentes tipos:

- **Adaptivo:** Modificar un producto de software, luego de haber entregado una versión del mismo, para mantener al producto utilizable en un entorno cambiante, el presente sistema es perfectamente adaptivo, esto lo permite directamente la funcionalidad del sistema operativo Unix o Linux, el entorno es muy estable, tomando en cuenta de que el presente sistema funcionará bajo el concepto de cliente servidor a través del servicio de secure shell el cual permite realizar un despliegue remoto a través de una red insegura; Este concepto permite realizar despliegue remoto modificando en una terminal una variable de ambiente (DISPLAY), claro siempre y cuando el sistema que sirve como servidor este correctamente configurado y en funcionamiento.

- **Correctivo:** Realizar modificaciones a un producto de software luego de haber entregado una versión, para corregir los errores detectados, en esta etapa del mantenimiento se hicieron algunas modificaciones para que el sistema sólo permita valores válidos para cada campo en donde se solicitan datos, en el caso de la elección de la imagen a guardar se tiene que introducir un entero, mientras que es más abierto el número de posibilidades de introducción del isovalor pudiendo ser este un número double, en el caso de que se deseé buscar una ruta diferente a la establecida para generar el despliegue también, no tiene problemas el sistema.

- **Optimizador:** Realizar modificaciones a un producto de software, luego de haber entregado una versión, para mejorar el performance, en esta sección del mantenimiento en lo que se refiere a optimización del sistema se revisó la implementación para optimizar el código en pruebas (haciendo uso de una utilería llamada top, la cual muestra los procesos y su consumo de memoria) se pudo observar el siguiente comportamiento:



Este proceso consistió en liberar memoria de los objetos que ya no eran necesarios en el proceso, en las imágenes anteriores puede verse el proceso Isosuperficies en donde el primero hacia un uso mayor de memoria.

- **Emergencia:** Tareas correctivas que se deben realizar y que no estaban planificadas para mantener operativo el Sistema, este proceso no aplicó, debido a que el sistema es operativo.

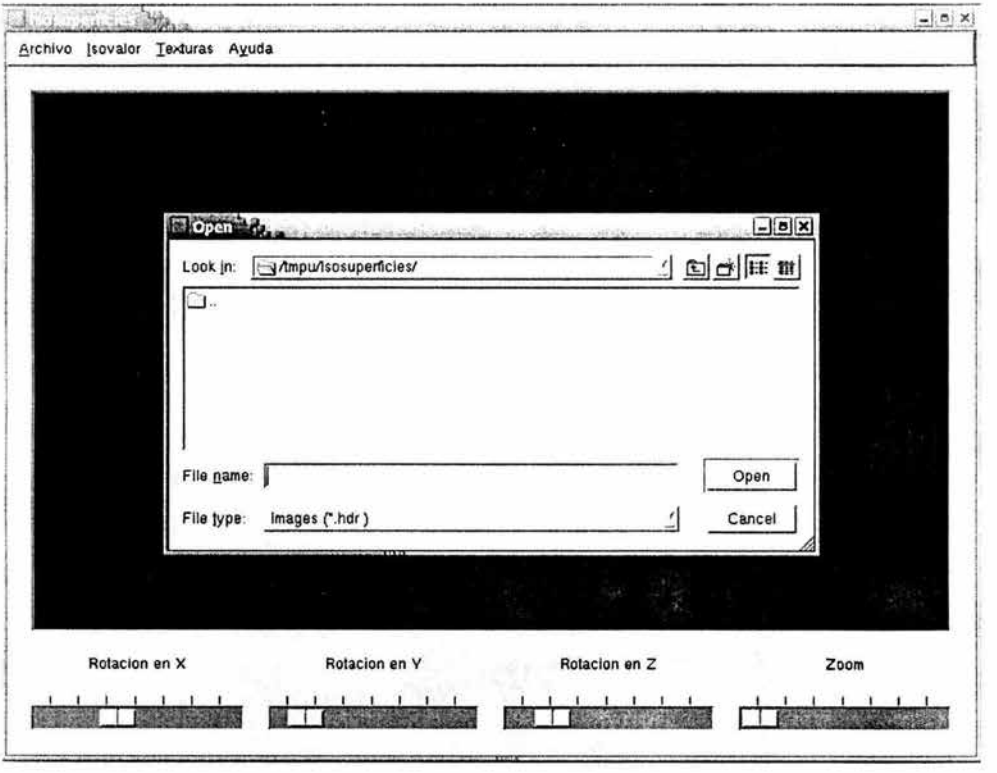
- **Preventivo:** Tareas correctivas que se deciden realizar para resolver potenciales problemas que pueden ocurrir, aquí ya se realizaron varios procesos para prevenir contingencias y que no sean realizados procesos con datos erróneos y pudiera con esto desencadenar en un error grave del sistema.

- **Problemas Software:** Es una confrontación humana con el software que causa dificultades, dudas o incertidumbre en el uso o evaluación del software. En algunos casos, los problemas pueden originarse como consecuencia de un desvío entre la Especificación de Requerimientos y el funcionamiento del Software, para este proceso se cambió el orden de los procesos que se llevan a cabo en donde no existió problema alguno debido a que el sistema no permite variaciones en el flujo de datos.

- **Requerimiento:** Funcionalidad requerida por un Usuario para resolver un problema o satisfacer uno o varios objetivos, en este caso se podrían añadir bastantes funcionalidades para hacer más sofisticado el sistema en si, como puede ser añadir más isosuperficies a un mismo despliegue, esto es en base a un conjunto de isovalores se generaran isosuperficies de diferentes valores y también de diferentes texturas para poder así obtener mayor información de forma visual.

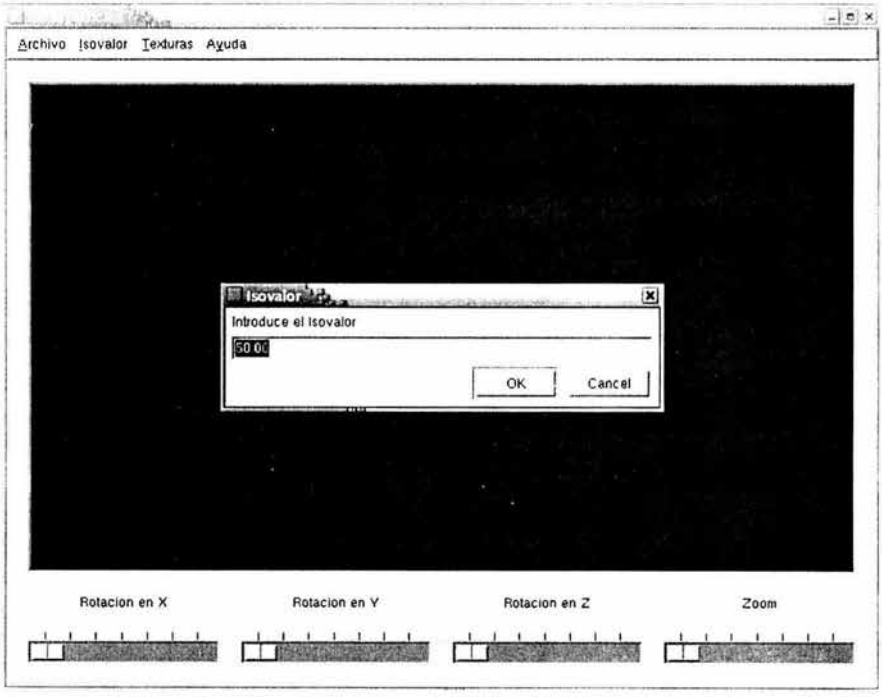
Guía rápida de uso

El uso del sistema es muy intuitivo, en primer lugar se debe abrir el archivo de cabeceras, el cual contiene toda la información para leer el archivo que contiene los datos volumétricos, para hacer esto hacer clic en el menú *Archivo -> Abrir Archivo*, lo cual abrirá la siguiente ventana.



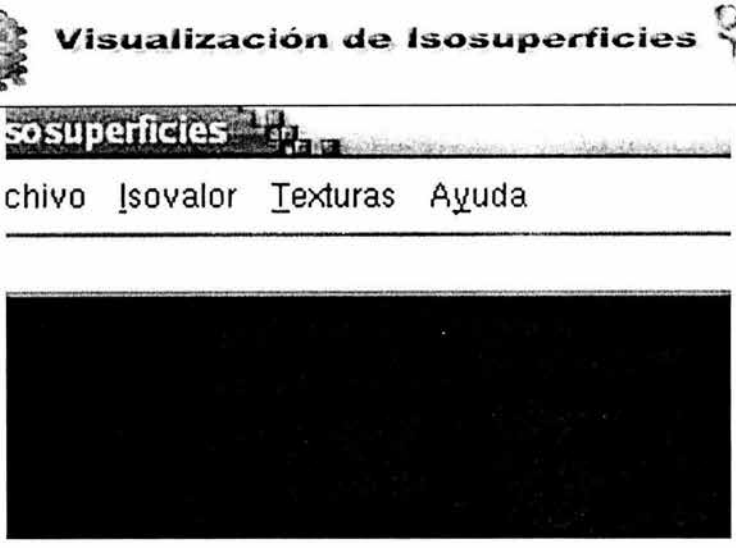
Pantalla de apertura del archivo HDR

El siguiente paso es realizar la elección del isovalor, por lo que debemos hacer clic en el menú en: *Isovalor -> Inserta Isovalor*, con esto se abre una ventana que contiene una caja de texto en donde se introducirá el isovalor.



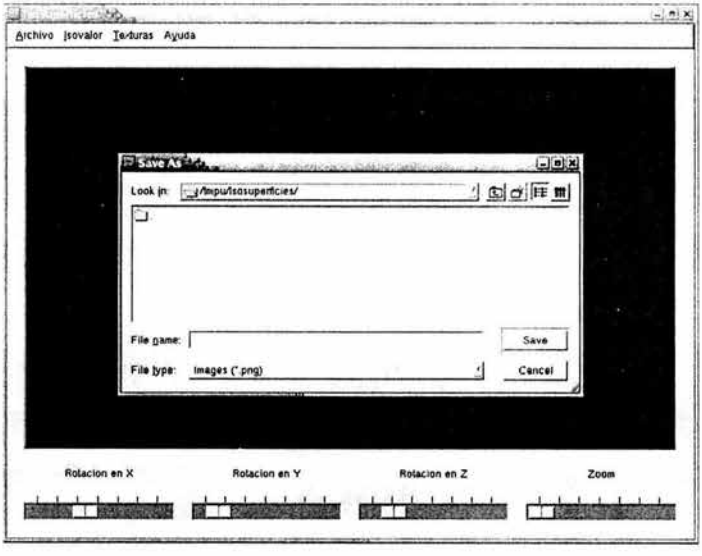
Ventana de introducción del isovalor

Con esto tendremos el despliegue de la isosuperficie en la ventana, opcionalmente se pueden elegir algunas texturas a elegir en el menú (haciendo clic en *Texturas-> textura_n*):



Menú de elección de texturas

También se tiene la opción de elegir algunos cortes de los datos volumétricos, el sistema generará una imagen de acuerdo al corte, esta imagen estará en formato pgm, para hacer esto hacer clic en el menú en *Archivo->Guardar Imagen*, al hacer esto se abrirá una nueva ventana para elegir el corte y el nombre que le será dado a la imagen



pantalla de guardar imagen