UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

VISUALIZADOR Y EVALUADOR DE MALLAS GEOMÉTRICAS MIXTAS 3D

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVÍL EN COMPUTACIÓN

JAVIERA ALEJANDRA MASCARÓ CUMSILLE

PROFESOR GUÍA:
NANCY HITSCHFELD KAHLER

MIEMBROS DE LA COMISIÓN:
CLAUDIO LOBOS YÁÑEZ
PATRICIO INOSTROZA FAJARDIN

SANTIAGO DE CHILE
DICIEMBRE 2010

Resumen

El objetivo del presente tema de memoria es diseñar y desarrollar un visualizador y evaluador de mallas geométricas mixtas en 3D, que provea diferentes formas de visualización, que permita evaluar la calidad de las mallas aplicando distintos criterios de evaluación sobre ellas, y que sea fácilmente extensible a nuevas funcionalidades.

Una malla geométrica es una colección de vértices, aristas y caras, que define la forma de un objeto complejo en base a polígonos (2D) y poliedros (3D). Se considerará en este trabajo la manipulación de mallas de superficie y mallas mixtas 3D compuestas por tetraedros, pirámides, prismas de seis vértices y hexaedros. Mientras más regulares son los elementos de una malla geométrica esta es de mejor calidad. Para evaluar la calidad de una malla, se aplican sobre los elementos que la componen los criterios de evaluación ángulo diedro, ángulo sólido, jacobiano y relación volumen - arista más larga al cubo.

En una primera etapa, se propone e implementa un diseño con enfoque orientado a objetos en donde se utilizan diferentes patrones de diseño. Se propone una estructura de datos para el manejo de mallas que privilegia la eficiencia al ejecutar algoritmos sobre ellas por sobre el espacio utilizado al almacenarlas en memoria. Se utiliza un subsistema que permite abrir y guardar mallas almacenadas en archivos de distintos formatos usando el patrón de diseño Facade. Se crean los elementos de las mallas a partir de los datos utilizando el patrón Factory Method. Se generan Iteradores sobre los elementos de la malla según sea necesario. Se utiliza el patrón de diseño Strategy para definir y aplicar los diferentes criterios de evaluación. Finalmente, se define el visualizador como un Singleton para asegurar una instancia única.

Luego, se procede con el desarrollo de una interfaz adecuada a los requisitos y fácil de usar que provee distintas funcionalidades dentro de las cuales se encuentran: abrir y guardar una malla geométrica, visualizarla, rotarla, trasladarla, realizar zoom-in y zoom-out, ver sólo la superficie de la malla, desplegar datos de los elementos, seleccionar elementos, escoger niveles de vecindad y observar un elemento con sus vecinos, aplicar criterios de evaluación sobre toda la malla o sobre vecindades, visualizar elementos que cumplen o no con los rangos de evaluación y mover puntos de la malla para mejorar su calidad.

Como resultado, se obtiene una herramienta extensible, flexible y fácil de usar, que provee distintos tipos de visualización de mallas geométricas, que permite evaluar la calidad de los elementos que las componen y, mejorarlas manipulando los puntos de elementos críticos y al mismo tiempo manteniendo su integridad.

Agradecimientos

A mi familia por su apoyo y buenos deseos, en especial a mis padres por apoyarme en todo mi proceso de formación, por incentivarme siempre a salir adelante y a pensar que merezco lo mejor. A mi hermana, por comprenderme y entenderme siempre en los momentos difíciles y entregarme una sonrisa cuando más lo necesitaba.

A mis amigos y compañeros que han estado junto a mí en distintos momentos durante estos 6 años. A Bea, por ser como mi hermana, porque a pesar de la distancia siempre sentí su apoyo incondicional. A Vero y Amgelo, pero estar ahí día a día dándome ánimo y apoyo, por levantarme en las derrotas y celebrar conmigo los triunfos.

Y en especial a mi Profesora Guía Nancy, por todo el conocimiento transmitido y en sobre todo, por su comprensión, disposición y paciencia.

Contenido

1. Introducción

1.1. Aspectos generales

Hoy en día las aplicaciones de computación gráfica son muy masivas y utilizadas en diversos campos. Uno de ellos es el modelamiento y simulación de situaciones reales a través de procesos computacionales.

Por ello, es importante poder representar de forma fidedigna información del mundo real a través de modelos de datos discretos que sean posibles de procesar [1].

Una forma útil de representar estos datos y así modelar objetos y sistemas son las Mallas Geométricas. A través de ellas es posible representar la realidad de forma discreta, pero bastante aproximada y así obtener modelos procesables y aplicar diferentes simulaciones físicas o matemáticas sobre ellos.

Una Malla Geométrica se compone de un conjunto de polígonos (2D) o poliedros (3D), que en conjunto representan información sobre un objeto en estudio [4].

Por esto, es importante mantener la calidad de estos modelos, para así obtener resultados que representen los procesos de forma lo más aproximada a la realidad posible, además de asegurar que, por ejemplo, simulaciones mediante métodos numéricos, tales como elementos o volúmenes finitos, produzcan resultados fidedignos. Existen diversas formas de medir y mejorar la calidad, observando las propiedades de los polígonos o poliedros que las componen. Mientras mayor sea la calidad de estos, se obtiene una mejor representación de los modelos.

En el presente trabajo, se diseñó y desarrolló una herramienta para el manejo de Mallas Geométricas en 3D, que permite visualizar la malla y las distintas propiedades de los objetos que la componen, analizar y calcular su calidad y, simular posible mejoras sobre ella.

Por otra parte, el diseño de la aplicación permite en un futuro agregar nuevas funcionalidades de forma fácil y rápida. Para ello, la aplicación se desarrolló utilizando programación orientada a objetos y patrones de diseño.

1.2. Justificación y motivación

Las mallas geométricas son utilizadas en muchas áreas de la ciencia y la tecnología, ya sea desde el punto de vista matemático o a través de su uso en el modelamiento gráfico 2D y 3D para representar la realidad. Dentro de sus distintas aplicaciones se encuentran entre otros:

- Discretización y modelamiento del dominio de problemas matemáticos complejos.
- Modelamiento y simulación de proceso Industriales, Mecánicos, Químicos, Mineros, etc.
- Desarrollo de herramientas CAD para su uso en Arquitectura e Ingeniería.
- Creación de entornos de realidad virtual.
- Representación discreta de superficies.
- Industria del entretenimiento: cine, videojuegos, etc.

En general, al utilizar mallas geométricas para realizar distintos modelamientos, se obtiene un conjunto de datos de gran tamaño y es importante poder, de alguna u otra forma, corroborar la correctitud y calidad de dichos datos y si el modelo representa la realidad que se deseaba modelar. Por ello, el poder visualizar estos resultados es de suma importancia, y se hace imprescindible el contar con una herramienta fácil de usar, que permita visualizar y analizar mallas almacenadas en distintos formatos, que provea diferentes modos de visualización, genere distintas estadísticas sobre los datos y, finalmente permita almacenar los resultados en el formato deseado.

Existe en la actualidad programas de visualización y modificación de mallas geométricas, dentro de los cuales se encuentran Tetview [12] y Geomview [13]. Sin embargo, Tetview sólo permite visualizar mallas de triángulos (2D) y tetraedros (3D) y visualiza los índices sobre sus elementos pero no provee evaluación de criterios de calidad, y Geomview se limita sólo a la visualización de mallas de poliedros (mallas de superficie) y no provee herramientas de análisis y evaluación de ellas.

1.3. Objetivos

1.3.1 Objetivo general

Diseñar y desarrollar un visualizador y evaluador de mallas geométricas mixtas en 3D, que provea diferentes formas de visualización de acuerdo a las necesidades del usuario, que permita evaluar la calidad de las mallas aplicando distintos criterios de evaluación sobre ellas, y que sea fácilmente extensible a nuevas funcionalidades.

1.3.2 Objetivos específicos

- Visualizar una malla geométrica 3D compuesta por tetraedros, prismas, pirámides y hexaedros, a partir de distintos formatos.
- Proveer distintos modos de visualización tales como: visualizar sólo la superficie, visualizar todos los elementos, visualizar los índices de los vértices y elementos, dado un vértice visualizar sus elementos vecinos, dado un arco visualizar sus elementos vecinos, y visualizar elementos transparentes.
- Evaluar la calidad de una malla geométrica de acuerdo a los distintos criterios aplicables a los elementos que la componen, entre ellos: ángulo diedro, ángulo sólido, jacobiano y razón entre el volumen y el largo más largo, de tal manera que permita al usuario evaluar su calidad.
- Permitir al usuario modificar dinámicamente la malla que visualiza observando instantáneamente las mejoras de calidad al hacerlo, pero asegurando que se mantiene la integridad de la malla y los poliedros que la componen.
- Almacenar la malla resultante en diferentes formatos.
- Diseñar la aplicación de forma que sea fácilmente extensible a nuevas funcionalidades, como por ejemplo agregar nuevos formatos de entrada o salida, agregar modos de visualización, agregar nuevos criterios para generar estadísticas.

1.4 Contenidos

1. Introducción: Se presenta una mirada general al tema y la motivación. Se explicitan los objetivos del trabajo a realizar.
2. Antecedentes: Se presenta el marco teórico y contenidos necesarios para el desarrollo del tema, elementos de ingeniería de software, y una mirada a herramientas similares disponibles.
3. Diseño Visualizador y Evaluador de Mallas: Se presenta la metodología a utilizar y los requerimientos y funcionalidades con que debe cumplir la herramienta a desarrollar. Luego, se explicita la estructura de datos y el diseño interno de la aplicación, utilizando diferentes patrones de diseño según sea conveniente.
4. Implementación: Se exponen los detalles de la implementación de la aplicación, incluyendo el uso de librerías y código pre-existente, manejo y almacenamiento de datos y, desarrollo de los algoritmos de cálculo de criterios.
5. Conclusiones: Se muestran las conclusiones obtenidas en este trabajo y el posible trabajo futuro sobre la aplicación desarrollada.
6. Referencias: Se presentan las fuentes de información bibliográfica utilizadas en relación al marco teórico y trabajos relacionados sobre el tema en estudio.
7. Anexos: Se presenta información adicional y de interés sobre el desarrollo de la memoria.

2. Antecedentes

En este capítulo se presentan los antecedentes relacionados con el área en que se desarrolla este tema de memoria. Entre ellos se incluyen tanto conceptos geométricos como de programación.

2.1. Conceptos geométricos

2.1.1. Geometría computacional

La Geometría Computacional es una rama de la computación que se dedica al estudio de algoritmos geométricos. Esta área busca esencialmente diseñar y analizar algoritmos para solucionar problemas geométricos de forma eficiente, relacionándose fuertemente con las matemáticas discretas y el análisis combinatorio.

La Geometría Computacional se vio fuertemente impulsada por el avance de la computación gráfica y el diseño y representación de objetos reales asistido por computador, sin embargo tiene muchas otras aplicaciones dentro de las que se incluyen la robótica (planificación de movimientos y problemas de visualización), diseño de circuitos integrados, sistemas de localización, estudio de comportamiento de sólidos y en general cualquier fenómeno físico, programación de comportamiento de máquinas, etc.

La simulación de objetos o fenómenos reales puede ser analizada en 2 o 3 dimensiones, en donde el problema en 2D es una simplificación del problema en 3D, que es bastante complejo. El tema de ésta memoria pretende facilitar el estudio de modelos en 3D.

2.1.2. Mallas geométricas

Una malla geométrica es una colección de vértices, aristas y caras, que define la forma de un objeto complejo en base a polígonos (2D) y poliedros (3D), es decir, en base a formas simples y conocidas. Se distinguen dos tipos de mallas:

- Mallas Estructuradas: se componen de celdas del mismo tipo y tamaño, como por ejemplo, rectángulos o triángulos en (2D) y hexaedros o tetraedros (3D).
- Mallas No Estructuradas: se componen de celdas del mismo o distinto tipo pero de tamaños diferentes. Existen en general, algunas zonas con celdas pequeñas y otras con celdas de mayor tamaño.

La ventaja de las mallas estructuradas con respecto a las mallas no estructuradas, es que son más fáciles de generar y de estudiar sus propiedades, sin embargo no permiten modelar problemas o formas complejas.

Dada una malla cualquiera, existen diversas maneras de analizar la calidad de los elementos que las componen (como se verá posteriormente), para así aplicar métodos que permiten refinarlas o mejorar los atributos de elementos que no cumplen con el estándar de calidad deseado y así mejorar la calidad general de la malla.

Una forma de mejorar la calidad de un elemento o un grupo de elementos de una malla, es el desplazamiento o cambio de posición de un punto que puede resultar crítico para la composición de los poliedros o polígonos que lo contienen. El cambiar de posición un punto no es trivial pues afecta las propiedades de cada elemento que lo contiene, y podría mejorarse la calidad de uno de ellos pero empeorar sustancialmente la de los otros. Además, es posible romper la integridad de la malla haciendo que el poliedro o polígono deje de serlo al modificar la posición de uno de sus puntos.

Otra forma de mejorar la calidad de una malla es refinándola. Una malla puede refinarse agregando puntos en los lugares críticos y generando así elementos de mejor calidad en dichas zonas. El punto se agrega dentro del poliedro o polígono crítico y luego se generan aristas desde cada vértice del elemento al punto agregado. Se debe verificar la validez de los nuevos elementos generados y que efectivamente estos sean de mejor calidad.

Es posible distinguir mallas geométricas características y sus propiedades de acuerdo en el espacio en el que se encuentran, ya sea 2D o 3D, tal como se detalla a continuación:

2.1.2.1. Mallas Geométricas 2D

En 2D, las mallas más comúnmente utilizadas son las mallas de triángulos, sobre todo en la modelación de superficies. Las mallas de triángulos se denominan triangulaciones, siendo la más conocida la triangulación de Delaunay.

La triangulación de Delaunay asegura que los triángulos que la componen son lo más equiláteros posible, aplicando un test llamado "test del círculo", para asegurar esta propiedad [4,6]. El test del círculo consiste en corroborar que para todo triángulo, la circunferencia circunscrita a él no contiene ningún vértice de otro triángulo, de lo contrario, se corrige la triangulación intercambiando la arista común de los dos triángulos contenidos en el círculo (Figura 1).

Figura 1: Dos triángulos que no cumplen la condición del "test del círculo", con posterior cambio de arista que hace que sí se cumpla la condición.

2.1.2.2. Mallas de Superficie

Las triangulaciones, como se mencionó anteriormente, son comúnmente usadas en la modelación de superficies. Las mallas de superficie se componen de una malla de polígonos, generalmente una triangulación, generada en base a una proyección de los datos en dos dimensiones (x,y), en donde luego se aplica la componente z o de altura a los datos (Figura 2).

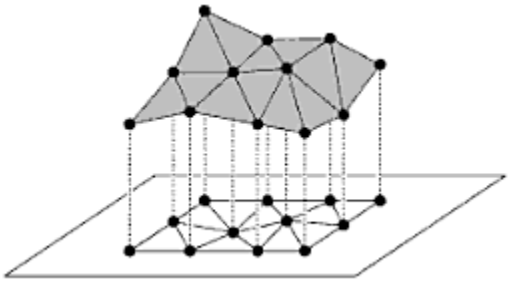


Figura 2: Triangulación de superficies.

Ya que este tipo de mallas se componen de elementos en 2D, es decir polígonos y no de poliedros, pero además se les agrega una componente z o de altura, se suele decir que son mallas en 2 ½D. Se utilizan en general para el modelamiento de terrenos.

2.1.2.3. Mallas Geométricas 3D

Una malla geométrica en 3D o de poliedros es una colección de vértices, aristas, caras y elementos en donde cada cara es compartida por a lo más 2 elementos [2].

Se puede extender el modelo de mallas 2D a 3D. Así, se obtiene el equivalente a la triangulación de Delaunay en tres dimensiones, denominada tetraedrización de Delaunay, la cual consiste en una malla de poliedros, en donde todos son tetraedros y cada uno de ellos cumple el equivalente test de la esfera [4,3]. Las mallas de tetraedros, así como también las de hexaedros, son mallas de poliedros muy utilizadas pues sus características y propiedades son conocidas.

Las mallas geométricas 3D son el objeto principal de estudio de este tema de memoria.

2.1.3. Elementos en mallas geométricas 3D

En el presente trabajo se estudian mallas mixtas en 3D compuestas por los siguientes elementos (Figura 3):

- Tetraedro: poliedro de 4 caras triangulares, 6 arcos y 4 vértices, en donde cada uno de ellos es compartido por 3 caras.

- Prisma: poliedro que consta de dos caras triangulares iguales y paralelas llamadas bases, y de caras laterales rectangulares. Se usarán prismas de 6 vértices.
- Pirámide: poliedro cuya base es un cuadrilátero cualquiera y cuyas caras laterales son triángulos con un vértice común, que es el vértice de la pirámide, teniendo en total 5 vértices**.**
- Hexaedro: o también llamado ortoedro, es un paralelepípedo ortogonal, es decir, posee 6 caras rectangulares y 8 vértices, donde cada vértice es compartido por 3 caras y las caras opuestas son iguales entre sí en forma y tamaño.

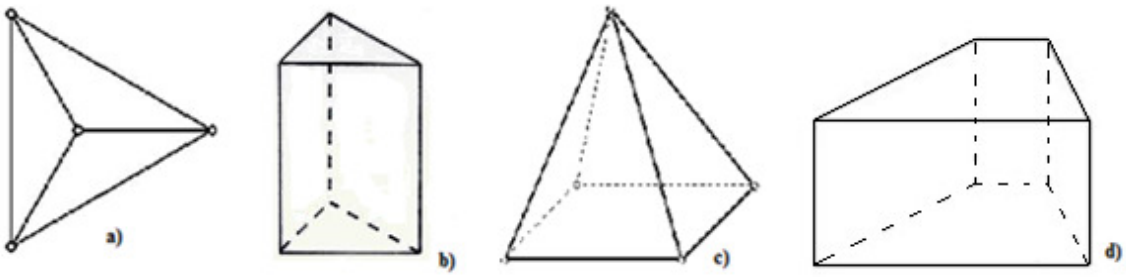


Figura 3: a) Tetraedro, b) Prisma, c) Pirámide, d) Hexaedro

2.1.4. Criterios de evaluación

Existen diversos métodos de evaluar la calidad de los elementos que componen mallas geométricas 3D, considerando que mientras más regulares estos sean se tiene mayor calidad, y así obteniendo modelos más uniformes de los objetos en estudio. Dentro de dichos criterios se encuentran:

2.1.4.1. Ángulo Sólido

Mide el tamaño aparente de un objeto. Es el ángulo espacial que abarca dicho objeto visto desde un punto dado, que se corresponde con la zona del espacio limitada por una superficie cónica.

Para calcular el ángulo sólido de una superficie, se proyecta el objeto sobre una esfera de radio conocido. Luego, el ángulo sólido es la relación entre el área de la superficie proyectada en la esfera (S) y el radio (R) de la esfera al cuadrado:

$$\Omega = \frac{S}{R^2}$$

En la figura 4 se puede observar el caso para la proyección de una superficie cónica.

La unidad de ángulo sólido es el estereorradián, definido como el ángulo sólido que teniendo su vértice en el centro de una esfera, delimita un área en la superficie de la misma igual a la de un cuadrado cuyos lados sean iguales a la longitud del radio. Es el equivalente tridimensional del radián [16].

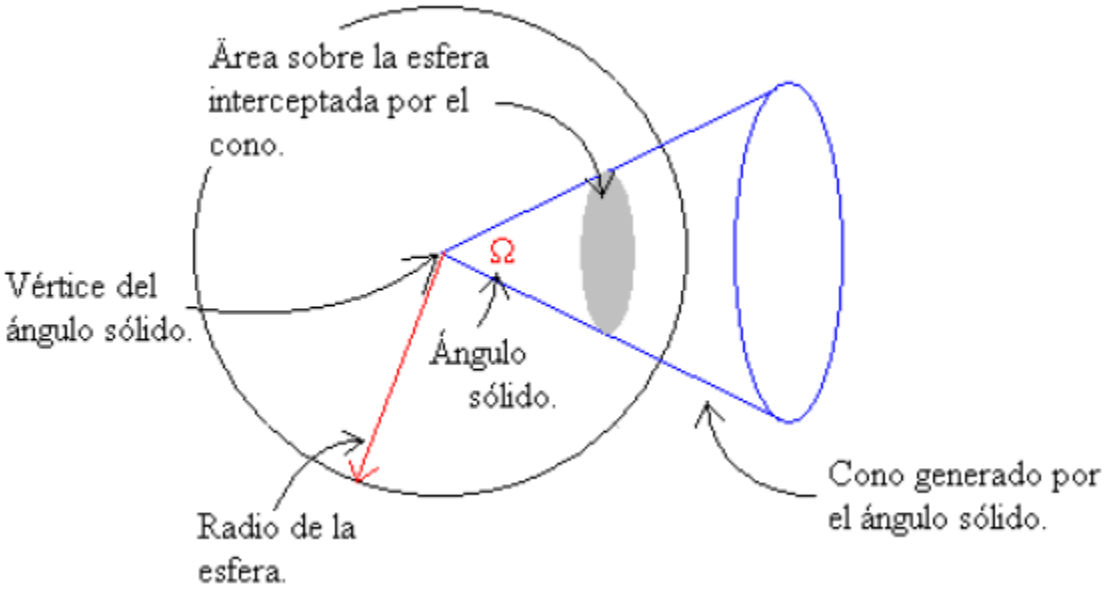


Figura 4: Ángulo Sólido

Para evaluar los elementos de una malla utilizando el ángulo sólido, se calcula el ángulo sólido asociado a cada vértice, identificando el ángulo máximo y mínimo. Para ello, se identifica la superficie o cara que se forma de las aristas que salen de cada vértice y con respecto a dicha superficie se realiza el cálculo del ángulo.

En este caso las superficies o caras proyectadas desde las aristas que salen de un vértice en un poliedro son triángulos o cuadriláteros (que se pueden dividir en 2 triángulos), y, si se traslada el vértice en estudio y la superficie formada al origen, se tiene que es posible calcular el ángulo sólido de una superficie triangular visto desde el origen, con vértices $\mathbf{R}_1$, $\mathbf{R}_2$ y $\mathbf{R}_3$, a través de la siguiente ecuación [16]:

$$\tan\left(\frac{1}{2}\Omega\right) = \frac{|\mathbf{R}_1\mathbf{R}_2\mathbf{R}_3|}{R_1R_2R_3 + (\mathbf{R}_1\cdot\mathbf{R}_2)R_3 + (\mathbf{R}_1\cdot\mathbf{R}_3)R_2 + (\mathbf{R}_2\cdot\mathbf{R}_3)R_1}$$

En donde $|\mathbf{R_1R_2R_3}|$ es el determinante de la matriz compuesta por dichos vectores, R_i denota la distancia escalar del punto i al origen y $\mathbf{R}_i \cdot \mathbf{R}_j$ denota el producto escalar entre ambos vectores.

2.1.4.2. Ángulo Diedro

Es el ángulo formado por dos semiplanos que parten desde una arista común (Figura 5).

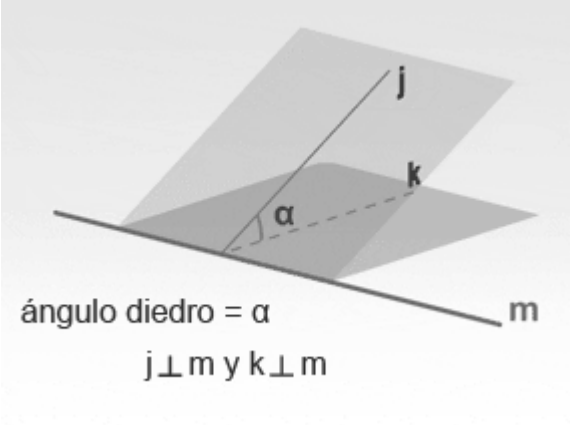


Figura 5: Ángulo Diedro

Para calcular el ángulo diedro, se debe encontrar un vector perpendicular, contenido en cada plano, a la recta en donde ambos se intersectan. Dichos vectores deben tener como origen un pivote común en la recta de intersección. Luego, el ángulo diedro es el coseno del producto punto entre ambos vectores [17].

Para evaluar los poliedros de una malla utilizando el ángulo diedro, se calcula el ángulo diedro formado entre todas las intersecciones formadas por las caras que componen al poliedro, identificando el ángulo máximo y mínimo.

2.1.4.3. Razón Volumen Arco más largo

Es la razón entre el volumen de una celda y el largo de la arista más larga al cubo.

Para calcular el volumen de un poliedro, se puede dividir dicho poliedro en un conjunto de tetraedros. En el caso particular de los poliedros en estudio en esta memoria, es posible dividirlos de la siguiente manera:

Si se numeran los vértices de cada poliedro (Figura 6):

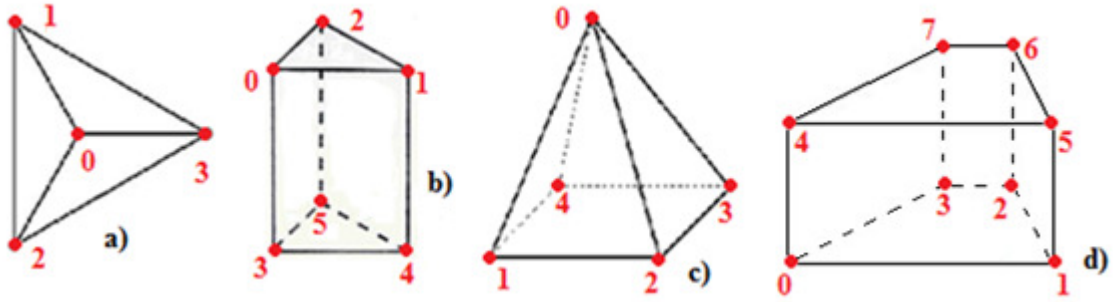


Figura 6: a) Tetraedro, b) Prisma, c) Pirámide, d) Hexaedro. Numerando en cada caso sus vértices.

Es posible dividir cada poliedro en los tetraedros formados por los siguientes vértices:

	Tetraedro	*Prisma*	*Pirámide*	*Hexaedro*
Vértices Tetraedro 1	*1,2,3,4*	*0,4,5,3*	*0,1,2,4*	*0,1,2,5*
Vértices Tetraedro 2	-	*2,4,5,0*	*0,2,3,4*	*5,7,6,2*
Vértices Tetraedro 3	-	*1,4,2,0*	-	*0,2,3,7*
Vértices Tetraedro 4	-	-	-	*4,7,5,0*
Vértices Tetraedro 5	-	-	-	*0,2,7,5*

Tabla 1: División de poliedros en tetraedros indicando los vértices.

Luego, es posible calcular el volumen de un tetraedro cualquiera con la siguiente fórmula:

$$V = \frac{1}{3!}\begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix}$$

Finalmente, para cada poliedro se suma el volumen de los tetraedros que lo forman obteniendo el volumen total. Este volumen se divide por la arista más larga del poliedro al cubo y se obtiene el valor del criterio.

2.1.4.4. Jacobiano de un Poliedro

Mide la deformación de un poliedro con respecto a su representación regular. Mientras mayor es la deformación, menor es la calidad del poliedro.

En cálculo vectorial, el jacobiano es el determinante de la matriz jacobiana de una función. La matriz jacobiana, es la matriz de las derivadas parciales de primer orden de la función.

En términos generales, para calcular el jacobiano de un poliedro, se obtiene una matriz representativa del poliedro en su versión regular y una matriz del poliedro en estudio, en base a ello se genera un polinomio que representa la relación entre ambos poliedros. Luego, se calcula el jacobiano de dicho polinomio.

El jacobiano de un poliedro toma valores entre 0 y 1. Si el valor es 1 se tiene un poliedro regular, y mientras el valor se acerca a 0 este más irregular y de menor calidad. Si el jacobiano del poliedro toma valores negativos, quiere decir que dicho poliedro es inválido o no es del tipo esperado, por ejemplo, si se tiene un poliedro que no es un prisma y se calcula su jacobiano con respecto a un prisma regular, el valor que se obtendrá será negativo.

2.1.5. Formatos de almacenamiento de mallas

En general, una malla geométrica se almacena guardando sus datos en archivos de texto con diferentes extensiones que definen un estándar para una posterior lectura e interpretación de dichos datos. A continuación se describe el formato de almacenamiento de algunas extensiones de interés.

- Archivos .off: es uno de los formatos más populares de GeomView. En él se numera una lista de vértices y luego una lista de polígonos planos (caras) descritos a en base a los vértices definidos previamente. Los vértices se numeran implícitamente y parten desde 0. El archivo se compone de la siguiente manera (el símbolo # es un comentario):

```
#OFF
numeroVertices numeroCaras numeroAristas
#tantos vértices como se indica en numeroVertices
x y z
x y z
#tantas caras como se indica en numeroCaras, NVertices índica
#cuantos vértice tiene la cara, los vértices se indican por #su
índice correspondiente
NVertices v1 v2 v3 ... vN
MVertices v1 v2 v3 ... vM
```

- Archivos .node: Contiene una lista de puntos en 3 dimensiones. En la primera línea del archivo de especifica el número de vértices, la dimensión en que se encuentran (siempre 3), el número de atributos y el indicador “boundary marker (0 o 1)”. Este último se utiliza para identificar puntos límites, para efectos de esta memoria no se utiliza. Un ejemplo de archivo.node:

```
# Número vértices, 3 dimensión, no tributos, no boundary marker
8  3  0  0
# Node Id, Coordenadas, atributos, boundary marker
1  0.0 0.0 0.0
2  1.0 0.0 0.0
3  1.0 1.0 0.0
4  0.0 1.0 0.0
5  0.0 0.0 1.0
6  1.0 0.0 1.0
```

```
 7  1.0 1.0 1.0
 8  0.0 1.0 1.0
```

- Archivos .smesh: Este tipo de archivos se compone de 4 partes: una lista de nodos, una lista de caras, una lista de agujeros, y una lista de atributos por región. Para efectos de ésta memoria, no se consideran las últimas 2 listas, así como tampoco el "boundary marker", ya que no se considera en el estudio mallas con agujeros ni se definirán regiones en las mallas. El archivo se compone de la siguiente manera:

```
<# de vértices> <dimensión> <# de atributos> <# boundary markerr>
<vértice #> <x> <y> <z> [atributos] [boundary marker]
...
<# de caras> <boundary markers>
<# de vértices> <vértice 1>...<vértice #> [boundary marker]
...
<# de agujeros>
<agujero #> <x> <y> <z>
...
<# de regiones>
<región #> <x> <y> <z> <número de región> <atributo de la región>
```

  Si la lista de nodos está vacía y se índica que el número de vértices es 0, quiere decir que la lista de nodos se encuentra en un archivo del mismo nombre pero de extensión .node.

- Archivos .m3d: En este tipo de archivos, se describe una lista de vértices y una lista de poliedros. Antes de cada vértice se tiene un indicador 1 o 3 que índica si está dada o no la dirección de proyección de dicho punto, y se númeran implícitamente partiendo desde 0. Antes de cada poliedro se tiene una letra que índica el tipo de dicho poliedro (H = hexaedro, T = tetraedro, P = pirámide, R = prisma). El archivo se compone de la siguiente forma:

```
<# de vértices>
<tipo de vértice> <x> <y> <z> <proy x> <proy y> <proy z>
...
<# de poliedros>
<tipo poliedro> <vértice 1>...<vértice #>
...
```

2.2. Conceptos ingeniería de software

2.2.1 Programación orientada a objetos

La programación orientada a objetos es un paradigma de programación basado en objetos y las interacciones que se dan entre ellos. Está fundada en base a varias técnicas e incluye herencia de objetos, abstracción y polimorfismo. Los objetos son entidades que poseen un estado representado por atributos, un comportamiento representado por sus métodos, y una identidad o identificador que lo diferencia de otros objetos [7,8]. La programación orientada a objetos define un programa como un conjunto de objetos que interactúan entre ellos para realizar distintas tareas.

Dado que el diseño de aplicaciones basadas en este paradigma de programación es muy complejo, se propone un conjunto de patrones de diseño, con el objetivo de facilitar esta tarea, y además generar código reusable, fácil de re implementar, extender metodologías, definir relaciones estables entre elementos, generar objetos extensibles, etc., de tal forma que el diseño es particular para el problema presentado, pero lo bastante general como para agregar funcionalidades a futuro [7].

2.2.2 Patrones de diseño

Los patrones de diseño son soluciones generales a problemas que ocurren recurrentemente, de tal manera que permiten reutilizar dicha solución. Se han definido un conjunto de patrones dados los problemas observados en el desarrollo de aplicaciones usando orientación a objetos.

A continuación, se presentan algunos de los muchos patrones de diseños existentes, lo cuales podrían ser de utilidad posteriormente en el diseño del presente trabajo [7]:

- Abstract Factory: Proporciona una interfaz que permite crear familias de objetos relacionados sin especificar sus clases concretas. Se utiliza cuando se quiere definir sistemas independientes de cómo se crean, componen y representan cada una de sus partes

y, cuando se desea proveer una librería de clases, revelando sólo la interfaz pero no su implementación.

- Composite: Compone objetos en estructuras de árbol para representar jerarquías generales o parciales. Permite al cliente tratar objetos individuales y composiciones de objetos de forma uniforme. Sirve para representar jerarquías, y permite ignorar la diferencia entre la composición de los objetos y el objeto individual.
- Facade: Provee una interfaz unificada para un conjunto de interfaces pertenecientes a un subsistema. Se define una interfaz de alto nivel que interactúa con el subsistema de forma más fácil. Permite proporcionar una interfaz sencilla para subsistemas completos y así mantenerlos de forma independiente de los elementos que interactúan con él.
- Factory Method: Define una interfaz para crear un objeto pero la subclase decide que clase instanciar. Permite a la clase aplazar la instanciación de la subclase. Se utiliza cuando una clase no puede anticipar el tipo de objeto que se necesitará crear, o cuando la clase delega en subclases las responsabilidades de ciertas tareas.
- Iterator: Proporciona una forma de acceso a los elementos de un objeto de agregación secuencialmente sin exponer su representación. Permite proveer una interfaz uniforme para recorrer distintas estructuras agregadas (Apoya la iteración polimórfica).
- Observer: Define una o más dependencias entre objetos de modo que cuando uno de los objetos cambia de estado, todas las dependencias de él son notificadas y actualizadas automáticamente. Es útil cuando un objeto requiere cambiar a los demás y no sabe quiénes son ni cuantos hay.
- Singleton: Asegura que una clase sólo posee una instancia y proporciona un punto de acceso global a ella.
- Strategy: Define una familia de algoritmos, encapsulando cada uno y haciéndolos intercambiables. Strategy permite al algoritmo variar independientemente de los clientes que lo usan. Es útil cuando se tienen clases relacionadas que sólo difieren en su comportamiento, o se necesitan muchas variantes del mismo algoritmo. Permite además, evitar la exposición de estructuras complejas y ocultar los datos del algoritmo.

- Template Method: Define el esqueleto de un algoritmo en una operación, diferenciando algunos pasos en subclases. Permite a las subclases redefinir ciertos pasos del algoritmo sin cambiar la estructura propia de dicho algoritmo. Permite evitar la duplicación de código, cuando un grupo de subclases tienen un comportamiento común que se centraliza en una sola clase.

2.3. Herramientas de Visualización Existentes

Existen algunas herramientas gratuitas de visualización de mallas geométricas en 3D. A continuación se describen 2 de las más usadas:

2.3.1. TetView

TetView [12] es un programa gráfico que permite visualizar mallas de triángulos y tetraedros. Se creó específicamente para visualizar mallas generadas y almacenadas a través de TetGen, un generador de mallas de tetraedros y triangulaciones, en donde el input/output son archivos de extensión .node, .poly, .smesh, .ele, .face. Sin embargo es capaz de reconocer mallas almacenadas en archivos de otras extensiones (.off, .mesh, .poly, .stl).

TetView está desarrollado en C++, y utiliza OpenGl para el rendering de los objetos. Se encuentra disponible para Linux, Windows y MAC OS y su última versión data de junio del 2004.

Esta herramienta es muy útil y presenta diversos modos de visualización, así como también permite la visualización de índices de los elementos. Sin embargo no permite visualizar mallas mixtas de poliedros ni provee evaluación de criterios de calidad sobre los elementos de una malla.

2.3.2. GeomView

GeomView [13] es un visualizador 3D interactivo que permite visualizar y manipular objetos en 3D. Permite controlar independientemente objetos y cámaras, luz, sombras, materiales, escoger un objeto a nivel de vértice, arista o elementos, etc., a través de paneles de control o del mouse. GeomView soporta datos de tipo: polígonos y vértices (.off), cuadriláteros, mallas rectangulares, vectores, superficies de Bézier, entre otros.

GeomView fue desarrollado para Unix pero es posible correrlo en Windows usando Cygwin, un ambiente de Linux para Windows. La última versión data de agosto del 2007.

Esta herramienta permite tener distintos modos de visualización y manipulación de los objetos. Sin embargo, no permite evaluar elementos utilizando criterios de calidad como los presentados en la sección 2.1.4 y la interfaz gráfica que presenta es difícil de manipular en primera instancia.

3. Diseño Visualizador y Evaluador de Mallas

En este capítulo se presenta la metodología utilizada para el desarrollo de la aplicación y los requerimientos que ésta de cumplir, se define la estructura de datos utilizada y se presenta el diseño de sus componentes principales.

3.1. Metodología

Inicialmente se realiza un análisis sobre las funcionalidades que debe tener la aplicación. Luego, se propone un diseño de la aplicación basado en el paradigma de la programación orientada a objetos y utilizando patrones de diseño, permitiendo que ésta sea fácil de extender y entender en trabajos futuros.

3.2. Requerimientos

El visualizador y evaluador de mallas a desarrollar debe permitir al usuario realizar las siguientes acciones:

- Leer un archivo que especifica una malla geométrica almacenada en alguno de los formatos de almacenamiento de mallas conocidos.
- Rotar y trasladar la malla en los ejes X, Y, Z.
- Realizar zoom-in y zoom-out sobre la malla.
- Ver los índices de cada elemento, ya sea vértices, aristas, caras o poliedros.
- Seleccionar un componente de la malla, ya sea vértices, aristas, caras o poliedro, clickeando sobre él, desde una lista de componentes o por el índice, para realizar posteriores cálculos sobre él.
- Desplegar los datos de los elementos de una malla.
- Desplegar los datos de los vértices de una malla.
- Ver sólo la superficie de una malla.
- Ver malla en modo wireframe (sólo se observan las aristas) o por polígonos coloreados.
- Elegir el color de visualización de una malla.

- Ver elementos transparentes en una malla
- Escoger y aplicar un criterio de evaluación sobre los elementos de una malla.
- Un vez aplicado el criterio de evaluación de la malla, permitir visualizar el valor del criterio por elemento, mostrar elementos que cumplen los rangos del criterio y los que no, y colorear los elementos de acuerdo al valor obtenido utilizando una escala de colores predeterminada (rojo los elementos más malos, azul los mejores).
- Permitir observar vecindades de vértices, aristas, caras o elementos, seleccionando el nivel de vecindad deseado. Aplicar criterios de evaluación a la vecindad seleccionada.
- Permitir modificar la malla cambiando la posición de un vértice seleccionado, restringiendo las condiciones de integridad sobre ella.
- Guardar una malla en el formato deseado.

3.3. Diseño de la Aplicación

A continuación se detalla el diseño de cada una de las partes del visualizador y analizador de mallas y los patrones de diseño utilizados. En el anexo A se encuentra el diagrama de clases completo de la aplicación.

3.3.1. Estructura de Datos

La representación de una malla de poliedros para efectos del desarrollo de una aplicación es un tema importante a considerar, pues, mientras más explicitas se encuentren las relaciones entre elementos, vértices, aristas y caras, más rápido es el procesamiento de información sobre los datos de la malla, pero se requiere más espacio para el almacenamiento de dichos datos [2,5].

Típicamente, en operaciones sobre mallas de poliedros, se procede primero a encontrar todas las aristas incidentes en un vértice, luego los vértices conectados por aristas, las aristas de cada cara, las caras a las que pertenece cada arista, y finalmente las caras de cada elemento [2].

Para representar la malla se definieron las siguientes estructuras de datos (Figura 7):

- Punto: Coordenadas de un punto en 3D, con sus valores en los ejes x, y, z.
- Arista: Se compone de un par de puntos. Posee un puntero a cada cara que la contiene.
- Cara: Conjunto de puntos que la componen ordenados con orientación positiva. Posee dos punteros, uno a cada elemento que la contiene, y una lista de punteros a las aristas que la componen.
- Elemento: Se compone de un contenedor de caras. De ésta clase heredan los diferentes tipos de poliedros que existen (prisma, hexaedro, tetraedro, pirámide) y sus diferentes propiedades, tales como, número de puntos, caras, aristas, etc. Posee punteros a las aristas y vértices q lo componen.
- Malla: Se compone de un contenedor de vértices, uno de aristas, uno de caras y, en el caso de una malla 3D, uno de poliedros (elementos).

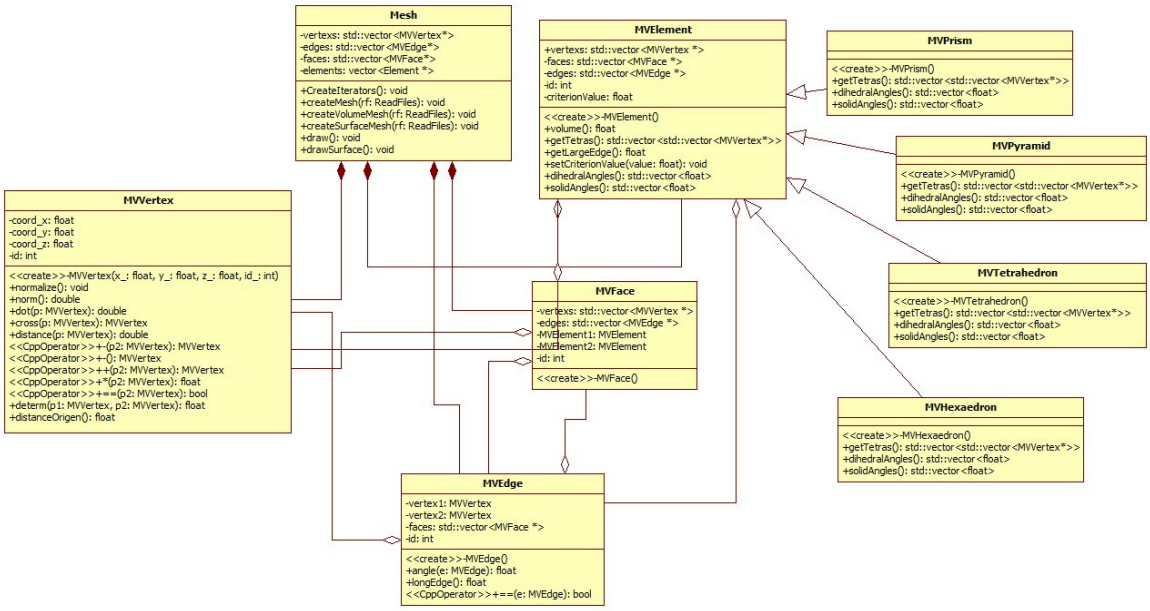


Figura 7: Estructura de Datos para representar una malla de poliedros.

En el caso de mallas de superficie, sólo se manejan los datos a nivel vértice, arista y cara (o polígono).

3.3.2. Lectura y Guardado de Archivos

Se posee, gracias a un trabajo pre-existente, un sistema gestor de archivos que permite abrir y guardar mallas en distintos formatos de almacenamiento. Se utiliza el patrón de diseño Facade para integrar este subsistema (Figura 8), en donde ReadFile y WriteFile rereciben las peticiones de lectura (abrir) y escritura (guardar) del usuario a través de Controller, que es la clase que ejecuta todos los componentes del visualizador, se comunican con el subsistema ejecutando la petición y recibiendo el resultado enviándolo de regreso al Controller, funcionando como un intermediario entre el visualizador y el subsistema.

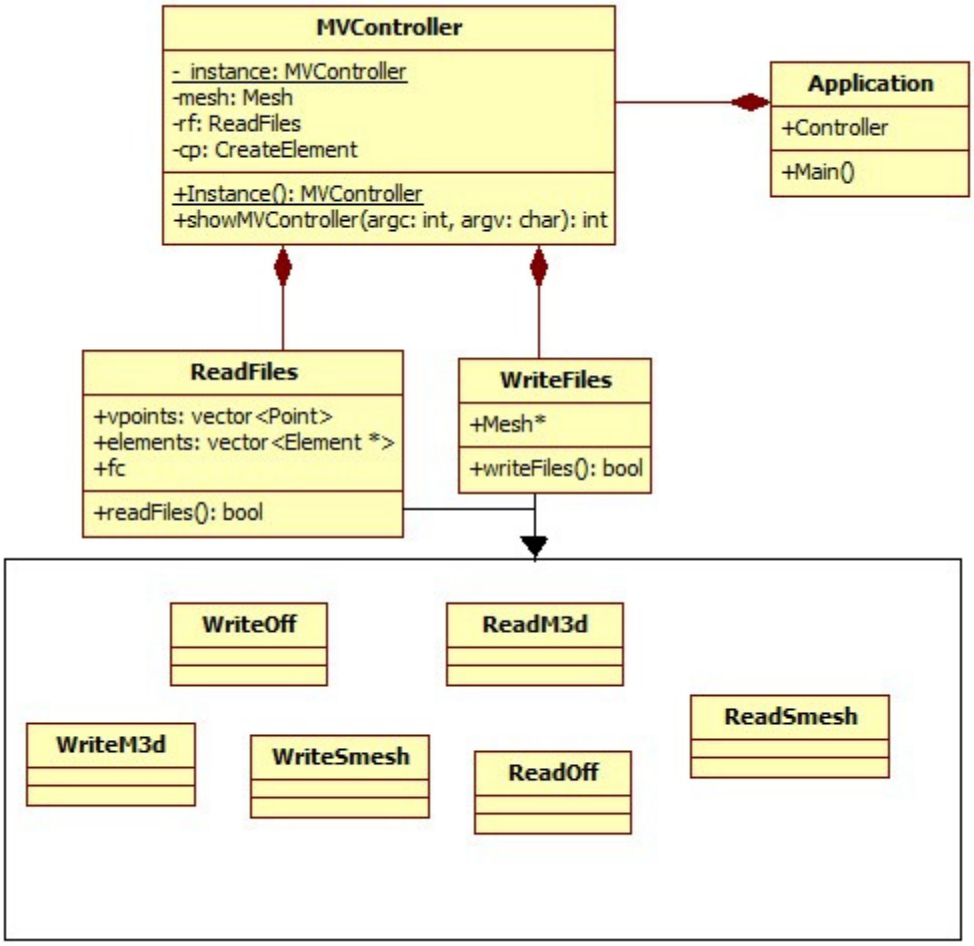


Figura 8: Facade sobre módulo de manejo de archivos.

3.3.3. Creación de los elementos de la malla

Una vez que se abre la malla, los datos leídos del archivo de almacenamiento son entregados a la clase Mesh, que representa la malla. Esta clase, genera los vértices (vertex), aristas (edge) y caras (face) y todas las relaciones entre dichos elementos almacenándolos en los contenedores respectivos. Luego se utiliza el patrón de diseño FactoryMethod para generar las clases que se encargan de crear los distintos poliedros contenidos en la malla y almacenarlos en el contenedor de elementos (Figura 9).

La clase Mesh posee un objeto de la clase CreateElement que se encarga de crear un elemento (poliedro) de la malla dados los datos. Por cada elemento, de acuerdo a sus características (número de vértices, número de caras, etc.), la clase CreateElement decide qué tipo de poliedro se tiene y se inicializa un objeto de la clase CreatePrism, CreateTetrahedron, CreateHexaedrom o CreatePiramid, las cuales heredan de FactoryElement, según corresponda. Estos, se encargan de crear respectivamente un objeto Prism, Tetrahedron, Pyramid o Hexaedron, los cuales heredan de Element, para ser almacenados en la malla.

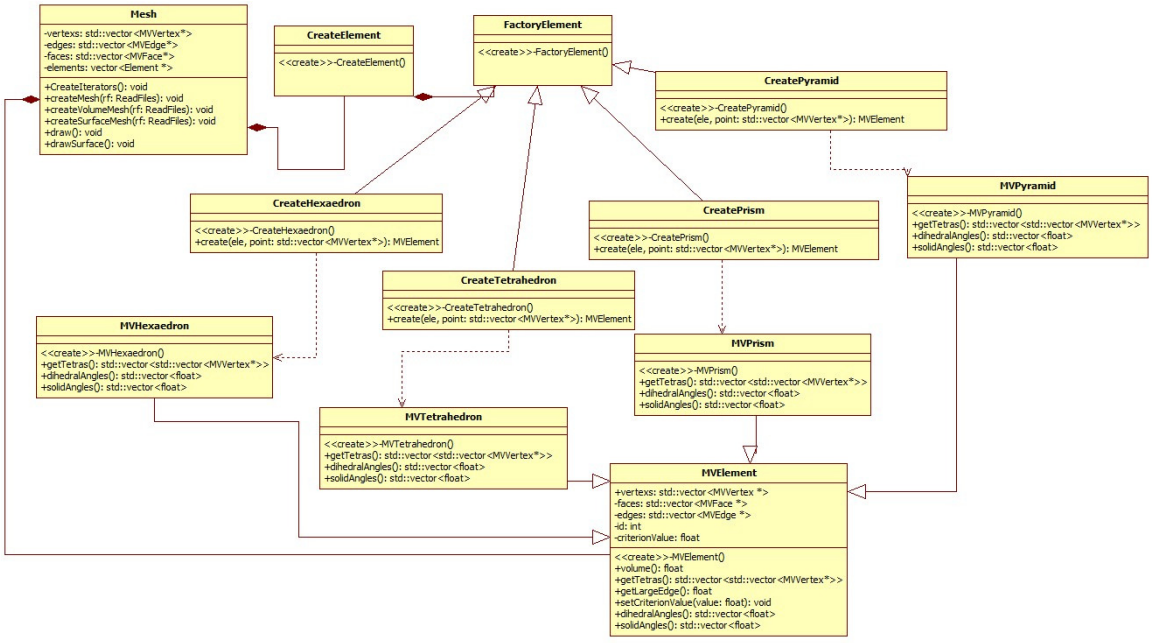


Figura 9: Creación de los elemento de la estructura de la malla.

3.3.4. Iteración sobre la Malla

Para iterar sobre los elementos de la malla se utiliza el patrón de diseño Iterator (Figura 10). Se tienen MeshIterator y FaceMeshIterator que poseen los métodos First(), para ir al primer componente, Next(), para ir al siguiente componente, y Current(), para obtener el componente actual sobre los cuales cada uno de ellos iteran.

MeshIterator recorre uno a uno los elementos de la malla. Se utiliza cuando se necesita calcular alguno de los criterios de evaluación sobre la malla. FaceMeshIterator recorre el contenedor de caras de la malla. Se utiliza cuando se desea hacer rendering de la malla.

La clase Mesh crea uno u otro Iterador según lo necesite.

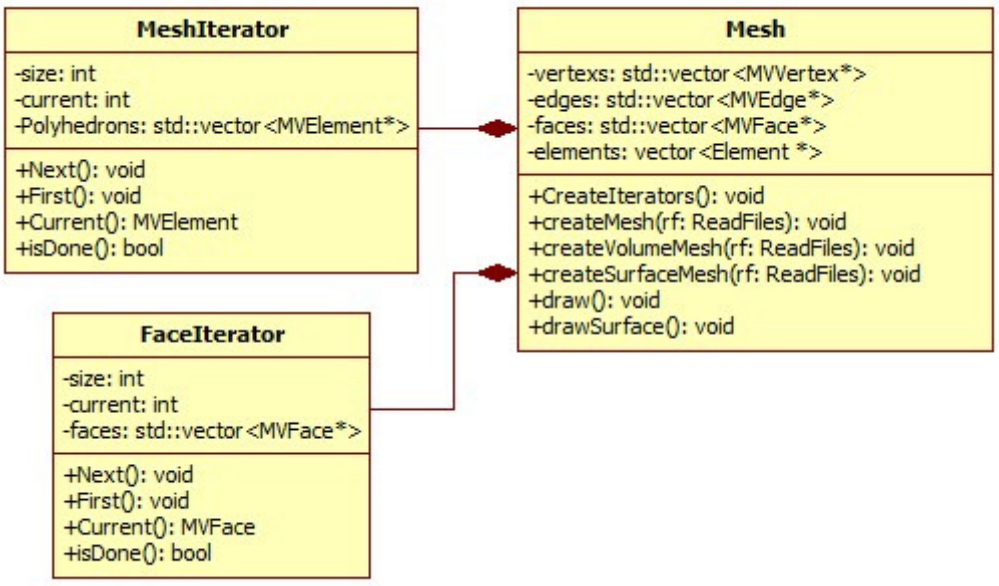


Figura 10: Iteradores sobre la estructura de la Malla.

3.3.5. Cálculo de Criterios

Ya que se necesita calcular distintos criterios sobre la malla, pero sin saber a priori cual de todos se desea calcular, se utilizó el patrón de diseño Strategy para modelar los criterios de calidad a aplicar sobre la malla (Figura 11).

La clase Mesh posee un objeto de la clase Criterion. De ésta clase heredan las clases:

- CAngleSolidMin y CAngleSolidMax, que se encargan de calcular el ángulo sólido mínimo y máximo de cada elemento respectivamente.
- CAngleDihedralMin y CAngleDihedralMax, que se encargan de calcular el ángulo diedro mínimo y máximo de cada elemento respectivamente.
- CJacobian, que se encarga de calcular el Jacobiano del elemento.
- CRateVolume, que se encarga de calcular la razón volumen y largo de la arista más larga al cubo.

Todas estas clases, redefinen el método apply() de Criterion. Luego, cuando se dice a la malla qué criterio debe calcular sobre sus elementos, se inicializa el objeto Criterion utilizando la subclase que corresponda y se aplica el método apply() sobre los elementos de la malla. El método apply recibe como parámetro el elemento a evaluar y retorna el valor del criterio, el cual se almacena en el elemento. Si se desea crear un nuevo criterio, sólo debe heredar de Criterion e implementar apply().

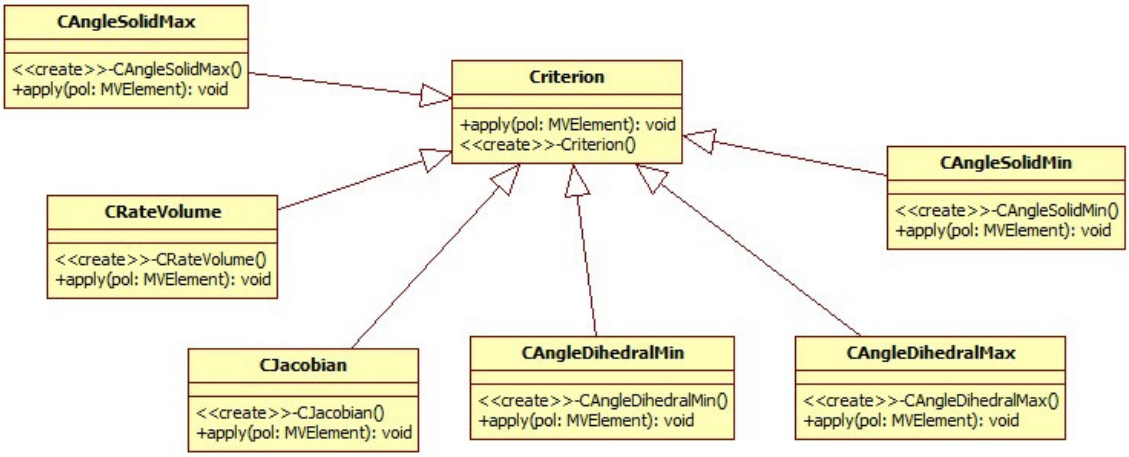


Figura 11: Patrón Strategy para cálculo de criterios sobre la malla.

3.3.6. Inicialización del visualizador y Componentes

Se tiene la clase Application, en donde se encuentra el Main del programa, crea una instancia de la clase Controller, la cual representa al visualizador de mallas. Se utiliza el patrón de diseño Singleton para asegurar que la instancia del visualizador es única.

La clase Controller es un objeto controlador a través de la cual interactúan todos los elementos de la aplicación:

- Crea la vista View del visualizador y la inicializa. La vista le entrega a Controller los requerimientos del usuario junto con el input entregado, y Controller con los datos necesarios a visualizar.
- Crea e instancia ReadFile y WriteFile. Envía la solicitud de lectura de archivos a ReadFile y recibe los datos leídos a través del gestor de archivos. Así mismo entrega los datos de la malla al gestor para el almacenamiento de la malla en un archivo usando WriteFile.
- Crea e instancia la malla Mesh que se desea visualizar, entregando como datos el input obtenido del gestor de archivos. De ser solicitado por el usuario, actualiza la malla en estudio y notifica de esto a la vista.

- La clase View es quien se encarga de crear la interfaz de visualización utilizando las librerías de Qt. Además, crea un objeto GlView, que consiste en una ventana de visualización que está contenida en la interfaz y que utiliza las librerías de OpenGl.

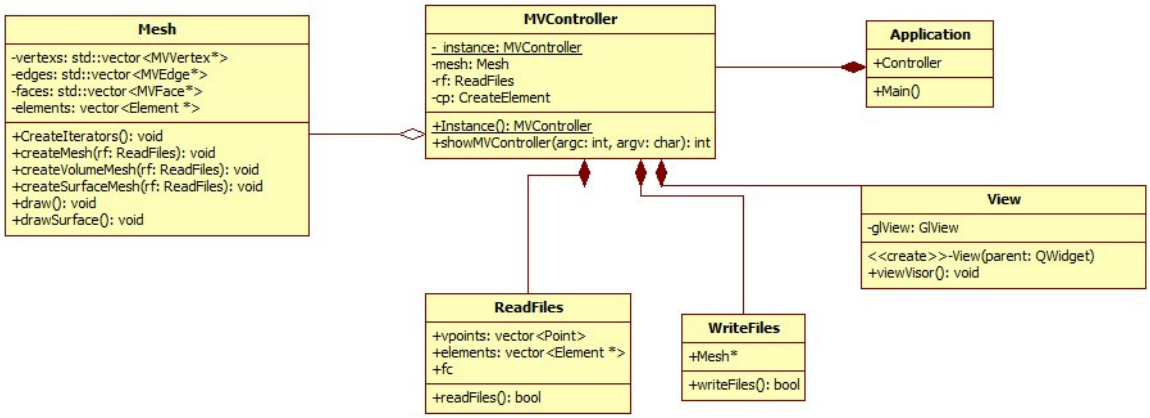


Figura 12: Inicialización de visualizador y componentes.

4. Implementación

En el presente capítulo se presentan detalles de la implementación de la solución propuesta, entre los que se encuentras las herramientas y recursos utilizados, los algoritmos presentes y las opciones disponibles a través de la interfaz de la aplicación.

4.1. Recursos Disponibles

4.1.1. Gestor de Archivos

Del desarrollo de un proyecto anterior, se encuentra disponible un gestor de archivos, el cual permite la manipulación de archivos de distintos estándares de almacenamiento de mallas geométricas.

Este gestor, permite abrir, y guardar archivos, y pasar un archivo de un formato a un archivo de otro formato. La información obtenida al abrir un archivo se almacena en una estructura de datos que se compone de las clases: punto, cara y elemento. De la clase elemento heredan los distintos poliedros básicos que componen las mallas en estudio. Estas clases son propias del gestor y no son las mismas que se definen en la estructura de datos de la aplicación en desarrollo. Los datos guardados por el gestor al abrir una malla, se utilizan para crear la estructura de datos propia definida previamente en el diseño.

Las funcionalidades de este gestor se encuentran disponibles para las siguientes extensiones de archivo: Ansys, M3d, Mdl, Mesh, Off, Smesh.

4.1.2. Librería Jacobiano

Se posee, también de un trabajo anterior, una librería que calcula el jacobiano de distintos poliedros. Esta librería posee una clase para el cálculo del jacobiano, de la cual extienden los cálculos del jacobiano de distintos poliedros. Además, posee todo el cálculo matemático necesario, lo que la hace fácilmente extensible a nuevos poliedros de ser necesario.

Se encuentra disponible el cálculo del jacobiano para: prismas, tetraedros, hexaedros y pirámides.

Esta librería se modificó de tal forma de hacerla compatible con la estructura de datos definida para este desarrollo. En particular, se identificó en donde se utilizaba la estructura original de elementos, caras, arista y vértices de la librería, y se reemplazó por el equivalente en la estructura de datos del visualizador.

4.2. Ambiente de Desarrollo

El desarrollo de la aplicación se llevó a cabo utilizando el lenguaje de programación C++. Se trabajó en sistema operativo Windows XP y se utilizó como IDE Visual Studio C++ 2008. A pesar de eso, es posible compilar y ejecutar la aplicación en sistema Linux,

Ya que la aplicación en si consta de una aplicación gráfica y es necesario realizar rendering de los elementos para visualizarlos, se utilizaron dos librerías gráficas Open Source disponibles que se detallan a continuación.

4.2.1. OpenGL

OpenGL es una librería gráfica que provee un estándar multiplataforma para producir gráficas 2D y 3D [9, 10, 11]. La interfaz de OpenGL consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Es masivamente usada en un gran número de aplicaciones gráficas.

OpenGL funciona como una máquina de estado, cuando se activan o configuran varios estados de la máquina, sus efectos perdurarán hasta que sean desactivados.

Esta librería permite dibujar escenas, y en este caso mallas geométricas, con una gran variedad de opciones, pero no provee las herramientas para crear interfaces de usuario aunque si es posible integrarla con librerías que si lo hacen, como por ejemplo Qt.

En este desarrollo, se utilizó de la librería las características que permiten dibujar polígonos y líneas, escribir en pantalla, modelar luz, sombreado y transparencias, definir una cámara sobre la escena y moverla de acuerdo a la conveniencia del usuario, entre otras.

4.2.2. Qt

Qt es una librería multiplataforma para el desarrollo de interfaces de usuario [14], y también para el desarrollo de programas sin interfaz gráfica como herramientas de consola y servidores.

Qt provee su propio IDE de desarrollo QtCreator para trabajar con C++. En ese ambiente de desarrollo, así como en todos los SDK (Kit de desarrollo de Software) de QT que están en su web, se tienen las librerías compiladas dinámicamente, es decir, al distribuir la aplicación deberá ir con sus dll.

Para utilizar Qt en otros ambientes, como por ejemplo Visual Studio, debe compilarse la librería de forma estática. En el caso de Visual Studio, es posible compilar la librería usando su propia consola. Primero se debe usar la herramienta configure que trae la librería de la siguiente forma:

```
configure -platform win32-msvc2008 -debug-and-release -static -nomake examples -nomake demos -no-qt3support -no-scripttools -no-openssl -no-webkit -no-phonon -no-style-cde -no-style-cleanlooks -no-style-plastique -no-sql-sqlite
```

La opción `-platform win32-msvc2008` indica el compilador que se está utilizando. Se dejan fuera los paquetes de Qt que no son necesarios, así como también los ejemplos y las demos, pues su compilación tarda varias horas y ocupa varios GB de memoria en disco. Luego se ejecuta `nmake` y se compilan las librerías. Al hacer esto en la consola de Visual Studio, Qt queda automáticamente integrado en el ambiente de desarrollo. Se debe agregar en la configuración de compilación el comando: /I $(QTDIR)\bin\moc.exe, pues al compilar las aplicaciones Qt utiliza este ejecutable para generar código dinámicamente que es necesario para la generación de la interfaz.

En el desarrollo de esta aplicación se utilizó la versión de Qt 4.6.3. Se utilizó de la librería las herramientas que permiten crear ventanas de visualización, crear y agregar menúes, cuadros de diálogo, cuadros de color, enviar señales entre la interfaz y el controlador de la aplicación, entre otras.

4.3. Manejo de Archivos

Como se mencionó anteriormente, se posee de un trabajo previo un gestor de archivos que permite abrir y guardar archivos de distintas extensiones para el almacenamiento de mallas geométricas. Este gestor posee las siguientes clases para lectura y escritura de archivos: ReadAnsys, ReadM3d, ReadMdl, ReadMeshSurface, ReadMeshVolume, ReadOff, ReadSmesh, WriteAnsys, WriteM3d, WriteMdl, WriteMeshVolume, WriteSmesh.

Se creó una clase ReadFile y una clase WriteFile. Estas clases, al abrir o guardar una malla respectivamente, verifican la extensión del archivo y de acuerdo a ello crean un objeto de la clase que corresponde, el cual se encarga de la lectura o escritura de datos.

Las clases ReadFile y WriteFile, permiten abrir archivos de todas las extensiones presentes en el gestor: Ansys, M3d, Mdl, MeshSurface, MeshVolume, Off y Smesh, pero sólo se trabajó con mallas almacenadas en M3d, Smesh y Off.

Este gestor de archivos provee además su propia estructura de datos, la cual se compone de una clase Element de la que heredan los distintos poliedros, una clase Face que representa una cara, una clase Point que representa un vértice y una clase FaceContainer que es un contenedor de caras. Esta estructura de datos se utiliza en el visualizador sólo al abrir archivo, pues fue creada y pensada para efectos de otra aplicación.

Las clases de este gestor que abren archivos que contienen mallas de superficie, almacenan un vector de Point y un FaceContainer. En el caso de archivos con mallas 3D, estas clases almacenan un vector de Point y uno de Element. Esta información es obtenida por la clase ReadFile y entregada al Controller, que tal como se vio en el Diseño en la sección 3.3.2, es quien se encarga de comunicar las distintas partes de la aplicación. Finalmente el Controller entrega esta información a la clase Mesh, para que se genere la malla sobre la cual se realizarán luego todas las operaciones solicitadas por el usuario.

Las clases del gestor que almacenan mallas en archivos, fueron modificadas para tomar los datos directamente desde la estructura de datos del visualizador y no de la original del gestor. Además se agregó la clase WriteOff para guardar mallas en archivos .off.

4.4. Almacenamiento y Manejo de Mallas

Una vez que se ha leído el archivo que contenía almacenada la malla, la clase Controller inicializa el objeto de tipo Mesh y le entrega los datos obtenidos. La clase Mesh identifica si se trata de una malla de superficie o de volumen y de acuerdo a eso ejecuta diferentes métodos: `createVolumeMesh o createSurfaceMesh`, respectivamente. A continuación se detalla cómo se crea la malla a través de estos métodos:

- `createSurfaceMesh`: en el caso de una malla de superficie, los datos que se tienen son un vector de puntos y un contenedor de caras. Para crear la malla se siguen los siguientes pasos:

  - Se crean todos los vértices obteniendo los datos del vector de puntos y se alamacenan.
  - Se itera sobre el contenedor de caras. Las caras en el contenedor contienen el Id de los vertices que la componen. Por cada cara se crea un objeto MVFace agregando una referencia a cada vértice.
  - Por cada cara agregada, se obtienen las aristas de dicha cara y se crean los objetos de tipo MVEdge.
  - Se verifica si la nueva arista ya estaba o no contenida en la malla, revisando las aristas ya alamcenadas. De no estar contenida previamente en la malla se agrega a ella y se agrega la referencia de la cara a la arista y la referencia de la arista a la cara. Si ya estaba contenida en la malla, sólo se agregan las referencias correspondientes, tanto en la arista como en la cara.

  Esto toma tiempo orden n*m, es decir n^2, n por la cantidad de caras y m por la cantidad de aristas, pero se realiza sólo una vez en toda la interacción con la malla.

- `createVolumeMesh`: en el caso de una malla de volumen, se tiene un vector de puntos y un vector de elementos. Cada elemento posee a su vez un vector de puntos en donde los índices del vector coinciden con los vértices ordenados como se vió en la figura 6 d ela sección 2.1.4.3. Para crear la malla se siguen los siguientes pasos:
  - Se crean todos los vértices obteniendo los datos del vector de puntos y se alamacenan.

- Se itera sobre el vector de elementos. La malla posee un objeto de la clase CreateElement. Se identifica que tipo de elemento es cada uno por el número de vértices que posee y se instancia FactoryElement con la subclase que corresponda (CreatePrism, CreatePiramid, CreateTetrahedron, CreateHexaedron).
- CreateElement crea un objeto MVElement del subtipo que corresponda (MVPrism, MVPyramid, MVTetrahedron, MVHexaedron) y este se agrega a la malla. Esto toma tiempo de orden n, correspondiente a la cantidad de elementos de la malla.
- Luego, se identifican las caras del elemento y se crean objetos del tipo Face. Se verifica si esa cara ya está en la malla, si no está se agrega y se agrega una referencia de la cara al elelemto y una del elemento a la cara. Si la cara ya está, sólo se actualizan las referencias tanto de la cara como de poliedro. Esto toma tiempo orden n, correspondiente al número de caras.
- Además, cada vez q se agrega una cara, se utiliza el mismo algoritmo que en el método anterior, sólo que esta vez también se actualizan las referencias del elemento. Esto toma tiempo orden n, correspondiente al número de aristas.

En total, esto toma tiempo orden n^3, pero se realiza sólo una vez en toda la interación con la malla.

Una vez finalizado este proceso se tiene la malla generada con las referencias de todos los elementos actualizadas. Es importante notar que sólo la clase Mesh contiene los elementos propiamente tal, las demás clases: MVElement, MVFace y MVEdge, sólo poseen referencias a los elementos, de modo que sólo basta actualizar los datos en los contenedores de Mesh para que se actualicen en toda la estructura.

4.5. Cálculo de Criterios

Para realizar el cálculo de criterios sobre los elementos de una malla se deben considerar dos objetos: el que representa el criterio y el iterador sobre la malla. La clase Mesh posee un objeto de la clase Criterion y uno de la clase MeshIterator. Luego, identifica que criterio es el que se desea calcular e instancia el objeto Criterion con la subclase que corresponda.

Finalmente, se itera sobre los elementos de la malla utilizando MeshIterator y se aplica el criterio sobre el elemento utilizando el método virtual `apply(MVElement * p)` de Criterion, almacenando en el poliedro el valor obtenido. Todos los datos que el criterio requiere saber para calcular su valor se encuentran contenido es el elemento.

A continuación se detalla cómo funciona el método apply() en cada tipo de criterio.

4.5.1. Ángulo Sólido

El ángulo sólido se aplica sobre los vértices de un elemento.

En este caso se poseen 2 criterios: CAngleSolidMax y CAngleSolidMin. Ambos calculan el ángulo sólido pero almacenan el valor del ángulo máximo y mínimo obtenido respectivamente. Estas clases ejecutan el método `solidAngles()` de MVElement, obteniendo el valor de todos los ángulos en el elemento, luego calculan el máximo y el mínimo respectivamente. A continuación se describe el método `solidAngles()`:

- Se itera sobre los vértices del elemento y se calcula el valor del ángulo sólido.
- Para calcular el valor del ángulo sólido de vértice se identifica la superficie que se genera de la proyección de todas las aristas que salen del vértice y que pertenece al elemento.
- Si ésta superficie es un triangulo, se aplica la fórmula expuesta en el capítulo 2.1.4.1.
- Si ésta superficie es un cuadrilátero, se divide este en 2 triángulos y se calcula el ángulo sólido para ambos utilizando la misma fórmula y sumando el resultado obtenido de ambos.

Esto toma tiempo constante sobre cada elemento. Si se calcula sobre todos los elementos toma tiempo lineal.

4.5.2. Ángulo Diedro

Se tiene que para un elemento la cantidad de ángulos diedros que posee se corresponde con la cantidad de aristas. Por lo tanto, aprovechando el hecho que cada arista posee una referencia a las caras que la contienen, se itera sobre las aristas del poliedro.

En este caso se poseen 2 criterios: CAngleDihedralMax y CAngleDihedralMin. Ambos obtienen el ángulo diedro con el método `dihedralAngles()` de MVElement calculando luego el máximo y mínimo respectivamente. El método `dihedralAngles()` se describe a continuación:

- Se itera sobre cada arista.
- Se busca en las referencias a las caras que la contienen las 2 que pertenecen al polígono.
- En cada cara se busca una arista que comparta un vértice con la arista en común.
- Se utiliza el método Gram-Schmidt[17] para encontrar un vector contenido en cada cara ortogonal a la arista común utilizando la arista encontrada en el paso anterior.
- Se trasladan ambos vectores a un punto pivote en la arista común y se calcula el ángulo formado entre ellos.

Esto toma tiempo lineal sobre cada elemento, pues la cantidad de componentes que se debe recorrer por cada elemento es contante. Si se calcula sobre todos los elementos toma tiempo lineal.

4.5.3. Razón Volumen Arista más Larga

Para el cálculo de este criterio, la clase MVElement posee un método `volume()` que calcula el volumen del poliedro como se explico en el punto 2.1.4.3. Además MVElement posee el método `getLargeEdge()` que calcula la arista más larga del elemento.

Luego, CRateVolume utiliza estos dos métodos obteniendo los valores que necesita y divide el volumen por el valor de la arista más larga al cubo.

Esto toma tiempo constante sobre de cada elemento. Si se calcula sobre todos los elementos de la malla toma tiempo lineal.

4.5.4. Jacobiano

En el caso de este criterio, se utiliza la librería que se posee para el cálculo del jacobiano. Esta librería se adaptó para funcionar con la estructura de datos definida para esta aplicación y posee las siguientes clases de interés:

- JacElement
- JacHexahedra
- JacPrism
- JacPyramid
- JacTetrahedra

En donde JacElement es clase la padre de todas las otras.

Finalmente para calcular el jacobiano de un poliedro, se tiene un objeto JacElement en MVElement. Cada subclase de MVElement instancia JacElement con la subclase que corresponda según su tipo. Con este objeto se calcula el valor del criterio.

4.6. Interfaz y Funcionalidad

Como ya se mencionó anteriormente, se utilizó Qt para el desarrollo de la interfaz de usuario (GUI). Los elementos de una GUI creada utilizando Qt se conectan con los distintos métodos que ejecutan una funcionalidad enviándoles una señal cada vez que el usuario realiza alguna acción ya sea de teclado o mouse.

Así, por cada funcionalidad hay una acción en la interfaz y un método que la ejecuta y deben conectarse usando:

```
connect(action, SIGNAL(triggered()), this, SLOT(method()));
```

Aquí, `action` es el elemento de la interfaz que envió la señal, en `SIGNAL` se especifica el tipo de señal, `this` es el objeto que envía la señal y en `SLOT` se indica la función a ejecuta.

En caso que esta acción ejecute alguna acción de respuesta sobre los elementos de la interfaz, entonces se envia del mísmo modo una señal a la interfaz.

Así, una interfaz de usurio se compone de menúes y elementos que involucran acciones que envian señales a métodos para que sean ejecutados, y de ser necesario estos envian una señal de regreso notificando a la interfaz que la acción ha sido realizada.

La interfaz de usuario de la presente aplicación (Figura 13), se compone de 3 grandes partes: Un menú principal (Figura 14), una barra de acciones simples y una ventana de visualización.

- Menú principal: se encuentra en la parte superior de la ventana de la aplicación y en el se presentan todas las opciones de visualización, las de evaluación con criterios y las de vecindades.
- Barra de acciones simples: Se encuentra en el costado izquierdo de la ventana y se compone de un conjunto de sliders que el usuario controla con el uso del mouse. Los 3 sliders superiores corresponden a la rotación, las 3 siguientes a la traslación, y el último es zoom.
- Ventana de Visualización: Se encuentra al centro derecho de la pantalla y es en donde se visualizan las mallas.

La ventana de visualización es, tal como su nombre lo dice, una ventana dentro de la ventana principal, en ella se realizan todas las acciones y cambios de estado con la librería OpenGL. Cuando se realiza una acción de rotación, traslación o zoom con los sliders, los métodos que reciben la señal de estas acciones inicializan los valores de la ventana de visualización que se usan en las funciones glRotate, glTranslate y glScale de OpenGL respectivamente.

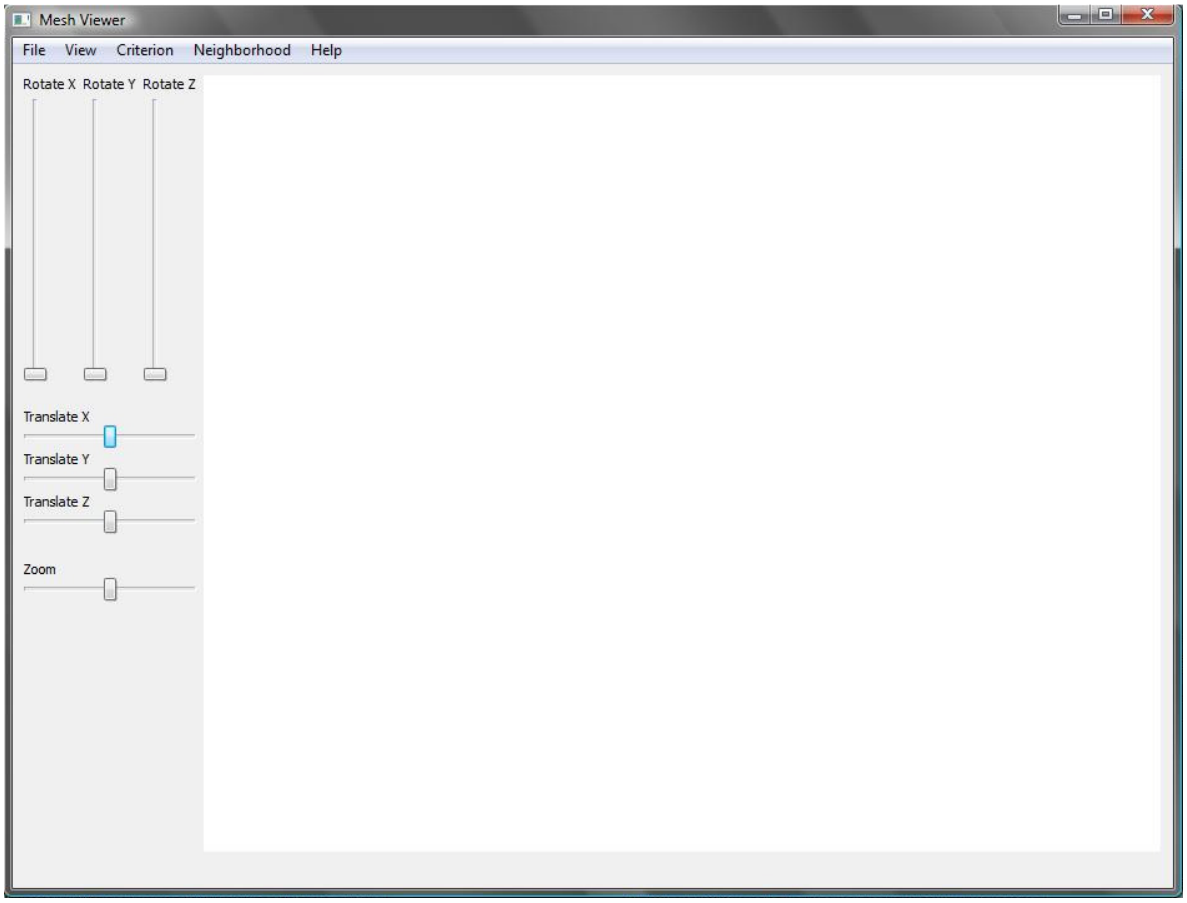


Figura 13: Interfaz de Usuario principal.

Figura 14: Menú de la Interfaz de Usuario

A continuación se muestran las funcionalidades presentes en la barra de menú principal de la interfaz:

- Abrir Archivo: En el menú File de la barra de menú se encuentra la opción de abrir un archivo. Al acceder, se abre un cuadro de diálogo (Figura 15) en donde se busca el directorio y se escoge la extensión del archivo que se desea abrir. Este cuadro de diálogo lo provee la librería de Qt, y retorna como respuesta la dirección y nombre del archivo. Esto se notifica a Controller, quien solicita al gestor de archivos que abra el archivo y lea los datos. Luego, pasa estos datos a Mesh, quien inicializa la malla. Finalmente Controller notifica a la ventana de visualización que la malla ha sido creada, está ejecuta el método `draw()` de Mesh y se dibuja la malla en pantalla. El método `draw()`, crea un iterador sobre la malla de tipo FaceIterator, y se dibujan las caras contenidas en la malla usando la primitiva `GL_POLIGON` de OpenGL.

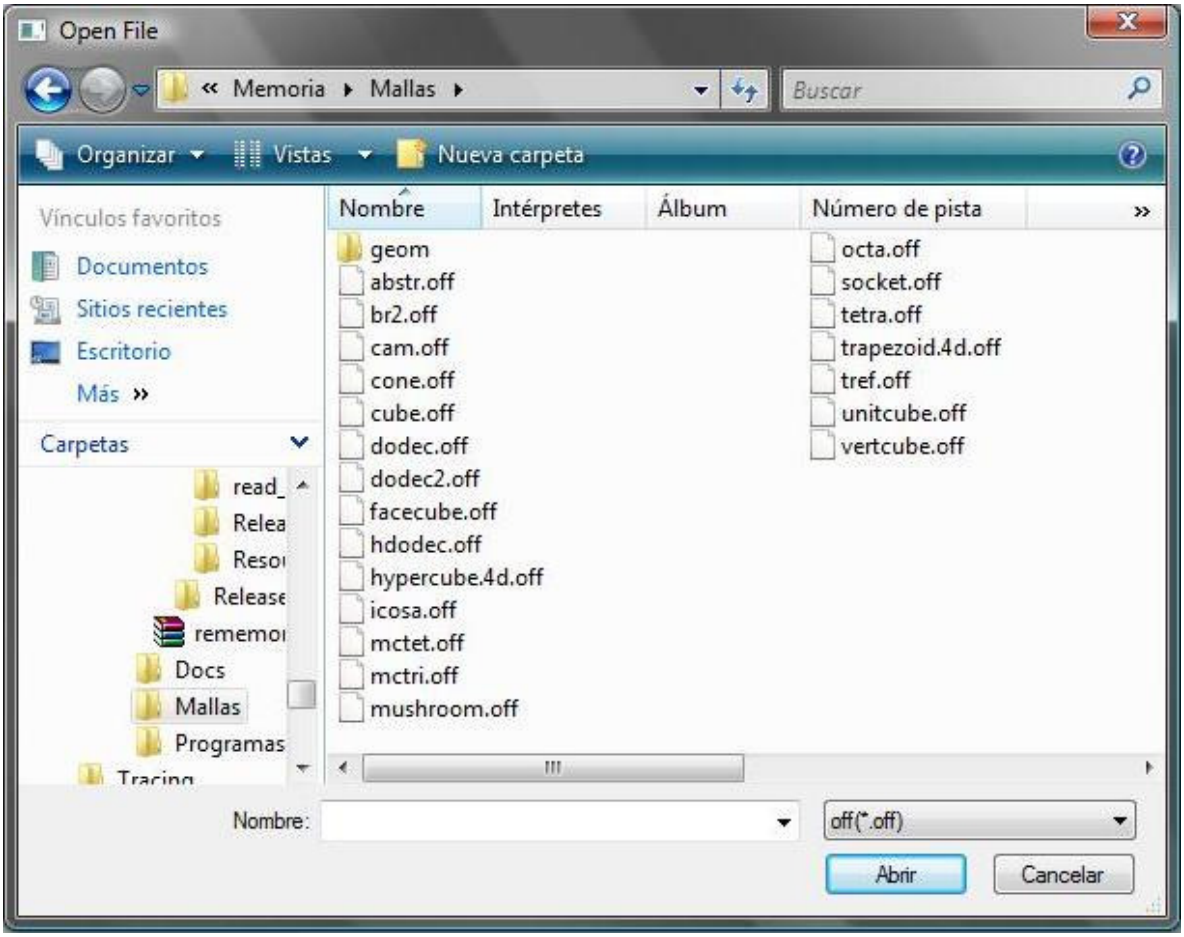


Figura 15: Abrir Archivo

- Guardar Archivo: En el menú File de la barra de menú se encuentra la opción guardar un archivo. Al acceder, se abre un cuadro de diálogo que al igual que el caso anterior es provisto por las librerías de Qt (Figura 16). Se obtiene el nombre del archivo, la extensión y la ubicación en donde se desea guardar y se notifica a Controller, quien obtiene los datos de la malla desde Mesh y solicita al gestor de archivos que almacene los datos, pasándole una referencia de Mesh.

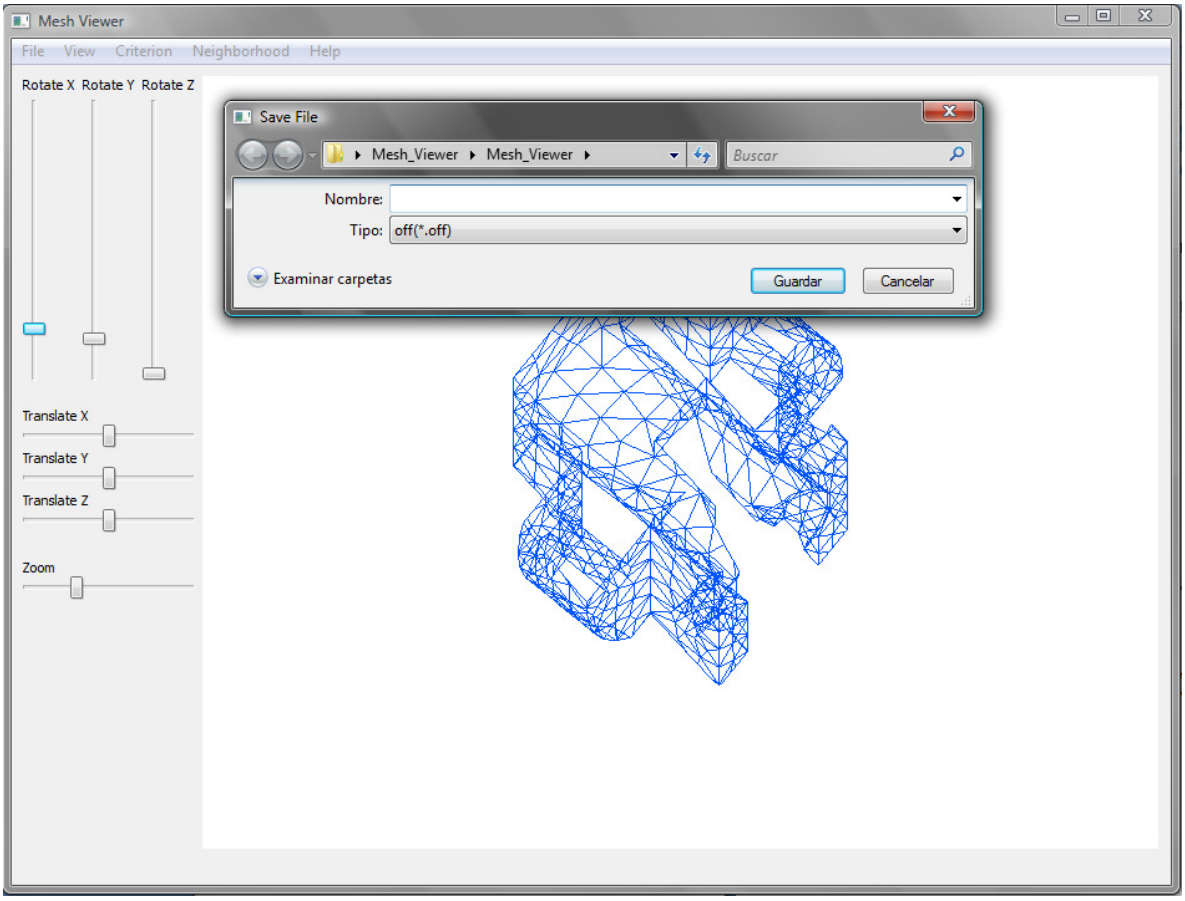


Figura 16: Guardar Archivo.

- View: Dentro de este menú se encuentran diversas acciones (Figura 17).

  Primero, se da la opción de ver los índices de los elementos (Figura 18). Dentro de esa opción, se despliega un submenú en donde se selecciona que índices se desea visualizar en pantalla: vértices, aristas, caras o elementos.

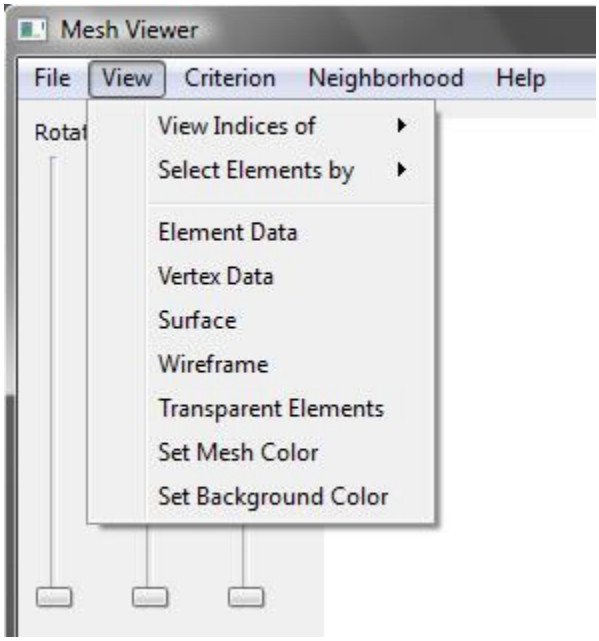


Figura 17: View Options

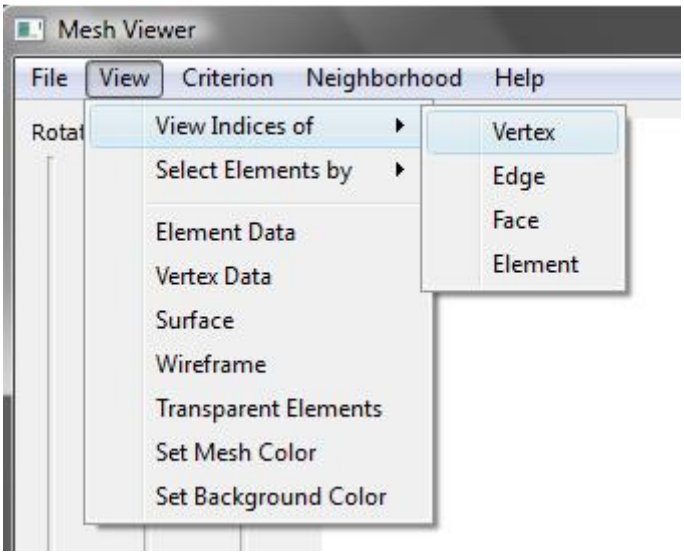


Figura 18: View Index

Luego, se da la opción de seleccionar un elemento (Figura 19). Se despliega un submenú con 2 partes. En la primera parte se escoge qué tipo de componente de la malla se desea seleccionar. En la segunda parte se indica cómo se quiere seleccionar el elemento. Si el usuario escoge la opción de seleccionar por Id, se abre un cuadro de diálogo (Figura 20)

en donde salen los Ids de los elementos y se permite al usuario seleccionar uno. Esto toma tiempo orden 1 pues el acceso al elemento es directo. Si el usuario escoge hacer clic con el mouse, la ventana de visualización escucha el evento de mouse y escoge el elemento más cercano al lugar donde se realizó el clic. Encontrar la componente más cercana toma orden n, correspondiente al total de componentes. Si el usuario escoge seleccionar por los datos de la malla, se le despliega una ventana con los datos de los elementos en donde puede escoger el elemento deseado. Este acceso también es directo y toma tiempo orden 1. En todos los casos el sistema guarda el Id y tipo del elemento seleccionado atento a cualquier acción posterior que el usuario requiera realizar sobre él.

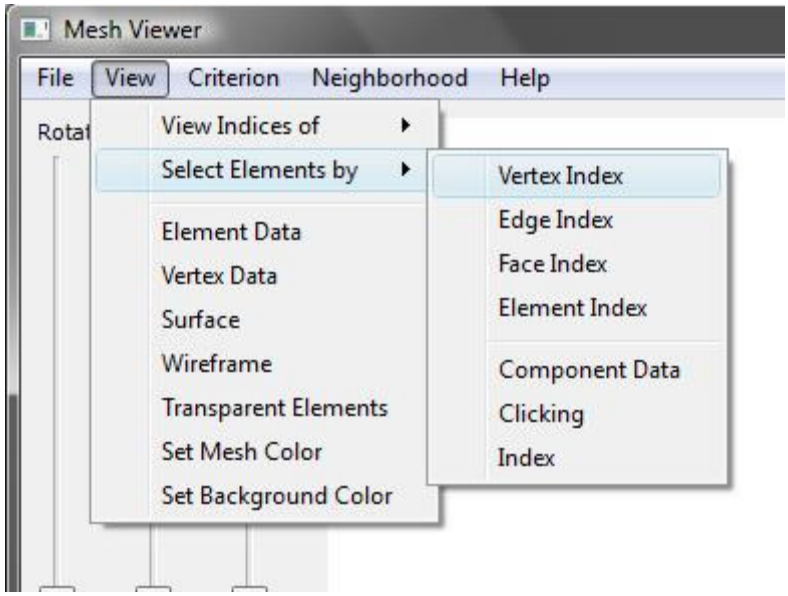


Figura 19: Select Element

Figura 20: Choose Element

Tanto Mesh Data y Vertex Data, despliegan una ventana en donde se pueden observar los datos numéricos de todos los elementos y vértices que componen la malla respectivamente. En el caso de los elementos, se muestran los índices a los vértices, en el caso de los vértices, se muestran los valores de las coordenadas. Para obtener estos datos, se recorre la malla utilizando MeshItarator y se obtienen los datos. Esto toma tiempo lineal.

Surface, permite visualizar sólo la superficie de una malla en 3D. Esto se logra dibujando sólo las caras que poseen referencia a un solo elemento y no son compartidas por 2 elementos de la malla. Para ello, Mesh posee el método `drawSurface()`, y View decide si debe utilizarlo o no. La operación de saber cuántos elementos comparten una cara toma tiempo orden 1, por lo tanto dibujar la malla de superficie toma tiempo lineal.

Wireframe y Transaparent Elements, hacen referencia específicamente a opciones de visualización que corresponden a opciones o estado de funciones de OpenGL. En el primer caso, se visualizan sólo las aristas de los elementos, de lo contrario estos de colorean. La opción de OpenGL que permite esto es `glPolygonMode(GL_FRONT, GL_LINE)`. En el segundo caso, se refiere a si se desea ver o no a través de los elementos, para ello OpenGL provee la opción de activar la variable `Gl_BLEND`.

Finalmente se da la opción de seleccionar el color en que se desea visualizar la malla. Para ellos se despliega un cuadro de diálogo de colores que provee Qt (Figura 21), el cual retorna el color escogido. Luego, en la ventana de visualización se inicializa con este color la variable que se utiliza en la función de OpenGL glColor. Del mismo modo se da la opción de cambiar el color del Background.

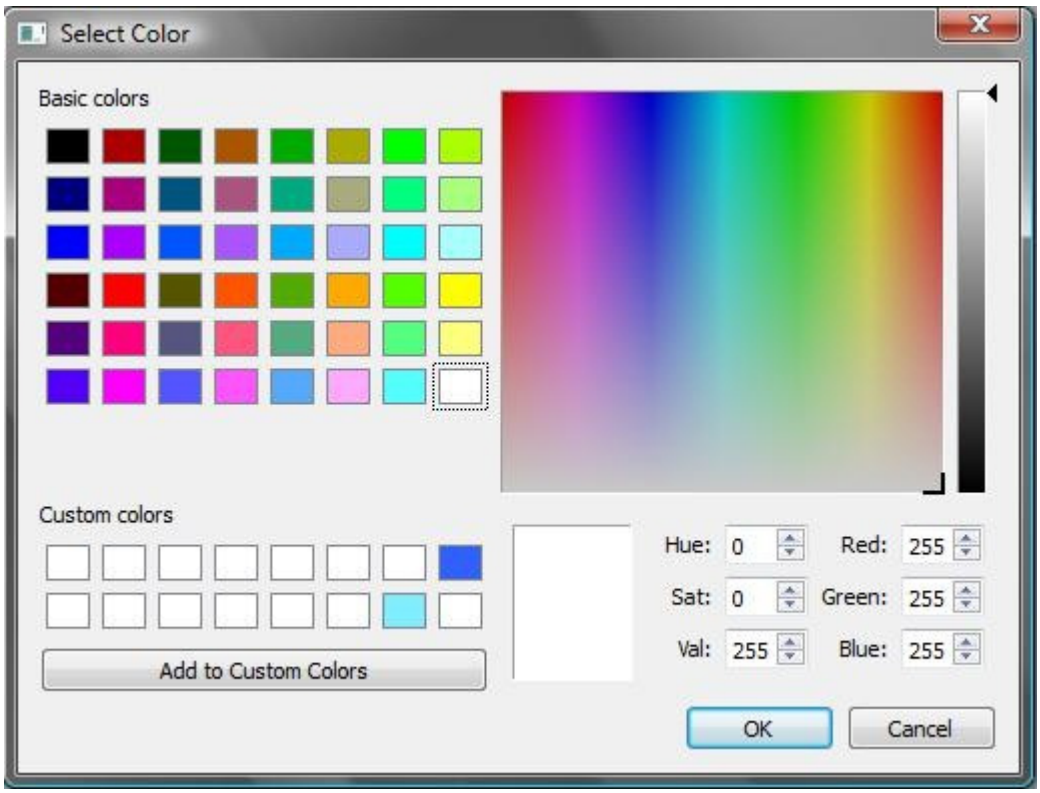


Figura 21: Cuadro de diálogo para la selección de color.

- Criterion: Aquí, se presentan al usuario las opciones sobre el cálculo de criterios de evaluación sobre la malla.

  Primero, se le da la opción de seleccionar que criterio desea aplicar sobre la malla (Figura 22). Esta selección se guarda y se notifica al Controller, quien indica a la malla que se debe iterar sobre sus elementos y aplicar el criterio de evaluación. Estos valores son almacenados en los elementos.

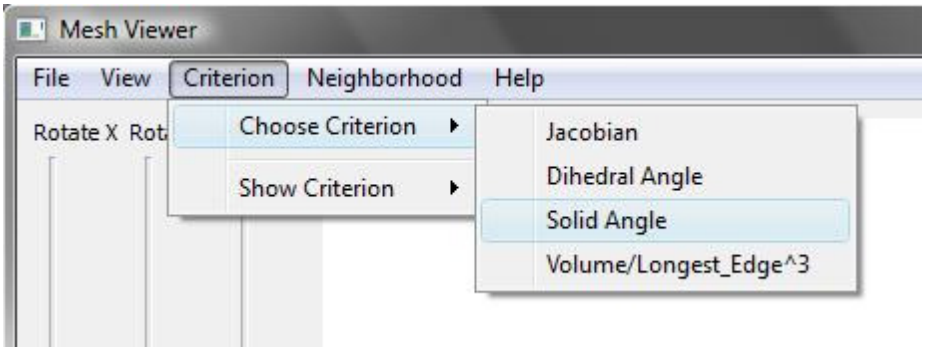


Figura 22: Choose Criteria

En el caso en que se ha seleccionado previamente solo elemento o una vecindad, el criterio sólo se evalúa sobre los elementos seleccionados y no sobre toda la malla.

Al seleccionar un criterio, se despliega un diálogo (Figura 23), en donde el usuario escoge el rango de valores dentro del cual se considera que los elementos son de buena calidad y así poder detectar fácilmente donde se encuentran los de mala calidad, para una posterior visualización de los datos obtenidos. En caso de escoger ángulo sólido o diedro, debe escoger también si será mínimo o máximo.

.

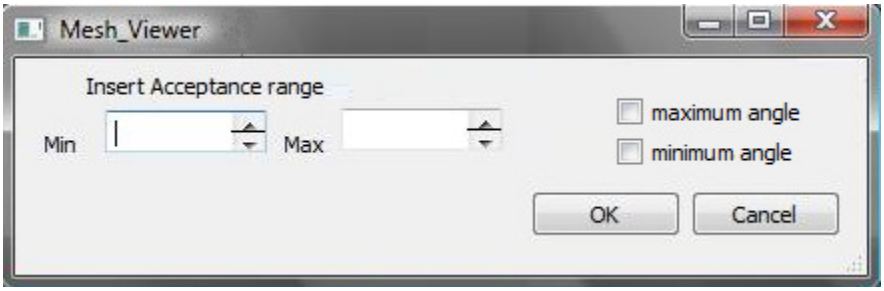


Figura 23: Choose Range.

Luego, se le da la opción al usuario de visualizar los resultados obtenidos al evaluar el criterio (Figura 24). Esta visualización se puede realizar de distintas formas.

Visualizar o no los elementos que cumplen con el rango de valores seleccionado previamente.

Desplegar un cuadro con los valores del criterio por elemento.

Visualizar que tan bueno o malo es un elemento por el color. Mientras más malos son los poliedros se ven más rojos, y mientas mejores son, se ven azules, pasando por la escala de valores intermedia entre estos colores.

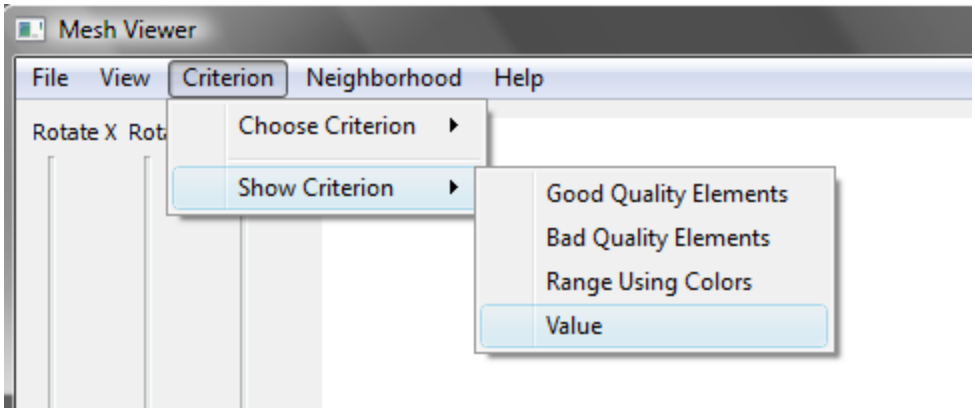


Figura 24: Show Criteria

- Neigborhood: Finalmente, se le da al usuario la opción de seleccionar una vecindad de elementos. Esta vecindad será visualizada y en caso de calcular un criterio, este se calculará solo en los elementos de la vecindad.

  Primero, se debe ir al menú Options y seleccionar el elemento que se quiere analizar. Gracias a que en la estructura de datos los elementos tienen referencias con sus vecinos, el cálculo de vecindad toma O(1) una vez que se ha encontrado el elemento. Encontrar el elemento toma tiempo O(1) o lineal dependiendo de cómo este se seleccione de a acuerdo a lo que se expuso anteriormente. En caso de hacer clic sobre un componente, se debe revisar componte a componente si se hizo clic en esa componente. Si la componente seleccionada es una cara o un elemento, es posible saber el punto donde se hizo clic está dentro fuera del elemento a través del método de orientación. Si las caras o elementos poseen sus vértices orientados positivamente, la orientación de cada arista con respecto al punto (en el caso de una cara) o de cada cara con respecto al punto (en el caso de un elemento), también debe ser positiva, de ser así el punto está dentro de la componente.

  El usuario debe escoger el nivel de vecindad deseado (Figura 25, 26). Para ello, se despliega un cuadro de texto en donde ingresa el número.

  El nivel de vecindad indica en qué nivel de profundidad se avanza por los elementos. Por ejemplo, si el nivel de vecindad es 1, sólo se ven los vecinos directos del elemento, si es 2, se ven los vecinos directos, y los vecinos directos de los vecinos, y así sucesivamente.

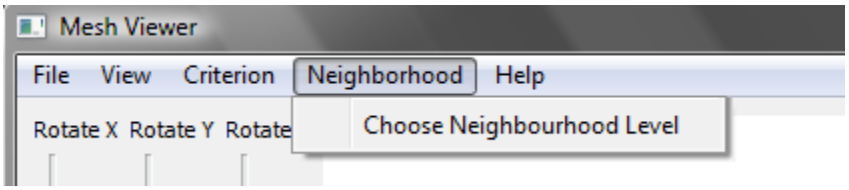


Figura 25:Menú Level of Neigborhood

Figura 26: Level of Neigborhood

Cabe destacar, que en caso de visualizar una malla de superficie, todas las opciones que involucran el manejo de poliedros se bloquean en el menú.

Finalmente, se provee al usuario la opción de modificar la malla moviendo un punto. El movimiento se realiza haciendo clic en el vértice con el mouse y desplazándolo por la pantalla. Este desplazamiento se realiza sólo en los ejes x e y, para desplazarse en z se debe usar el slider de traslación disponible.

Cada vez que el usuario desea mover un punto de posición, se chequea que no se rompa la integridad de los elementos que lo contienen, chequeando que el jacobiano de los nuevos elementos se mantenga positivo. De lo contrario el cambio no se realiza.

En los Anexos B, C, se pueden observar una malla de volumen y otra de superficie.

5. Conclusiones

5.1 Resultados Obtenidos

En este trabajo de memoria se diseñó y desarrolló una aplicación que permite leer una malla desde un archivo especificado en distintos formatos, visualizarla, evaluar su calidad, modificarla de forma restringida y almacenar el resultado en un archivo.

En el diseño de la aplicación se utilizó el paradigma de la programación orientada a objetos y patrones de diseño, lo que facilitará el desarrollo posterior de funcionalidades.

Se definió una estructura de datos, en donde cada componente de la malla posee referencias a componentes que contiene y a componentes que lo referencian, esto con el fin de permitir un acceso eficiente desde un elemento a otro. Así, el cálculo de relaciones de vecindad se efectúa sólo en la creación de la malla y no cuando se interactúa con ella y sus elementos.

Se obtuvo como resultado una aplicación en donde se pueden apreciar 4 módulos importantes:

- Gestor de archivos: Es el módulo que se encarga de abrir mallas almacenadas en archivos de texto con distintas extensiones y de almacenarlas con distintas extensiones.
- Creación de la malla: Es el módulo que, una vez que se obtienen los datos desde un archivo, crea los elementos a almacenar en la estructura de datos definida y genera de manera correcta todas las relaciones entre los elementos.
- Criterios de evaluación: Es en donde se encapsulan los distintos algoritmos involucrados en la evaluación de calidad de la malla.
- Interfaz y funcionalidad: Se genera la interfaz gráfica a través de la cual se recibe todo el input del usuario. En ella, se explicitan todas las funcionalidades disponibles en la aplicación.

Estos 4 módulos se comunican entre sí a través de una instancia única del visualizador y es posible modificar cada uno sin alterar el funcionamiento de los otros gracias al uso de patrones de diseño. Además, agregar nuevos componentes a la malla, agregar nuevos criterios, soportar

otros formatos de almacenamiento, es fácil y rápido gracias al uso del paradigma de programación orientado a objetos.

Todo esto produjo como resultado una herramienta para la visualización, evaluación y manipulación de mallas geométricas mixtas en 3D, extensible, flexible y fácil de usar y de entender.

5.2 Trabajo Futuro

Existen varias directrices en las que esta aplicación puede extenderse y mejorar, entre ellas se encuentran:

- Soportar más formatos de archivos de almacenamiento de mallas, para ello sólo es necesario agregar la opción en las clases que se comunican con el gestor de archivos. .
- Agregar más modos de visualización. En particular, manejar propiedades de los elementos de la malla, como pueden ser especificación de color o materiales. En algunos casos estos atributos vienen dados en los archivos de almacenamiento, por lo que se tendría que modificar el sistema de lectura y almacenamiento de archivos.
- Permitir la selección y manejo de regiones dentro de la malla. Dar la misma funcionalidad que se tiene sobre las vecindades a estas regiones.
- Generalizar el visualizador a todo tipo de malla 2D y 3D, y no sólo a mallas mixtas compuestas por tetraedros, hexágonos, prismas y pirámides. Para ello, se deben definir los métodos necesario a nivel de Element y no sólo al nivel de quienes heredan de él.
- Definir y generar estadísticas con los resultados obtenidos al aplicar los distintos criterios de evaluación sobre las mallas. Permitir visualizar y guardar éstas estadísticas en algún formato.
- Estudiar y agregar otros criterios de evaluación de mallas. En particular, agregar criterios de evaluación para mallas en 2D.
- Agregar algoritmos de refinamiento de mallas que permitan mejorar su calidad, permitiendo que el usuario seleccione cual desea usar.

- Integrar un generador de mallas que permita crear distintos tipos de mallas a través de un set de datos dado previamente.
- Mejorar la eficiencia de los algoritmos de creación de mallas. En partículas, podrían usarse otras estructuras de datos como apoyo a esto, como tablas de Hash.

6. Referencias

[1]. Watt. Advanced Animation and Rendering Techniques. Theory and Practice, Addison Wesley, Wokinghm, 1992.

[2]. Foley, Van Dam, Feiner, Hughes, Computer Graphics: Principles and Practice, Second Edition, Add. Wesley, 1990.

[3]. De Berg, M. Van Kreveld, M. Overmars and O. Schwarzkopf, Computational Geometry, Algorithms and Applications, Second, Revised Edition, Springer Berlin 2000.

[4]. Herbert Edelsbrunner. Geometry and topology for mesh generation, Cambridge University Press, 2001.

[5]. Pascal Jean Frey and Paul-Louis George, Mesh Generation: applications to finite elements, Hermes Science, 2000.

[6]. Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars . Computational geometry: algorithms and applications. Third edition, Hardcover, 2008.

[7]. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Element of Reusable Object Oriented Software Addison-Wesley, 1995.

[8]. Cay Horstmann. Object-Oriented Design & Patterns. John Wiley & Sons, Inc., 2004.

[9]. M. Woo, J. Neider, T. Davis, OpenGL Programming guide. The official guide to learning OpenGL. Versión 1.1, Add Wesley, 1996.

[10]. D. Hearn and M. P. Baker, Computer Graphics with OpenGL. Third edition, Pearson Prentice Hall, 2004.

[11]. OpenGL: http://www.opengl.org, última visita 14 de diciembre 2010.

[12]. TetView: http://tetgen.berlios.de/tetview.html, última visita 14 de diciembre 2010.

[13]. Geomview: http://www.geomview.org, última visita 14 de diciembre 2010.

[14]. QT: http://qt.nokia.com/products/, última visita 14 de diciembre 2010.

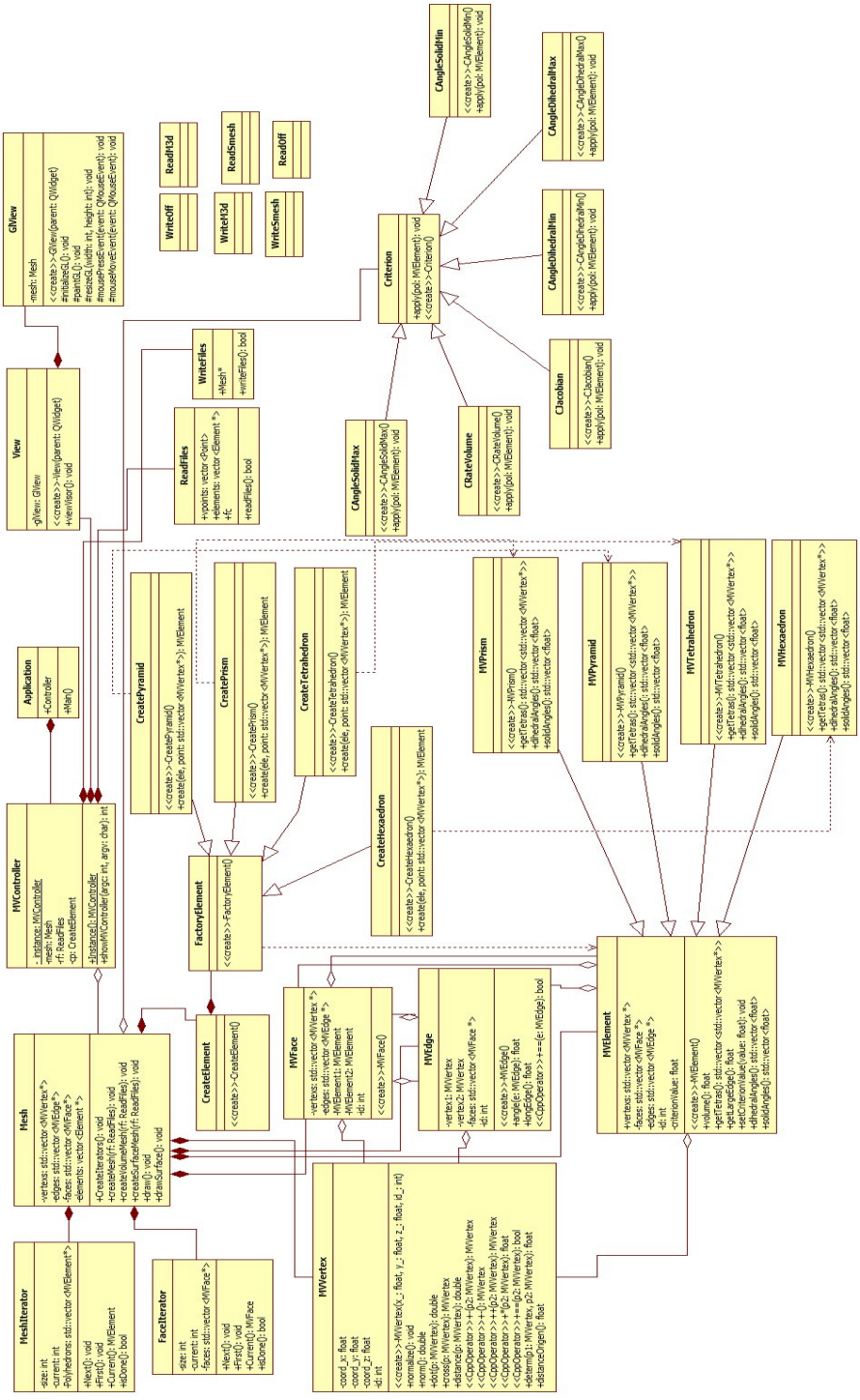
[15]. Bjarne Stroustrup. The C++ Programming Language. Addison-Wesley, 2002. Edicion especial.

[16]. Van Oosterom, A. Strackee, J. (1983). The Solid Angle of a Plane Triangle. Laboratory of Medical Physics and Biophysics, University of Nijmegen. En IEEE Transactions on Biomedical Engineering.

[17]. http://es.wikipedia.org/wiki/Proceso_de_ortogonalizaci%C3%B3n_de_Gram-Schmidt

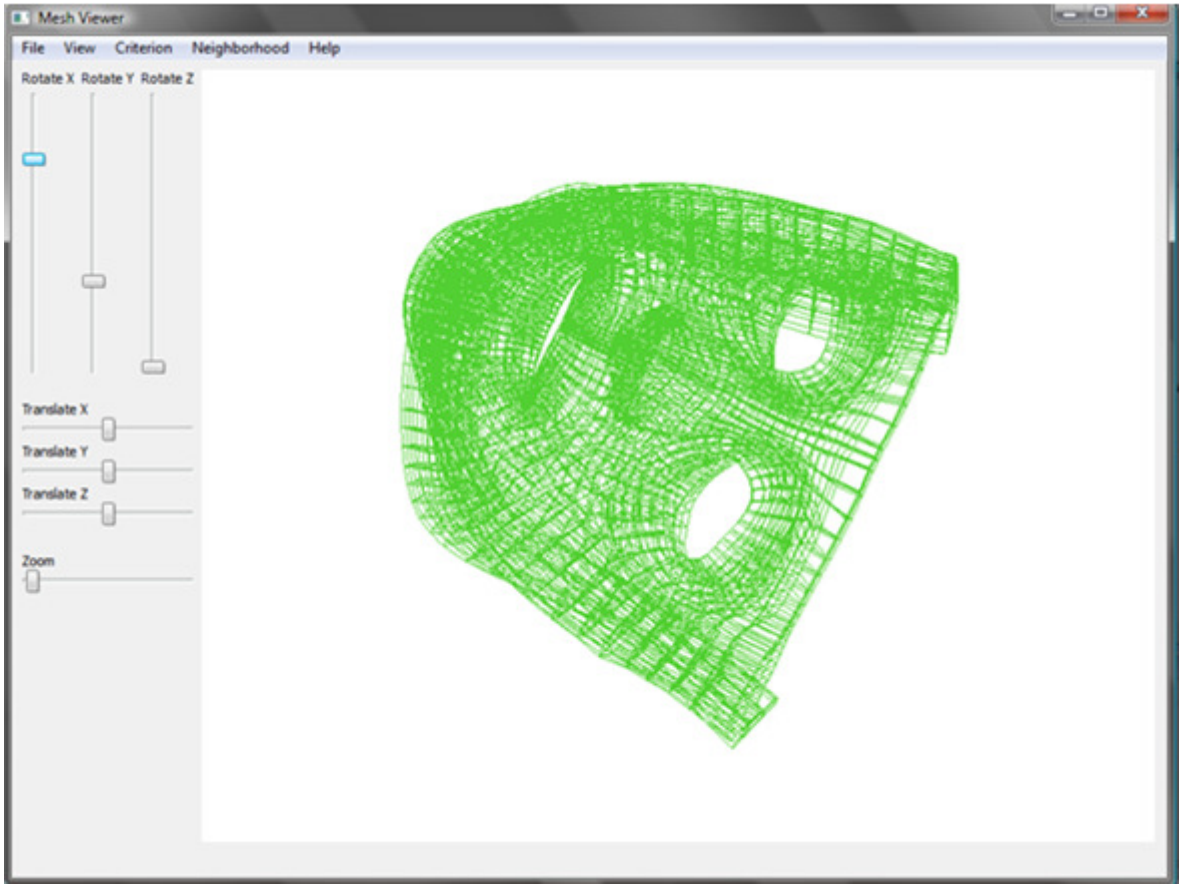
7. Anexos

Anexo A

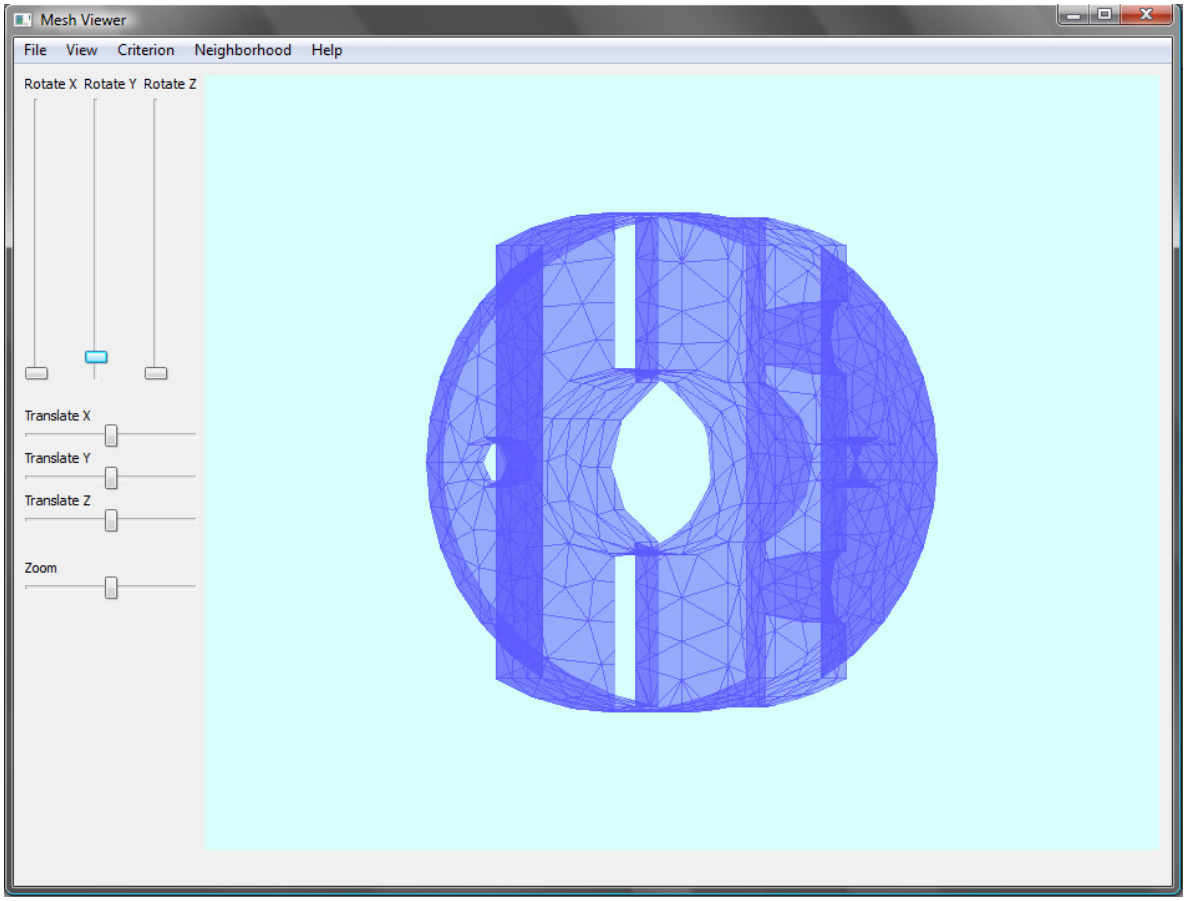


Anexo B

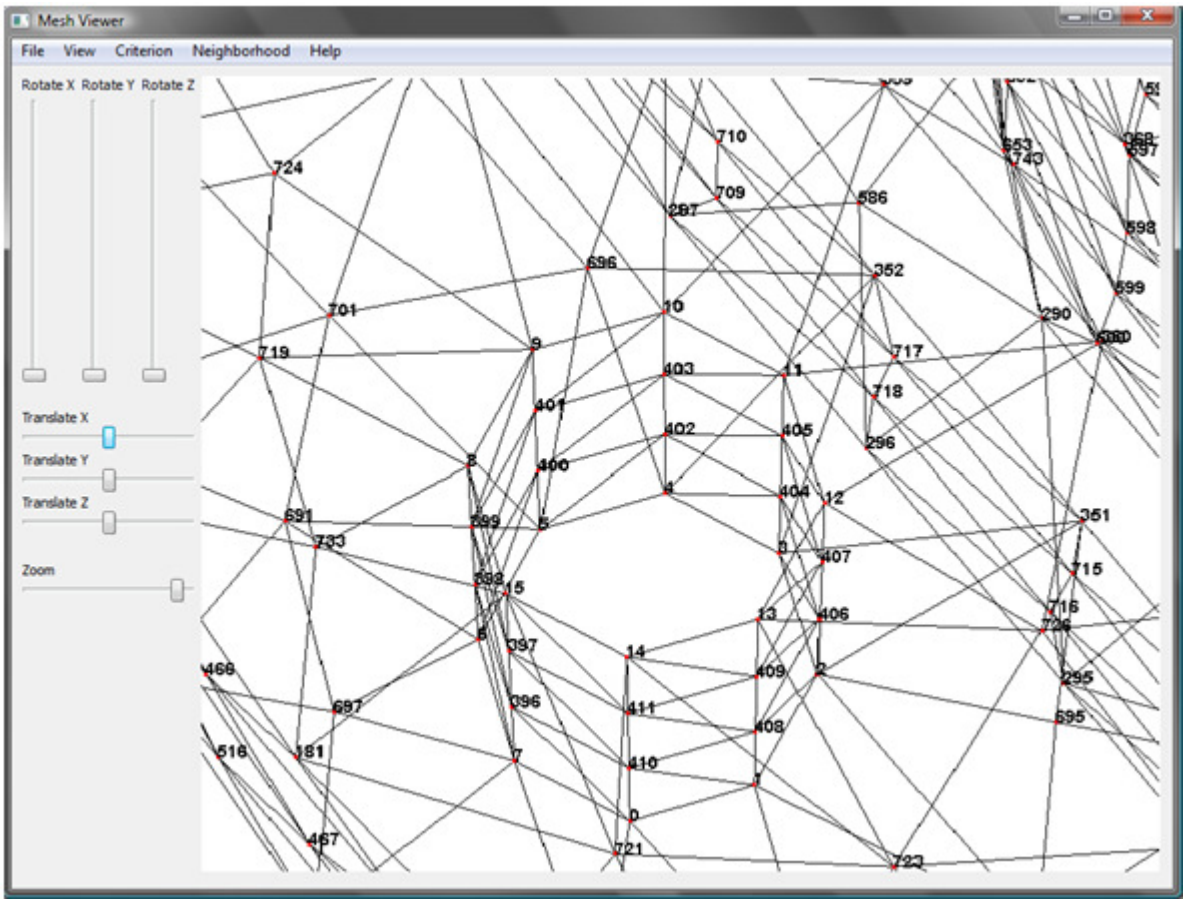
Malla de Volumen



Malla de superficie con vista de elementos transparentes.



Visualización del índice del vértice en una sección de una malla.



Visualización valor de un criterio