Universidad Central "Marta Abreu" de Las Villas

Facultad Matemática Física Computación

Representación de propiedades continuas, granulares y mixtas en sólidos

Autor:

Heikel Yervilla Herrera

Tutores:

Dr. Carlos Pérez Risquet

Lic. Alcides Viamonte Esquivel

Ciencias de la Computación 2007

"Obrar es fácil, pensar es difícil, obrar según se piensa es aún más difícil."

Johann Wolfgang Goethe

A mis padres por su infinito amor…

A mi abuela…

"Agradecer es nobleza"

A mi mamá por todo su esfuerzo, apoyo y preocupación.

A mi papá y mi abuela por su confianza.

A Damiana por saber esperar.

A mi familia.

A Yaidel por todo el tiempo compartido.

A Ismay y Alcides por toda su ayuda.

A Anabel por enseñarme a escribir.

A Alejandro por convertirse en un buen amigo.

A todos los estudiantes y profesores del Aula Cimne.

A mis compañeros de estudio.

A todos aquellos que de una forma u otra me han dado su apoyo.

Resumen

El siguiente trabajo aborda el tema de la visualización científica. El objetivo principal es el desarrollo de una aplicación vinculada con la visualización de datos escalares y vectoriales. Se trata, además, todo el proceso de visualización desde un punto de vista teórico. Se explican las técnicas de visualización a partir de las características de los datos. Para datos escalares discretos se desarrollaron nuevas técnicas: representación de cubos y polígonos en el interior y exterior del cuerpo principal respectivamente. Las bases de este trabajo son las primitivas geométricas y los predicados de geometría computacional. En la implementación se utilizaron herramienta como: C#, OpenGL y GTK-Sharp. El software analiza los resultados obtenidos a partir de la aplicación del Método de Elementos Distintos en cuerpos sólidos.

Abstract

This work is about scientific visualization. The aim in this work it is to develop a software appliance for scalar and vectorial visualization. This work also explains the concepts and processes of visualization. Visualization techniques are tightly tied to visualized data. For discontinuous data visualization were developed two new algorithms: representation with cubes and polygon in interior and exterior of main structure, respectively. An important part here it is about primitives and predicates of computational geometry. The software is implemented using C#, OpenGL and GTK-Sharp and it is used to analyze results of Particle Methods in solid structures.

Índice

Introducción

Al utilizar los resultados del cálculo de modelos numéricos convencionales (diferencias finitas, volumen finito, elementos finitos) es posible obtener de forma aproximada el campo de cierta magnitud sobre un sólido. En los métodos tradicionales, esta magnitud es continua, o en algunos casos presenta discontinuidades aisladas. El Método de Elementos Finitos es insuficiente para varios tipos de problemas: aquellos que implican grandes cambios geométricos o deformaciones del modelo de análisis, o intentan predecir el comportamiento de medios con un fuerte grado de discontinuidad inherente.

Nuevos métodos permiten reproducir el comportamiento natural discontinuo de ciertas propiedades físicas en el medio o estructura. Por ejemplo, los parámetros de movimiento de las partículas aisladas en un gas, el campo de tensiones en un mortero sometido a carga, o el grado de ionización en una macro-molécula con radicales múltiples. También existen fenómenos donde se aprecian magnitudes con comportamiento mixto, como el campo de velocidades en distintas partes de la sección transversal de un tubo. Resulta interesante visualizar y analizar las propiedades físicas obtenidas por estos nuevos métodos.

Formulaciones del Método de Elementos Distintos o Método de Partículas[1] han venido a engrosar el mundo de las investigaciones de materiales y su comportamiento. Hasta el momento no se visualiza información acerca de las propiedades físicas de las partículas generadas. Además resulta imposible obtener conclusiones a partir de una interpretación directa de los datos, dado el enorme volumen de información. Hasta el momento las imágenes obtenidas de las formulaciones del MED se confeccionan mediante el uso del software PovRay, pero solo es posible una representación parcial de los datos. Además, se conoce que los campos en la superficie están dados por una función cuya

[1] MED

imagen puede ser continua, densamente discontinua o mixta. Por tales motivos se necesita una aplicación para la visualización de dichos campos.

El objetivo principal de esta investigación es diseñar algoritmos para la visualización de campos escalares continuos, densamente discontinuos y mixtos sobre la superficie de un objeto sólido generado mediante el método de partículas, y confeccionar una herramienta de software que implemente estos algoritmos.

Este objetivo puede ser desglosado en:

- Diseñar algoritmos de visualización que permitan la representación gráfica de un campo escalar en el espacio euclídeo tridimensional, que puede estar definido por una función continua, continua a trozos o discreta.
- Implementar los algoritmos de visualización diseñados.
- Diseñar e implementar una interfaz que permita la interacción del usuario con la imagen final y la comprobación de la eficacia del diseño e implementación de los algoritmos de visualización.

La visualización de datos escalares continuos es un tema bastante tratado en la actualidad, lo que no sucede con los de datos discretos. En un inicio se manejaron dos ideas fundamentales: utilizar una herramienta ya confeccionada y adaptar nuestros datos o elaborar un software que manejara todos los datos y que no tuviera problemas de licencias. La primera es, sin dudas, una buena solución pero conllevaba a que el software utilizado no fuera totalmente libre de restricciones; por otra parte al estudiar códigos ajenos se notó que sería necesario lidiar con complejidad superflua para resolver el problema. Una vez evaluados las ventajas y desventajas se decidió por la realización de un software de visualización.

En un área como la visualización científica un software relacionado con el tema introducirá nuevos puntos de referencia en el desarrollo de aplicaciones cada vez más potentes, precisas y útiles al usuario. Pero, ¿cómo extender el uso del enfoque de visualización para la representación de campos escalares continuos a campos densamente discontinuos y mixtos sobre la superficie de un sólido?

A lo largo de este trabajo se trata de responder esta interrogante y otras que surgirán a medida que el lector se introduzca en el tema. La culminación exitosa de este trabajo permitirá enriquecer las investigaciones realizadas hasta el momento en nuestro grupo de trabajo.

Capítulo I Visualización científica.

El ser humano obtiene una gran cantidad de información a través de la visión, se estima que un 50% de las neuronas del celebro humano están dedicadas a la percepción. El sistema visual es un buscador de patrones de extrema fuerza y sutileza, con tan solo observar una imagen durante un corto tiempo el cerebro obtiene información detallada del objeto de estudio. La visualización de información es, hoy, un tema recurrente en innumerables disciplinas tanto de importancia social como científica.

1.1. Introducción a la visualización científica.

El auge de las tecnologías fotográficas en las últimas décadas ha traído consigo imágenes con una gran calidad, lo que influye notablemente en la cantidad y calidad de información que se obtienen a partir de la imagen. En la mayoría de las disciplinas científicas los datos son almacenados para una posterior visualización [2]. Cada día el mundo de la computación se hace más potente y se ha convertido en una herramienta imprescindible para científicos e ingenieros a la hora de combinar diseños e imágenes.

1.1.1. La visualización como herramienta de entendimiento y análisis.

La visualización facilita el entendimiento de características de los datos almacenados, al usar técnicas estándares para la reproducción de imágenes en computadoras comienzan a aparecer características y/o propiedades totalmente invisibles en otros gráficos tradicionales. No solo muestra información de los mismos datos sino de la manera en que fueron coleccionados por lo que tiene un significativo valor en el control de la calidad [3].

La visualización es la generación de una imagen mental o real de algo abstracto o invisible [4]. En la computación y el diseño con fines de ingeniería este

término tiene un significado más específico, es la transformación y posterior representación de datos científicos y conceptos, abstractos, en imágenes. [3-5]

1.1.2. Visualización asistida por computadoras.

El primer impulso por utilizar las computadoras en la representación de datos fue en la década de 1960. La visualización científica, de ahí en lo adelante, ha ido de la mano del desarrollo de la computación. Hoy es imposible obtener una buena representación gráfica sin ayuda de un ordenador. [5]

La capacidad de almacenamiento y procesamiento de las computadoras es enorme. El volumen de información también crece considerablemente haciendo la búsqueda un proceso lento y costoso. Las computadoras, ya sean para generar estas bases de datos como para obtener una acertada visualización, han influido en el desarrollo científico y en la toma de decisiones a partir de las imágenes obtenidas.

“La visualización asistida por computadoras no es más que una sencilla y unificada colección de técnicas computacionales para la visualización científica” [5]. La visualización simplifica el entendimiento, análisis y comunicación de modelos, conceptos y datos en la ciencia y la ingeniería. En [6] se refiere a la visualización científica como la transformación de datos científicos y abstractos en imágenes. Es una forma especial de la visualización.

Grandes cantidades de datos provenientes de procesos industriales, económicos o ingenieriles son almacenados diariamente con un solo objetivo: obtener información. Las herramientas de visualización científica constituyen el intermedio entre ambos procesos.

El rápido crecimiento de las aplicaciones de visualización en áreas ingenieriles demuestra los grandes beneficios dentro del proceso de diseño. Un mejor

entendimiento de los beneficios en costo, productividad y exactitud se perciben en los siguientes puntos [3, 5].

- Disminución de pruebas físicas. Estas pruebas llevadas al ordenador reducen costos en tiempo y materiales. La visualización permite a los ingenieros observar fenómenos que en la práctica resultarían difíciles, peligrosos o imposibles.
- Integración entre el diseño y el análisis. Permite que pequeñas modificaciones en el diseño se muestren inmediatamente para obtener un correcto análisis en un corto tiempo.
- Incremento en la complejidad y sofisticación en el análisis. Las técnicas de visualización examinan fenómenos de los cuales no es fácil un correcto análisis.
- Diseños óptimos. Permite cambiar parámetros del modelo tales como el tamaño y la forma de manera automática, lo que posibilita el desarrollo del análisis y la optimización del propio diseño. Las herramientas de visualización permiten a los ingenieros evaluar automáticamente cambios de diseño y análisis de resultados.
- Beneficio económico. Se obtiene mayor exactitud de los diseños.

El uso de la visualización científica no está restringido exclusivamente a la investigación, se usa en otras aplicaciones ingenieriles y de diseño: diseño de ropa, industrial, automovilístico y aviones; ingeniería genética, producción química y farmacéutica, exploración mineral y de combustibles, diseño de plantas químicas y de producción energética, animación para la producción de efectos especiales en la industria audiovisual y la medicina.

No se puede dejar pasar por alto la gran influencia de los métodos numéricos en el proceso de visualización. El método de elementos finitos y la dinámica de fluidos en pocos años han logrado un gran impacto en los algoritmos de visualización científica.

1.1.3. El proceso de análisis y visualización.

Muchos diseños ingeniériles tienen en común componentes de modelado, análisis y visualización. El primero y el último son a menudo descritos en la práctica como pre-proceso y post-proceso respectivamente. Estos términos se refieren a la secuencia modelado-análisis-visualización donde se obtiene un diseño determinado, si los resultados son insatisfactorios el ciclo comienza nuevamente hasta cumplir las expectativas [5].

Este modelo tradicional ha evolucionado. Las tendencias actuales combinan las funciones y hacen un proceso interactivo (véase Figura 1) Permite la interacción de cada componente y una retroalimentación de los demás a cualquier cambio. Este modelo es el más idóneo para la modelación-visualización y la obtención de un diseño totalmente adecuado a las exigencias tecnológicas.

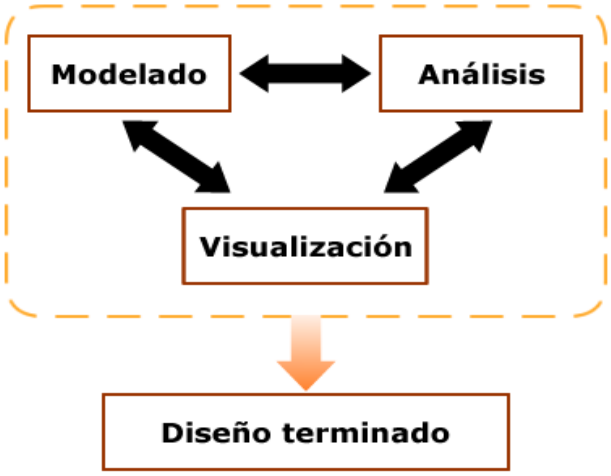

Figura 1 Integración entre el modelado, el análisis y la visualización para obtener un diseño determinado.

Los pasos que se desarrollan en el proceso de modelado o pre-proceso descritos en [5] incluyen:

- Descripción del modelo geométrico que describe el problema.
- Modelación analítica. Crea datos necesarios para el análisis a partir del modelo geométrico. Incluye la creación de mayas que aproximen dicho

modelo y la definición de los materiales y/o propiedades físicas de los elementos.

- Generación de las condiciones iniciales del modelo, entre las cuales están los estados iniciales, cargas y las condiciones de frontera.

Por su parte durante la visualización o post-proceso descrita en [5] se desarrollan operaciones como:

- Aplicación de métodos de visualización específicos: generación de imágenes que describan el fenómeno.
- Transformaciones del modelo: permiten cambiar interactivamente su percepción. Entre las transformaciones más conocidas se encuentran: la rotación, traslación y el escalado.
- Animación de la imagen en función del tiempo: la representación a través del tiempo y la navegación dentro del espacio.
- Manipulación del modelo resultante: incluye técnicas que permiten reducciones y cortes en la imagen, permite con ello un mejor análisis.

1.1.4. Uso de la visualización, adquisición de los datos.

La visualización científica se emplea, como se ha visto anteriormente, con varios fines que pueden englobarse en tres modos de uso muy particulares. En [7] se describen de esta forma:

- Análisis Exploratorio: se tiene un conjunto de datos sin una hipótesis específica. Estos se someten a un proceso de búsqueda interactiva de información que trae como resultado una visualización para sostener una hipótesis sobre el conjunto de datos.
- Análisis confirmativo: se tiene un conjunto de datos sobre los que se plantea una hipótesis. Se realiza un procesamiento de dichos datos que genera una visualización mediante la cual se valide o refute la hipótesis planteada.

- Presentación de un resultado: se conocen hechos específicos sobre los datos y se efectúa el proceso de visualización enfatizando en su veracidad.

Aunque las fuentes de datos para la visualización sean varias se engloban en tres categorías. [7]

- Reales: se refiere a datos obtenidos del mundo real mediante la medición por instrumentos como barómetros, termómetros, satélites o censores. El tamaño de las bases de datos es variable y puede superar los Gigabytes e incluso Terabytes cuando las mediciones provienen de análisis astronómico o aplicaciones militares. Estas bases de datos contienen errores producto de imprecisiones a la hora de almacenar, de apreciación en el momento de medir o por desperfectos en los equipos de medición.
- Teóricos: se refiere a datos obtenidos por medio de simulaciones computacionales basadas en modelos matemáticos. Estos datos dependen de la veracidad y la profundidad del modelo, así como de su diseño inicial.
- Artificiales: se refiere a datos creados por el ser humano: datos artísticos, de efectos especiales y animaciones en filmes. Estos datos pueden llegar hasta el rango de los Terabytes.

1.1.5. Tuberías de visualización.

Para la obtención de una representación, los datos deben ser tratados y procesados, este proceso puede ser explicado en términos de múltiples etapas conectadas entre sí: tuberías[2] [7-10] (Véase Figura 2).

[2] "Tuberías" (*pipeline*) termino ampliamente aceptado.

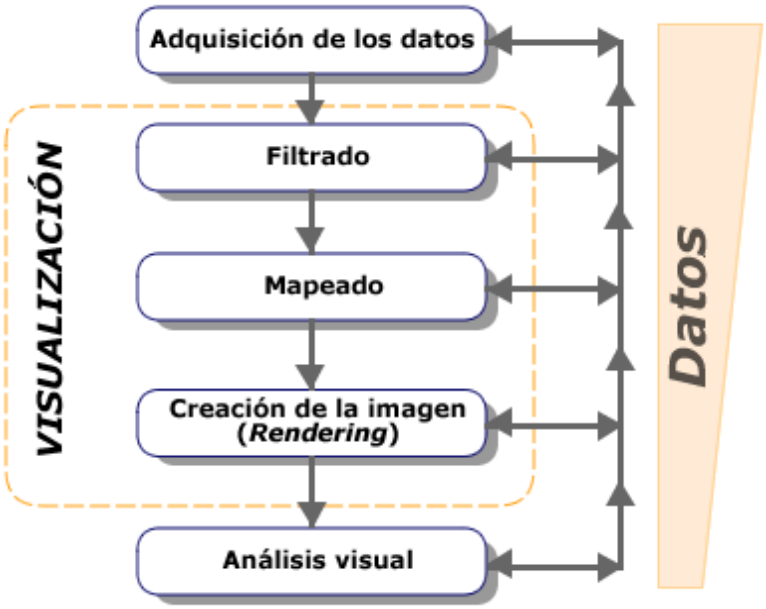

Figura 2 "Pipeline" de la visualización (tomado de CAGD and Scientific Visualization:Holgers Theisel)

La figura muestra la estructura general por la que transitan los datos desde su adquisición hasta el análisis visual de la imagen.

El primer paso consiste en la adquisición de los datos, el cual también es considerado el pre-paso de la tubería. Estos datos provienen de medidas, simulaciones, modelaciones u observaciones (proceso explicado en el epígrafe anterior). La manera de adquisición y el tamaño de la base de datos Para la visualización cien no importa [8].

En el proceso de filtrado se realizan operaciones de perfeccionamiento, reducción, filtrado, aproximación y extracción a partir de los datos originales con el objetivo de obtener meta datos. Se aplican algoritmos para completar los datos de una base de información incompleta [8, 10].

La reducción de los datos es necesaria si el conjunto de datos es muy largo para proceder a una visualización directa o si contienen fuertes redundancias. Los meta datos reúnen información cualitativa y cuantitativa relacionada con la base

de datos. Estos se usan para dirigir y controlar el siguiente paso en la visualización [7, 8, 10].

El proceso de mapeado es realmente el centro de la canalización. Los datos filtrados se llevan a primitivas geométricas: puntos, líneas, polígonos, u otros sus atributos de dibujo: color, transparencia, etc. Una vez desarrollado este proceso los datos se encuentran en condiciones de ser representados; la imagen o secuencia de imágenes es generada y pueden ser analizadas por el especialista o científico [7, 8].

1.2. Las mallas y la visualización.

Las mallas geométricas son utilizadas en la representación de cuerpos volumétricos. La topología de una malla de superficie es representada como una lista de elementos (caras) no estructurada donde cada cara define una n-tupla de índices en una lista de puntos .[5, 10-12]

Generalmente las mallas son usadas en representaciones en tres dimensiones (3D). Las más conocidas son las mallas de polígonos y las de triángulos. Su composición en ambos casos consiste en un conjunto de vértices, caras y bordes.

Se consideran bordes a los segmentos exteriores del polígono o triangulo. Las cara son denotadas como el interior del triángulo o polígono excluyendo los bordes y vértices. La superficie de la malla es el conjunto de puntos de la unión de las caras, bordes y vértices [12]. (Véase Figura 3)

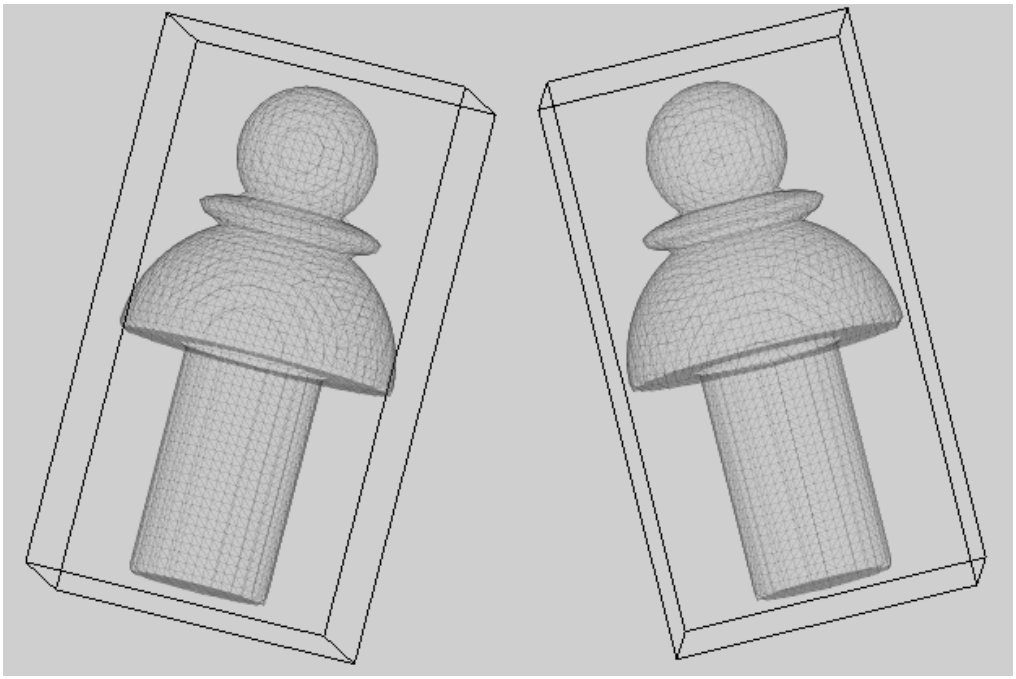

Figura 3 Malla de superficie. (Elaboración propia)

En la construcción de mallas geométricas tridimensionales uno de los algoritmos más utilizados es *marching cubes*.

1.3. Descripción de las técnicas de visualización científica.

Según Theisel, la visualización científica encuentra una representación visual apropiada para darle al conjunto de datos, con el fin de permitir un efectivo análisis y evaluación [7]. A lo largo de este epígrafe se abordarán de manera superficial algunas técnicas de visualización de superficies, volúmenes y las técnicas de visualización científicas más conocidas.

1.3.1. Modelos de visualización.

Cuando se habla de visualización de datos y computación gráfica se hace necesaria la definición de un modelo que permita representar los aspectos particulares del medio estudiado. Estos modelos son generalmente sencillos y

gracias a nuestro sistema de visión, se obtienen imágenes sumamente complejas [5].

Para modelar objetos se debe entender primero el espacio donde se encuentra. Teniendo un objeto y un punto de referencia, se necesita la posición relativa de ese objeto con respecto al punto de referencia [5].

Los objetos se representan de dos formas: explícitas o implícitas. En la primera se conoce de forma exacta de la posición del objeto, sea un punto, una línea o una superficie. El ejemplo más típico de este caso es cuando se emplean mallas para la representación de cuerpos. En la forma implícita no se conoce de forma exacta la ubicación de la superficie sino que está dada por un lugar geométrico o una función implícita. Por ejemplo, la esfera se representa implícitamente por su ecuación $1 = x^2 + y^2 + z^2$ y su superficie queda definida como un conjunto de puntos que se encuentra a una misma distancia del centro [5].

1.3.2. Clasificación de los algoritmos de visualización.

Los algoritmos de transformación de los datos son la cabeza de la visualización. Estos se clasifican según la estructura y el tipo de transformación. La estructura se refiere a los efectos tiene la transformación sobre la topología[3], geometría y atributos de los datos. El tipo se refiere a los datos con que se va a operar [10].

La clasificación según la estructura puede ser de cuatro formas [10]:

- Transformaciones geométricas: varían la geometría de los datos de entrada pero no su topología. Si se traslada, rota o se cambia de tamaño el conjunto de datos, su topología no cambia, solo las coordenadas de los puntos.

[3] Topología: rama de las matemáticas que trata especialmente de la continuidad y de otros conceptos más generales originados de ella como las propiedades de las figuras con independencia de su tamaño o forma.

- Transformaciones topológicas: alteran la topología pero no la geometría y los atributos de los datos.
- Trasformaciones a los atributos: convierten los atributos desde una forma a otra o crean nuevos atributos desde el conjunto de datos. Mantiene la estructura de manera inalterable. Ej. cálculo de la magnitud de un vector.
- Combinación de trasformaciones: combinación de trasformaciones que cambian la estructura y los atributos de los datos. Ej. líneas de contorno y superficies de contorno.

Los atributos se caracterizan en escalares, vectoriales, tensoriales o informaciones generales. Debido a los diferentes tipos de datos se decide hacer algún tipo de clasificación para los algoritmos, esta categoría según [3, 10] incluye:

- Algoritmos escalares: operan con datos escalares. ej. Representación de la temperatura en un mapa geográfico.
- Algoritmos vectoriales: se utilizan cuando los datos son vectores, la representación se hace mediante flechas, las cuales tienen dirección y magnitud. ej. Flujo de aire alrededor de una superficie, corriente de agua en un estanque.
- Algoritmos tensoriales: algoritmos aplicados sobre datos tensoriales[4]. Ej. Algoritmos que describen la deformación de algún material elástico. [13, 14]
- Algoritmos combinados: se agrupan los algoritmos que no pertenecen a ninguna de las clasificaciones anteriores o combinaciones de ellos. Ej. combinación de atributos escalares y vectoriales.

[4] Tensor: cierta clase de entidad geométrica que generaliza los concepto de escalar, vector y operados lineal de manera que sea independiente de cualquier sistema de coordenadas elegido.

1.3.3. Técnicas de visualización.

El objetivo de la visualización científica es mejorar el entendimiento de la información generada. La dificultad de este proceso consiste en encontrar la técnica de visualización que obtenga los resultados apropiados tanto para científicos como ingenieros.

Las técnicas de visualización son la base de todo el proceso explicado en el epígrafe 1.1.3, dependen en gran medida del tipo de dato (escalar, vector o tensor) y de la dimensión del dominio en que se representará (1D, 2D o multidimensional). Es importante conocer los objetivos de la visualización y así poder escoger la técnica de mayor adaptación para obtener los resultados más apropiados.

Las técnicas de visualización se dividen en técnicas orientadas a píxel, puntos, líneas, símbolos, superficies y volúmenes y a su vez estas varían en cuanto al tipo de dato –escalares, vectoriales y tensoriales– que van a representar [7, 10]. Esta no es la única clasificación de las técnicas de visualización, existen otros autores que solo las dividen en cuanto al tipo de dato a representar

1.3.3.1. Algunas técnicas orientadas a superficies y volúmenes.

La representación de superficies o volúmenes es esencial a la hora de obtener una interpretación del objeto. Da una idea general de la forma de la figura, algo bien acogido por los usuarios finales de cualquier software de visualización.

Visualización mediante trazado de rayos (*Ray Tracing*)

Ray-Tracing es una técnica de rendereo que determina la visibilidad de la superficie trazando rayos de luz desde el observador al objeto, en la practica por cada píxel se envía un rayo desde la cámara o observador [5, 15]. El objeto puede ser una malla regular o un conjunto de primitivas (CSG). Este método conceptualmente simple y flexible. Existen una gran variedad de algoritmos para extender y mejorar el *ray-tracing* [11].

Mapeo de textura (*Texture maping*)

El “mapeo” de textura es una técnica orientada a superficies. Para aplicarla se debe definir un mapa, que define una imagen, de la superficie del objeto, este juega el papel de la textura en la superficie [5]. Es una poderosa forma de mejorar la riqueza visual del objeto resultante dado el gran nivel de detalle que se obtiene a través de las texturas. Se adquieren de imágenes reales escaneadas, por simples algoritmos o por métodos más elaborados basados en procesos estocásticos.

Técnicas basadas en puntos

La representación de superficies y las técnicas de rendereo basadas en puntos son muy populares. El punto es la más simple y fundamental entidad geométrica definida, como primitiva de representación es muy utilizada en los últimos tiempos [16]. El conjunto de datos considerado para dichas técnicas es una colección densa de puntos de la superficie del objeto organizados en diferentes tipos de estructuras –la más usada es el octree–. Un ejemplo de esta técnica es **qsplat.**

1.3.3.2. Introducción a las técnicas de visualización de datos escalares.

La representación de datos escalares es la más común. Las técnicas posibles a aplicar son bastantes diversas. A continuación se presenta una pequeña explicación de varias de ellas, para un mayor detalle véase Epígrafe 2.2.

Mapeo de colores (*Color Maping*)

El mapeo de colores es una de las técnicas más comunes por las facilidades que brindan las librerías gráficas en el trabajo con colores y sombras. Existen dos maneras básicas para asignar los colores a partir de valores escalares: una se basa en buscar en una tabla de colores y la otra en utilizar una función de transferencia para cada una de las componentes del color [10]. Dada la simplicidad y la importancia de los colores en la visualización, esta técnica es, a menudo, utilizada en conjunto con otras técnicas de visualización científica.

Líneas y superficies de contorno

A menudo se observa, fundamentalmente en mapas geográficos, líneas de colores que encierran determinadas regiones, esta es una técnica orientada a líneas para datos escalares en 2D/3D. Comúnmente se conoce como iso-líneas (*isoline*) para la representación en 2D y iso-superficies (*isosurface*) para espacios tridimensionales. Tanto las líneas de contorno como las superficies de contorno encierran regiones con valores constantes en 2D (planos) y 3D (superficies) respectivamente.[7, 10]

Coordenadas paralelas

Coordenadas paralelas *parallel coordinates* es una técnica orientada a líneas pero para datos escalares multidimensionales. Se tiene un conjunto de puntos

de n-dimensión, se sitúan n-ejes paralelos (*x1,x2,…,xn*), y cada punto P con coordenadas (*p1,p2,…,pn*) es representado con una línea de secuencia definida como la coordenada *pi* en el eje *xi* [5, 7, 17]. En la práctica consiste situar una línea vertical por cada dimensión de los datos, y unir cada valor de los puntos con líneas que a menudo son enriquecidas con color par mejorar la imagen..

Interpolación directa de colores

La interpolación directa de colores (*direct color interpolation*) sobre un objeto es una técnica derivada de *color maping.* Se basa en la asignación de colores en los vértices de un polígono, luego se interpolan hacia el interior. Solo es aplicada sobre datos escalares continuos. [5, 7]

Codificación de color por caras

La codificación de colores por caras (*element face color coding*) es una técnica semejante a la anterior, solo que los colores luego de asignarlos no se interpolan por la superficie. La asignación se consigue a través algún algoritmo matemático. En la mayoría de los casos se utiliza el promedio de todos los valores de los datos en la cara o también puede ser utilizado el mínimo o máximo valor. Esta técnica puede aplicarse a datos continuos o discretos indistintamente.

Representación mediante pequeñas figuras

Cuando los datos son escalares y tienen algún tipo de discontinuidad las técnicas anteriores no son aplicables. Una metodología seguida es la representación mediante subestructuras o pequeñas figuras similares con una codificación de color dentro de la estructura original [18]. Esta técnica es conocida por algunas bibliografías como *tiny cube* (pequeño cubo). Las

subestructuras son cubos, esferas u otras en dependencia de lo que se quiere lograr con la representación.

Cortes realizados al volumen

Esta técnica permite ver el interior de los cuerpos mediante cortes. En la práctica se intercepta un plano con el volumen y se obtiene como resultado dos o más cuerpos volumétricos. [5] Su utilización conjugada con otras técnicas permite al observador obtener información no solo de la superficie sino también del interior del objeto. Los cortes se realizan utilizando planos o algún tipo de superficie que permita extraer fragmentos o dividir en partes el cuerpo original.

1.3.3.3. Introducción a las técnicas de visualización vectorial.

El vector es uno de los tipos de datos más comunes utilizados para representar disímiles fenómenos tanto naturales como científicos. El vector representa magnitudes donde es importante considerar la dirección, el sentido, y en determinadas situaciones la longitud, características que lo asocian en gran medida con movimiento. Los datos vectoriales representan la dirección de la fuerza aplicada sobre un cuerpo o la dirección de la velocidad del viento o el agua. Es común su utilización en materias que trabajan con fluidos.

Representación mediante flechas (*arrow*)

Para la representación de datos vectoriales se utilizan flechas dada la similitud cogno-asociativa de este símbolo con la representación de un vector. Por cada vector en la base de datos se utiliza una flecha que comienza en el origen y orientada en la misma dirección de este. [10] Una base de datos de gran tamaño provocaría una utilización infructífera del método. La metodología a seguir entonces sería aplicar algún algoritmo de filtrado. Se pueden asociar colores a las flechas para enriquecer el dibujo y hacer la compresión más fácil. Esta

técnica para espacios tridimensionales no es totalmente factible pues entorpece el entendimiento de la posición y orientación de los vectores.[5, 10]

Líneas de fluido o tramas de desplazamiento (*streamlines*)

Muestra el movimiento de partículas tanto en el interior como exterior del cuerpo. Esta técnica permite la representación continua de campos vectoriales, lo que evita la interpolación mental al aplicar la técnica basada en flechas. La idea consiste en dibujar mediante líneas la trayectoria de las partículas en el medio. Es más aplicable en espacios de dos dimensiones pues para un número mayor de dimensiones la representación de las trayectorias y su integración se hace más compleja y deben ser manejadas con cuidado para evitar errores.

1.4. Herramientas utilizadas.

Escoger herramientas para el desarrollo de un software siempre es un proceso polémico. Cada una tiene sus ventajas y desventajas y se debe hacer un estudio mesurado antes de cometer errores.

Se necesita un lenguaje de programación potente y multiplataforma capaz de comunicarse con alguna herramienta de programación gráfica portable. Nuevas tecnologías como .Net y Mono-Project han aparecido con una gran cantidad de facilidades. Su utilización en cualquier tipo de proyecto es muy común.

Luego de estudiar de cada una de las herramientas disponibles se decidió como lenguaje de programación el C#, cuyo compilador genera código para .Net. La herramienta gráfica utilizada fue OpenGl. GTK-Sharp, extensión de GTK para .Net, fue el instrumento escogido para la implementación de la interfaz gráfica. (Véase Epígrafe 3.1)

1.5. Estado del arte.

Una vez convencidos de los beneficios de la visualización científica grandes compañías se han dado la tarea del desarrollo de software para la visualización de datos científicos e ingenieriles. IBM (International Business Machines), Kitware (Compañía productora de software de visualización), SGI (Silicon Graphics, Inc), el departamento de desarrollo e investigación científica Departament for Scientific Visualization of Honrad-Zuse-Zentrum für infromation stechnik Berlin (ZIB), entre otros llevan la vanguardia en el campo de la visualización científica.

A continuación se mencionarán varios softwares producidos por algunas de estas compañías y algunas especificaciones de sus características.

1.5.1. OpenDX

Sistema de visualización de propósito general, es la versión libre del programa de visualización IBM Data Explorer (DX). Originalmente desarrollado por IBM para computadoras RS/6000. DX es un entorno de programación visual (similar a otras aplicaciones RAD) para procesar datos y generar visualizaciones multidimensionales. Las facilidades que brinda este software son realmente grandes. Requiere para su funcionamiento computadores con gran cantidad de memoria RAM para trabajar con conjuntos de datos grandes y/o visualizaciones de media-alta resolución.

Soporta una buena gran cantidad de formatos internacionales de almacenamiento de datos. Debido a la generalidad de su modelo de datos y a la flexibilidad de la programación visual OpenDX no esta dirigido a alguna rama de la ciencia en particular. Proporciona técnicas de visualización científica para la representación de datos escalares y vectoriales en 2D y 3D tales como mapeo de colores, líneas de contorno, superficies de contorno, líneas de fluido, representación mediante símbolos, entre otras. [19]

Este software facilita la incorporación de nuevos módulos para la visualización científica por el usuario, realizados mediante pequeños ficheros escritos en C. Es una de las propuestas para poder representar nuestros datos utilizando la característica antes descrita.

1.5.2. Amira.

Es un sistema de visualización y modelación de datos en espacios tridimensionales. Permite visualizar conjuntos de datos científicos provenientes de áreas como medicina, biología, química, física e incluso ingeniería. Amira es un software modular orientado a objetos; para la visualización y algunas operaciones computacionales son usados módulos independientes. Es un software licenciado, cada paquete es agregado de forma independiente que requiere de una licencia separada.

Facilita al usuario una amplia cantidad de técnicas de visualización científica: rendereo de volúmenes, superficies de contorno, cortes y reconstrucción de volúmenes y líneas de contorno para una gran cantidad de formatos de datos. Este es un software con muchas facilidades pero la principal desventaja es que no es libre. [20]

1.5.3. Iris Explorer

Es un programa de visualización de datos en 3D de altas prestaciones que permite realizar animaciones y manipulación de datos. Incluye una extensa librería de módulos de visualización. Las aplicaciones son desarrolladas en forma de red o mapa de módulos. La principal desventaja de este software es que no es de distribución libre.

Este software permite visualización, análisis y animaciones 3D. Utiliza toda la potencia gráfica de OpenGL. Incluye un gran número de módulos/rutinas

predefinidas Es fácilmente extensible, mediante nuevos módulos implementados por el usuario. Presenta una amplia gama de técnicas de visualización.

1.5.4. Advanced Visual Systems (AVS)

El AVS es un paquete de la visualización científico basado en el paradigma de programación visual de redes de flujo de datos. Proporciona una herramienta de desarrollo visual que le permite al usuario construir módulos reusables para visualizar, analizar, manipular y desplegar los datos. Por otra pueden ser escritos en C, C++ o FORTRAN.

Permite la visualización de datos escalares y vectoriales. Contiene una gran variedad de técnicas de visualización científica para el rendereo de volúmenes, extracción de volúmenes, visualización de superficies de contorno, líneas de contorno, líneas de fluido, entre otras.

También existen otros softwares de propósito general muy aceptados dentro de la comunidad de visualización científica como el Persistence of Vision Raytracer (PovRay). El XD3D es otro software de visualización, quizás no tan potente como los anteriores, pero sencillo, libre y multiplataforma. Otros softwares utilizados para la visualización científica utilizados por ingenieros son el MatLab y el Matematic, utilizados en el grupo para la visualización del empaquetamiento del MED.

1.6. Conclusiones parciales.

Una vez analizados diversos temas relacionados con la visualización científica se puede arribar a las siguientes conclusiones:

- La visualización facilita la comprensión de mundos abstractos, simulados, modelados y reales.

- La visualización científica se ha convertido en una práctica de mucho interés y aplicación. El desarrollo de software de visualización no se ha quedado al margen, los ingenieros y científicos cuentan con softwares de muchas capacidades – MatLab, Opendx, Amira, etc. – sin embargo estos son de propósito general o en muchos casos requieren del pago de licencias para su uso.
- Las nuevas formulaciones del Método de Elementos Distintos identifican diferentes comportamientos (continuo, discreto o mixto) de los campos en la superficie e interior de las estructuras generadas.
- Se hace necesario el desarrollo de nuevos softwares que sean capaces de representar el comportamiento continuo, discontinuo y mixto de las partículas resultado de las nuevas formulaciones del Método de Elementos Distintos.

- La selección de las técnicas a emplear es un proceso importante para la obtención de la representación final de los resultados y debe realizarse teniendo en cuenta las características de los datos.
- El desarrollo de cualquier aplicación trae consigo un estudio profundo de las herramientas a utilizar.

Capítulo II Obtención de los datos y técnicas de visualización científica.

Como se ha podido apreciar los datos juegan el papel fundamental en el proceso de visualización. Para lograr una efectiva visualización se necesitan datos precisos aunque estos no sean reales y procedan de simulaciones numéricas. A pesar de la calidad de los resultados de los equipos de medición la simulación es un mecanismo poderoso que cada día se hace más común. Antes de abordar el tema de la visualización y sus técnicas se profundizará en la generación del medio mediante el MED.

2.1. Generación de los datos. Método de elementos distintos.

Con mayor frecuencia el método de elementos finitos se queda estrecho para muchos tipos de problemas. Se han desarrollado nuevos métodos, entre ellos el MED.

El MED emplea una serie de nodos distribuidos por todo el medio, obtenidos mediante la generación de una nube de punto. La generación de esta nube inicial de puntos implica un costo computacional menor que el de la construcción de una malla [21].

Utilizando los resultados del cálculo de modelos numéricos convencionales (diferencias finitas, volumen finito, elementos finitos) es posible obtener de forma aproximada el campo de cierta magnitud sobre un sólido. En los métodos tradicionales, esta magnitud es continua, o en todo caso presenta discontinuidades aisladas.

Los métodos de partículas permiten reproducir el comportamiento natural discontinuo de ciertas propiedades físicas en el medio. Por ejemplo, los parámetros de movimiento de las partículas aisladas en un gas, el campo de

tensiones en un mortero sometido a carga, o el grado de ionización en una macro-molécula con radicales múltiples. También existen fenómenos donde se aprecian magnitudes con comportamiento mixto, como el campo de velocidades en distintas partes de la sección transversal de un tubo.

Los métodos de partículas resuelven múltiples problemas de ingeniería. En ellos se emplean diversos tipos de partículas, las más generalizadas son las esféricas por su simplicidad geométrica y por la forma en que se pueden detectar las vecinas y el contacto entre ellas. Estos métodos se clasifican según su aplicación en dos grandes grupos: Macroestructurales y Microestructurales, aspecto que lo define el tamaño de las partículas que conforman el medio.

En el enfoque macroestructural del MED, cada cuerpo geométrico de la simulación tiene un símil físico. Este enfoque se emplea básicamente para modelar medios discontinuos, donde cada cuerpo de la simulación representa una partícula de un material granular. Mientras que en el enfoque microestructural el cuerpo geométrico no tiene, por lo general, un símil físico y se emplea para modelar medios continuos. [22]

En [22, 23] se tienen dos formulaciones diferentes del MED. Las primitivas preferidas son las esferas, sin embargo, a partir de ellas es posible obtener una gran variedad de formas construyendo conjuntos de esferas considerados como un cuerpo rígido En estos momentos se trabaja en el empaquetamiento de partículas como fase previa del MED utilizando estructuras no esféricas: polígonos y poliedros, teniéndose ya algunos resultados concretos. [24, 25] (Véase Figura 4)

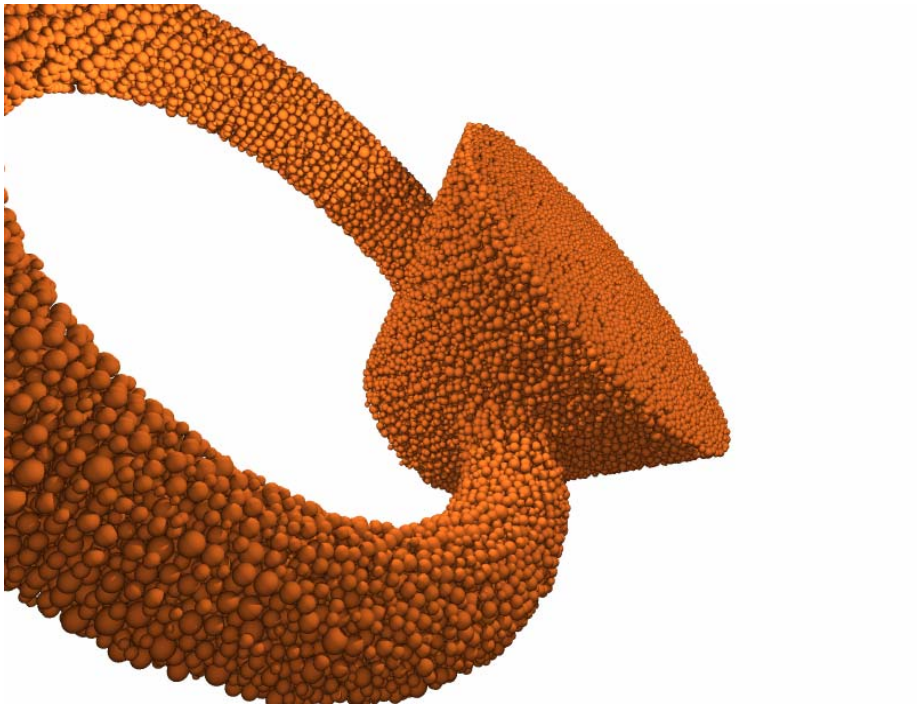

Figura 4 Objeto geométrico resultado del empaquetamiento mediante esferas del MED

2.2. Especificación de las técnicas de visualización científica.

Las técnicas de visualización son la base de toda herramienta de visualización. Como resultado del MED se obtiene un comportamiento bastante cercano a la realidad atendiendo a las propiedades físicas de las partículas obtenidas a partir de interacciones entre ellas. Este epígrafe se encarga de profundizar en algunas de las técnicas mencionadas en los epígrafes 1.3.3.2 y 1.3.3.3 utilizadas para la representación de propiedades físicas en sólidos generados mediante el MED.

2.2.1. Técnicas de visualización aplicadas sobre datos escalares.

La mayoría de las técnicas de visualización existentes pueden ser aplicadas a datos escalares siempre dependiendo del comportamiento de estos, ya sea escalares continuos, discontinuos o mixtos. A continuación se explican en detalle cada una de las técnicas implementadas.

2.2.1.1. Mapeo de colores.

El mapeo de colores es una sencilla y conocida técnica de visualización escalar que consiste en corresponder cada valor con un color. Esta asignación es implementada buscando dentro de una tabla de colores (*color lookup table*). El mapeo a través de la tabla de colores se realiza de la siguiente forma: la tabla contendrá un arreglo de colores y los valores máximo y mínimo de los valores escalares que serán mapeados. El índice dentro de la tabla de colores para cualquier valor escalar v se busca según la siguiente formula donde n se corresponde con la cantidad de colores:

$$i = \begin{cases} 0, v \leq \min \\ n-1, v \geq \max \\ n\left(\dfrac{v-\min}{\max-\min}\right) \end{cases}$$

Una forma más eficaz es la utilización de una función de transferencia para cada una de las componentes del color (RGBA Red Green Blue Alpha). Este método reduce algunos problemas que puede traer consigo la utilización de la tabla de colores (cambios bruscos de coloración). Estos cambios se deben a que la tabla de colores no es más que una representación discreta de una función de transferencia.

En el proyecto se utilizó una función de transferencia para cada una de las componentes de color obteniéndose resultados favorables. Al utilizar una función de transferencia para cada una de las componentes del color se obtiene una interpolación suave de colores (Véase Figura 5). Esta función no es necesario conocerla de forma explícita, tan solo conociendo el color para valores específicos es posible obtener una correcta interpolación.

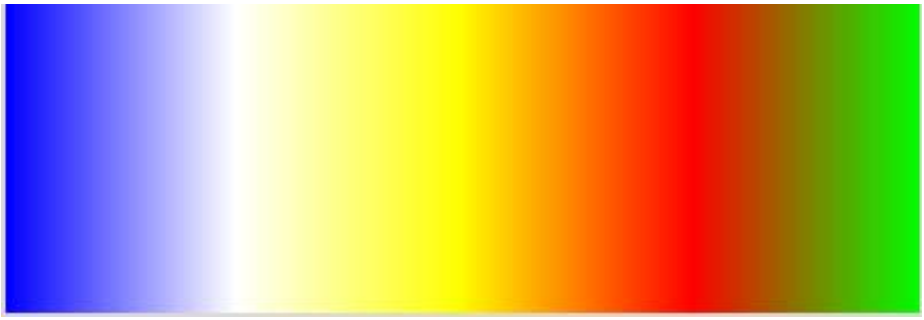

Figura 5 Interpolación de colores utilizando una función de transferencia para cada una de las componentes del color.

El mapeo de colores es una técnica que a menudo es combinada con otras técnicas de visualización. La interpolación directa de colores y la codificación de colores por caras son dos ejemplos de su utilización. (Véase Figura 6)

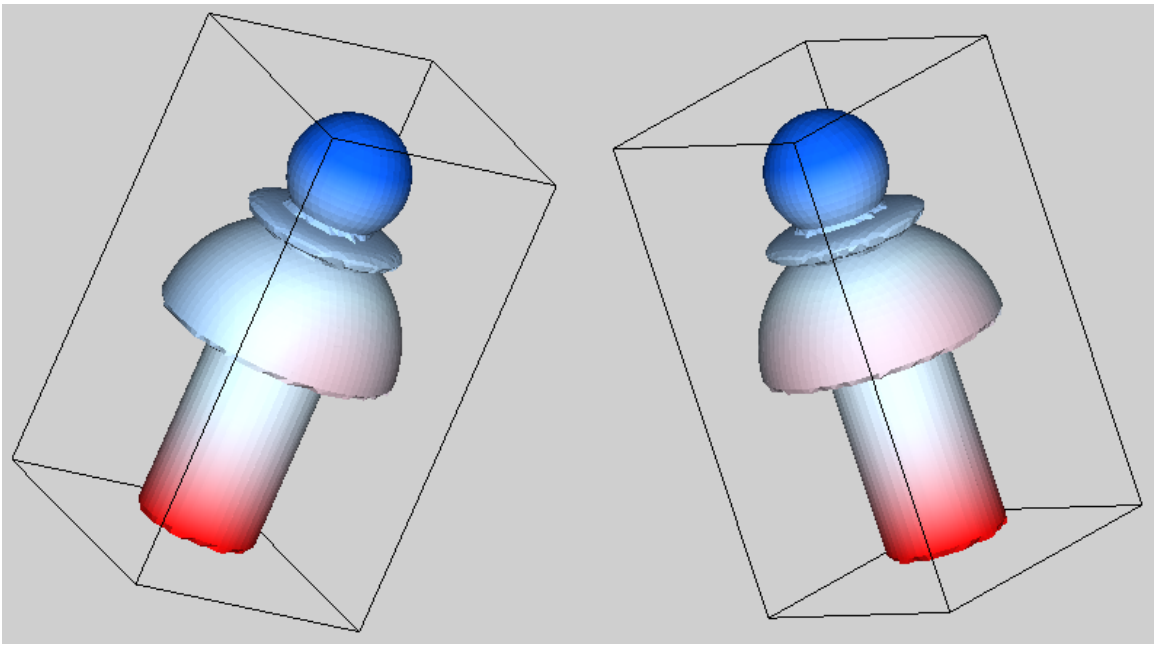

Figura 6 Uso del mapeo de colores para dibujar el comportamiento de la función en la superficie del cuerpo aplicando la técnica de Interpolación de Colores

2.2.1.2. Líneas de contorno

Otra extensión del mapeo de colores son las iso-líneas (líneas de contorno). Son utilizadas para representar regiones con determinados valores constantes en la superficie de un objeto. La línea representa la frontera entre las regiones

$F(x) < c$ y $F(x) > c$ o sea donde $F(x) = c$, sea c un valor de contorno y x un punto de n-dimensiones del conjunto de datos. [10] (Véase Figura 9)

Existen varios métodos para obtener las iso-líneas, la mayoría trabajan sobre discretizaciones del espacio. El espacio es dividido en cuadrados aunque en algunos casos se usan otros tipos de figuras pero el procedimiento para la obtención de las líneas sería semejante.

Un método consiste en encontrar una celda y rastrear las demás buscando a través de las celdas vecinas hasta llegar a ella misma o que no quede ninguna otra en la base de datos. Este procedimiento se repite para cada una de las líneas de contorno. Otro método, mucho más utilizado que el anterior, esta basado en la conocida técnica de “divide y vencerás” considerando las celdas de manera independientes. Esta técnica es conocida en la bibliografía como *marching squares*. (Véase Figura 8)

El método *marching squares* considera que cada línea pasa por una celda en un número finito de formas (2^4 posibles combinaciones). Haciendo un análisis de las combinaciones posibles se puede prescindir de varias de ellas por simetría para tan solo quedarnos con 4 casos de los 16 originales. [10] (Véase Figura 7).

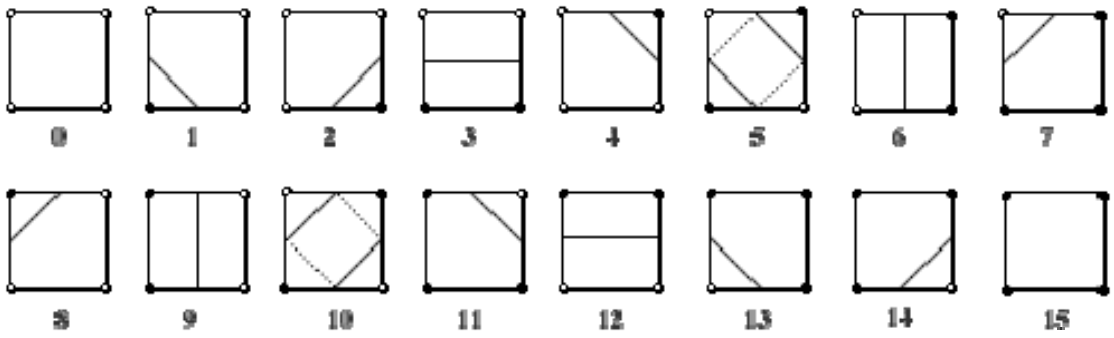

Figura 7 Casos considerados en la técnica "marching squares" los cuales representan las formas en que una línea puede interceptar una celda. Los casos 5 y 10 son ambiguos

En esta figura anterior se hace mención a ciertas ambigüedades que se presentan durante la obtención de las iso-líneas. La forma de tratarlas es sencilla pues para cada ambigüedad la cantidad de posibles soluciones es pequeña.

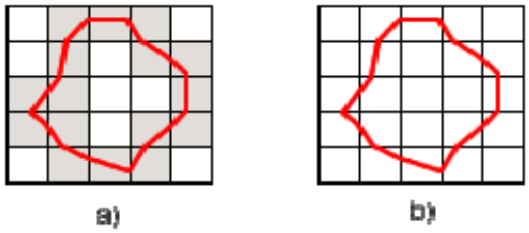

Figura 8 Obtención de las líneas de contorno sobre una cuadrícula. a) Cuadricula con las intercepciones antes de obtener las líneas de contorno. b) líneas de contorno.

Esta técnica es aplicada sobre cuerpos bidimensionales. En este proyecto se toma la malla como el espacio sobre el cual serán calculadas y luego dibujadas las líneas. Las líneas de contorno son utilizadas fundamentalmente en mapas geográficos refiriéndose a la temperatura y relieve. Esta técnica puede ser extendida a cuerpos tridimensionales y es conocida como superficies de contorno.

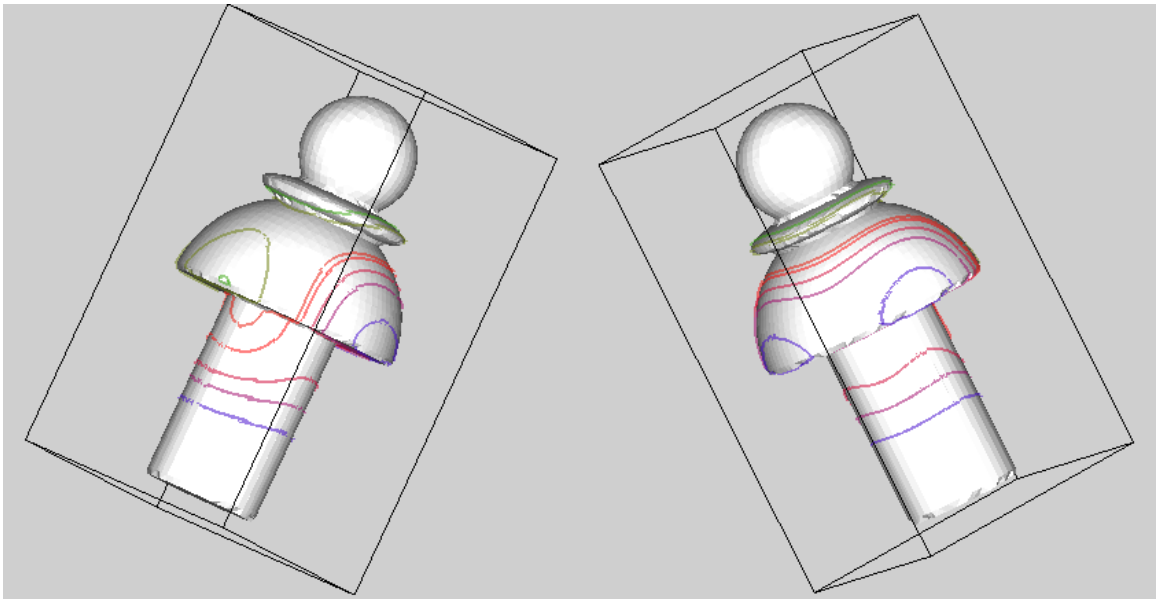

Figura 9 Líneas de contorno.

2.2.1.3. Superficies de contorno

La técnica iso-líneas es aplicada a espacios tridimensionales, pero en lugar de obtenerse una línea el resultado es una superficie. De manera similar, la superficie esta formada por todos los puntos que cumplen con $F(x) = c$. (Véase Figura 11)

La idea es la misma, el espacio es particionado en cubos, figura geométrica tridimensional, fácil de manipular y de gran aceptación. Este procedimiento es conocido como *marching cubes*; es una extensión del algoritmo *marching squares* pero para espacios tridimensionales. Al aplicar este método se obtiene una malla de superficie. La cantidad de combinaciones posibles para este algoritmo es 2^8 pero por simetría pueden ser reducidas a solo 15 casos. (Véase Figura 10a) [10]

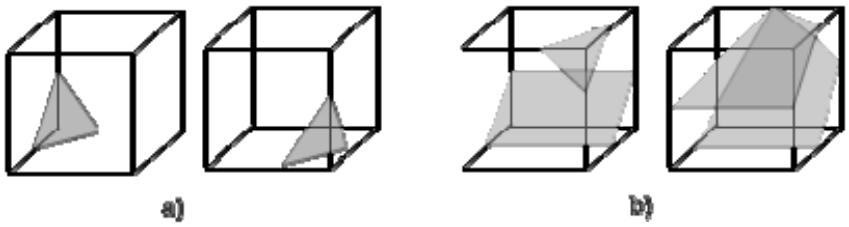

Figura 10 a) Dos combinaciones posibles en el algoritmo "marching cubes" que pueden ser reducidas a una por simetría. b) Uno de los casos ambiguos y una posible solución (caso complementario).

En la figura anterior se muestra dos combinaciones reducidas a una por propiedades de simetría y una de las ambigüedades presentes a la hora de aplicar la técnica. Resolver estas ambigüedades en espacios tridimensionales no es tan simple como en el caso bidimensional.

En [10, 26, 27] se plantean varios métodos para solucionar dichas ambigüedades. La metodología sugerida en [10] es la teselación de cubos

mediante tetraedros usando una técnica llamada *marching tetrahedra*. Este método genera inconvenientes ya que produce una gran cantidad de triángulos y requiere hacer una orientación previa de los tetraedros lo que puede provocar abolladuras en la superficie. Otra metodología consiste en evaluar el comportamiento asintótico y escoger los casos en que se puede romper o mantener la superficie. Este método es llamado *asymptotic desider* y es ampliamente abordado en [26]. Consiste en tratar cada una de las ambigüedades con casos complementarios. La solución utilizada en este proyecto es utilizando otro método que también utiliza casos complementarios los cuales son detallados en el Apéndice I.

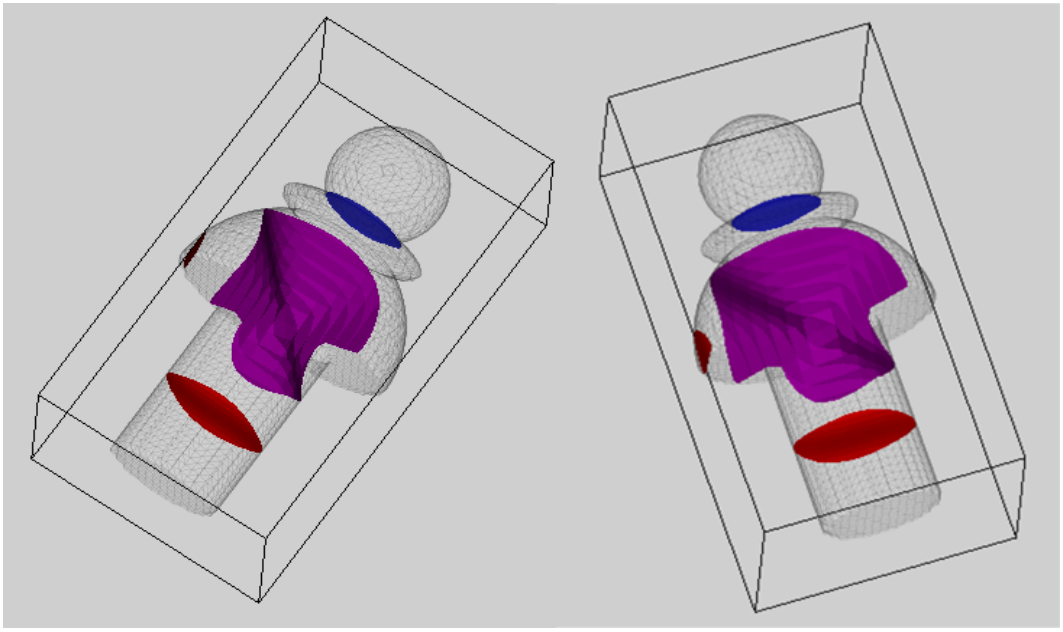

Figura 11 Iso-superficies.

2.2.1.4. Representación mediante pequeñas figuras: Cubos y Polígonos.

La representación de datos escalares discretos no es un tema muy abordado en lo referente a la visualización científica. Se hizo una búsqueda tanto de aplicaciones como artículos relacionados con el tema y no se obtuvieron resultados satisfactorios, solo se hace referencia a la representación mediante

pequeños cuerpos o figuras (en la mayoría de los casos esferas, las cuales son mucho más sencillas a la hora de la manipulación).

En este trabajo se desarrollaron dos algoritmos para la representación de datos escalares discretos. La idea es la misma: utilizar figuras para representar los puntos discretos. Las figuras usadas fueron cubos y polígonos, los cuales simbolizaran los puntos discretos en el interior y exterior del cuerpo respectivamente.

Uno de los algoritmos consiste en simbolizar con cubos las discontinuidades en la misma posición en que se encuentran, aplicándole cierta transparencia al objeto principal. El otro algoritmo, un poco más complejo, radica en llevar estas discontinuidades a la superficie del objeto principal representándolas con polígonos. Además se desarrolló una combinación de ambos algoritmos para obtener una imagen que permita obtener mayor cantidad de información.

La representación de cubos en el interior del cuerpo, específicamente en la misma posición en que se encuentra el punto discontinuo, no presenta una alta complejidad tanto algebraica como computacional. Consiste en generar un cubo en la misma posición en que se encuentra el punto a visualizar. Las coordenadas de la esquina inferior izquierda del cubo coinciden con las coordenadas del punto y a partir de un radio, se generan los demás vértices.

Para colorear la superficie puede ser utilizada cualquier técnica estándar de visualización de datos continuos, interpolación directa de colores o codificación de colores en las caras de la malla, aplicándole cierta transparencia a la superficie para poder ver la imagen y todos los cubos en su interior. Esta técnica permite un mayor realismo puesto que el especialista o ingeniero observa las discontinuidades en el lugar exacto donde se encuentran. Tiene la desventaja de que en conjuntos de datos con una gran cantidad de discontinuidades no se

observe con claridad pues pueden originarse solapamientos entre los cubos. En este caso seria necesario algún algoritmo de filtrado. (Véase Figura 12)

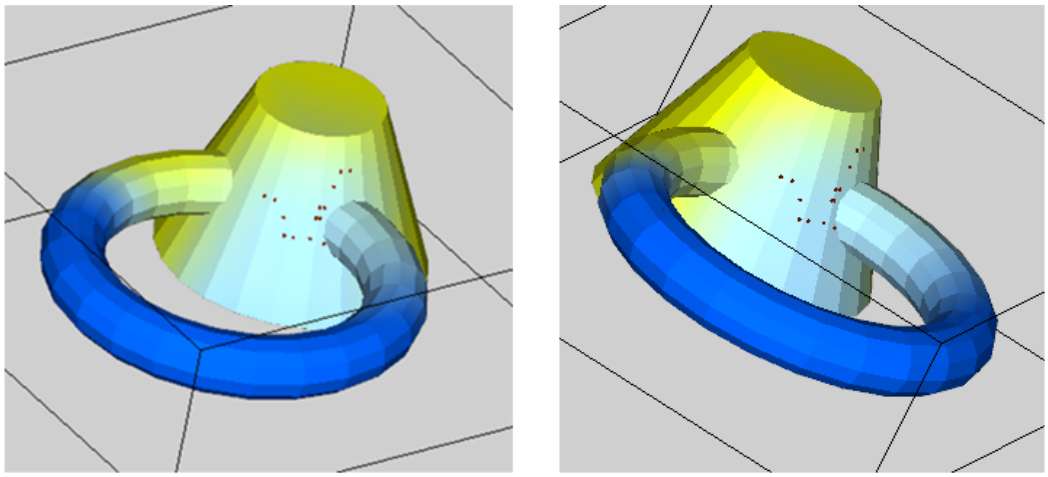

Figura 12 Representación de datos discretos mediante polígonos regulares

El otro algoritmo de visualización de datos escalares discontinuos desarrollado consiste en trasladar las discontinuidades a la superficie del cuerpo. Este método es mucho más costoso que el anterior pero tiene la ventaja de que al llevar las discontinuidades al exterior del cuerpo se tiene una idea de la forma y la posición con respecto a la superficie. La idea reside en dibujar polígonos de n lados en las caras del objeto primario, específicamente, de las caras más cercanas al punto discontinuo, la que le queda de frente. Entiéndase como la cara que queda de frente a aquella que contiene el plano que al proyectar el punto, la proyección está dentro de la cara.

El algoritmo consiste en buscar, por cada punto que representa una discontinuidad, el vértice de la malla más próximo. Puesto que este algoritmo es bastante costoso se utilizó una estructura de particionado del espacio para organizar los vértices de la malla (Véase Epígrafe 3.3.1). La búsqueda dentro de esta estructura se hace utilizando el algoritmo del vecino más cercano implementado utilizando el kdtree. (Véase Algoritmo IX).

Una vez encontradas todas las caras que contienen este vértice, las cuales son las candidatas a contener el polígono que simbolice el punto discontinuo, se buscan las que quedan completamente de frente, para dibujar definitivamente los polígonos. Para encontrar estas caras se proyecta el punto en cada uno de los planos que representan y se busca cual de estas proyecciones pertenece a la cara que dio lugar al plano correspondiente Estas serían las que quedan de frente al punto que representa la discontinuidad.

Una vez encontradas todas las caras en las cuales se va a dibujar se generan los polígonos que van a simbolizar las discontinuidades en el cuerpo. Dada la cantidad de lados, el radio y el centro del polígono se generan todos los vértices exteriores utilizando rotaciones vectoriales (Véase Algoritmo V), tanto el radio como la cantidad de lados son especificados por el usuario, el centro coincide con la proyección del punto en el plano. La importancia de este algoritmo viene dada en la interpretación que el usuario pueda obtener de la imagen resultante. Como los polígonos son solo dibujados en las caras más cercanas que quedan de frente se obtiene una idea de la posición de las discontinuidades con respecto a la superficie del cuerpo. (Véase Figura 13)

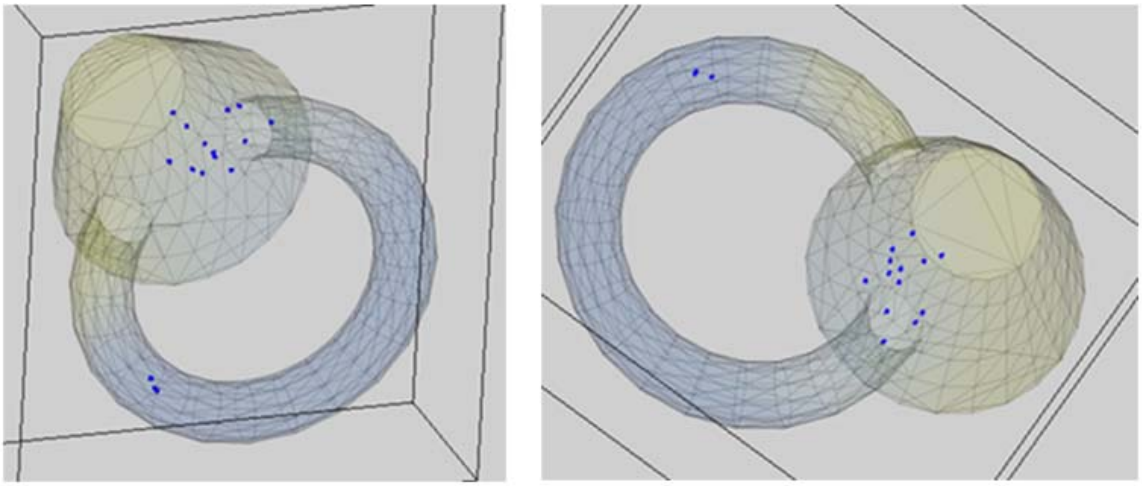

Figura 13 Representación mediante cubos.

Estas dos técnicas son combinadas para aumentar sus capacidades y obtener mayor cantidad de información. (Véase Figura 14)

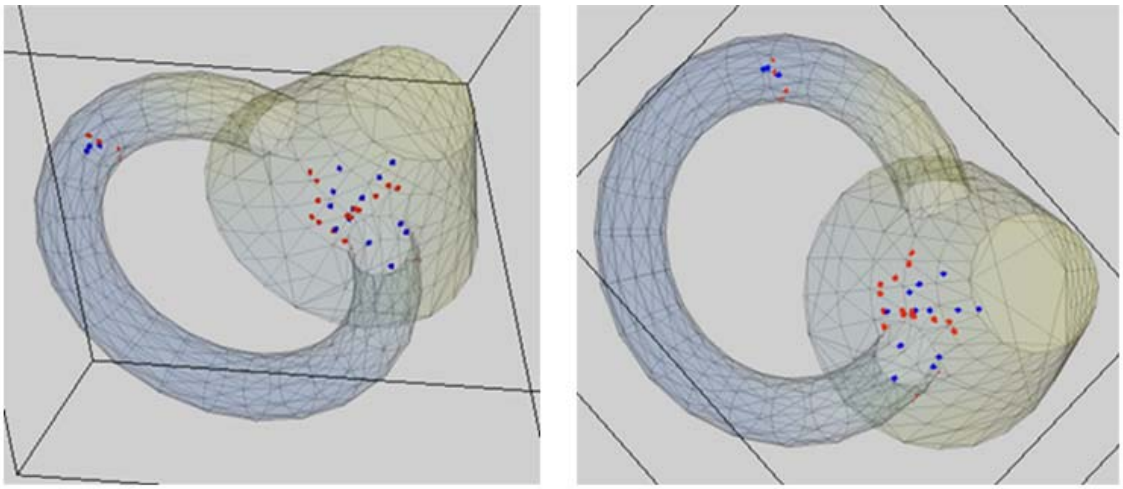

Figura 14 Combinación de las representaciones de discontinuidades mediante cubos (azul) y polígonos (rojo).

2.2.2. Técnicas de visualización aplicadas sobre datos vectoriales.

Los vectores son segmentos con magnitud y sentido. Una visualización natural para datos vectoriales consiste en dibujar una línea por cada vector en la base de datos. La línea comenzaría en el origen del vector y estaría orientada en la dirección que indican de sus componentes. [10]

Es una técnica base para la mayoría de las técnicas de visualización aplicadas sobre datos vectoriales. Una variación sencilla es la utilización de flechas o algún otro símbolo derivado en lugar de líneas debido su relación cogno-asociativa. Conjuntamente puede ser aplicado algún color tanto a las líneas como a las flechas dependiendo de la magnitud del vector.

Los vectores muchas veces son asociados con el movimiento (velocidad o desplazamiento). Estas propiedades son simbolizadas mediante líneas continuas. Esta técnica es conocida como líneas de fluido (*streamlines*). Existen

otras representaciones semejantes con algunas variaciones conocidas como *streaklines* y trayectorias de partículas.

La técnica utilizada en este trabajo es líneas de fluido. Se corresponden con integrales a través de una curva s que satisface la ecuación $s = \int Vds$ con $s = s(x,t)$ para un tiempo particular t. La integración de la línea de fluido se utilizó Runge-Kutta de orden 4 definido a continuación. Se escoge como cabeza para la evaluación un punto cualquiera en el interior del cuerpo. (Véase Figura 15)

$$k_1 = hf(x)$$
$$k_2 = hf(x + k_1/2)$$
$$k_3 = hf(x + k_2/2)$$
$$k_4 = hf(x + k_3)$$
$$x' = x + k_1/6 + k_2/3 + k_3/3 + k_4/6$$

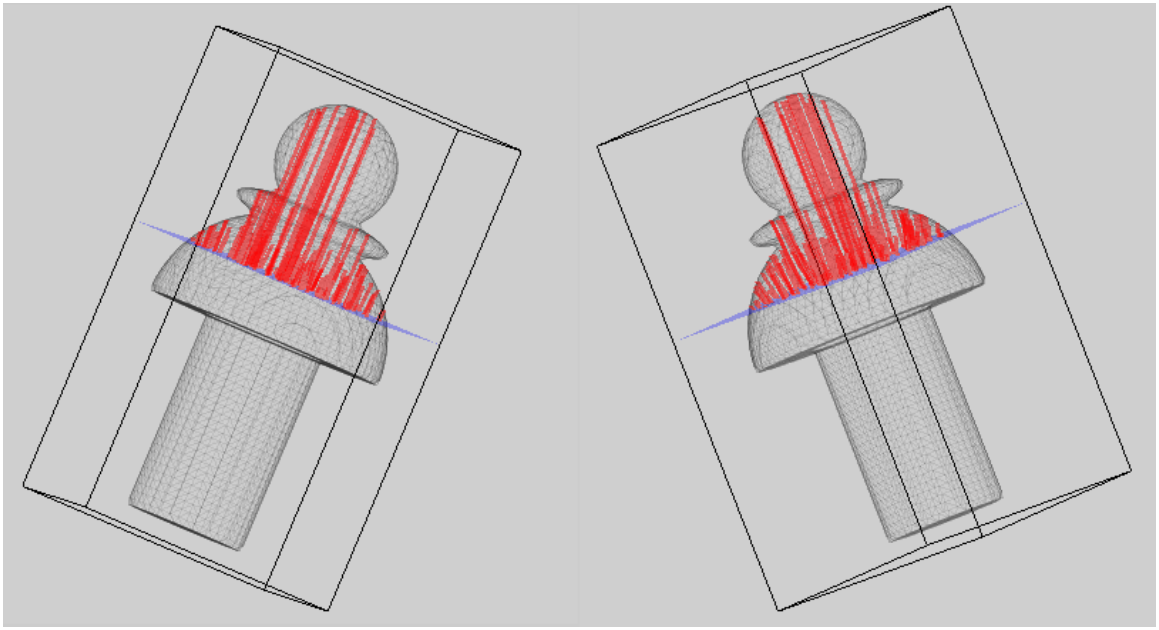

Figura 15 Líneas de fluido.

2.3. Conclusiones parciales.

Luego de dilucidar cada una de las técnicas de visualización aplicadas se arribó a las siguientes conclusiones:

- El MED permite obtener el comportamiento natural de las partículas.
- Cada técnica de visualización científica está diseñada con propósitos específicos.
- Para poder hacer un uso correcto de las técnicas deben ser comprendidas completamente pues se puede caer en errores a la hora interpretar la imagen obtenida.
- Muchas veces estas técnicas no son suficientes para satisfacer las necesidades reales de cualquier proyecto y es necesario combinarlas o hasta diseñar nuevas.
- Se utilizaron técnicas como la interpolación directa de colores, líneas de contorno, superficies de contorno, líneas de fluidos y representación mediante figuras.
- Se desarrollaron dos variantes de la técnica para datos escalares discretos descrita: representación mediante cubos y polígonos en el interior y el exterior del cuerpo respectivamente. Además se combinan para obtener mayor cantidad de información.

Capítulo III Implementación y desarrollo de DataVis (Data Visualization)

A continuación se exponen cada una de las herramientas de desarrollo y una visión general de la aplicación respecto a la ingeniería de software. *DataVis*[5] es una aplicación que se basa fundamentalmente en primitivas geométricas y predicados de geometría computacional. En esta sección se especifican el conjunto de primitivas, estructuras de búsqueda espacial y almacenamiento de datos más algunos algoritmos básicos utilizados en el trabajo.

3.1. Herramientas de desarrollo.

A continuación se explican cuáles y por qué se escogen cada una de las herramientas utilizadas en el desarrollo de la aplicación.

3.1.1. Lenguaje de programación. Plataforma .NET y Mono Project

Decidirse por un lenguaje de programación es siempre una tarea difícil y que trae contradicciones a cualquier programador debido a las ventajas de cada uno de los existentes en la actualidad.

Durante los últimos 20 años, C y C++ han sido los lenguajes elegidos para desarrollar aplicaciones comerciales y de negocios. Proporcionan un altísimo grado de control al programador permitiéndole el uso de punteros y muchas funciones de bajo nivel. Sin embargo, cuando se comparan lenguajes, como Microsoft Visual Basic con C y C++, uno se da cuenta de que aunque C y C++ son lenguajes más potentes, se necesita más tiempo para desarrollar una aplicación.

[5] DataVis: aplicación que se diseñó.

Con la idea de buscar un lenguaje rápido para desarrollo pero que a la vez permita interactuar con las tecnologías clásicas usadas desde C y C++ se escogió el C#. Este un lenguaje de programación simple, moderno, derivado de C; completamente orientado a objetos y con un sistema de tipos seguro. Este lenguaje combina la alta productividad de Microsoft Visual Basic y la eficacia bruta de C++. [28]

La iniciativa .NET Framework de Microsoft supone un cambio importante en la metodología de desarrollo de software. Este entorno está construido usando una arquitectura que permite a los lenguajes de desarrollo de software trabajar juntos, compartiendo recursos y código, para proporcionar a los programadores las avanzadas herramientas necesarias para construir aplicaciones de escritorio y de Internet. La plataforma .NET se compone de cuatro partes: un entorno común de ejecución, un conjunto de bibliotecas de clases, un grupo de lenguajes de programación y el entorno ASP.NET. Una herramienta relativamente nueva dirigida al desarrollo de .NET es el lenguaje de programación C#. Con la utilización de esta plataforma se logran aplicaciones mucho más estables y se simplifica el trabajo. Por otra parte, surge Mono Project, proyecto que brinda el software necesario para desarrollar y ejecutar aplicaciones .Net en plataformas Windows, OS X, Unix, Linux y Solaris. Es un proyecto de código abierto y cuenta con una comunidad numerosa de desarrolladores por lo que se encuentra en constante crecimiento.

3.1.2. OPENGL (Open Graphics Library).

OpenGL es un estándar creado por Silicon Graphics en el año 1992 para la representación gráfica de diseños en 2D/3D. Está formado por un conjunto de APIs (*Application Programming Interface*) que actúan como interfaz para el hardware gráfico y permiten obtener imágenes de alta calidad. Actualmente es una de las tecnologías más empleadas en el diseño de aplicaciones 3D. Cuenta con la facilidad de tener implementaciones libres y propietarias en múltiples

plataformas. OpenGL fomenta el desarrollo de aplicaciones con incorporación de rendereo de grandes conjuntos de datos, mapeo de texturas, efectos especiales y otras poderosas funciones de visualización [1, 29-33].

OpenGL tiene dos propósitos principales:

- Ocultar la complejidad de la interfaz con las diferentes tarjetas gráficas, presentando al programador una API única y uniforme.
- Ocultar las diferentes capacidades de las diversas plataformas de hardware, requiriendo que todas las implementaciones reporten el subconjunto de funcionalidad de OpenGL que implementan.

La operación básica de OpenGL es aceptar primitivas tales como puntos, líneas y polígonos, y convertirlas en píxeles. Este proceso es realizado por una tubería gráfica conocida como la Máquina de Estados de OpenGL.

OpenGL se divide en tres partes funcionales [1, 29-32]:

- Librería OpenGL: proporciona todo lo necesario para acceder a las funciones de dibujado básico de OpenGL.
- Librería GLU (OpenGL Utility Library): librería de utilidades que proporciona acceso rápido a algunas de las funciones más comunes de OpenGL a través de la ejecución de comandos de más bajo nivel.
- GLX (OpenGL Extension to the X Window System): proporciona funciones para interactuar con un sistema de ventanas X Window. Está incluido en la propia implementación de OpenGL (su equivalente en Windows es la librería WGL, externa a la implementación de OpenGL).

Además de estas tres librerías, la librería GLUT (OpenGL Utility Toolkit) proporciona una interfaz independiente de la plataforma para crear aplicaciones de ventanas totalmente portables. Existen interfaces de lenguaje para OpenGl en un gran número de ellos: Java, C, C++, C# y VB. Tao es la interfaz de OpenGL para programación en C# utilizando las plataformas Mono y .NET.

3.1.3. Sistema de ventanas (Gtk-Sharp).

GTK es un grupo de bibliotecas o rutinas para desarrollar interfaces gráficas de usuario (GUI). GTK es la abreviatura de ***G****IMP* ***t****ool****k****it* (conjunto de rutinas para GIMP). Inicialmente fue creado para desarrollar el programa de manejo de imágenes GIMP por lo que esta diseñada encima de GDK, sin embargo actualmente es muy usada por muchos otros programas en los sistemas GNU/Linux. Esta licenciada usando la licencia LGPL, por lo que puedes desarrollar software abierto, software libre, o incluso software no libre.

GTK esta basada en tres bibliotecas: [34]

- GLib: biblioteca de bajo nivel, estructura básica de GTK y GNOME. Proporciona manejo de estructura de datos para C, portabilidad, interfaces para funcionalidades de tiempo de ejecución "*runtime*" como ciclos, hilos, carga dinámica o un sistema de objetos (gobject).
- Pango: biblioteca para el diseño y rendereado de texto; hace hincapié especialmente en la internacionalización. Es el núcleo para manejar las fuentes y el texto de GTK+2.
- ATK: biblioteca para crear interfaces con características utilizadas por personas discapacitadas o minusválidas. Pueden usarse útiles como lupas de aumento, lectores de pantalla, o entradas de datos alternativas al clásico teclado o ratón de ordenador.

Básicamente GTK es una interfaz orientada a objetos para programadores de aplicaciones. Aunque está escrita completamente en C, está implementada usando la idea de clases y funciones de retro-llamada[6] . Actualmente existen interfaces de GTK para programar en una amplia gama de lenguajes entre ellos C#. GTK-Sharp es la extensión a GTK para programación en la plataforma .Net. con C#.

[6] retro-llamada: *callback*, punteros a función

GTK, además de ser software abierto, presenta muchas más ventajas que sus similares. El *Backend* gráfico de GTK (Cairo) es más universal que el de Winform (GDI+) por solo compararla con una librería similar. Este es uno de los motivos por lo que muchos programadores se deciden a usar esta herramienta.

3.1.4. Generación de la documentación (Doxygen).

Muchas veces se encuentran fragmentos de código realmente conflictivos a la hora de interpretar lo que hacen, sin embargo si este código estuviese comentado sería mucho más fácil entenderlo y rehusarlo. Documentar código es un proceso tedioso pero es un proceso que, a la larga, va a evitar muchos problemas y que, en la medida de lo posible es conveniente adoptar y hacer que forme parte de del estilo corriente de programación, haciendo que los comentarios y la documentación sean una parte tan habitual de la programación como crear un bucle.

Muchos programadores y compañías se han dado a la tarea de desarrollar softwares capaces de generar documentación de manera automática a partir de ficheros de código fuente. Softwares como el NDoc, Doxygen, ROBODoc, Doc-O-Matic, Natural Doc, entre otros son algunos de los más conocidos.

Doxygen genera documentación en cierta medida para ficheros escritos en C# y su uso es bastante sencillo. Fue el software escogido para dicho fin. Genera un navegador de documentación *on-line* (en HTML) y/o un manual de referencia *off-line* para un conjunto de documentos de ficheros fuente. Además soporta la generación de ficheros RTF (MS-Word), Postscript, PDF y HTML comprimido. La documentación se extrae directamente de los ficheros fuente, lo que hace que sea más fácil mantenerla consistente con el código fuente. Puede ser configurado para extraer la estructura del código de ficheros fuentes no documentados. La documentación que se anexa al trabajo fue completamente generada con esta herramienta.

3.2. Diseño general de DataVis.

La programación por capas es un estilo de programación en la que el objetivo primordial es la separación de la lógica de negocios de la del diseño. La ventaja principal es que el desarrollo se puede llevar a cabo en capas independientes, permitiendo que en caso de algún cambio sólo se modifique la capa requerida sin tener que revisar y modificar todo el código. A lo largo de este epígrafe se abordará este tema. En lo que respecta a la implementación solo se explicaran con detalle aquellas clases que se consideran esenciales o que constituyan un patrón para las demás.

El diseño de mayor auge es el de tres capas: la capa de presentación o usuario, la capa de negocio o lógica y la capa de datos.

- Capa de Presentación: la que ve el usuario, Se comunica solamente con la capa del negocio y se encarga de notificar cualquier evento ocurrido. Se corresponde con una interfaz gráfica programada en GTK-Sharp.
- Capa de Negocio: encargada de enlazar la capa del usuario con la capa de datos. Cualquier solicitud del usuario es procesada haciendo peticiones a la capa de datos y luego enviando la respuesta al usuario. Se corresponde con un sistema de clases implementadas en C#.
- Capa de Datos: encargada de almacenar toda la información con que se va a trabajar en el sistema. Solo se comunica con la capa lógica. Se corresponde con un sistema de clases implementadas en C#.

Se comenzará explicando el funcionamiento del proyecto por las clases encargadas de almacenar los datos. Vale recordar que se procesarán datos continuos, discontinuos y mixtos; además se hizo una extensión para poder visualizar cuerpos generados por el MED. Se debe aclarar ademas que muchas de estas clases forman parte de otro proyecto por lo que contienen atributos y

métodos que son más relevantes para este. A continuación (Diagrama 1) se muestran las clases principales que soportan los datos en la aplicación.

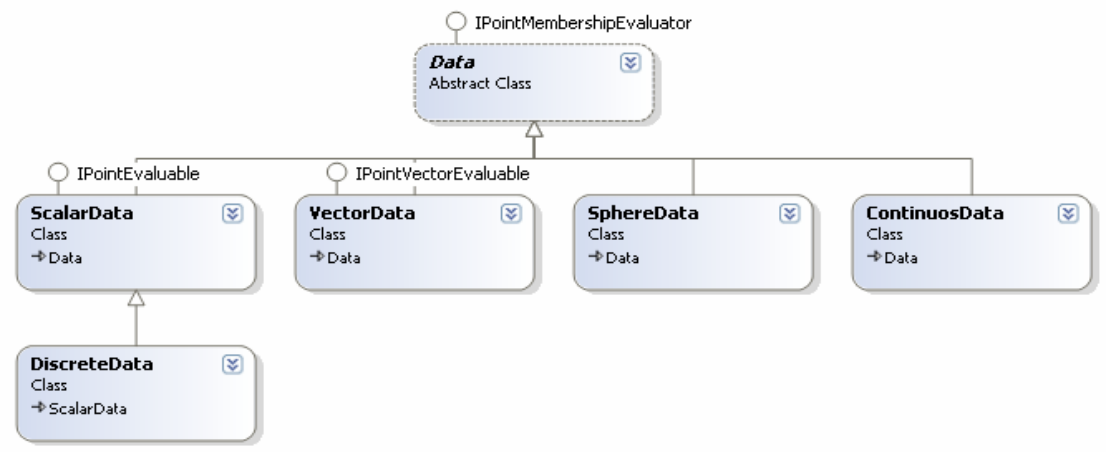

Diagrama 1 Clases encargadas de almacenar información

El diagrama anterior muestra una clase principal, **Data**, de la que heredan varias clases **VectorData**, **ContinuousData**, **SphereData**, **ScalarData** y a su vez de la clase **ScalarData** hereda la clase **DiscreteData**. Este es un diagrama orientado a objetos simple donde existe una clase base de la cual heredan las demás. El símbolo adherido en la parte superior de algunas clases indica que ellas implementan alguna interfaz, a la derecha del símbolo se especifica su nombre. Ej: la clase **ScalarData** implementa la interfaz **IPointEvaluable**.

Se implementó otro sistema de clases encargadas del funcionamiento del software las cuales forman la llamada capa de negocio, que para una mejor comprensión son divididas en tres grupos:

- El grupo integrado por aquellas clases encargadas de todo el calculo de geometría computacional de la aplicación, **Point_3D**, **Vector**, **Line**, **Polygon**, **Triangle**, **Plane**, **Cube**, entre otras. Para las especificaciones de estas clases y algunos de sus algoritmos implementados véase el epígrafe 3.2.2
- El grupo que implementa cada una de las técnicas de visualización utilizadas, entiéndase ello como los cálculos necesarios para obtener los

datos a representar, entre ellas están **TinyPolygon**, **Isolines**, **Isosurface**, **StreamLines**, **DirectColorInterpolation**, entre otras.

- El conjunto de clases encargadas de visualizar los datos. Todas las clases que pertenecen a este grupo implementan la interfaz *IDrawable,* entre ellas están **ExteriorPoligonDrawer**, **CubeDrawer**, **MeshDrawer**, **LinesDrawer**, **SphereDrawer**, entre otras.

La capa de usuario es la interfaz con la cual el usuario va a interactuar. Consiste en un conjunto de clases, que implementan cada uno de los componentes utilizados. Estos son desarrollados utilizando los *widgets* de GTK-Sharp. A continuación se expondrán cada uno de estos grupos. Para una completa documentación véanse los anexos al trabajo.

3.2.1. Clases contenedoras de datos.

La clase **Data** es el ente principal para el almacenamiento de los datos en la aplicación, es una clase abstracta. Esta clase será explicada con detalle por la importancia que encierra para el proyecto y además porque constituye un patrón para las demás clases. (Véase Diagrama 1).

La función de esta clase es almacenar datos. Sus atributos son: **fmesh**, objeto de tipo **Mesh** y **freaderfile**, objeto de tipo **TextReader**, encargado de leer los datos desde un fichero. Contiene dos métodos esenciales, un método virtual **isIn,** encargado de determinar la interioridad de un punto dentro de las estructuras que contiene la clase y otro método, **outerCube**, que calcula el cubo exterior que circunscribe el objeto principal. (Véase Diagrama 2)

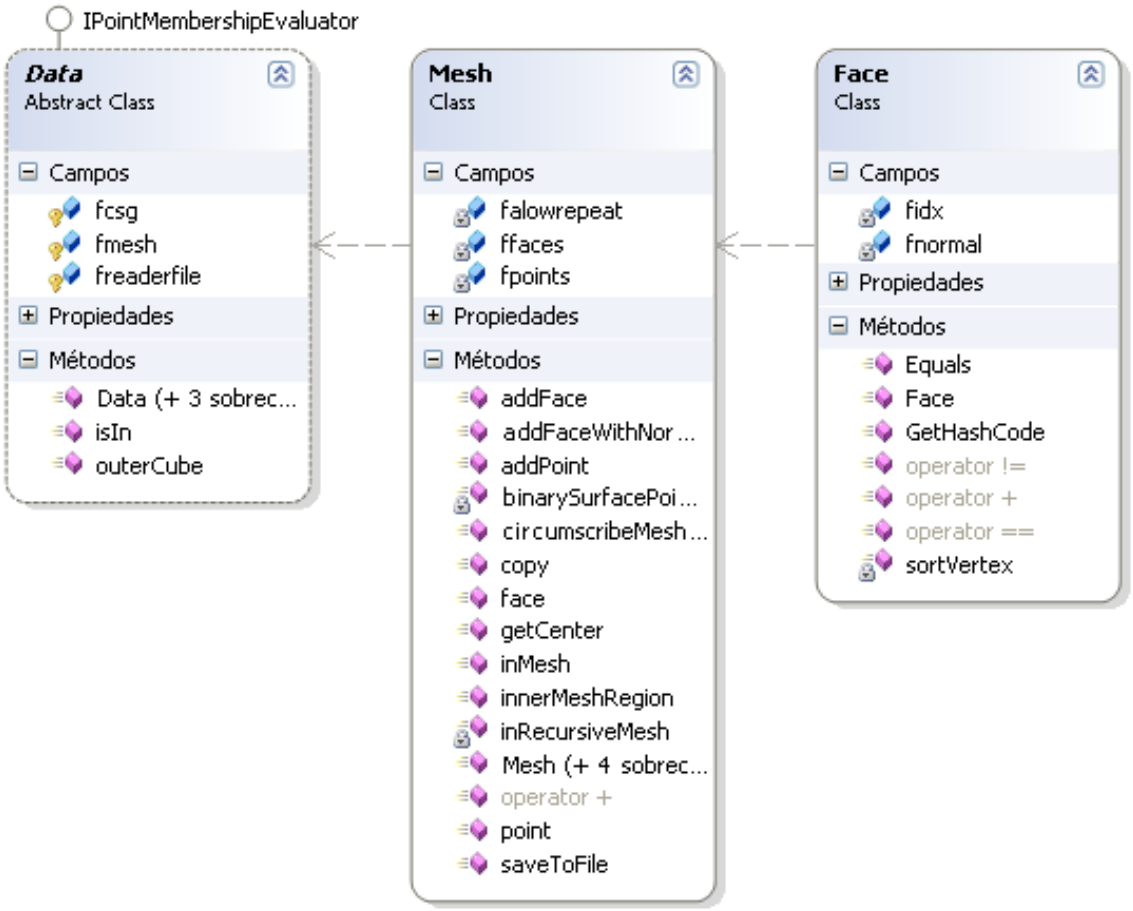

Diagrama 2 Vista detallada de las clases Data, Mesh y Face.

El elemento esencial de esta clase (Data) es la malla, estructura fundamental dentro del proyecto; sobre los datos contenidos en ella se realizan la gran mayoría de las operaciones de visualización. Sus atributos principales son: **fpoints:** lista de todos los vértices de las caras y f**ases:** lista de todas las caras, como las caras son triángulos esta lista es de tripletas de índices a los puntos.

Los métodos más importantes en la clase **Mesh** son:

- **addFace:** permite adicionar una cara a la malla.
- **getCenter:** calcula en centro geométrico de la malla.
- **innerMeshRegion:** calcula un cubo exterior a la malla.
- **inMesh:** permite encontrar la interioridad de un punto en la malla. Para una mayor explicación de este método véase Epígrafe 3.4

3.2.2. Algunas primitivas básicas y algoritmos implementados.

Las primitivas geométricas constituyen la base del desarrollo de este trabajo. A continuación se expondrán con cierto nivel de detalle basándose en tres aspectos: su definición, las operaciones que pueden realizar y algunos predicados de interés relacionados.

3.2.2.1. Punto.

El punto es un ente geométrico fundamental que cumple, según Hilbert, con los axiomas de incidencia, orden, congruencia, paralelismo y continuidad para un espacio geométrico euclídeo. No puede ser definido a partir de otras primitivas. En un espacio n-dimensional a un punto se le asigna una n-tupla sobre el campo escalar de números reales. [23]

La distancia entre dos puntos es uno de los cálculos notables concernientes al punto, se deduce utilizando el teorema de Pitágoras. Está dada por la siguiente ecuación:

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ...(x_n - y_n)^2} \qquad (1)$$

3.2.2.2. Recta.

La recta es el lugar geométrico del subespacio lineal generado por un vector y un punto. La representación paramétrica viene dada precisamente por este enunciado. Los puntos que forman una recta que pasa por el punto p_0 en la dirección del vector v vienen dados por:

$$p = p_0 + t \cdot v \qquad (2)$$

Entre los predicados de interés y que involucran al punto y a la recta está la distancia entre ambos, la cual está dada por la menor distancia, definida según (1), entre el punto y todo los puntos de la recta. Otro predicado utilizado en este

proyecto se refiere a la proyección de un punto en una recta. (Véase Algoritmo I,Figura 16).

Algoritmo: Proyectar un punto en una recta

```
proyectar punto en recta (punto p)
{
    construir vector u con origen en un punto de la recta y
dirección hacia el punto p
    hacer  r = producto escalar entre u y v
    si r = 0
        si el punto es interior(u y v proporcionales)
            retornar p
        sino
            retornar p0
    sino
         retornar p0 + r * v
}
```

Algoritmo I Proyección de un punto en una recta

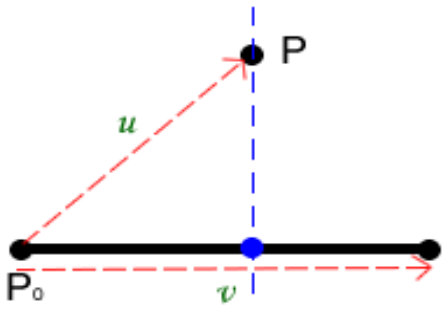

Figura 16 Proyección del punto P sobre la recta definida por el punto Po y el vector v

3.2.2.3. Vector.

En algunas bibliografías se define el vector solamente como un “segmento orientado” pero dicha definición no es lo bastante rica para entender su verdadero concepto y puede traer consigo ciertas confusiones. Un vector es un segmento de cierta longitud (denominada módulo), al cual se le asignan propiedades adicionales como la dirección y sentido.

Las magnitudes vectoriales están asociadas con una n-tupla en dependencia de la dimensión del espacio. Sobre los vectores geométricos se definen operaciones matemáticas tales como suma, resta, multiplicación vectorial, multiplicación escalar, multiplicación por un escalar, entre otras.

Sean los vectores $x = (x_1, x_2, x_3)$ e $y = (y_1, y_2, y_3)$ sobre los cuales se definen las siguientes operaciones:

- Suma y resta:

  $$x + y = (x_1 + y_1, x_2 + y_2, x_3 + y_3)$$

  $$x - y = (x_1 - y_1, x_2 - y_2, x_3 - y_3)$$

- Producto vectorial

  $$x \times y = (x_2 \cdot y_3 - x_3 \cdot y_2, x_3 \cdot y_1 - x_1 \cdot y_3, x_1 \cdot y_3 - x_2 \cdot y_1)$$

- Producto escalar

  $$x \cdot y = (x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3)$$

- Multiplicación por un escalar

  $$\alpha \cdot x = (\alpha \cdot x_1, \alpha \cdot x_2, \alpha \cdot x_3)$$

- Módulo

  $$\|x\| = \sqrt{x_1^{\,2} + x_2^{\,2} + x_3^{\,2}}$$

Rotar un vector en el espacio no es un procedimiento simple, esa característica adherida a los vectores fue desarrollada por medio de los cuaterniones. (Véase Algoritmo II)

Algoritmo: Rotar vector en el espacio

```
rotar vector (cuaternion h)
{
      hacer hc =  conjugada de h
      hacer q = h * v * hc
      retornar vector construido a partir del quaternion q
}
```

Algoritmo II Rotación de un vector en el espacio

3.2.2.4. Plano.

El plano es otro de los entes geométricos fundamentales al igual que el punto y la recta. Suele ser definido a partir de otras primitivas. Para un espacio de dos dimensiones, se supondrá que existe un solo plano que contiene todos los puntos del espacio. En un espacio de tres dimensiones, la definición del plano se introduce usando conceptos de secciones anteriores.

Dado cierto punto p_0 y un vector v, existe un plano asociado a ellos formado por todos los puntos $x = (x, y, z)$ tales que:

$$v \cdot x - p_0 = 0 \qquad (3)$$

El plano es uno de los elementos de mayor connotación para el desarrollo de este trabajo. En el plano se desarrollan operaciones tales como la proyección de un punto (Véase Algoritmo III, Figura 17a) y su relación con una recta. (Véase Algoritmo IV, Figura 17b).

Algoritmo: Proyectar un punto en un plano

```
proyectar punto en plano (punto p)
{
     hacer vector v a partir de un  punto del plano y el punto p
     hacer r = producto escalar entre el vector  v y la normal del
plano
     retornar p - r * n
}
```

Algoritmo III Proyección de un punto en un plano.

Algoritmo: Relación entre una recta y un plano

```
relación recta-plano(recta r)
{
     calcular a = producto escalar entre la normal del plano y un
vector punto que pertenece a la recta
     calcular b = producto escalar entre la normal del plano y un
punto que pertenece al plano
     si b = 0
          retornar la recta es paralela o coplanar
     sino
          retornar el punto en el que cada una de las componentes es
la suma de las componentes  de un punto que pertenezca a la recta
con la multiplicación del cociente de a y b y las componentes de un
vector que pase por la recta.
}
```

Algoritmo IV Relación entre una recta y un plano. La recta puede ser coplanar, paralela al plano o interceptarlo en un punto.

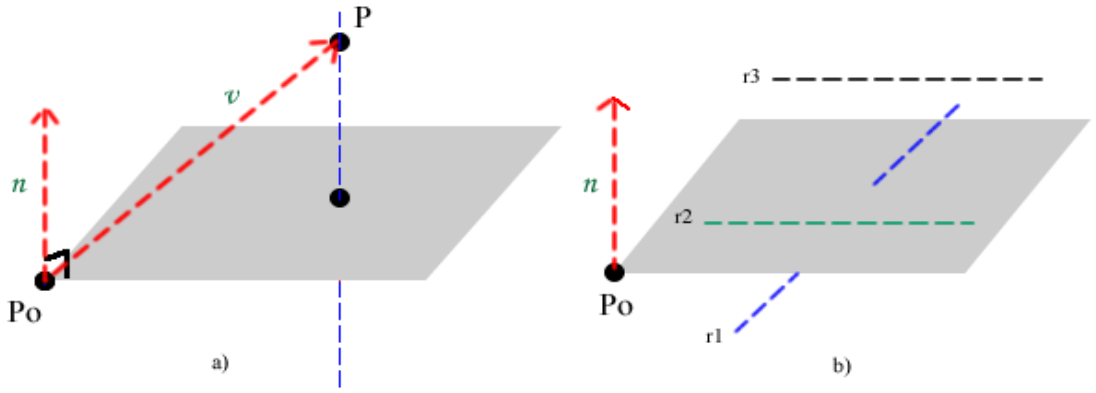

Figura 17 a) Proyección de un punto en un plano. b) Relación entre una recta y un plano: r1 corta al plano en un punto, r2 corta al plano en infinitos puntos (coplanar) y r3 no se intercepta con el plano (paralela).

3.2.2.5. Polígono.

Un polígono es una figura geométrica plana limitada por segmentos rectos consecutivos no alineados llamados lados. La palabra proviene del griego poly "muchos" y gonos "ángulos". En este trabajo solo se utilizan polígonos regulares ya que no es necesario un alto nivel de detalle a la hora de la visualización.

El polígono regular queda definido por un punto p (centro del polígono), un radio r (longitud del centro a cada uno de sus vértices) y la cantidad de los lados. Los vértices del polígono son generados dinámicamente mediante los cuaterniones y su conveniente notación para representar rotaciones (Véase Algoritmo V).

Algoritmo: Construcción de polígonos regulares.

```
construir polígono(vector v)
{
      calcular el eje alrededor del que se va a rotar(u)
      calcular ángulo de rotación  alpha = 360/n
      construir cuaternión h = cos(alpha/2) + u * sen(alpha/2)
      desde i = 0 hasta n hacer
      {
            rotar v
            agregar el punto obtenido a la lista de puntos
      }
      retornar lista de puntos
}
```

Algoritmo V Obtención de los puntos que determinan los lados del polígono a partir de rotaciones vectoriales.

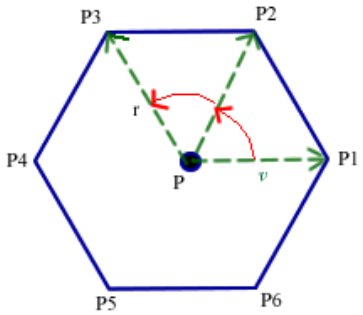

Figura 18 Polígono de 6 lados generado a partir de rotaciones de vectoriales.

3.2.2.6. Triángulo.

El triángulo es un polígono de tres lados. Es el elemento fundamental de las mallas de superficies utilizadas en este trabajo y es parte esencial de los predicados de interioridad en estas estructuras. El triángulo esta definido por los puntos que determinan sus vértices.

El predicado de mayor connotación es el que se encarga de determinar la relación entre un punto y un triángulo, el algoritmo utilizado es conocido como *Barycentric Coordinate*. Este algoritmo es fundamental en la representación de puntos discretos en la superficie de un objeto. (Véase Algoritmo VI, Figura 19). Otro algoritmo de relevancia es el que determina la relación de una recta y un triangulo en un espacio tridimensional. (Véase Algoritmo VII)

Algoritmo: Relación punto-triángulo

```
punto interior(punto p)
{
      Calcular vectores
      hacer v0 = C - A
      hacer v1 = B - A
      hacer v2 = P - A
      Calcular producto escalar de todos contados
      hacer dot00 = producto escalar entre v0 y v0
      hacer dot01 = producto escalar entre v0 y v1
      hacer dot02 = producto escalar entre v0 y v2
      hacer dot11 = producto escalar entre v1 y v1
      hacer dot12 = producto escalar entre v1 y v2
      Calcular coordenadas del baricentro
      hacer invDenom = 1 / (dot00 * dot11 - dot01 * dot01)
      hacer u = (dot11 * dot02 - dot01 * dot12) * invDenom
      hacer v = (dot00 * dot12 - dot01 * dot02) * invDenom
      retornar verdadero si u > 0 y v > 0 y u + v < 1
}
```

Algoritmo VI Técnica "Barycentric Coordinate" para determinar la interioridad de un punto en un triángulo.

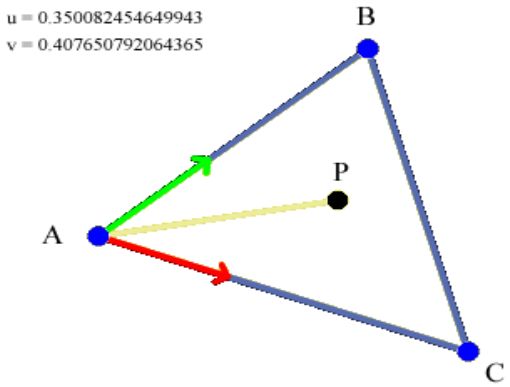

Figura 19 Relación entre el punto P y el triángulo determinado por los puntos A, B y C utilizando la técnica "Barycentric Coordinate".

Algoritmo: Intercepción de una línea y un triángulo

```
intercepción triangulo-recta(linea l)
{
        construir plano a partir del triangulo
        encontrar el punto de intercepción entre la recta y el plano
        determinar si ese punto es interior o exterior al triángulo
        si el punto es interior
            retornar la recta intercepta el triángulo
        sino
            retornar la recta no intercepta el triángulo
}
```

Algoritmo VII Intercepción entre una recta y un triángulo.

3.2.2.7. Cubo.

El cubo o hexaedro regular es un poliedro de seis caras cuadrados congruentes. Es uno de los llamados sólidos platónicos. Un cubo, además de ser un hexaedro, puede ser clasificado también como paralelepípedo recto y rectángulo, pues todas sus caras son de cuatro lados y paralelas dos a dos, e incluso como un prisma de base cuadrada y altura equivalente al lado de la base.

En este trabajo el cubo esta definido por un punto que representa la esquina frontal inferior izquierda y la longitud de un lado. El cubo es, junto a los polígonos, el elemento principal de la representación de datos escalares discretos y continuos a trozos.

3.2.2.8. Cuaternión.

Los cuaterniones, en matemática, son una extensión no conmutativa de los números complejos. No es un término reciente, fueron descritos por primera vez en 1843 por el irlandés William Rowan Hamilton y aplicados a transformaciones geométricas en tres dimensiones. Aunque en determinados trabajos

matemáticos teóricos o prácticos son sustituidos por vectores tienen especial importancia en aplicaciones que involucren rotaciones. [35]

En álgebra moderna los cuaterniones están formados por cuatro componentes reales y pueden expresarse como: $H = \{a + bi + cj + dk : a, b, c, d \in \Re\}$

Sean los cuaterniones $x = a_1 + b_1 i + c_1 j + d_1 k$ e $y = a_2 + b_2 i + c_2 j + d_2 k$. Las operaciones algebraicas quedan definidas de la siguiente forma.

- Suma y resta:

  $x + y = (a_1 + a_2) + (b_1 + b_2)i + (c_1 + c_2) j + (d_1 + d_2)k$

  $x - y = (a_1 - a_2) - (b_1 - b_2)i - (c_1 - c_2) j - (d_1 - d_2)k$

- Multiplicación: se realiza de forma similar al producto de polinomios teniendo en cuenta:

  $i^2 = j^2 = k^2 = ijk = -1$

  $ij = k,\ jk = i, ki = j$

  $ji = -k, kj = -i, ik = -j$

- Multiplicación por un escalar: $\alpha \cdot x = (\alpha \cdot a_1 + \alpha \cdot b_1 i + \alpha \cdot c_1 j + \alpha \cdot d_1 k)$

- Módulo: $x = a_1 - b_1 i - c_1 j - d_1 k$

Los cuaterniones presentan una notación conveniente para representar orientaciones y rotaciones. Característica que les confiere una especial importancia en la computación gráfica, robótica, navegación, satélites, entre otras. [36]

Sea $q = xi + yj + zk$ un punto (o un vector) del espacio, u un vector unitario del mismo espacio y θ un número real. La rotación alrededor del eje $(0, u)$ de un ángulo θ envía el punto q sobre el punto $q' = x'i + y'j + z'k$ dado por:

$q' = h \cdot q \cdot \tilde{h}$ donde $h = \cos(\theta/2) + u \cdot sen(\theta/2)$ [7]

[7] h: cuaternion unitario

Diagrama 3 Vista detallada de las clases mencionadas.

3.2.3. Clases encargadas de la visualización.

Se implementaron clases cuya única función fuese la de calcular los datos para la visualización. Estas son las clases más importantes dentro de la aplicación ya que constituyen el paso intermedio entre los datos y el proceso de dibujado;

desarrollan cada una de las técnicas de visualización científica utilizadas. Entre ellas: **TinyPolygon**, **StreamLines**, **Isosurface, Isoline,** entre otras.

Siguiendo la filosofía anterior de tan solo explicar algunas clases e inferir que las demás son semejantes se explicará la clase **TinyPolygon.** Su objetivo es hacer los cálculos correspondientes para obtener los cubos y los polígonos que serán dibujados en el interior y exterior del cuerpo primario respectivamente. Sus atributos principales son: **fdata** (datos discretos) y **fsize** (tamaño del lado de los cubos). (Véase Diagrama 4)

Los métodos de esta clase se encargan de obtener los datos para la visualización de los cubos y los polígonos, **getCubes** y **organizePointInFace** respectivamente.

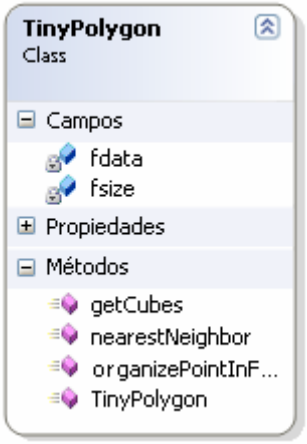

Diagrama 4 Vista detallada de la clase TinyPolygon.

Una vez obtenidos los datos para la visualización se pasará al proceso de dibujado. Siguiendo el mismo orden se hablará de las clases encargadas de dibujar los cubos y los polígonos, **CubeDrawer** y **ExteriorPolygonDrawer** respectivamente.

Los atributos de estas dos clases se corresponden con los datos obtenidos a partir de la clase **TinyPolygon** y valores asociados a la forma en que serán representados: tamaño, color, visibilidad, opacidad, entre otros. Además

implementan la interfaz *IDrawable* por lo que solo necesitan implementar el método **draw** que es el encargado de dibujar (Véase Diagrama 5)

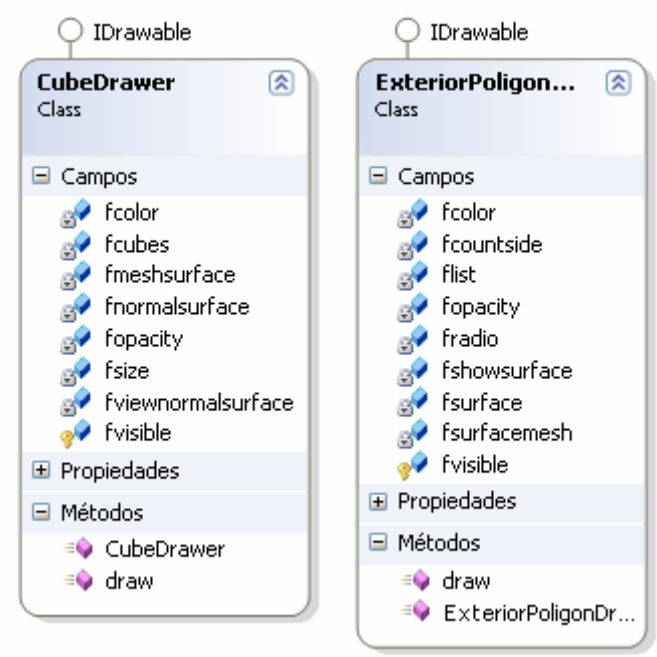

Diagrama 5 Vista detallada de las clases CubeDrawer y ExteriorPolygonDrawer.

3.2.4. Interfaz de usuario.

Se desarrolló utilizando la biblioteca GTK-Sharp. Se implementó un paquete de clases **CustomWidgets** que contiene cada uno de los componentes visuales utilizados. Entre estos nuevos elementos y los widgets de la biblioteca prima una relación de herencia para darles la funcionalidad deseada. Se usó la extensión para OpenGL de GTK (**GTKGLExt**) que permite dibujar, utilizando OpenGL, en los controles de GTK. La interacción del usuario con la escena se realiza por medio de los eventos del teclado. Las propiedades gráficas de la visualización se especifican y controlan por medio de OpenGL.

A continuación se muestra un pequeño diagrama con algunas clases pertenecientes a GTK-Sharp y clases implementadas en **CustomWidgets**:

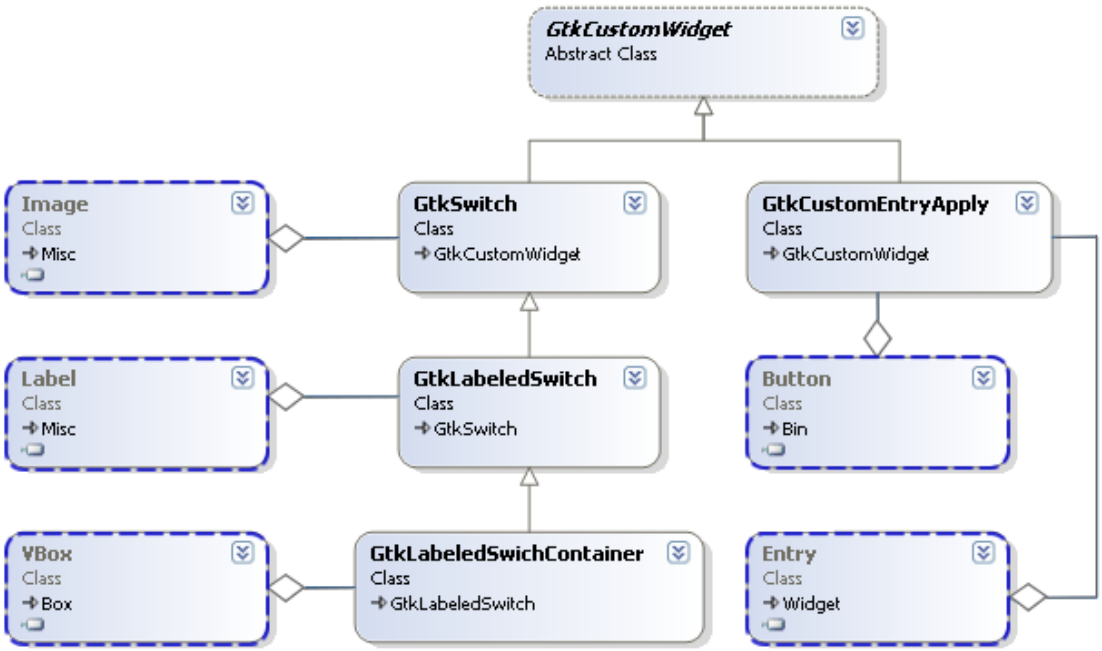

Diagrama 6 Algunas de las clases implementadas en el paquete CustomWidgets

3.3. Estructuras de consulta espacial.

Las estructuras de consulta espacial permiten localizar objetos en el espacio. La complejidad computacional[8] es siempre un tema recurrente al hablar de algoritmos de búsqueda de información en cualquier estructura de datos. Cuando se refiere a complejidad computacional se refiere a los recursos requeridos durante el cálculo para resolver un problema dado. Los recursos comúnmente estudiados son el tiempo (aproximación al número de pasos para resolver un problema) y el espacio (aproximación a la cantidad de memoria para resolver un problema).

Las estructuras utilizadas son el Kdtree, utilizado en la búsqueda del vecino más cercano y el Octree, estructura utilizada en la generación de mallas de superficie. A continuación se explica con detalle cada una de estas estructuras y el uso dado en este proyecto.

[8] El termino “complejidad computacional” aquí denomina a los indicadores “complejidad temporal” y “complejidad espacial”.

3.3.1. Árbol k-dimensiones (Kdtree).

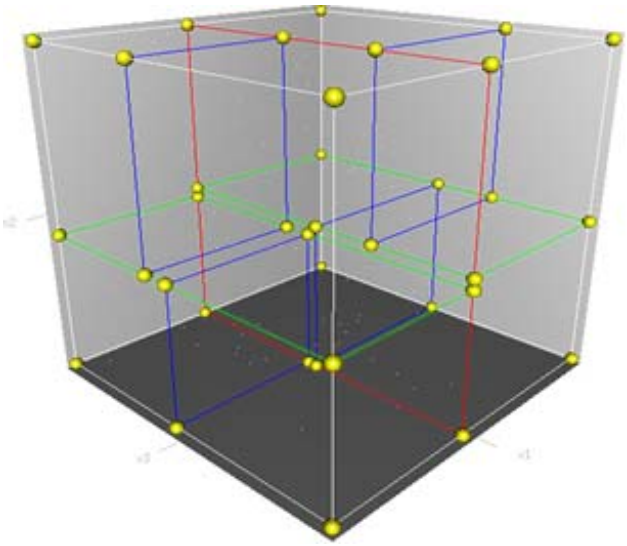

Figura 20 Un árbol kd tridimensional. La primera división (rojo) corta la celda raíz (blanco) en dos subceldas, que son divididas a su vez (verde) en dos subceldas. Finalmente, cada una de esas cuatro es dividida (azul) en dos subceldas. Dado que no hay más divisiones, las ocho finales se llaman hojas. Las esferas amarillas representan los nodos del árbol.

El problema que se presenta consiste en buscar el vecino más cercano a un punto en un conjunto de puntos en el espacio euclidiano de dos y tres dimensiones. La búsqueda del vecino más cercano es un problema habitual en una gran variedad de ramas: descubrimiento de conocimiento, clasificación y reconocimiento de patrones, estadística, compresión de datos, inteligencia artificial, entre otras.

En [37] [38] [39] se presenta el kd-tree[9] como una de las estructuras más usadas para la búsqueda del vecino más cercano. El árbol kd es una estructura de datos de particionado del espacio que organiza los puntos de un espacio euclídeo de k dimensiones. Es un caso especial de los árboles BSP. La problemática que dio lugar a la decisión de su utilización estuvo dada porque deben hacerse reiteradas búsquedas en una lista de puntos lo cual sería muy costoso. La construcción del árbol estático a partir de n puntos es de orden $\Theta\, n \log n$ y la búsqueda $\Theta \log 2n$ según la notación establecida en [40].

[9] del ingles k-dimensional tree.

La idea, detrás de esta estructura de datos, es asociar a cada nodo de un árbol una caja, y que la relación de paternidad desde el punto de vista del árbol, sea de inclusión de las cajas asociadas a los nodos (véase Figura 20). Esta idea es compartida, en particular, con otra estructura, conocida como quadtree/octree. La diferencia entre ambas es el número de hijos que tiene cada nodo. En el caso del kdtree son solo dos, mientras que en el quadtree/octree son cuatro/ocho, dependiendo de la cantidad de dimensiones del espacio (de ahí deriva el nombre de esta última estructura). En el kdtree, los hijos de un nodo representan la división de la caja del nodo padre según un eje el cual se va alternando por niveles de profundidad.

Para conseguir que cualquier prototipo (puntos en nuestro caso) tenga la misma probabilidad de estar a un lado o a otro del hiperplano y por tanto construir un árbol lo más equilibrado posible, se suele elegir el hiperplano de forma que se situé en la mediana de los valores de la coordenada discriminante. El árbol se construye de la siguiente forma: se van ordenando los puntos por la coordenada discriminante en cada nivel, se obtiene la mediana, se crea el nodo con la información de la mediana y se pasa a construir los dos hijos a partir de los subconjuntos resultantes de dividir la lista principal por la posición de la mediana. (Véase Algoritmo VIII).

Algoritmo: Construcción de un árbol kd (kd-tree)

```
construir kd-tree(lista de puntos , profundidad)
{
    si la lista no esta vacia
    {
        hacer coordenada = profundidad mod 3
        ordenar arreglo por coordenada
        construir nodo a partir de la mediana
        hacer hijo izquierdo = construir kd-tree(subconjunto a la
izquierda de la mediana, profundidad + 1)
        hacer hijo derecho = construir kd-tree(subconjunto a la
derecha de la mediana, profundidad + 1)
    }
}
```

Algoritmo VIII Construcción de un kd-tree a partir de una lista de puntos.

La búsqueda se realiza de la siguiente forma: dada una muestra x se compara la coordenada de x que es discriminante para ese nodo con el valor de corte v (la mediana por coordenada discriminante) y se procede en la dirección según esa comparación. Si $x[c] + dnn \leq v$ (donde dnn es la distancia al vecino más cercano hasta el momento) el hijo derecho no puede contener al vecino más cercano, de forma similar si $x[c] - dnn \geq v$ el hijo izquierdo tampoco lo puede contener. (Véase Algoritmo IX). De esta forma se busca solo en las ramas en que este el vecino más cercano al punto dado.

Algoritmo: Búsqueda del vecino más cercano en un kd-tree

```
Variable cd: coordenada discriminante
buscar vecino más cercano(kdtree nodo raiz, punto p)
{
        hacer vecino más cercano = nodo raiz
        hacer dnn = distancia entre p y vecino más cercano
        si  x[1] + dnn ≤ v
              buscar vecino más cercano en el hijo izquierdo
        si  x[1] – dnn  ≥ v
              buscar vecino más cercano en el hijo derecho
}
```

Algoritmo IX Utilización del kd-tree en la búsqueda del vecino más cercano.

3.3.2. Árboles octales (Octree).

La técnica iso-superficie especificada en el capítulo anterior precisa de la construcción de mallas de superficie. Algunos de los esquemas utilizados para representar sólidos y volúmenes se basan en la descomposición del espacio y utilizan estructuras jerárquicas para almacenar el modelo. Entre estos se encuentran: los árboles BSP, los árboles octales y distintas extensiones realizadas a estos últimos.

El árbol octal es una de las estructuras de particionado del espacio más utilizadas. Su nombre proviene del ingles “oct” + “tree”. Los octree tienen la característica de que cada nodo, siempre que no sea una hoja, tiene ocho hijos. Estos árboles son semejantes a los quadtree (árbol para particionar el espacio

de dos dimensiones) en su funcionamiento. La utilización del octree permite crear una eficiente estructura jerárquica del espacio.

Otra característica importante es que relacionado a cada nodo se encuentra un color, el cual indica la posición del cubo con respecto al objeto. Los colores utilizados son el blanco, el negro y el gris, el blanco indica que el cubo se encuentra completamente dentro de la figura, el negro indica lo contrario y el gris que el cubo se encuentra en la superficie del cuerpo. Si un nodo es de color blanco o negro indica que todos sus hijos son del mismo color del padre.

Cada nodo en el octree subdivide el espacio en ocho subespacios llamados octantes. Para la construcción del árbol se repite este proceso hasta una profundidad n que en la medida en que esta sea mayor el cuerpo resulta más exacto.

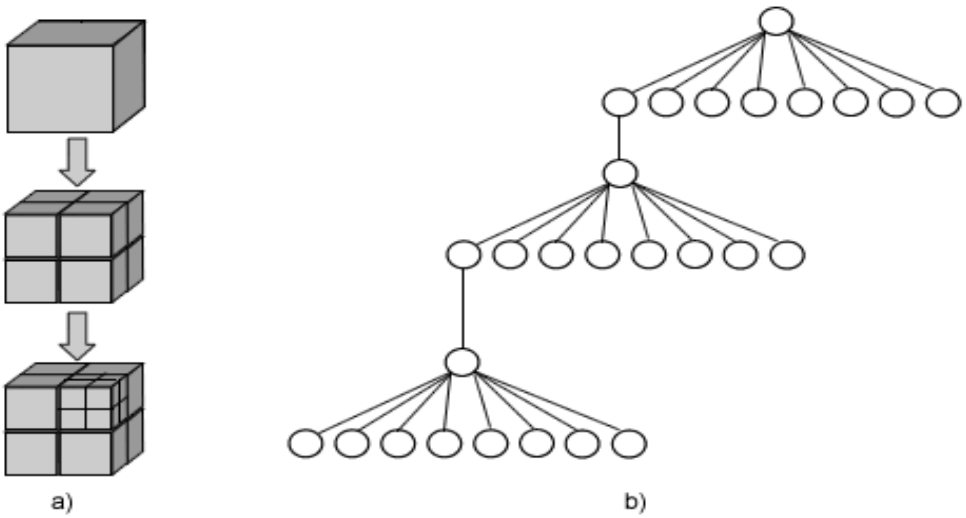

Figura 21 Subdivisión del espacio en cubos. a) representación gráfica del particionado mediante cubos. b) estructura arbolea del particionado mediante cubos.

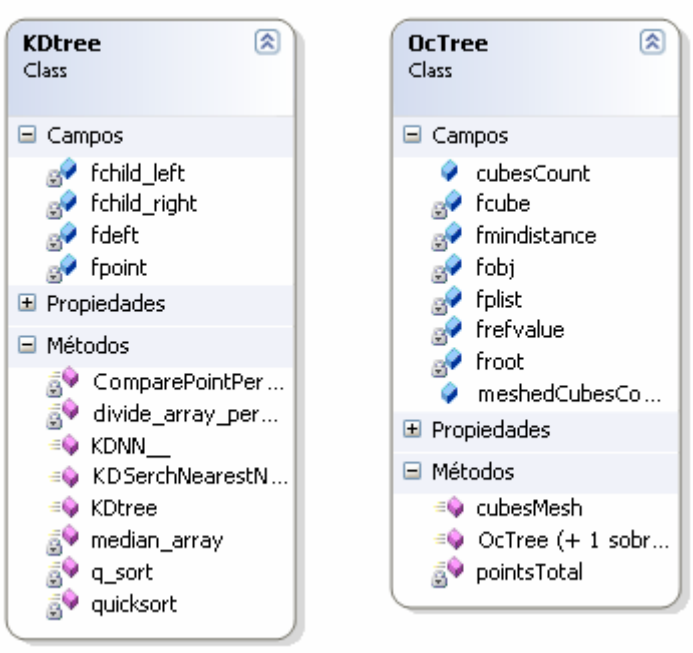

Diagrama 7 Vista detallada de las estructuras utilizadas

3.4. Mallas geométricas de superficie. Predicado de interioridad.

Las mallas de superficie constituyen una de las estructuras más usadas en la modelación y representación de cuerpos. La utilización de mallas en este trabajo es fundamental puesto que además de representar el cuerpo sobre ellas se realizan cálculos relacionados a las técnicas de visualización científica.

El primer paso para la utilización de una malla es su generación, de lo cual se encarga la técnica *marching cubes*. Este trabajo está orientado más hacia su manipulación y por ello se hará referencia al algoritmo que se encarga de encontrar la relación entre un punto y una malla. La idea general consiste en trazar una semirecta a partir del punto y contar la cantidad de veces que intercepta el objeto. (Véase Algoritmo X)

Algoritmo: Construcción de polígonos regulares.

```
relación malla-punto( punto p)
{
        hacer semirecta s con el punto p y un vector aleatorio
        interceptar s con cada una da las caras de la malla
        contar la cantidad de veces que intercepta s la malla
        si cantidad par
             retornar exterior
        sino
             retornar interior
}
```

Algoritmo X Determina si un punto es interior o exterior a una malla.

Este algoritmo tiene varias desventajas, una de ellas es que el rayo puede tocar la malla por un conjunto no numerable de puntos (por ejemplo el rayo intercepta la malla por uno de sus vértices, un segmento o una cara) o que la toque sin atravesarla, en cualquiera de ambos casos es imposible determinar la interioridad del punto con respecto a la malla. La probabilidad de que ocurra una eventualidad de este tipo para una malla sería infinitamente más pequeña que la probabilidad de funcionamiento correcto del algoritmo. Hasta el momento nunca se ha observado que el algoritmo falle debido a algún evento de este tipo. La otra desventaja es que es necesario analizar cada una de las caras de la malla por lo que el algoritmo es bastante costoso[10].

3.5. Conclusiones parciales.

A partir de lo expuesto anteriormente se pueden arribar a las siguientes conclusiones:

- Se puso el mayor empeño en utilizar prácticas modernas de programación al implementar DataVis.
- Son utilizadas la POO y el diseño de tres capas por las grandes ventajas que brindan a la hora de desarrollar una aplicación.

[10] pueden utilizarse técnicas de búsqueda espacial para acelerar el proceso.

- La utilización de proyectos de código abierto como Mono permitieron el desarrollo de una aplicación libre y multiplataforma.
- Los predicados de geometría computacional juegan un papel fundamental a la hora de dar una solución precisa a un problema geométrico real.
- La utilización de estructuras de datos cuando se trabaja sobre conjuntos de datos extensos es esencial a la hora de mejorar la complejidad computacional de cualquier algoritmo.

Capítulo IV Manual de usuario.

En esta sección se expondrán cada uno de los casos de uso en DataVis; desde el proceso de importación de los datos hasta la visualización y la manipulación de las propiedades de la imagen obtenida. Para una mejor interpretación se dividirá este manual en dos partes: una relacionada con la importación de los datos y la otra con la visualización y la manipulación de la imagen.

4.1. Proceso de importación de los datos.

En DataVis se visualizan datos escalares continuos, discontinuos, densamente discontinuos y campos vectoriales, además se implementó un módulo para la visualización del empaquetamiento mediante esferas del Método de Partículas. Esta sección esta dedicada a aspectos relacionados con el fichero de datos y el formato del mismo. Para lograr la visualización es necesario la definición del cuerpo y de los datos; ya sean escalares o vectoriales.

4.1.1. Vista principal de la aplicación.

La ventana principal es bastante simple:

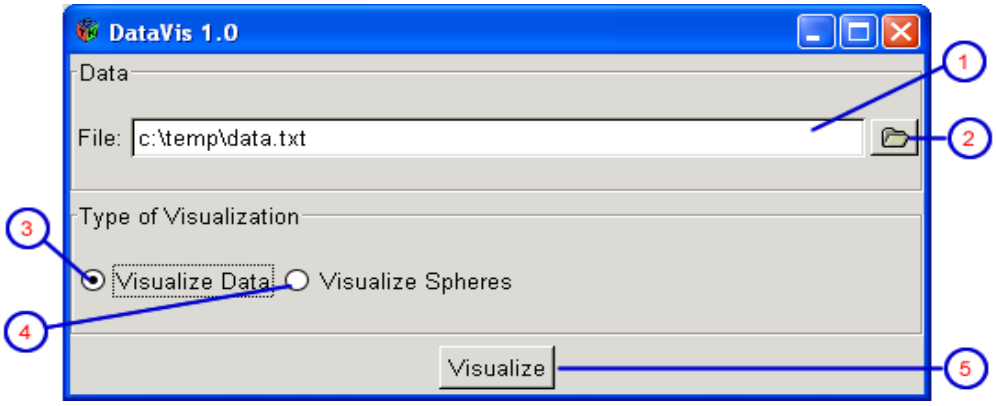

Figura 22 Vista principal de DataVis 1.0.

La anterior vista muestra los componentes esenciales para importar los datos de la aplicación:

- Cuadro de texto (1): Muestra el camino completo del fichero de datos que será cargado por la aplicación. Este camino puede editarse manualmente.
- Botón (2): Al hacer clic se muestra un diálogo que permite buscar el fichero de datos de la aplicación.
- Marcar *Visualize Data* (3): Indica que se visualizará utilizando las técnicas de visualización científicas implementadas.
- Marcar *Visualize Spheres* (4): Indica que se visualizarán datos provenientes del empaquetamiento mediante esferas del MED.
- Botón *Visualize* (5): Al hacer clic se toman los datos para realizar la visualización de los controles anteriores y se desarrolla el proceso de visualización correspondiente. En caso de que el fichero contenga errores de formato se mostrará un mensaje indicando el tipo del error.

4.1.2. Especificaciones del formato del fichero de datos.

La especificación del formato de archivo de entrada esta relacionada con su intención semántica. El formato del fichero principal sobre el que trabaja DataVis permite especificar la geometría del cuerpo y los datos (escalares y vectoriales). No se hablará de la visualización del empaquetamiento con esferas ya que este no es el objetivo principal del trabajo.

El fichero de datos tiene un formato complejo pues se especifica el cuerpo con primitivas (CSG) o como con una malla; además se da información acerca de los datos escalares y vectoriales. Para la representación del cuerpo mediante CSG se utiliza un pequeño intérprete. El formato está separado por secciones, una para cada clase de información mencionada anteriormente. Todas las secciones son opcionales. Aunque no tiene caso prescindir del CSG y la malla al mismo tiempo. (Véase Fichero 1)

```
%CSG%
.............especificacion del CSG
%CSG%

%MESH%
.............especificacion de la malla de superficie
%MESH%

%SCALARDATA%
.............especificacion de los datos escalares
%SCALARRDATA%

%DISCRETEPOINT%
.............especificacion de los datos escalares discretos
%DISCRETEPOINT%

%VECTORDATA%
.............especificacion de los datos vectoriales
%VECTORDATA%
```

Fichero 1 Archivo principal de la aplicación.

4.2. Visualización y manipulación de propiedades.

Al importar los datos correctamente la aplicación se encuentra en condiciones de llevar a cabo la visualización. La ventana de visualización esta dividida verticalmente. El área izquierda contiene la imagen resultante del proceso de visualización El área derecha contiene controles para manipular los objetos y las técnicas de visualización (superior derecha) y controles para modificar las propiedades de cada uno de ellos (inferior derecha). (Véase Figura 23).

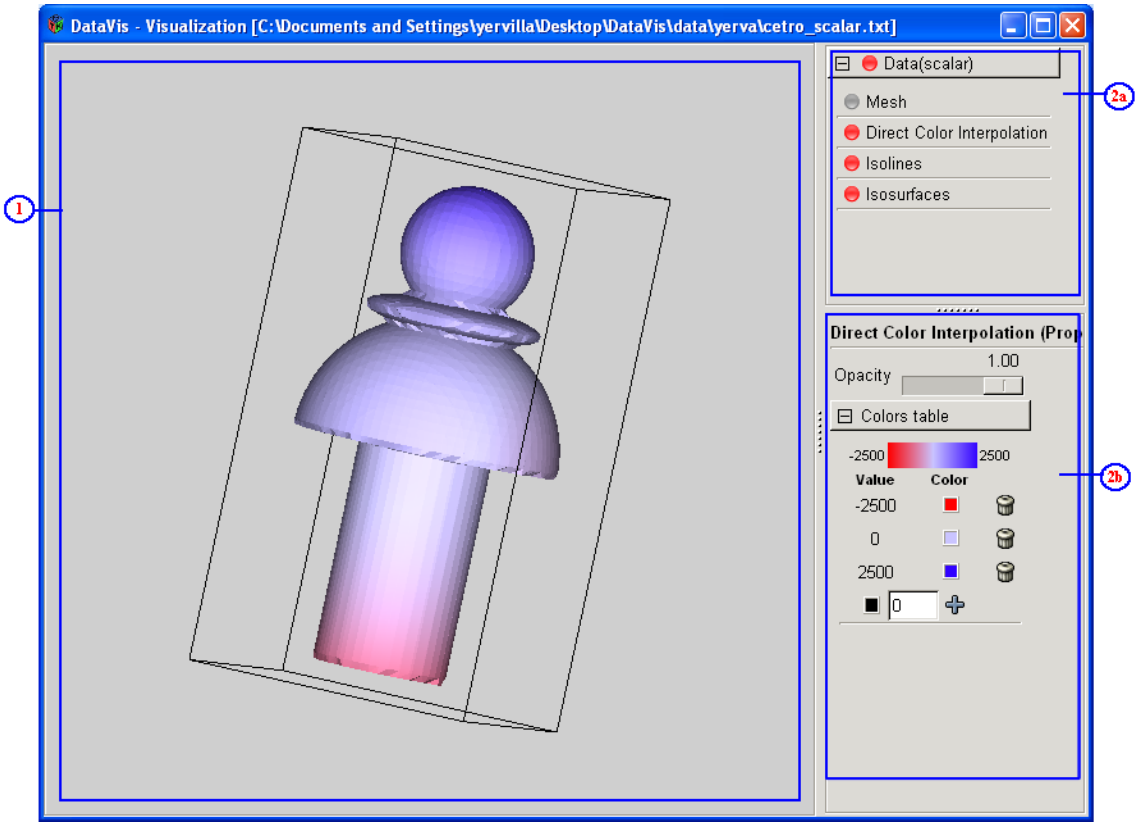

Figura 23 Vista completa de la ventana de visualización. (1) área de visualización, (2a) área contenedora de controles encargados de manipular las técnicas de visualización (panel de técnicas), (2b) área contenedora de controles para manipular las propiedades de la técnica activa.

4.2.1. Área de vista o visualización.

El área de visualización es el espacio fundamental dentro de la ventana. En esta se realizan las transformaciones de rotación, traslación y escalado. Estas transformaciones se llevan a cabo mediante combinaciones de teclas. Las combinaciones posibles son:

- **Arrow-right**: rotación hacia la derecha.
- **Arrow-left**: rotación hacia la izquierda.
- **Arrow-up**: rotación hacia arriba.
- **Arrow-down**: rotación hacia abajo.
- **Ctrl+arrow-rigth**: mover hacia la derecha.
- **Ctrl+arrow-left**: mover hacia la izquierda.
- **Ctrl+arrow-up**: mover hacia arriba.

- **Ctrl+arrow-down**: mover hacia abajo.
- **Numpad addition (+)**: aumentar la imagen (zoom).
- **Numpad addition (-)**: disminuir la imagen (zoom).

4.2.2. Editor de propiedades.

A lo largo del trabajo se ha hecho notar que para cada conjunto de datos se aplican varias técnicas de visualización. Estas técnicas al igual que el objeto pueden ser mostradas u ocultadas. Además cada uno de ellos tiene propiedades que pueden ser modificadas. A continuación se explica la funcionalidad de cada una de las áreas destinadas a editar propiedades.

4.2.2.1. Panel de Técnicas.

En el Panel de Técnicas se muestran las técnicas que son aplicadas al conjunto de datos. Cada conjunto de datos se presenta por un cuadro desplegable que contiene cada una de las técnicas y objetos asociados a los datos importados.

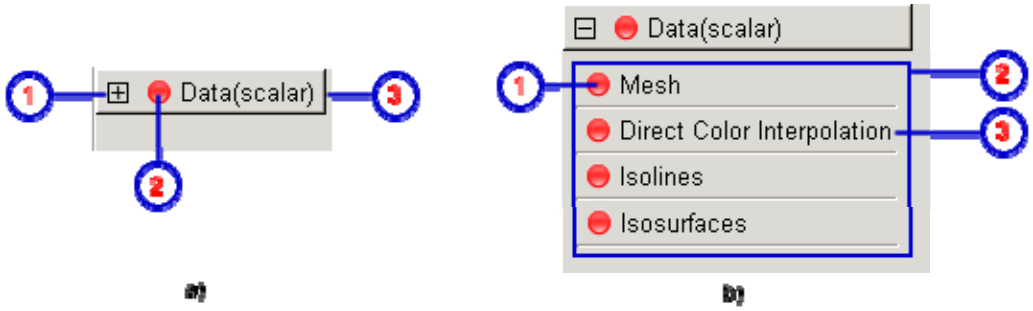

Figura 24 Vista del área de manipulación de objetos y técnicas cuando los datos importados son escalares. a) vista sin desplegar. b) vista desplegada.

En la Figura 24a se muestran los componentes de un cuadro desplegable:

- (1): Control de despliegue. Permite desplegar y ocultar las técnicas que se están aplicando al conjunto de datos.
- (2): Control de visibilidad. Permite establecer la visibilidad de todas las técnicas que están siendo aplicadas al conjunto de datos.
- (3): Breve descripción de los datos.

Por su parte en la Figura 24b) se muestra el contenido desplegado del conjunto de datos:

- (1): Control de visibilidad de la técnica. Permite establecer la visibilidad de la técnica. Contenido del cuadro de datos.
- (2): Muestra todas las técnicas aplicadas al conjunto de datos correspondiente al cuadro de datos.
- (3): Nombre de la técnica.

El contenido del Panel de Técnicas varía en dependencia de las características de los datos importados. Se tendrá más de un conjunto de datos en el Panel de técnicas cuando en el fichero se especifiquen datos vectoriales, pues se obtienen cuatro nuevos conjuntos de datos escalares que son la magnitud y los valores de las tres componentes. Cada uno de los cuadros desplegables asociados a un conjunto de datos va a contener todas las técnicas que puede aplicar DataVis dependiendo del tipo de datos. Las técnicas para datos escalares son: interpolación directa de colores en la superficie del cuerpo (*Direct Color Interpolation*), líneas de contorno (*Isolines*) y superficies de contorno (*Isosurfaces*); en caso de que se especifiquen datos discretos se adicionarían las técnicas cubos interiores (*Interior Cubes*), polígonos exteriores (*Exterior Polygon*) y una combinación de ambas. En el caso de los datos vectoriales sólo se aplica líneas de fluido (*Streamlines*). Adicionalmente a cada cuadro de datos se añade el objeto malla que describe la superficie del cuerpo (Mesh).

4.2.2.2. Inspector de propiedades

Esta área se contiene los componentes encargados de manipular las diferentes técnicas de visualización.

Propiedades del objeto malla:

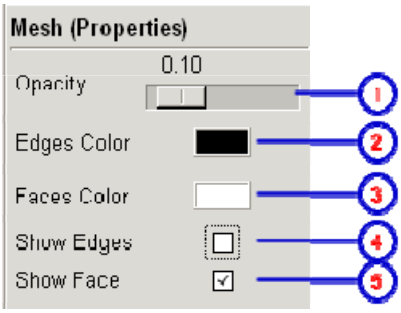

Figura 25 Vista de controles asociados al objeto malla.

Donde:

- Componente (1): Controla la opacidad de la malla.
- Componente (2): Controla el color de los bordes de la malla.
- Componente (3): Controla el color de las caras de la malla.
- Componente (4): Muestra u oculta los vértices de la malla.
- Componente (5): Muestra u oculta las caras de la malla.

Propiedades asociadas a la técnica interpolación directa de colores:

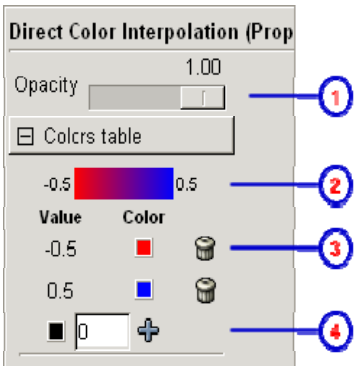

Figura 26 Vista de controles asociados a la técnica interpolación directa de colores.

Donde

- Componente (1): Opacidad de los colores.
- Componente (2): Vista de la interpolación según la tabla de colores.
- Componente (3): Controla los colores en la tabla, se pueden cambiar y eliminar.

- Componente (4): Permite adicionar colores a la tabla.

Propiedades asociadas a la técnica *isolines*:

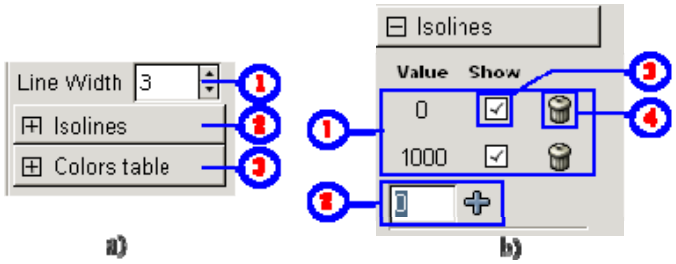

Figura 27 Vista de los controles asociados a las iso-líneas. a) vista general. b) lista desplegable de las iso-líneas.

Donde:

- Componente (1a): Ancho de las líneas de contorno.
- Componente (2a): Lista desplegable de líneas de contorno.
- Componente (3a) Tabla de colores asociada a las iso-líneas.
- Componente (1b): Conjunto de iso-líneas.
- Componente (2b): Permite adicionar nuevas iso-líneas.
- Componente (3b): Muestra u oculta la iso-línea asociada.
- Componente (4b): Elimina la iso-línea asociada.

Propiedades asociadas a la técnica *isosurface*:

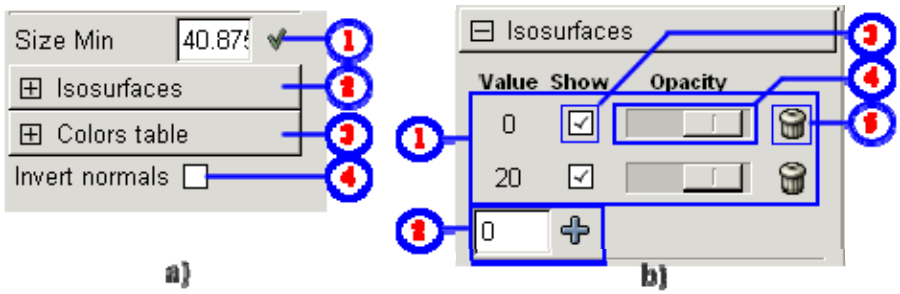

Figura 28 Vista de los controles asociados a la técnica isosurface. a) vista general. b) lista desplegable de las iso-superficies

Donde:

- Componente (1a): Permite modificar el tamaño de los cubos que simbolizaran cada uno de los puntos discontinuos
- Componente (2a): Lista desplegable de todas las iso-superficies.

- Componente (3a): Tabla de colores.
- Componente (3a): Invierte el valor de las normales de las iso-superficies.
- Componente (1b): Conjunto de iso-superficies.
- Componente (2b): Permite adicionar nuevas iso-superficies.
- Componente (3b): Muestra u oculta la iso-superficie.
- Componente (4b): Controla la opacidad de la iso-superficie asociada.
- Componente (5b): Elimina la iso-superficie asociada.

Propiedades asociadas a la técnica *interior cubes*:

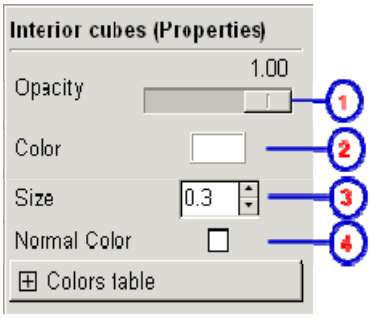

Figura 29 Vista de controles asociados a la técnica *Interior Cubes*.

Donde:

- Componente (1): Controla la opacidad de los cubos.
- Componente (2): Color de los cubos.
- Componente (3): Longitud del lado de los cubos.
- Componente (4): Al marcar este control los cubos toman el color según la tabla de colores asociada.

Propiedades asociadas a la técnica *exterior polygon*:

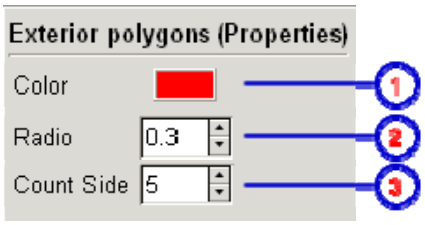

Figura 30 Vista de controles asociados a la técnica *Exterior Polygon*.

Donde:

- Componente (1): Color de los polígonos.
- Componente (2): Radio de los polígonos
- Componente (3): Cantidad de lados de los polígonos.

La combinación de ambas técnicas mostrará todos los componentes.

Propiedades asociadas a la técnica *streamline*:

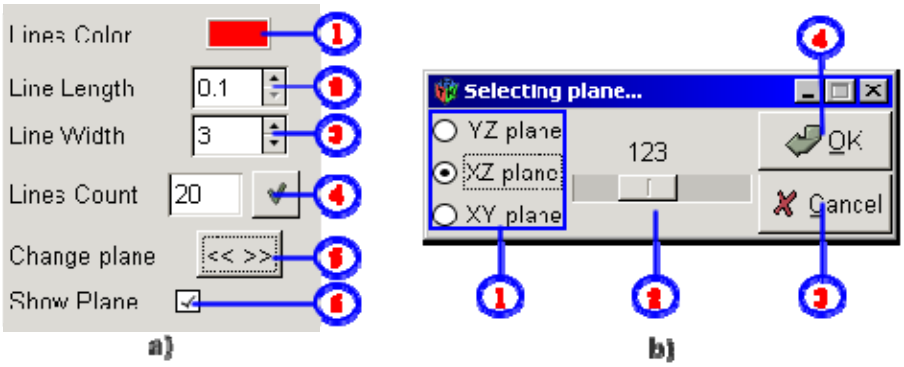

Figura 31 a) Vista de controles asociados a la técnica streamline. b) ventana encargada de manipulación del plano a partir del cual se generarán las líneas de fluido

Donde:

- Componente (1a): Color de las líneas de fluido.
- Componente (2a): Longitud del paso en la construcción de las líneas.
- Componente (3a): Ancho de la línea.
- Componente (4a): Cantidad de líneas a mostrar
- Componente (5a): Muestra una ventana encargada de intercambiar loa planos desde serán generadas las líneas.
- Componente (6a): Muestra el plano desde se generan las líneas.
- Componente (1b): Intercambia los planos.
- Componente (2b): Posición del plano activo.
- Componente (3b): Cancelar cambios.
- Componente (4b): Aplicar los cambios.

4.3. Conclusiones parciales.

Luego de analizar el manual de usuario de DataVis se arribo a las siguientes conclusiones:

- Se permite visualizar conjuntos de datos escalares y vectoriales utilizando de técnicas de visualización científica.
- La interacción con la imagen se realiza de manera sencilla.
- Al modificar las propiedades tanto del objeto como de las técnicas de visualización se logra obtener mayor información de la imagen final.

Conclusiones

Este trabajo introduce una herramienta de visualización de propiedades escalares y vectoriales en sólidos. El MED ha permitido obtener el comportamiento natural de las partículas (continuo, discontinuo y mixto). DataVis brinda al usuario técnicas de visualización científica para la representación de dichos campos.

En la implementación de DataVis se utilizaron herramientas libres sobre la plataforma .Net; las cuales junto al proyecto de código abierto Mono Project permitieron el desarrollo de una aplicación libre y multiplataforma.

La interfaz visual de la aplicación de la posibilidad de interactuar con la imagen resultante. De manera sencilla se pueden modificar cada una de las propiedades del objeto principal y de las técnicas de visualización; lo que influye notablemente en la calidad de la información que puede ser obtenida.

Recomendaciones

Se recomienda para el mejoramiento del trabajo:

- Implementar nuevas técnicas que permitan extraer información del interior del objeto.
- Implementar otras técnicas para enriquecer la visualización de datos vectoriales.
- Implementar técnicas para la visualización de datos tensoriales.
- Introducir animaciones a la imagen resultante.
- Introducir mayor nivel de interacción con las técnicas aplicadas.
- Continuar mejorando cada una de las técnicas de visualización científica implementadas.

Referencias Bibliográficas:

1. *OpenGL Reference Manual*: Addison-Wesley Publishing Company.
2. Crosby, A.W., *The Measure of Reality : Quantification in Western Europe, 1250-1600*. London, Cambridge University Press, 1997.
3. Ware, C., *Information Visualization Perception for Design*. 2da Edición ed. 2004.
4. Wikimedia Foundation, I. *Visualization*. 2007 [cited 2007; Available from: www.wikipedia.org.
5. Gallagher, R.S., *Computer Visualization: Graphics Techniques for Engineering and Scientific Analysis*. 1994.
6. Wikimedia Foundation, I. *Scientific Visualization*. 2007 [cited 2007; Available from: www.wkipedia.org.
7. Theisel, D.H., *Scientific Visualization. Compact course*. 2000.
8. Theisel, H., *CAGD and Scientic Visualization*. 2001, Rostock.
9. Camp, K.-L.M.D.M., *High Performance Visualization of Time-Varying Volume Data over a Wide-Area Network*. 2000: University of California-Davis.
10. Charles D. Hansen, C.R.J., ed. *The Visualization Handbook*. 2005.
11. Crawfis, R.A., *New Techniques for the ScientificVisualization of Three-Dimensional Multi-variate and Vector Fields*. 1989, University of California, Davis: California. p. 140.
12. Fabio, R., *From point cloud to surface:The modeling and Visualization problem.* 2003.
13. Heinbockel, J.H., *Introduction to Tensor Calculus and Continuum Mechanics*. 1996: Department of Mathematics and Statistics Old Dominion University
14. Sadd, M.H., *Elasticity. Theory, Applications And Numerics*. 2005, Rhode Island.
15. J. J. Jiménez, R.J.S., F. R. Feito, *Optimizing Ray-Tracing for Complex solids.* 2001.
16. Miguel Sainz, R.P., *Point-based rendering techniques.* 2004.
17. Salgado Milán, E., *Visualization Techniques.* 2004.
18. Dinesh P. Mehta, S.S., *Models and techniques for the Visualization of Labeled Discrete Objects.* ACM, 1991.
19. *IBM Visualizatuion Data Explorer User's Guide*. 7ma Edition ed. 1997: IBM Coorporation.
20. Mercury Computer System, Z., *Amira 4.1 User's Guide*. 2005: Mercury Computer System, ZIB.
21. Esquivel, A.V., *Noms*. 2005.
22. Morales, I.P.P., *Método de elementos distintos.*, in *Matemática*. 2006, UCLV: Santa Clara. p. 171.
23. Esquivel, A.V., *Noms*. 2006, UCLV: Santa Clara.
24. Valera, R.L.R., *Biblioteca de clases para el empaquetamiento de partículas en el MED*, in *Departamento de Computación, Facultad MFC*. 2007, UCLV: Santa Clara, Cuba.
25. Brito, Y.P., *Implementacion del empaquetamiento en el MED para diferentes tipos de partículas* in *Departamento de Computacion, MFC*. 2007, UCLV: Santa Clara, Cuba.

26. Gregory M. Nielson, B.H., *The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes.* 1991.
27. Chien-Chang Ho, F.-C.W., Bing-Yu Chen, Yung-Yu Chuang, Ming Ouhyoung, *Cubical Marching Squares: Adaptive Feature Preserving Surface Extraction from Volume Data.* EUROGRAPHICS, 2005. **24**.
28. Jeff Ferguson, B.P., Jason Beres, Pierre Boutquin, Meeta Gupta, *La biblia de C#*. 2003: ANAYA MULTIMEDIA.
29. *OpenGL Programming Guide I*: Addison-Wesley Publishing Company.
30. *OpenGL Programming Guide II*: Addison-Wesley Publishing Company.
31. García, J., *Curso de introducción a OpenGL*. 2003.
32. Wikimedia Foundation, I. *OpenGl*. 2007 [cited 25-04-2007; Available from: www.wikipedia.org/wiki/OpenGl.
33. *OpenGl Oficial Page*. 2007 [cited 2007; Available from: www.opengl.org.
34. *GTK Oficial Site*. 2007 [cited 2007 01-07-07]; Available from: www.gtk.org.
35. Wikimedia Foundation, I. *Quaternions*. 2007 [cited 2007 18-04-2007]; Available from: http://en.wikipedia.org/wiki/Quaternions.
36. Wikimedia Foundation, I. *Quaternions and spatial rotation*. 2007 [cited 2007 18-04-2007]; Available from: http://en.wikipedia.org/wiki/Quaternions and spatial rotation.
37. Sunil Arya, D.M.M., *An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions.* 1994.
38. Atramentov, S.M.L.A., *Efficient Nearest Neighbor Searching for Motion Planning.* 2001.
39. Ryusuke Sagawa, T.M., Katsushi Ikeuchi, *Effective Nearest Neighbor Search for Aligning and Merging Range Images.* 1999.
40. Skiena, S.S., *The Algorithm Design Manual*. 1997, New York: Springer-Verlag, New York.

Apéndices:

Apéndice I Casos complementarios para resolver ambigüedades en "marching cubes".

Como ya se ha explicado el método "*marching cubes*" es utilizado en la generación de mallas tanto para la representación de objetos como de superficies de contorno. Se ha planteado que las 256 combinaciones posibles pueden ser reducidas a 15 por propiedades de simetría. (Véase Figura 10a)

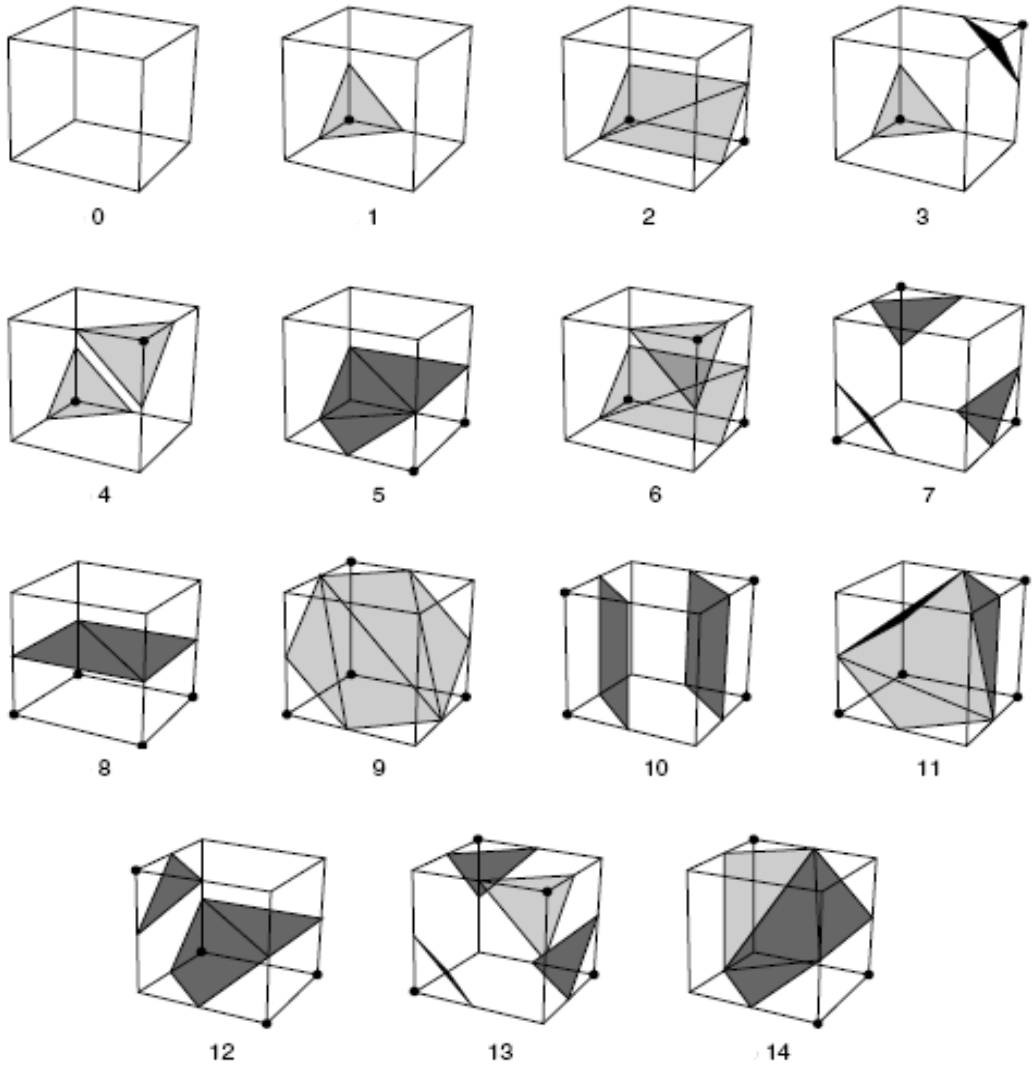

Figura 32 Reducción de los 256 casos a 15.

Entre los casos representados en la figura anterior el 3, 6, 7, 10, 12 y 13 presentan algún tipo de ambigüedad La manera de tratarlas es creando casos complementarios e introduciéndolos en la tabla da casos (Véase Figura 33). Existe una técnica “asymtotic decider”, mencionada anteriormente, que resuelve las ambigüedades con mayor precisión. Estos casos resuelven el problema de la aparición de hoyos en la malla. Este método es aplicado tanto para generar la malla de superficie como para la obtención de las superficies de contorno.

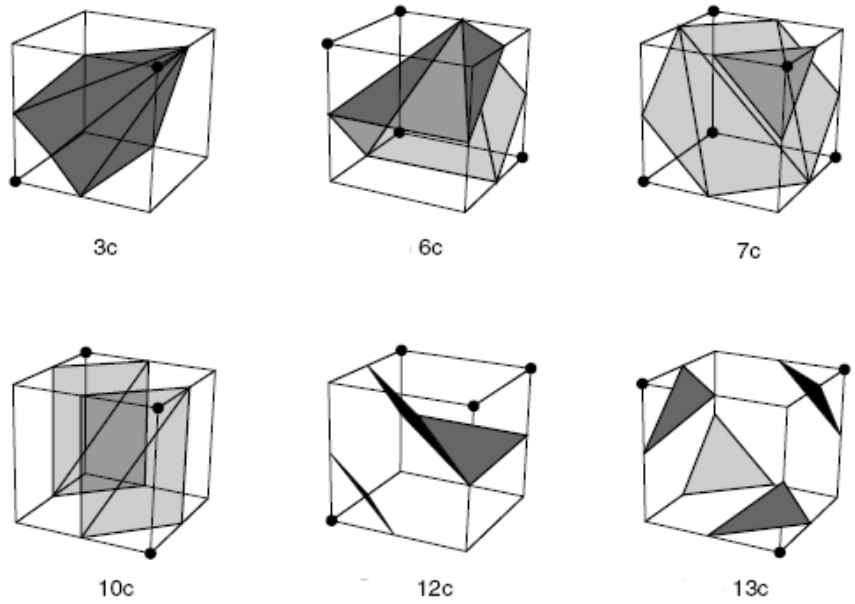

Figura 33 Casos complementarios para tratar las ambiguedades.