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

EXTENSIÓN Y MEJORAMIENTO DE HERRAMIENTA DE GENERACIÓN DE MALLAS GEOMÉTRICAS EN DOS DIMENSIONES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

JORGE VALENZUELA MARTÍNEZ

PROFESOR GUÍA
NANCY HISTCHFELD KAHLER

MIEMBROS DE LA COMISIÓN:
MAURICIO PALMA LIZANA
MARIA CECILIA RIVARA ZUÑIGA

SANTIAGO DE CHILE
OCTUBRE 2007

Índice

Resumen

“Extensión y Mejoramiento de Herramienta de Generación de Mallas Geométricas en Dos Dimensiones”

El objetivo general de esta memoria es extender y mejorar una herramienta interactiva gráfica que permite al usuario definir y visualizar el proceso de generación de mallas geométricas en dos dimensiones, agregándole nuevos algoritmos de refinamiento de mallas, nuevas funcionalidades y mejorando las existentes.

Inicialmente se tiene una aplicación que tiene la funcionalidad de mostrar y procesar una malla en dos dimensiones con un algoritmo de refinamiento/mejoramiento dado con ciertos criterios definidos por el usuario.

Luego de analizar la aplicación existente, se decide rediseñar una parte de sus procesos, considerando características en su diseño como extensibilidad, y modularidad. Se decide hacer una interfaz para el usuario, desde cero, la que comunica con la nueva aplicación. Se agregan nuevas funcionalidades a la aplicación, tal como la selección de regiones para su posterior refinamiento. Un nuevo algoritmo de refinamiento es agregado a la aplicación.

Como resultado se tiene una herramienta para visualizar y manejar mallas geométricas en dos dimensiones, que es poderosa, extensible, flexible y simple de usar, junto con nuevas funcionalidades y algoritmos, con respecto de la aplicación original.

Agradecimientos

A mis padres, por su continuo apoyo durante todo mi período estudiantil.

A mi profesora guía, Nancy Hitschfeld, por su paciencia, y su excelente disposición para atender mis consultas.

Me gustaría también agradecer la ayuda recibida en el contexto del proyecto Fondecyt No. 1061227

1 Introducción

1.1 Antecedentes Generales

La generación de una malla es un paso fundamental para la resolución de problemas reales de ingeniería mediante métodos numéricos tales como elementos finitos y volúmenes finitos. Actualmente, existen diversas aplicaciones en el ámbito profesional y de entretenimiento que utilizan esta estrategia para desarrollar modelos cada vez más realistas.

En este momento, existen tres tipos de mallas geométricas que son utilizadas en ambientes 2D, las triangulares, cuadriculadas y mixtas. La estrategia es representar el objeto que se va a modelar en base a alguna de estas mallas, siendo la calidad de cada elemento, una variable que influye directamente en la precisión y el costo computacional del modelo que se va a implementar.

Para generar una malla automáticamente se realizan en general, los siguientes procesos:

- Modelar (representar) la geometría del objeto (Generar malla inicial).

- Mejorar la malla, con el objetivo de mejorar la calidad de los elementos, sin insertar nuevos puntos, si es posible.

- Refinar la malla inicial de acuerdo con ciertos criterios que afectan la densidad de puntos requerida por la aplicación.

- Visualización de la malla inicial refinada/mejorada.

- Operaciones de de-refinamiento de la malla (implica eliminar algunos puntos) o suavizado (implica mover algunos puntos en la malla).

En cada operación de refinamiento y/o mejoramiento, los algoritmos usados debieran estar optimizados para insertar la menor cantidad de puntos posibles.

1.2 Motivación

Este trabajo se basa en la implementación de un visualizador interactivo de mallas en dos dimensiones realizado por Javier Irarrázabal[1] para su memoria de título.

Este visualizador posee actualmente lo siguiente:

- Interfaz gráfica para visualizar mallas 2D.
- Mejoramiento de mallas mediante algoritmos LEPP-Delaunay
- Generación de malla inicial.

Dentro de los aspectos que no funcionan bien dentro del programa actual se encuentran:

- Problemas de visualización, por ejemplo, no limpiar la pantalla antes de visualizar la siguiente malla.
- Problemas de usabilidad, interfaz poco intuitiva.
- Manejo inapropiado de entrada de usuario, entradas no válidas hacen que el programa termine su ejecución antes de lo esperado.
- No es capaz de procesar como entrada una malla que posea arcos y/o puntos flotantes.
- Faltan algunas funcionalidades de la interfaz, como por ejemplo, especificar zonas limitadas de refinamiento.

1.3 Objetivos

1.3.1. Objetivo General

La presente memoria tiene como objetivo extender y mejorar esta herramienta interactiva gráfica que permite al usuario definir y visualizar el proceso de generación de mallas geométricas en dos dimensiones, agregándole nuevos algoritmos de refinamiento de mallas, nuevas funcionalidades y mejorando las existentes.

1.3.2. Objetivos Específicos

1. Diseñar e implementar una nueva interfaz, considerando las necesidades y perfil del usuario objetivo.

2. Agregar nuevos algoritmos de refinamiento, en este momento se encuentra implementado principalmente LEPP-Delaunay. Se desea agregar otros algoritmos, como *Voronoi Point Insertion*.

3. Agregar refinamiento por regiones, ofreciendo la posibilidad de que se pueda refinar sólo una parte de la malla. Las regiones pueden definirse por una figura geométrica simple, o un polígono. De esta forma, se restringe el conjunto de puntos que se van a insertar a sólo los que se encuentran dentro de los elementos que intersectan la región especificada.

4. Organizar el código de la aplicación tal que como resultado se tenga un programa estructurado, que permita una gran extensibilidad, y sencilla comprensión por parte de terceros, incorporando el uso de patrones de

diseño, que sean adecuados para el desarrollo a realizar, usando las buenas prácticas de ingeniería de software.

1.4 Metodología

A continuación se muestra los pasos que se seguirán en el trabajo para el cumplimiento de los objetivos. Estos pasos están orientados a tener un mejor entendimiento acerca de los temas involucrados en el trabajo y un acercamiento hacia las herramientas que serán utilizadas para el desarrollo de la aplicación.

Los pasos a seguir son los siguientes:

- Determinar ambiente de desarrollo.
- Familiarización con las herramientas que se utilizarán (C++, GTK, LibMesh).
- Familiarización con la aplicación actual, las operaciones que es capaz de realizar, y de que forma las realiza (código).
- Rediseño de la interfaz gráfica
- Estudio algoritmos y estrategias de refinamiento, LEPP, *Voronoi Point Insertion*, criterios, regiones.
- Implementación de interfaz gráfica nueva.
- Implementación algoritmo de Voronoi para refinamiento de mallas 2D
- Implementación de uso de regiones (poligonales, circulares) para refinamiento / mejoramiento.

1.5 Contenido de la Memoria

Esta memoria está organizada en seis grandes capítulos:

Introducción

Este capítulo, describe la motivación, presenta los objetivos y desarrolla la metodología a usar.

Aplicación Original

En este capítulo se revisan los detalles del diseño e implementación de la aplicación original, y sus problemas. En particular, se detalla un diagrama de la aplicación original, el estado actual de la interfaz gráfica, librerías externas que se ocupan, diagrama de clases general, y detalle de implementación de algunos algoritmos.

Diseño

En este capítulo se revisa el diseño de la nueva aplicación, sus características, y la estructura interna del programa. En particular, se revisa los patrones de diseño que fueron usados al momento de diseñar y estructurar la aplicación, el diseño de la interfaz de usuario, y un diagrama general de clases del sistema.

Implementación

En este capítulo se discutirán los detalles de implementación de la aplicación. En particular, se discuten acerca del ambiente de desarrollo y bibliotecas utilizadas. Se especifican detalles de la implementación de la interfaz, los algoritmos de refinamiento y/o mejoramiento, las clases que representan criterio y región.

Conclusiones

En este capítulo se presentan las conclusiones finales de este trabajo. Se comenta adicionalmente el posible trabajo futuro sobre la aplicación.

Referencias

Referencias usadas dentro de esta memoria.

2 Aplicación Original

En este capítulo se revisan los detalles del diseño e implementación de la aplicación original, y sus problemas. En particular, se detalla un diagrama de la aplicación original, el estado actual de la interfaz gráfica, librerías externas que se ocupan, diagrama de clases general, y detalle de implementación de algunos algoritmos.

2.1 Funcionalidades de Aplicación Original

La aplicación actualmente permite:

- Cargar una geometría en memoria desde un archivo
- Generar una malla Delaunay inicial a partir de la geometría
- Refinar la malla completa con el algoritmo LEPP-Delaunay, pudiendo elegir tres criterios diferentes.

2.2 Interfaz

A continuación se muestra la interfaz gráfica y la forma en que se visualiza el proceso de generación de una malla a través de ésta. Los pasos implementados son: (a) carga de geometría desde un archivo (b) generación de la malla inicial (c) mejoramiento de la malla a través del algoritmo LEPP-Delaunay, aproximado o completo.

La interfaz actual consiste en una zona con las opciones en la parte superior, y el espacio para la visualización de la malla 2D, en la parte inferior. La figura 2.1 muestra una geometría cargada, a la que se le ha generado una triangulación de Delaunay inicial.

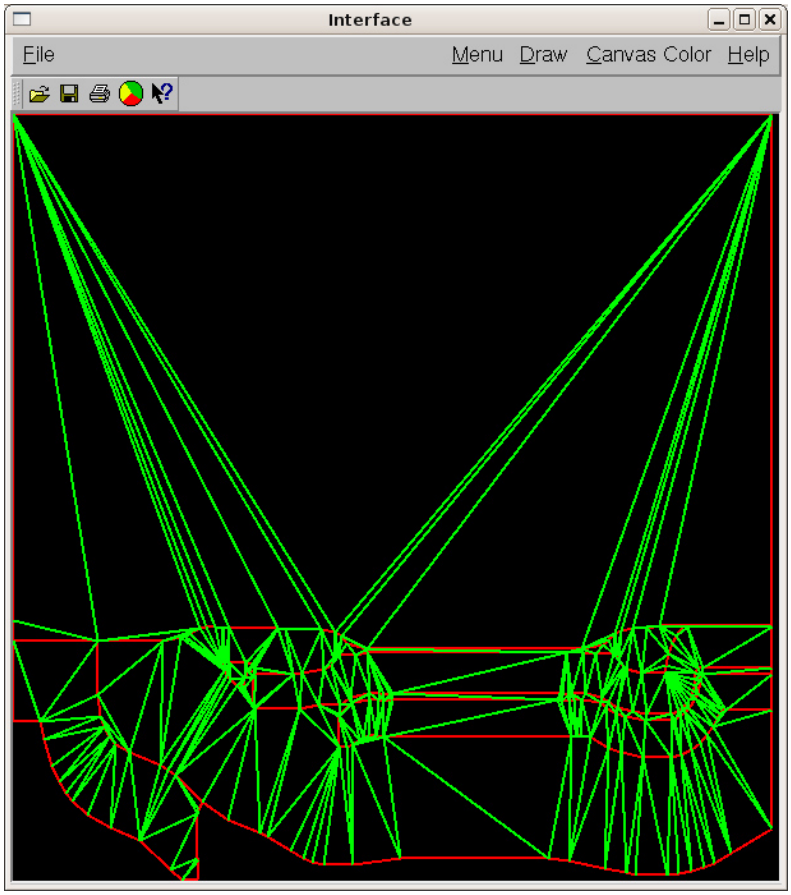

Figura 2.1: Geometría cargada, con una triangulación inicial

En la figura 2.2 se puede observar un cuadro de diálogo que se usa actualmente para elegir los diferentes criterios que se usan junto con el algoritmo LEPP-Delaunay.

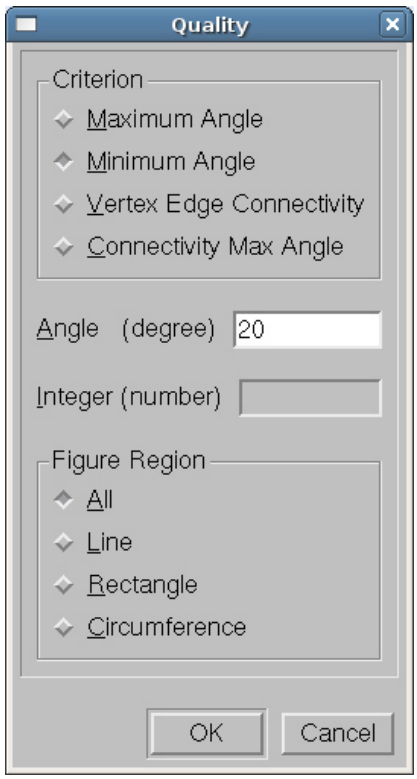

Figura 2.2: Cuadro de diálogo con los criterios del algoritmo

En la figura 2.3 se puede observar la misma malla mejorada con el algoritmo LEPP-Delaunay, con un criterio de ángulo mínimo de 20º.

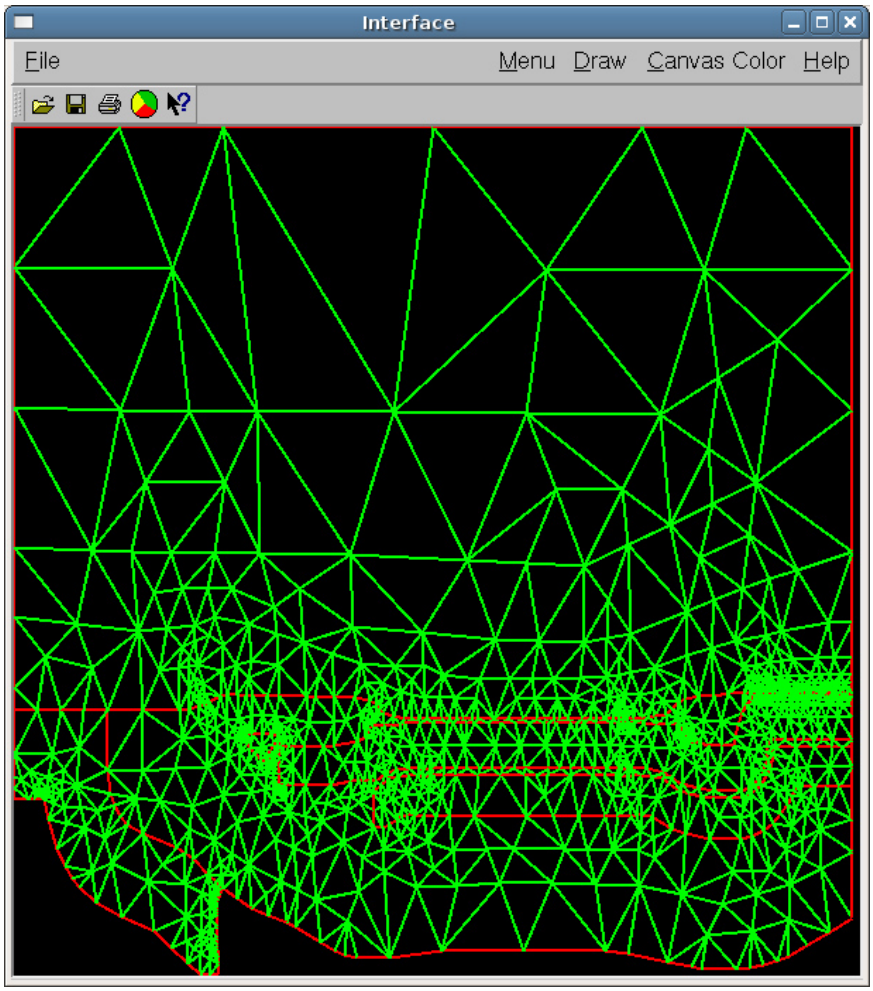

Figura 2.3: Malla mejorada con LEPP-Delaunay

2.2.1 Visualizador

Para la implementación del visualizador de la malla se usó la librería Qt, la cual es una biblioteca multiplataforma para desarrollar interfaces de usuario. Es principalmente usada en KDE, el manejador de ventanas para linux. Qt no es una librería completamente gratuita, y debido a esto la licencia puede ser restrictiva en ciertos casos.

Ya que la aplicación trabaja en mallas de dos dimensiones, se usa las funciones de dibujo estándar que provee Qt en dos dimensiones, para representar la malla en la pantalla.

2.2.2 Diagrama de la Interfaz

A continuación se presenta un diagrama que muestra la interacción entre el usuario y la interfaz original. La aplicación interactúa sólo con un usuario.

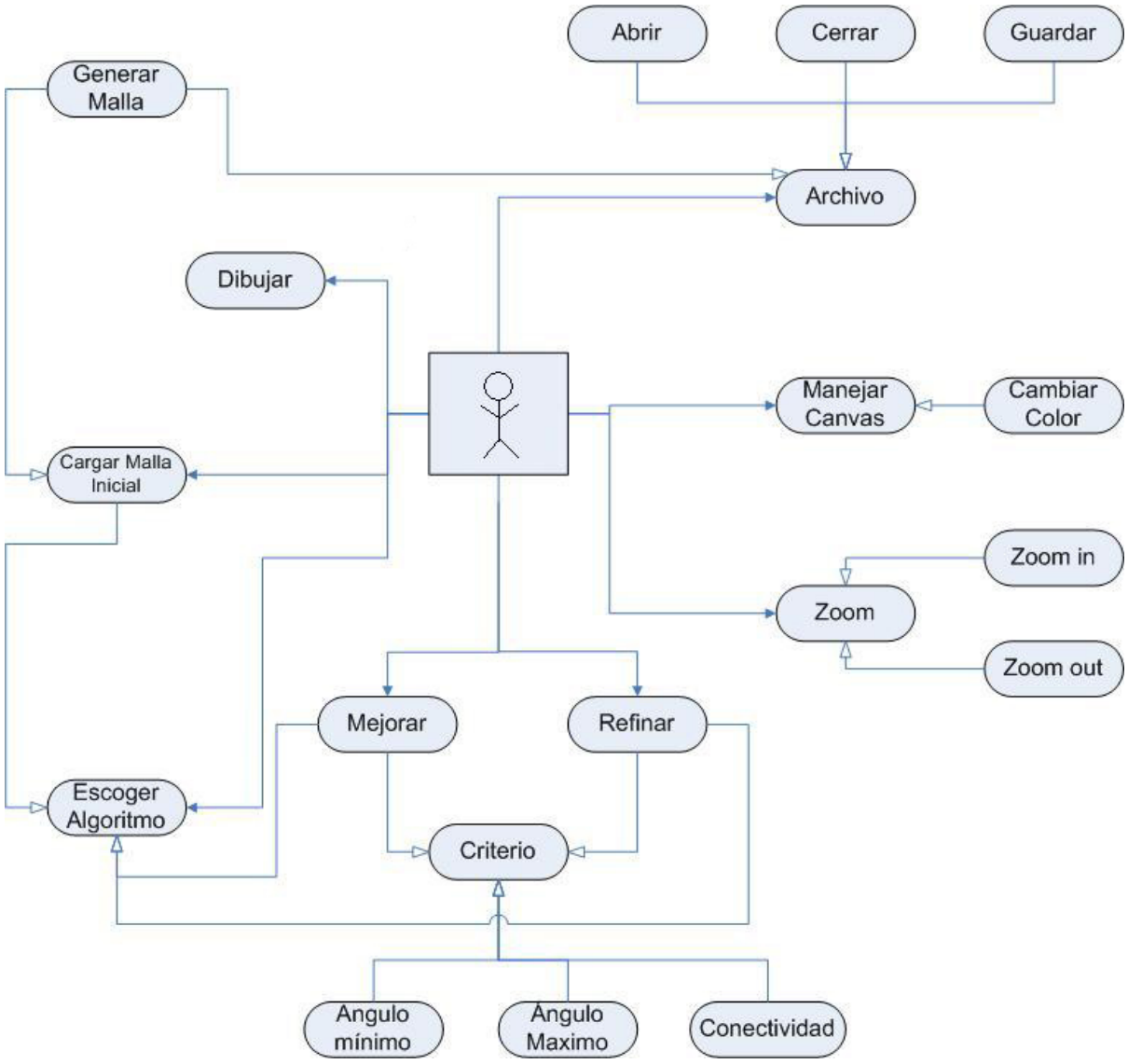

Figura 2.4: Diagrama de interacción con el usuario

2.3 LibMesh

Esta librería se encarga de mantener la representación de la malla en la memoria, y de manejar todas las operaciones básicas de cambios o modificaciones sobre ella, tales como agregar nuevos elementos/puntos, borrarlos o de relacionar dos elementos.

La aplicación original se basa fuertemente en esta librería, y dada la funcionalidad que provee también fue usada para la aplicación final.

En el contexto de dos dimensiones la librería provee objetos para representar puntos, segmentos, y elementos como triángulos y/o polígonos. La malla se representa como el conjunto de todos estos objetos.

LibMeshPoint: Clase que se usa para representar un punto P(x,y) en dos dimensiones dentro de la malla. Un objeto de esta clase contiene punteros a los elementos en los cuales se encuentra este punto.

LibMeshPoints contiene una lista de todos los LibMesh2DPoint que existen en la malla.

LibMeshEdge: Clase que se usa para representar un segmento dentro de la malla. Esta formada por dos puntos, y este objeto sirve como base para elementos más complicados como triángulos o polígonos. Cada objeto de esta clase contiene punteros a los puntos que la conforman, y también a los elementos de los cuales son parte (para una malla 2D sólo tiene sentido que un arco pertenezca a dos elementos).

LibMeshEdges contiene una lista de todos los LibMesh2DEdge que existen en la malla.

LibMeshElement: Clase que representa un elemento en dos dimensiones dentro de la malla, un elemento puede variar entre un vértice simple hasta polígonos simples con multiples vértices y arcos. En general se usa para representar triángulos. Cada objeto de esta clase contiene punteros a todos objetos que la conforman, ya sea puntos, o segmentos. En el caso del triángulo son tres puntos y tres segmentos a los que se referencia.

LibMeshElements contiene una lista de todos los LibMeshElement que existen en la malla.

LibMeshGrid: Clase que se usa para definir una malla en dos dimensiones, contiene punteros a las listas de puntos, arcos y elementos (LibMeshPoint, LibMeshEdge, LibMeshElement) que representan en conjunto la malla.

El mayor problema al usar esta biblioteca fue la falta de documentación apropiada. En ciertos casos hubo que

2.4 Diagramas de Clases

En esta sección, se describen las clases más importantes que originalmente estaban implementadas en el sistema, junto con algunas relaciones básicas generadas entre ellas.

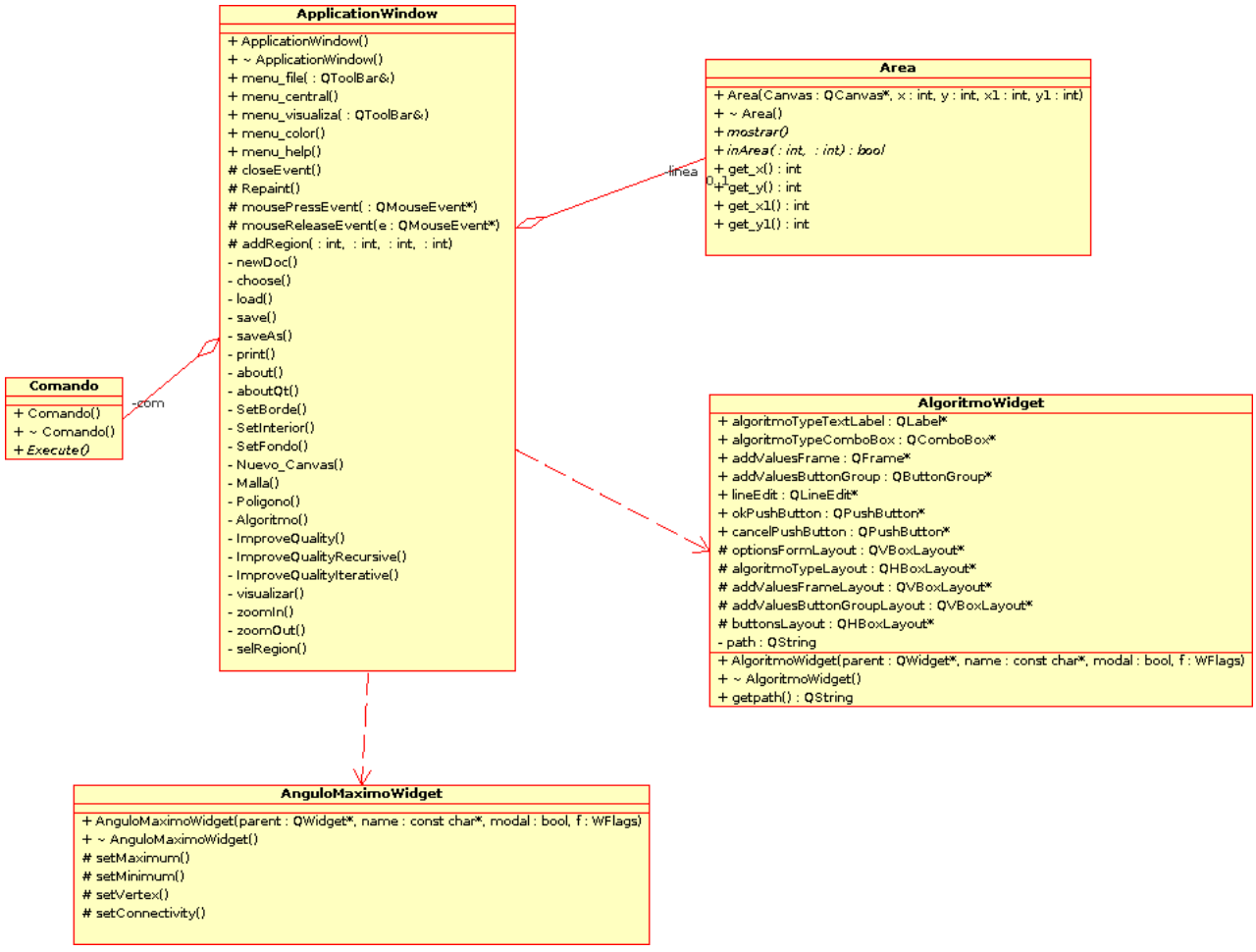

Figura 2.5: Diagrama de Clases del Sistema - Aplication Window

En la figura 2.5 se observa el diagrama de clases que modela la interfaz de la aplicación. La clase ApplicationWindow es la encargada de construir el visualizador, manejar carga de archivos de geometría en memoria, manejar el area

de dibujo y cerrar la aplicación. Estas acciones se implementan en la clase principal, y son activadas mediante señales desde la GUI implementada en base a la librería Qt.

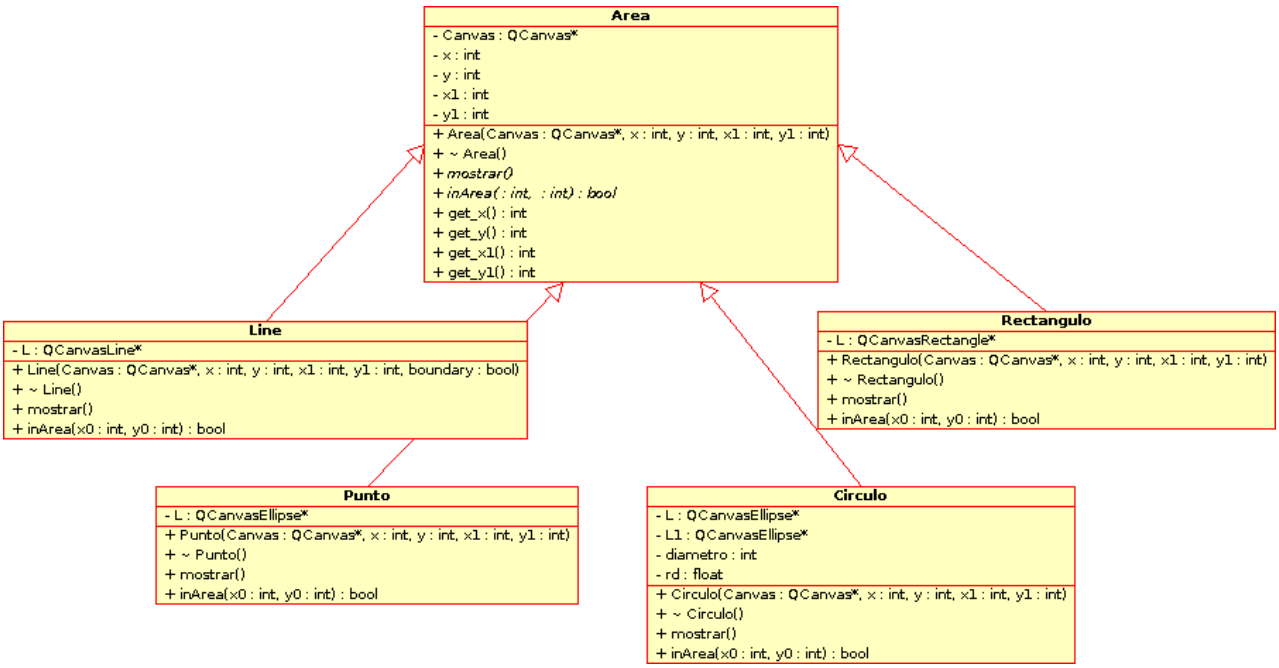

Figura 2.6: Diagrama de Clases del Sistema - Area

En la figura 2.6 se observa el diagrama de clases en torno a la clase Area, la cual fue diseñada para especificar regiones que limiten la zona donde se mejorará y/o refinará una malla. En este momento, estos algoritmos se aplican siempre dentro de toda la malla.

Ninguna de estas clases está implementada aún, sólo está la estructura generada que se pensó en ese momento sería la mejor para almacenar y especificar áreas, y agregar nuevos tipos si es que fuese necesario.

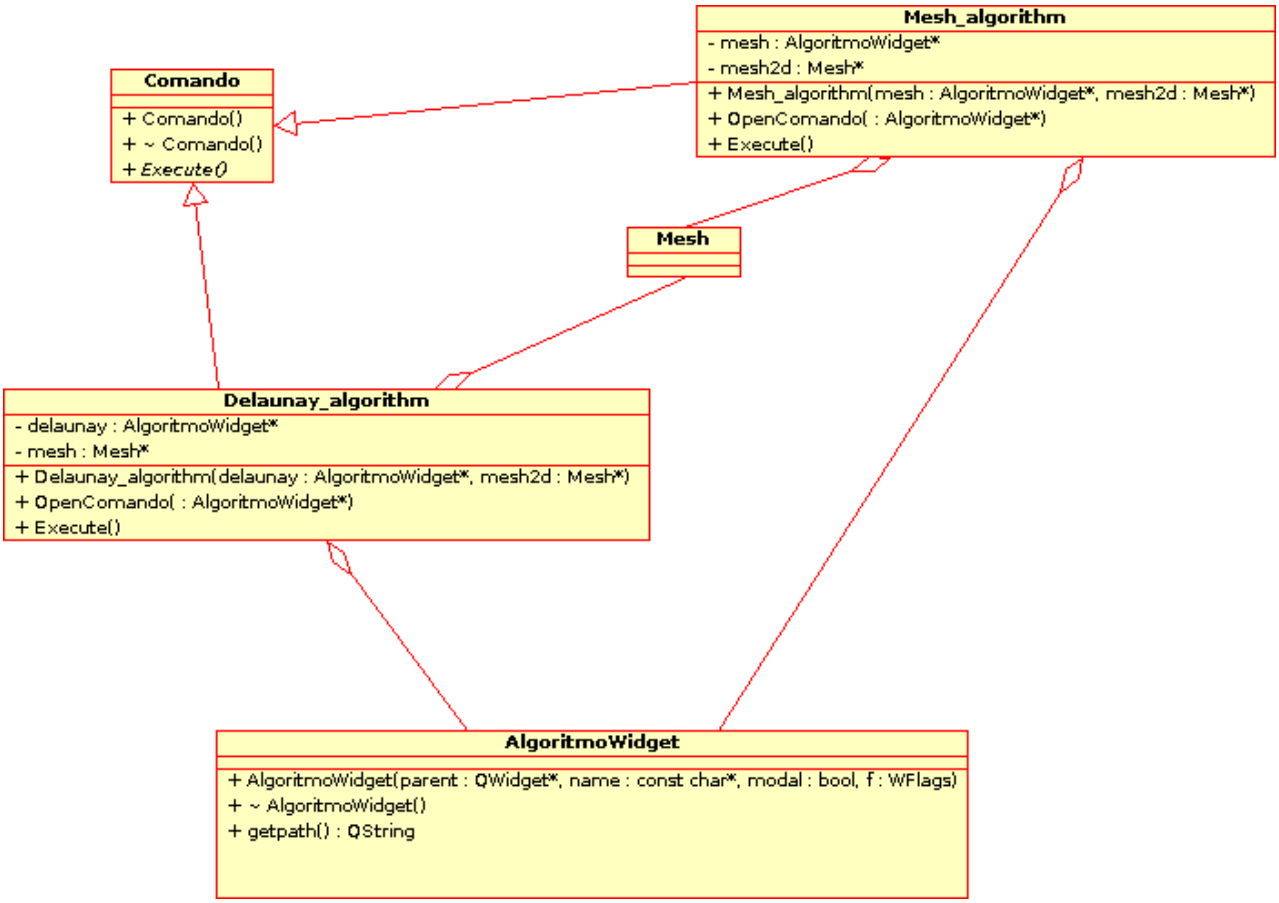

Figura 2.7: Diagrama de Clases del Sistema – Algoritmos de malla

En la figura 2.7 se observa cómo están definidas las clases para los algoritmos de generación de mallas. Ambas clases Delaunay_algorithm y Mesh_algorithm implementan el patrón de diseño Command para su operación. Además usan LibMesh para su operación interna. Estos algoritmos instancian AlgoritmoWidget

el que se encarga de preguntar al usuario el algoritmo apropiado para sus necesidades.

Hay muchas operaciones que en vez de estar asociadas una clase acorde con su funcionalidad, estas asociadas a la clase principal, AplicationWindow, y están implementadas en esa clase, por ejemplo, el cargar una malla en memoria. No es posible con este esquema, por ejemplo, cambiar la interfaz del programa para usar una interfaz de línea de comando, pues habría funcionalidades que no estarían implementadas (las tiene implementadas la misma interfaz).

2.5 Algoritmo Lepp Delaunay

Este algoritmo de mejoramiento se basa en la inserción de nuevos puntos dentro de la malla para mejorar los triángulos que no satisfacen los criterios definidos para un buen triángulo.

Definición: Para cualquier triángulo t_0 de una triangulación x, el Longest-Edge Propagation Path (Camino de propagación por el arco más largo) de t_0 - o LEPP(t_0) - será la lista ordenada de todos los triángulos $t_0,t_1,t_2,\ldots,t_i,\ldots,t_{n-1}$, t_n, tal que t_i, es el triángulo vecino de t_{i-1} por su arco más largo, para i=1,2,…,n.

Definición: Dos triángulos adyacentes serán llamados triángulos terminales si comparten el arco más largo.

Definición: Para cualquier triangulación t, cualquier arco interior l será llamado un arco terminal en t si este arco es un arco más largo común entre los dos triángulos que comparten l.

El punto que se va a elegir para la inserción será el punto medio del arco terminal del LEPP del triángulo a mejorar, o en su defecto, si el triángulo más largo no es borde y el triangulo tiene el segundo arco más largo en el borde, se elige el punto medio de este segundo arco. [5]

Algoritmo

- Se calcula el conjunto de triángulos que no satisfacen el criterio establecido.
- Para cada triangulo t que no satisfaga el criterio establecido:
    - Si es un triángulo borde con un arco borde l_b, y éste no es el arco más pequeño, entonces inserta un punto en la mitad de l_b. Se asegura de que los nuevos triángulos mantengan la propiedad de Delaunay, invirtiendo diagonales de ser necesario.
    - Si no es un triángulo borde, entonces se calcula el LEPP del triángulo. Se busca el primer triángulo borde t^* distinto de t. y su borde l_b, tal que no sea el arco más pequeño de t^*.
        - Si l_b existe, entonces se inserta en el punto medio de l_b. Se asegura la propiedad Delaunay, invirtiendo diagonales de ser necesario.
        - Si l_b no existe, entonces se usa el arco terminal del LEPP de t. Se inserta en el punto medio del arco terminal, luego se verifica la propiedad Delaunay, invirtiendo diagonales de ser necesario.
- Se calcula nuevamente el conjunto de triángulos que no satisfacen el criterio establecido, se itera hasta que el conjunto este vacío.

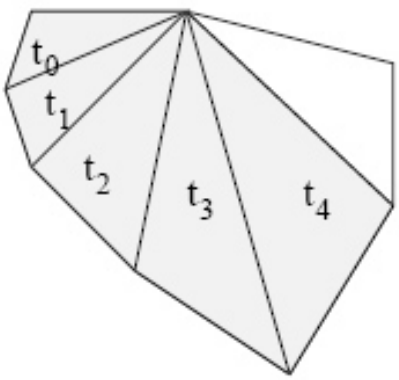

Figura 2.8: LEPP de t_0.

3. Diseño

En este capítulo se revisa el diseño de la nueva aplicación, sus características, y la estructura interna del programa. En particular, se revisa los patrones de diseño que fueron usados al momento de diseñar y estructurar la aplicación, el diseño de la interfaz de usuario, y un diagrama general de clases del sistema.

3.1 Patrones de Diseño

Command: Este patrón utiliza una interfaz que agrupa mediante herencia cada uno de los comandos concretos. Cada comando provee un método llamado execute(), el cual es el responsable de ejecutar las acciones asociadas al comando. También participa en este patrón una o más clases que implementan las acciones asociadas al comando. Además, se usa una clase cliente que crea el comando específico y una clase invocadora que llama al comando.

Se usa para:

- Parametrizar los objetos según acción a ejecutar
- Encolar y ejecutar solicitudes a distintos tiempos
- Permitir opción de deshacer.

Este patrón se utiliza para encapsular requerimientos basados en diferentes acciones por realizar. En la aplicación se utilizará este patrón en el modelamiento de opciones del menú. Así, cada operación de la que es capaz la aplicación corresponde a un comando. Cuando el usuario ejecuta una acción a través del menú de la interfaz gráfica, se genera una instancia de un comando específico encapsulando todos sus parámetros.

Template: Este patrón se utiliza para definir el esqueleto de un algoritmo en alguna operación, dejando algunos métodos para ser redefinidos por las clases que heredan. Así las subclases pueden implementar de acuerdo a su necesidad sin necesidad de salirse del esquema de la clase padre.

Se usa cuando:

- Se desea implementar las partes invariantes de un algoritmo y permitir que las subclases implementen los comportamientos variables.

- Existe un comportamiento común entre varias clases que debiera ser ordenado y factorizado por una clase padre.

Se usa este patrón para implementar las funcionalidades de refinamiento por regiones, y/o criterios para definir si un triangulo debe refinarse. La funcionalidad de estos elementos es invariante en su forma general, para un criterio, por ejemplo, se consulta si un elemento lo cumple o no, independiente del criterio a utilizar; para una región, se consulta si un elemento está dentro de ella o no, independiente de la región a usar. Los detalles específicos del algoritmo de cada criterio o región son redefinidos por sus correspondientes subclases, manteniendo la clase padre intacta.

Strategy: Este patrón define y encapsula una familia de algoritmos que resuelven un mismo problema y permite intercambiar fácilmente el algoritmo que un cliente usa. Además del intercambio y la manera limpia de resolver el problema, este patrón permite agregar otros algoritmos que se quiera utilizar más adelante en la aplicación, permitiéndole ser altamente extensible. Este patrón define una interfaz común para cada clase que implementa un algoritmo, la cual consiste en un método que debe ser implementado en las diferentes clases. Además define una clase contexto, la cual mantiene una referencia al algoritmo o estrategia específica y la aplica.

Con la integración de nuevos algoritmos para la generación de mallas y mejoramiento/refinamiento se puede usar un patrón de diseño Strategy para agrupar los diferentes algoritmos que trabajan sobre el mismo dominio, como por ejemplo, LEPP-Delaunay y *Voronoi Point Insertion*.

3.2 Interfaz de Usuario

Al diseñar una nueva interfaz de usuario siempre es importante tomar en cuenta la usabilidad: se debe diseñar una interfaz que sea fácil de usar y de entender para el usuario final, y es aquí justamente donde fallaba la interfaz original. La interfaz antigua no era en absoluto intuitiva, era necesario buscar dentro de todos los menús para encontrar la funcionalidad que se buscaba. Los cuadros de diálogo abarcaban más información de la que realmente se requería consultar, efectivamente confundiendo al usuario. Esta nueva interfaz debe incluir menús, iconos, botones, etc. en forma organizada y limpia, representando cada funcionalidad de manera separada y clara.

La interfaz contiene, dentro de la ventana principal, una ventana visualizador OpenGL, la que es interactiva con el usuario. Esta característica le da una usabilidad única al programa. El usuario puede ver de manera inmediata los cambios que planea realizar, los puntos y las zonas que está seleccionando, antes de ejecutar un algoritmo.

Desde la interfaz se pueden acceder las siguientes funcionalidades:

- Cargar en memoria una geometría desde un archivo.
- Guardar una malla o geometría a un archivo.
- Generar una malla Delaunay inicial en base a una geometría en memoria.
- Refinar una malla con el algoritmo seleccionado en la interfaz, usando el criterio seleccionado, y limitado por una región seleccionada
- Seleccionar el criterio a utilizar
- Seleccionar los algoritmos a utilizar

- Seleccionar una región para refinar
- Seleccionar lenguaje de la interfaz
- Capacidades de zoom

La nueva interfaz usa elementos visuales estándares de la biblioteca de GTK, lo que lo hace visualmente similar a cualquier aplicación gnome. El usar este tipo de elementos tiene ventajas, sobre todo si se piensa en un usuario que utilice gnome a diario:

- Los usuarios aprenden a usar el programa más rápidamente, ya que los elementos de la interfaz se ven y se comportan de la forma que ellos están acostumbrados.
- Tanto usuarios principiantes como avanzados podrán realizar tareas en forma rápida y fácil, ya que la interfaz no será confusa y no hará las cosas más difíciles.
- La aplicación tendrá una vista atractiva que encajará con el resto del escritorio.

La aplicación original usaba una interfaz basada en la librería qt, esta librería tiene básicamente las mismas capacidades que GTK, se decide finalmente por GTK por las siguientes razones: Qt tiene limitaciones en el uso de la licencia que GTK no posee, los objetos por defecto de GTK son visualmente más agradables que los de Qt, y se poseía una aplicación de prueba con código abierto realizada en GTK que se usó como referencia, para crear la interfaz final.

3.3 Diagrama de Clases

A continuación se describirá las clases más importantes que conforman el programa actualmente.

3.3.1 Interfaz

La interfaz del programa está descrita principalmente por la clase GUIVentanaPrincipal, la que se encarga de mostrar y manejar la ventana principal de la interfaz gráfica del programa y toda la interacción del usuario con el programa. SimpleGLScene se encarga de manejar el frame OpenGL que se encuentra dentro de la interfaz gráfica.

Ambas clases GUIVentanaPrincipal y SimpleGLScene usan partes de la librería GTK para mostrar las ventanas y los objetos visuales característicos del framework GTK.

SimpleGLScene tiene una referencia a la clase Malla que es la que contiene todos los métodos y operaciones sobre la malla geométrica. Si se deseara cambiar la interfaz solo sería necesario invocar a Malla y sus métodos desde otra clase que implemente la nueva interfaz.

En la figura 3.1 se muestra las clases GUIVentanaPrincipal, y SimpleGLScene, además de otras clases similares GUIDialogCriterio y GUIDialogAlgoritmos, que sirven para mostrar ciertos cuadros de diálogo que se utilizan dentro de la interfaz.

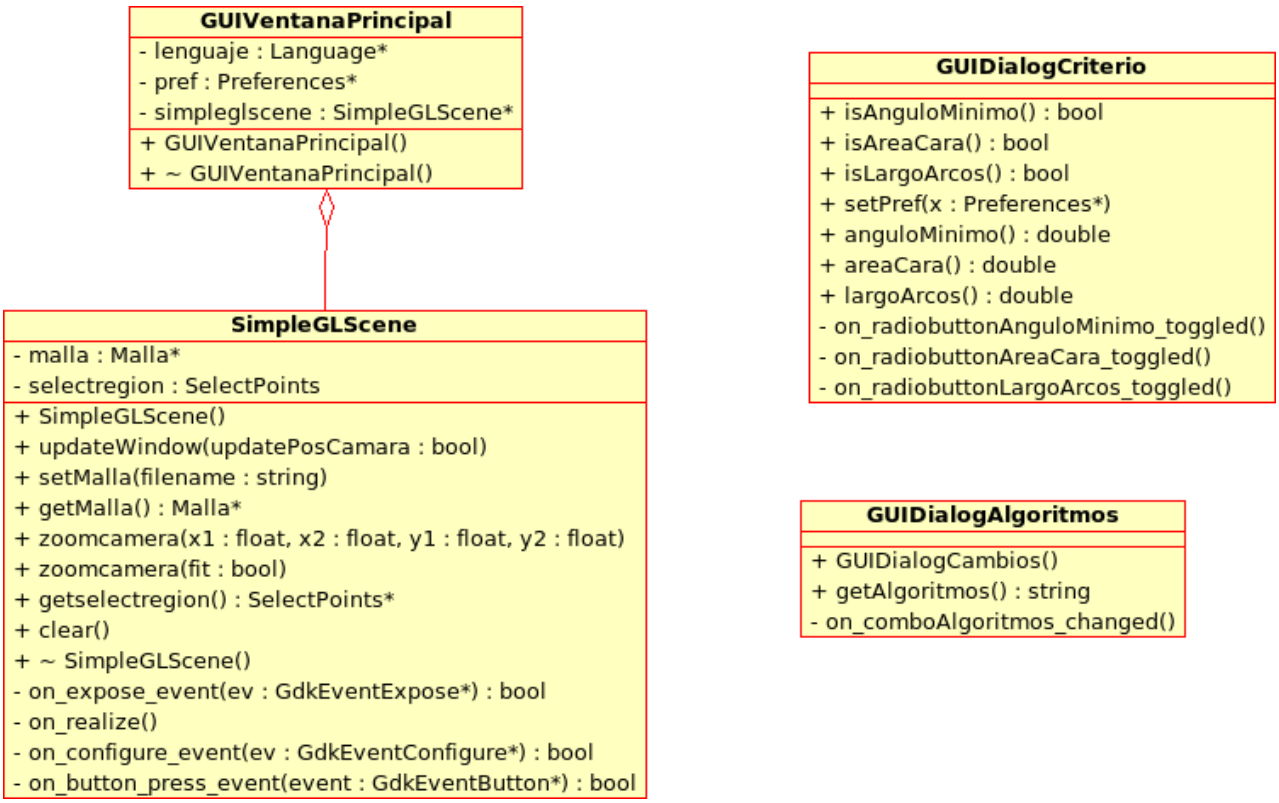

Figura 3.1: Clases interfaz Ventana Principal

3.3.2 Command

Se utiliza command para generar, almacenar, y ejecutar todos los comandos generados por el usuario hacia la interfaz y la malla.

En el ejemplo de la figura 3.2 la interfaz GUIVentanaPrincipal ejecuta un comando del tipo QualityCommand (Algoritmo Lepp-Delaunay quality) con parámetros y una región dada LPRegion. Así como existe un comando concreto implementado para el algoritmo LEPP-Delaunay, existen otros para implementar el algoritmo *Voronoi Point Insertion*, y para definir otros comandos conectados con la interfaz.

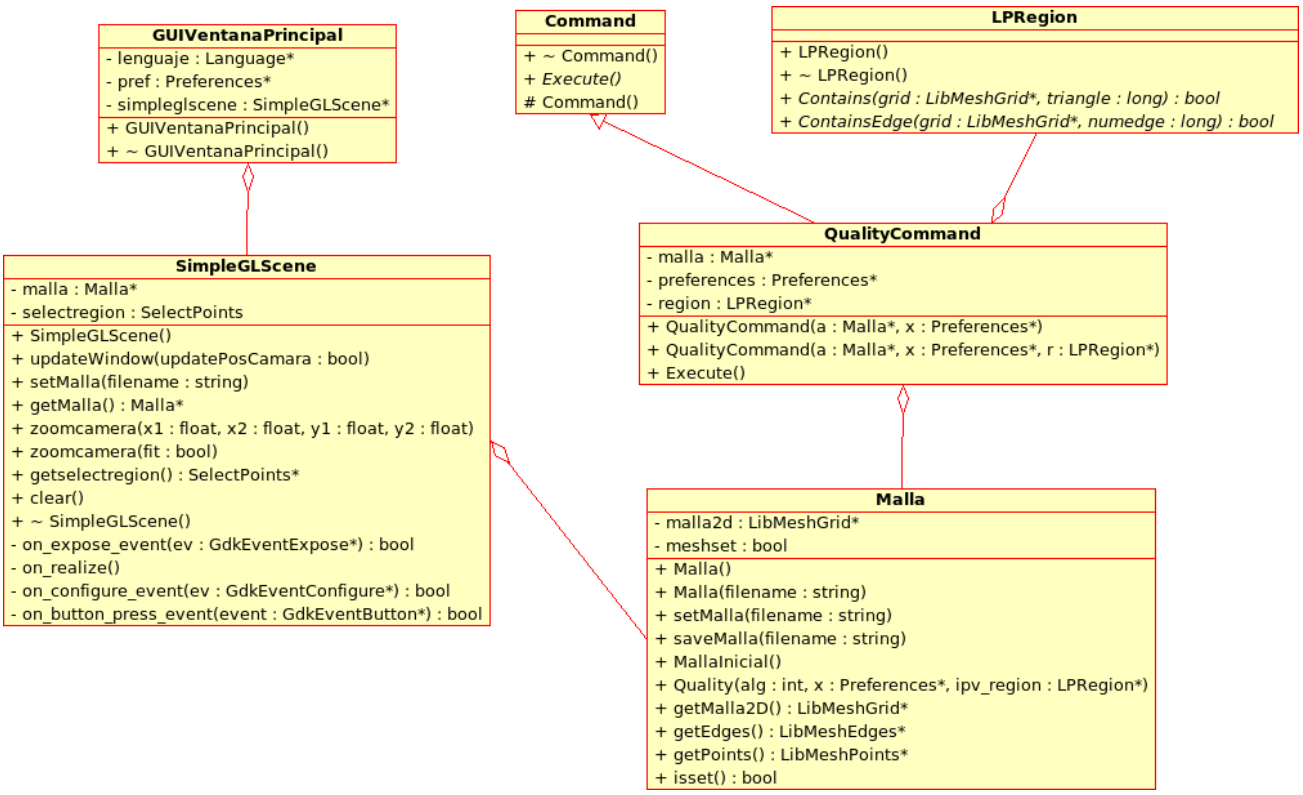

Figura 3.2: Ejemplo de un comando para refinar una malla.

3.3.3 Algoritmos de Refinamiento

En la figura 3.3 se observa el diagrama de clases correspondiente a los algoritmos de refinamiento de LEPP-Delaunay.

Todos los algoritmos se basan en una clase padre LPLeppBasedAlgorithms, de la cual LPIterativeApproximation, LPRecursiveApproximation y LPQualityDelaunayTriangulation heredan de él.

Se agrega la función Generate(LPRegion* ipv_region, long criterion_code, DList& parameters, long testing_level, long approx_level), a todos los subalgoritmos, la que considera en sus calculos el uso de regiones.

En la figura 3.4 se observa el diagrama de clases correspondiente a los algoritmos de refinamiento de *Voronoi Point Insertion*. Para estas clases también se crean dos métodos distintos, uno que incluye uso de regiones y otro que no.

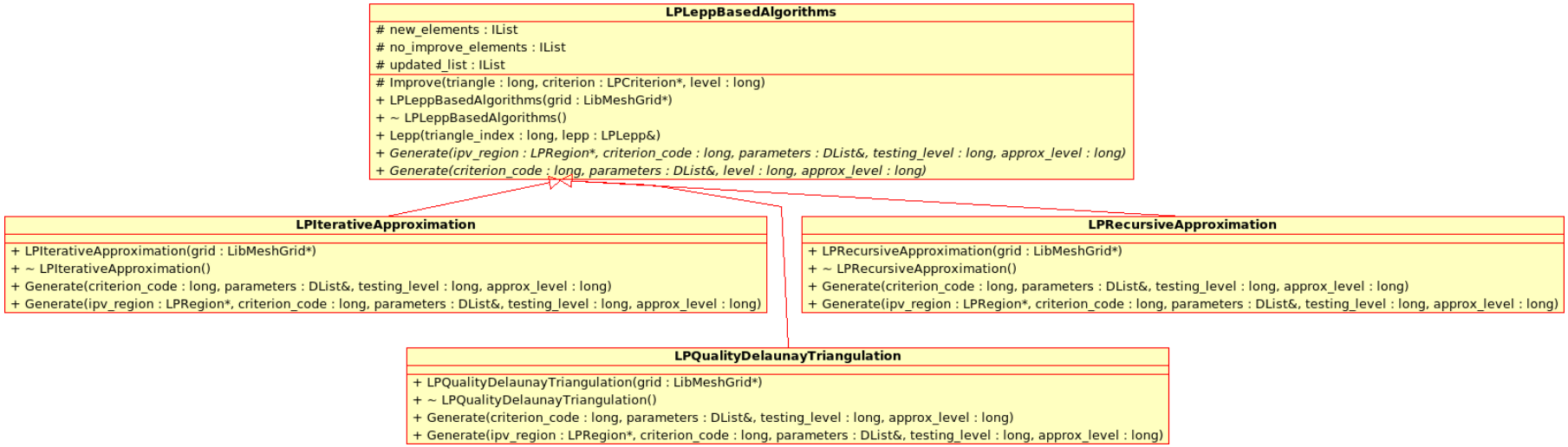

Figura 3.3: Diagrama de clases de algoritmos de refinamiento Lepp-Delaunay

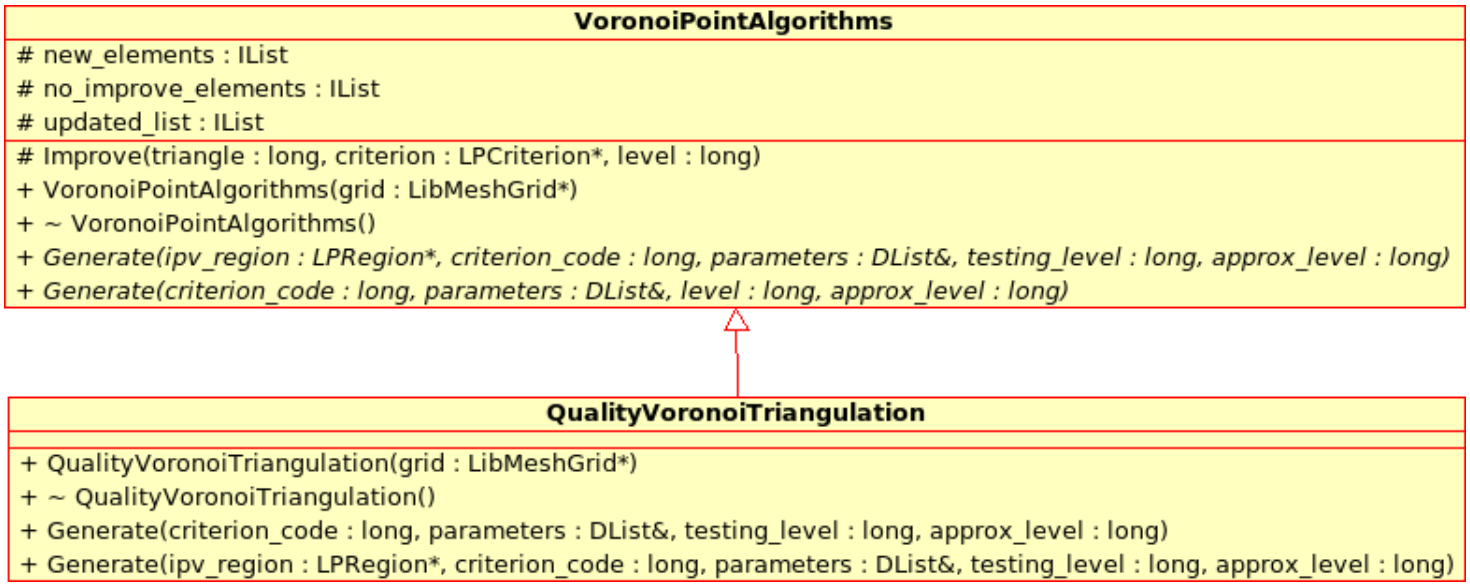

Figura 3.4: Diagrama de clases de algoritmos de refinamiento *Voronoi Point Insertion*

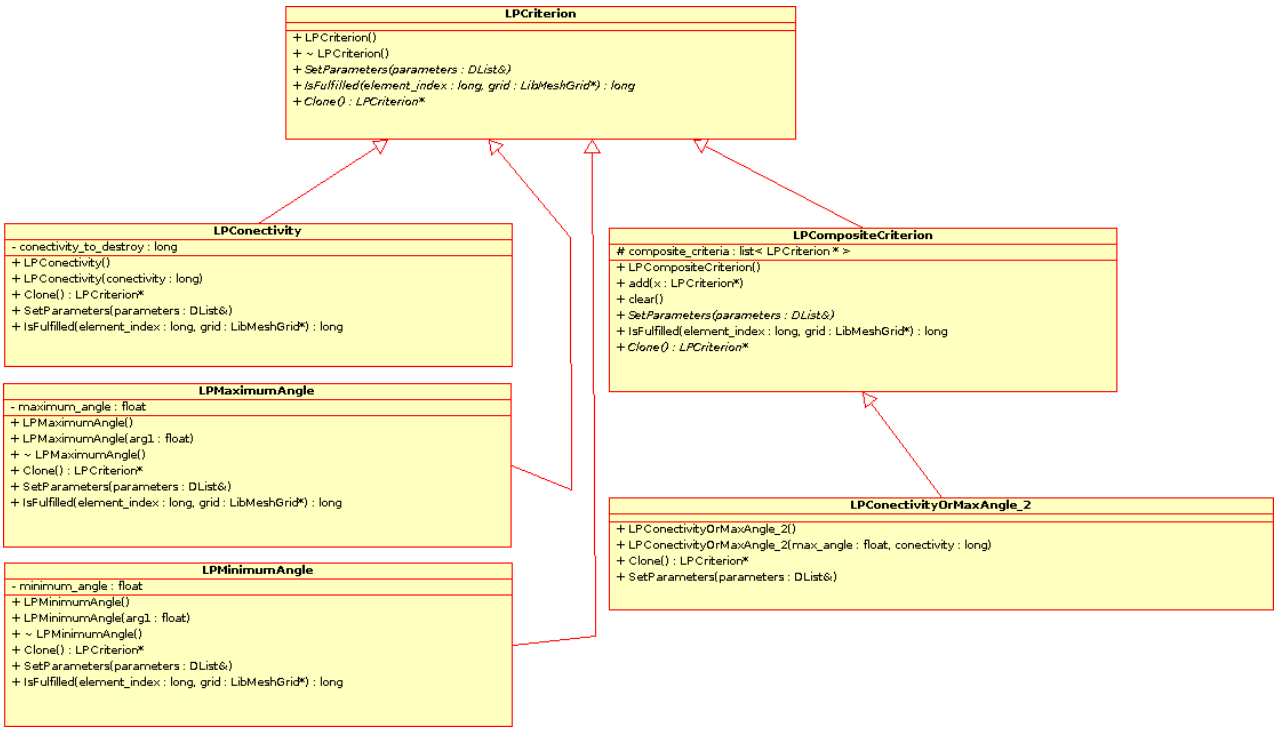

Figura 3.5: Diagrama de clases para los criterios usados en los algoritmos de mejoramiento de malla.

3.3.4 Criterio

Las clases LPCriterion y sus hijas se usan para definir y calcular si es que un triángulo dado cumple con la condición o condiciones especificadas. Se puede ver el diagrama de clases de criterio en la figura 3.5.

LPCriterion contiene un método virtual IsFulfilled(long element_index, LibMeshGrid *grid) retorna *true* si es que el triángulo satisface el criterio establecido, *false*, si es que no lo cumple. Este método debe ser implementado por cada criterio nuevo que se quiera usar dentro del programa.

LPCompositeCriterion es una nueva clase que permite crear nuevos criterios en base a uniones o composiciones de criterios simples ya creados.

3.3.5 Regiones

Las clases LPRegion se encargan de definir, manejar y verificar las regiones dentro de una malla. Se puede ver el diagrama de clases de criterio en la figura 3.6.

LPRegion posee un método virtual Contains(LibMeshGrid *grid, long triangle) que retorna true si es que el triángulo dado se encuentra sobre la región establecida, *false* si la región no contiene el triángulo. Contains es un método virtual, que debe ser implementado por cada tipo (clase) de región.

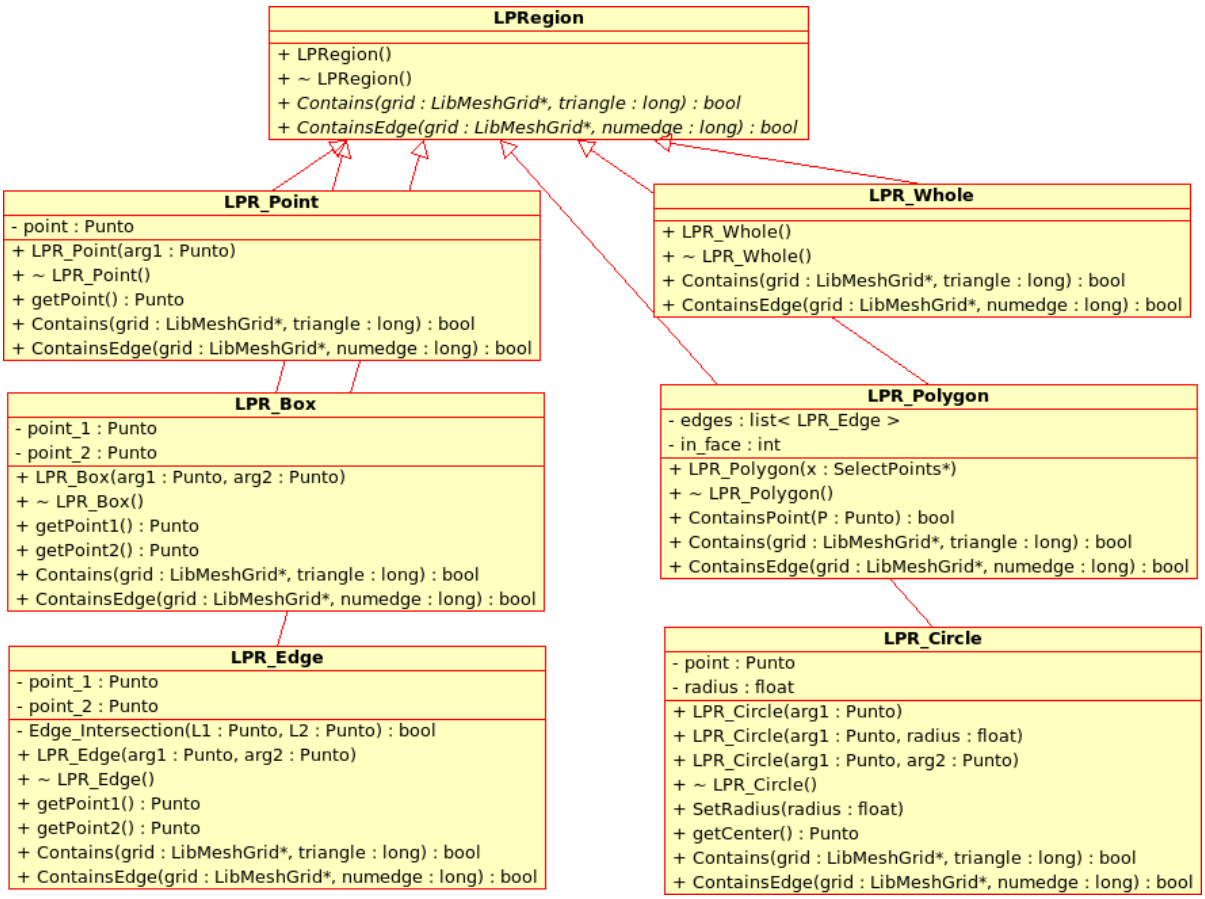

Figura 3.6: Diagrama de clases para definir regiones de refinamiento dentro de una malla.

4 Implementación

En este capítulo se discutirán los detalles de implementación de la aplicación. En particular, se discuten acerca del ambiente de desarrollo y bibliotecas utilizadas. Se especifican detalles de la implementación de la interfaz, los algoritmos de refinamiento y/o mejoramiento, las clases que representan criterio y región.

4.1 Ambiente

La aplicación fue desarrollada sobre una máquina basada en el sistema operativo Linux, distribución Gentoo con kernel 2.6.16 r12. El lenguaje de programación usado es C++, con la versión 3.4.6 del compilador GNU (gcc). Este equipo se encuentra físicamente ubicado en las oficinas del Departamento de Ciencias de la Computación.

Uno de las detalles más rescatables de la implementación de la aplicación original era el uso de una librería para manipular estructuras geométricas, LibMesh, en este trabajo se ha decidido reutilizar esta librería, pues provee la funcionalidad básica para la manipulación de una malla y sus elementos. Dado que no es opensource, sólo se puede utiliza específicamente en la máquina ubicada en las oficinas del DCC.

El equipo es accedido de manera remota, vía SSH, y un túnel SSH del puerto X, para efectos de interactuar visualmente con el programa, sin estar físicamente en las oficinas DCC.

El Ambiente de Desarrollo Integrado (IDE) utilizado para la generación e implementación de las clases en C++ fue KDevelop versión 3.3.4., bajo Linux Ubuntu 6.10.

4.2 Bibliotecas

La aplicación usa varios conjuntos de bibliotecas. Éstas son GTK+, GTKmm, y GTKGlExtmm; GTK+, GTKmm, OpenGL y LibMesh se utilizaron para la creación de la interfaz de usuario de la aplicación. GTKGlExtmm se utilizó para la implementación del visualizador OpenGL en tiempo real integrado en la aplicación.

Se decidió utilizar estas bibliotecas, porque fueron las que más se acomodaban a los requerimientos inmediatos de la aplicación, y/o en las que se tenía cierto nivel de dominio previo.

La interfaz de la aplicación original usaba la librería QT, si bien esta librería provee casi las mismas funcionalidades que GTK, pero QT tiene una licencia de uso mucho más restrictiva que la que tiene GTK.

A continuación se entrega una descripción y una referencia de estas bibliotecas.

- GTK+: Provee un conjunto de bibliotecas para la construcción de interfaces gráficas de usuario (GUI). Está desarrollado en C. Es una de las bibliotecas más utilizadas en el desarrollo de aplicaciones gráficas para Linux.

- GTKmm: Es la interfaz oficial escrita en C++ para la biblioteca GTK+. GTKmm permite utilizar el lenguaje C++ para el desarrollo de la interfaz gráfica. Junto con ello se obtienen todos los beneficios de la programación orientada a objetos como lo son la herencia, extensibilidad, programación por componentes, etc.

- GTKGlExtmm: Es una extensión OpenGL para GTK+. Esta extensión provee objetos adicionales para GTK+ los cuales tienen la característica de soportar renderizado OpenGL sobre ellos. Tiene una interfaz en C++ para que se pueda utilizar libremente con ese lenguaje.

- OpenGL: Es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. OpenGL permite al usuario abstraerse del rasterizado sobre la pantalla o de las funciones específicas de la tarjeta gráfica.

- LibMesh: Esta biblioteca define una estructura estándar para manipular de manera más sencilla las complejas mallas geométricas en dos dimensiones. Provee una estructura de datos flexible para manejar los elementos componentes de las mallas en dos dimensiones.

4.3 Interfaz

La nueva interfaz de la aplicación está dividida en cuatro secciones importantes: el menú (1), barra de botones (2), pantalla visualizadora (3), y un log de mensajes (4).

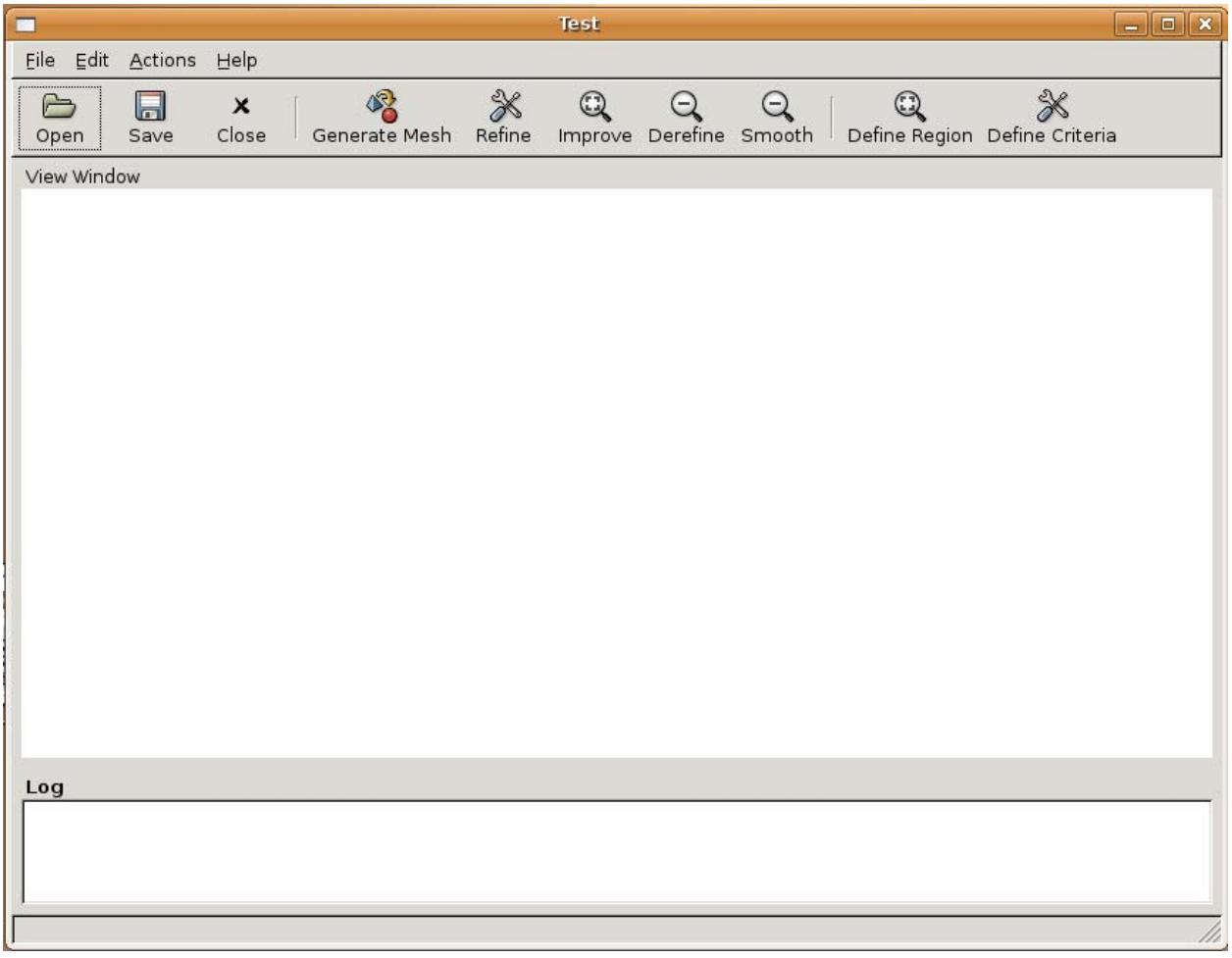

Figura 4.1: Interfaz de Usuario del Programa

4.3.1 Menú

En el menú están organizados todos los comandos posibles dentro del programa, para su fácil acceso por parte del usuario. Actualmente, existen cuatro submenús principales, Archivo (1), Editar (2), Acciones (3), Ayuda (4).

- Archivo: Contiene los comandos para manejar archivos externos, como por ejemplo abrir y guardar mallas 2D.

- Editar: Contiene los comandos para editar las preferencias del programa - -- como el lenguaje -, los criterios que se van a usar dentro de los algoritmos, la forma de elegir una región para refinar o mejorar.

- Acciones: Contiene los comandos para ejecutar cambios sobre la malla, como por ejemplo mejorar una malla mediante un algoritmo deseado, o generar una malla inicial a partir de un PSLG.

- Ayuda: Contiene comandos para mostrar información adicional sobre el programa o la malla que actualmente está cargada en el programa.

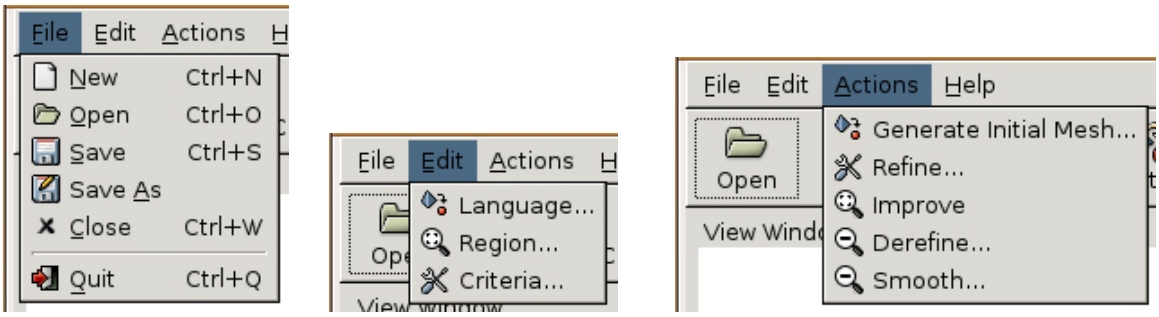

Figura 4.2: Submenús Archivo, Editar y Acciones.

4.3.2 Barra de Herramientas

Existe una barra de herramientas que contiene los comandos más importantes del programa, para su acceso de manera fácil y directa. Dentro de la barra se tienen los siguientes comandos: Abrir, Guardar, Cerrar (Menú Archivo). Generar Malla Inicial, Refinar, Mejorar, Derefinar, Suavizar (Menú Acciones). Definir Criterio, Definir Región (Menú Editar).

Figura 4.3: Barra de Herramientas, contiene los comandos más importantes.

4.3.3 Visualizador

El visualizador es la sección más importante de la ventana principal, pues es en ésta, donde se dibuja la malla geométrica. En este visualizador se tiene un objeto GTKGlExtmm que permite renderizar OpenGL sobre él. Además de ser una pantalla para dibujar la malla geométrica y los cambios sobre ella, también se utiliza para leer entrada del usuario, tales como selección en la pantalla o regiones de refinamiento.

4.3.3.1 Selección en Pantalla

Es posible definir una figura o área de selección en el visualizador. Esta selección puede representar ciertas formas o regiones dentro del programa. Para esto se leen los píxeles que son seleccionados por el usuario en secuencia mediante el mouse, según el tipo de selección que se esté realizando

4.3.4 Log de Mensajes

En la parte inferior de la ventana principal del programa se tiene un log de mensajes que se usa para informar al usuario acerca de mensajes importantes, tales como la selección de comandos, ejecución de éstos, problemas con los archivos o procesos, etc.

4.4 Manejo de Malla 2D

4.4.1 Generación de Malla Inicial

Una vez que se tiene cargado un PSLG dentro del programa, se puede generar una triangulación inicial mediante este comando. LibMesh incluye un simple comando para generar una triangulación inicial, luego de que se ha cargado una malla dentro de su estructura.

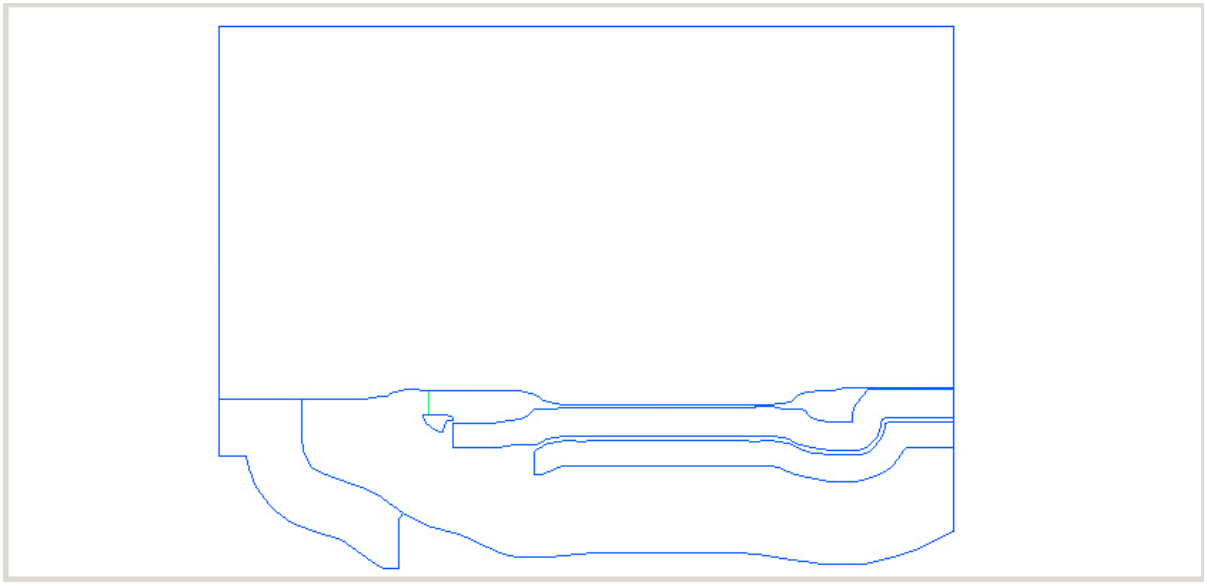

Figura 4.4: PSLG cargado en el programa.

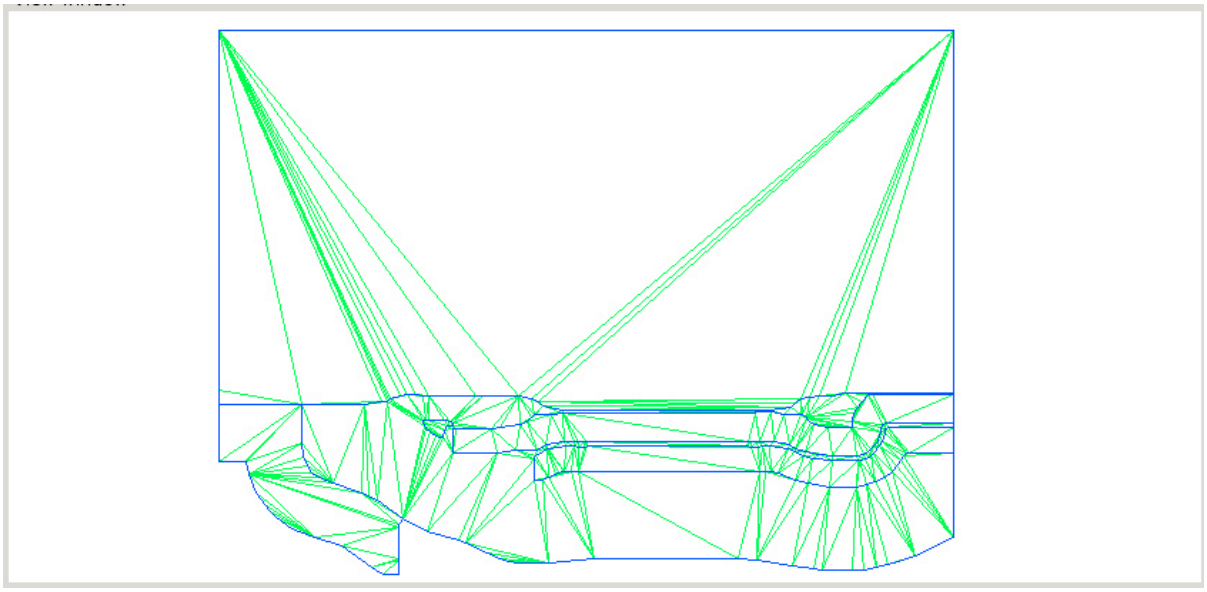

Figura 4.5: Malla después de generada la malla inicial a partir del PSLG.

4.4.2 Refinamiento/Mejoramiento

Una vez que se tiene una triangulación inicial se puede refinar la malla, de manera que ésta cumpla con ciertos criterios preestablecidos. Se desea evitar "malos" triángulos, En general, se considera malos, los triángulos con ángulos muy agudos o muy obtusos.

Para este refinamiento se puede elegir entre un set de diferentes algoritmos, cada uno con un énfasis distinto:

- Algoritmos basados en LEPP
    - Quality
    - Iterative
    - Recursive
- *Voronoi Point Insertion*

Figura 4.6: Cuadro de diálogo de selección del algoritmo

En general, estos algoritmos manejan dos listas que referencian a los triángulos de la malla: (a) Lista de triángulos *malos* y (b) Lista de los que no se refinarán. (a) contiene todos los triángulos que aún no cumplen el criterio dado, - por ejemplo, que todos sus ángulos interiores sean mayores que a 30º - mientras que (b) contiene una lista de triángulos que no se mejorarán, a pesar que no cumplen con el criterio. Estos últimos son triángulos que tienen aristas en el borde de la figura, y, a pesar de que puedan tener ángulos muy agudos (u obtusos), no son mejorados debido a que se desea mantener la geometría original del objeto que se está modelando. La única forma de mejorar este último tipo de triángulos es modificando la geometría del objeto.

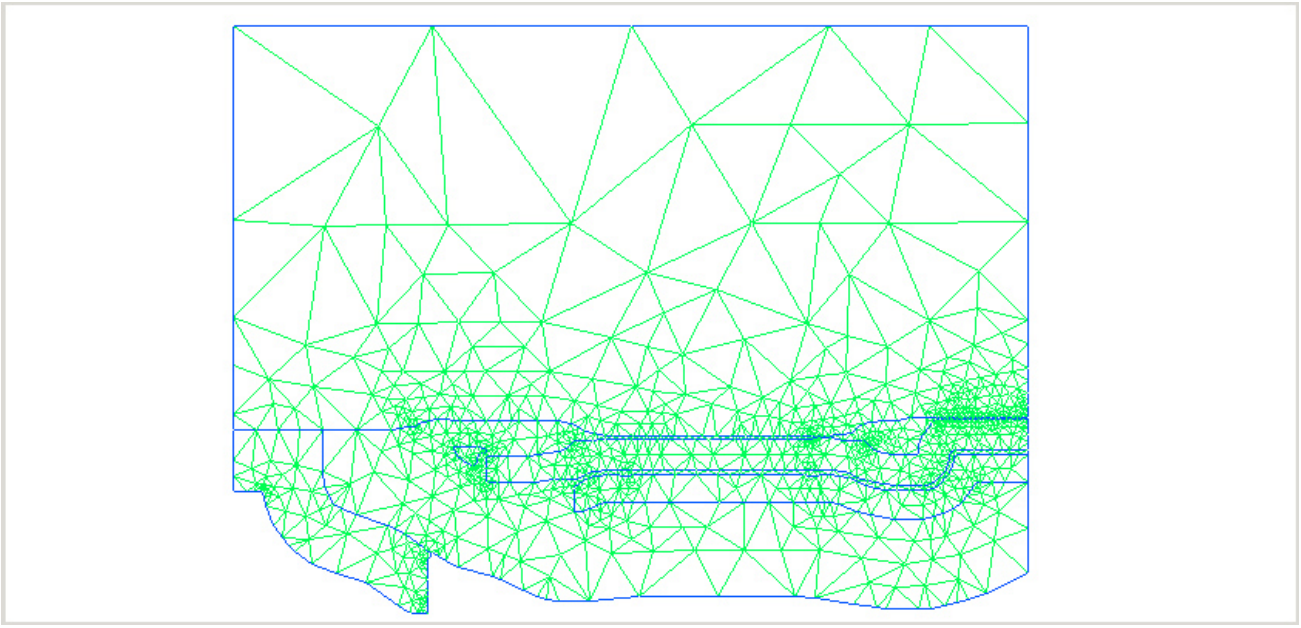

Figura 4.7: Malla refinada mediante algoritmo LEPP.

El diagrama de clases está diseñado de tal manera que es sencillo agregar nuevos algoritmos a este set preestablecido, si se deseara en un futuro cercano.

4.4.2.1 Algoritmo Voronoi Point Insertion

Este algoritmo de refinamiento se basa en la inserción de nuevos puntos dentro de la malla para mejorar los triángulos que no satisfacen los criterios definidos para un buen triángulo.

El punto a insertar se define como el centro de Voronoi del triángulo que se va a mejorar, es decir, el centro del circumcírculo correspondiente al triángulo. En base a este nuevo punto y sus elementos vecinos se crean nuevos triángulos, los que, en general, serán mejores que el original, mejorando efectivamente de esta manera la malla. [3]

Algoritmo:

- Para cada triángulo t que no cumpla el criterio establecido
    - Se calcula el centro de Voronoi del triángulo a mejorar.
    - Si el centro de Voronoi se encuentra dentro del triángulo
        - Si el triángulo tiene algún arco en el borde, se debe verificar si el centro de Voronoi se encuentra dentro de la circunferencia diametral del arco borde (si es más de uno se verifica en todos)
            - Si se encuentra en un arco borde, entonces se inserta un punto nuevo en el punto medio del arco borde. Si existe más de un arco borde en el cual el punto se contenga dentro de su circunferencia diametral, entonces se utiliza el arco más largo.
            - Si no lo es, entonces se usa el centro de Voronoi para insertar un punto nuevo.
        - Se crean nuevos triángulos, y nuevos arcos, en base a los puntos del triángulo original y el punto nuevo. Se destruye el triángulo original.
    - Si el centro de Voronoi se encuentra afuera del triángulo
        - Se busca el triángulo en el cual se encuentra
        - Si el triángulo es borde, se debe verificar si el centro de Voronoi se encuentra dentro de la circunferencia diametral del arco borde (si es más de uno se verifica en todos)

- Si se encuentra en un arco borde, entonces se inserta un punto nuevo en el punto medio del arco borde. Si existe más de un arco borde en el cual el punto se contenga dentro de su circunferencia diametral, entonces se utiliza el arco más largo.

- Si no lo es, entonces se usa el centro de Voronoi para insertar un punto nuevo. Se utiliza los vértices para crear tres nuevos triángulos y arcos junto con el centro de Voronoi.

  - Luego de que se crean los nuevos triángulos, se debe asegurar que satisfagan la condición de Delaunay, si no se cumple, se realizan los cambios de diagonales necesarios.

- Se itera hasta que no existan triángulos que no cumplan con el criterio establecido.

4.4.3 Criterios

Son medidas de referencia para decidir cuándo un triangulo deja de ser un mal triángulo. En general todos los algoritmos de refinamiento comparten estos mismos tipos de criterio, así que se puede elegir el tipo de criterio que se utilizará en el programa, independientemente del tipo de algoritmo que se va a usar.

Los criterios que se usan en el programa actualmente son:

- Ángulo Máximo
- Ángulo Mínimo
- Conectividad

Pudiendo, en algunos casos ser compuestos, Ángulo Máximo y Conectividad, por ejemplo. Este fue re-implementado agregando la clase virtual LPCompositeCriterion. (Ver figura 3.5)

Figura 4.8: Cuadro de diálogo de selección del criterio

El diagrama de clases está diseñado de tal manera que es sencillo agregar nuevos criterios a este set preestablecido, si se deseara en un futuro cercano.

4.4.4 Regiones

En general, la operación de refinamiento se realiza sobre toda la malla, a menos que se indique lo contrario.

Una región permite especificar una zona de la malla que debe ser refinada. En general se representan como una figura geométrica sobre la malla, pero podría abstraerse a una lista de elementos (triángulos) que no tuvieran ninguna relación directa con otro.

Es posible definir en el programa, regiones de refinamiento, es decir, regiones en las cuales se selecciona cierto numero de triángulos, los cuales serán refinados, y el resto de los triángulos será ignorado. Actualmente, se usan sólo para definir regiones limitadas de refinamiento, pero estas regiones podrían extenderse a otros usos alternativos, por ejemplo, guardar solo una parte de la malla definida por la región, también sería posible eliminar ciertos triángulos que estén definidos por una región.

Para esto se aprovecha el hecho de que los algoritmos manejan una lista de triángulos que no se refinarán, así, una vez definida la región, todos los triángulos que no cumplan con las condiciones de la región – estar dentro de ella, o intersectarla - son agregados a la lista de triángulos que no se refinarán, de esta forma sacándolos efectivamente del proceso de refinamiento.

Las regiones implementadas en la aplicación actual son:

- Punto
- Línea
- Rectángulo
- Circunferencia
- Polígono

4.4.4.1 Implementación de la clase LPRegion

El método principal es Contains (bool Contains(LibMeshGrid *grid, long triangle) , que retorna *true* si es que la región contiene al triángulo dado y *false*, si es que no.

Ésta es una clase virtual, que debe ser heredada y su método Contains implementado, por las distintas clases de regiones que se deseen construir.

LPR_Whole:LPRegion

Representa toda la malla, es una región que contiene todos los elementos de la malla.

La función Contains devuelve siempre valor true.

LPR_Circle:LPRegion

Representa una circunferencia de centro P y radio r. Todo triángulo intersectado o contenido en ella, se define como dentro de la región.

La función Contains busca primero si es que hay algún vértice del triángulo a una distancia menor o igual a r del punto centro P, si es que existe alguno el triángulo, está inmediatamente dentro de la región, luego se busca la intersección entre la circunferencia y los tres arcos del triángulo, si existe, entonces el triángulo pertenece a la región, en caso contrario, la función retorna falso.

LPR_Box:LPRegion

Representa un rectángulo de vértices no contiguos P_1 y P_2. Todo triángulo intersectado o contenido en este rectángulo, se define como dentro de la región.

La función Contains revisa primero si algún vértice del triángulo se encuentra dentro de la zona del rectángulo, aprovechando la forma ortogonal de éste. Si no se encuentra ningún vértice dentro, entonces se verifica la intersección de los arcos del rectángulo con los arcos del triángulo. Si existe intersección, entonces el triángulo pertenece a la región; en caso contrario la función retorna falso.

LPR_Edge:LPRegion

Representa un segmento de una recta de vértices P_1 y P_2. Todo triángulo que sea intersectado por este segmento se define como dentro de la región.

La función Contains revisa si hay intersección entre el segmento de recta y alguno de los arcos del triángulo, si es así retorna true, si no, false.

LPR_Polygon:LPRegion

Representa un polígono simple, con vértices P_1, P_2, … , P_i, … ,P_n; n $\geq$3. Todo triángulo que se encuentre al interior del polígono, o que intersecte con algún arco del polígono, se define como dentro de la región.

La función Contains revisa primero si algún vértice del triángulo se encuentra dentro de la zona interior del polígono. Si no se encuentra ningún vértice dentro, entonces se verifica la intersección de los arcos del rectángulo con los arcos del polígono.

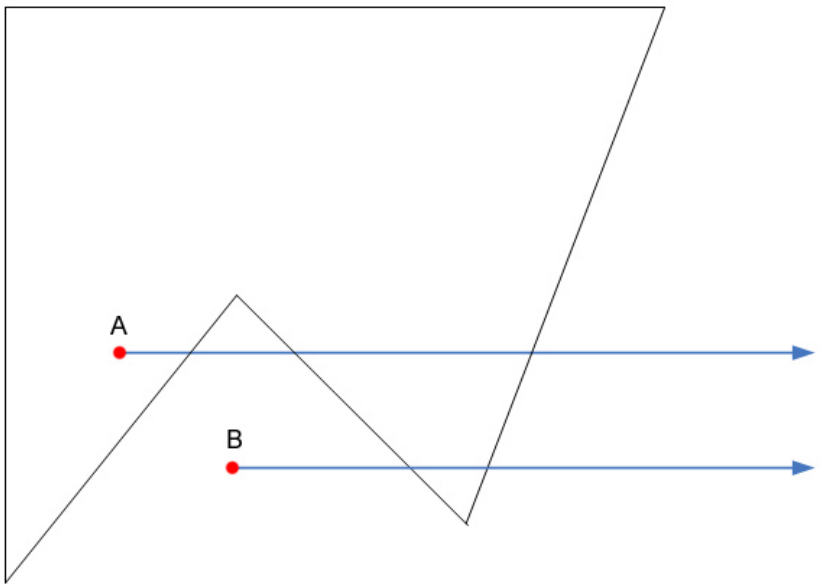

Figure 4.9: Ray casting y verificación de puntos dentro de un polígono

Para verificar que un vértice se encuentra dentro del polígono, se utiliza una técnica de ray casting, en la que se traza un rayo desde el punto por verificar en forma horizontal, y se verifica las intersecciones que tiene con el polígono, si el numero de intersecciones es par, entonces el punto se encuentra fuera del polígono, si es impar, entonces se encuentra al interior del polígono.

En la figura 23 se puede ver como proyectando un rayo horizontal desde el punto A se tienen 3 intersecciones con el polígono, numero impar, por lo tanto se encuentra dentro del polígono. El punto B tiene dos intersecciones, por lo tanto está fuera del polígono. El chequeo de intersecciones se realiza en base a cada uno de los segmentos, por lo tanto los puntos tangentes no son un problema.

LPR_Point:LPRegion

Representa un punto P dentro de la malla. Es una región diferente de las demás, pues no contiene nada geométricamente. Se dice que un triángulo se encuentra en esta región cuando el punto P está al interior del triángulo.

La función Contains verifica si es que el punto P está al interior del triángulo, ya sea estrictamente al interior o en alguno de los bordes.

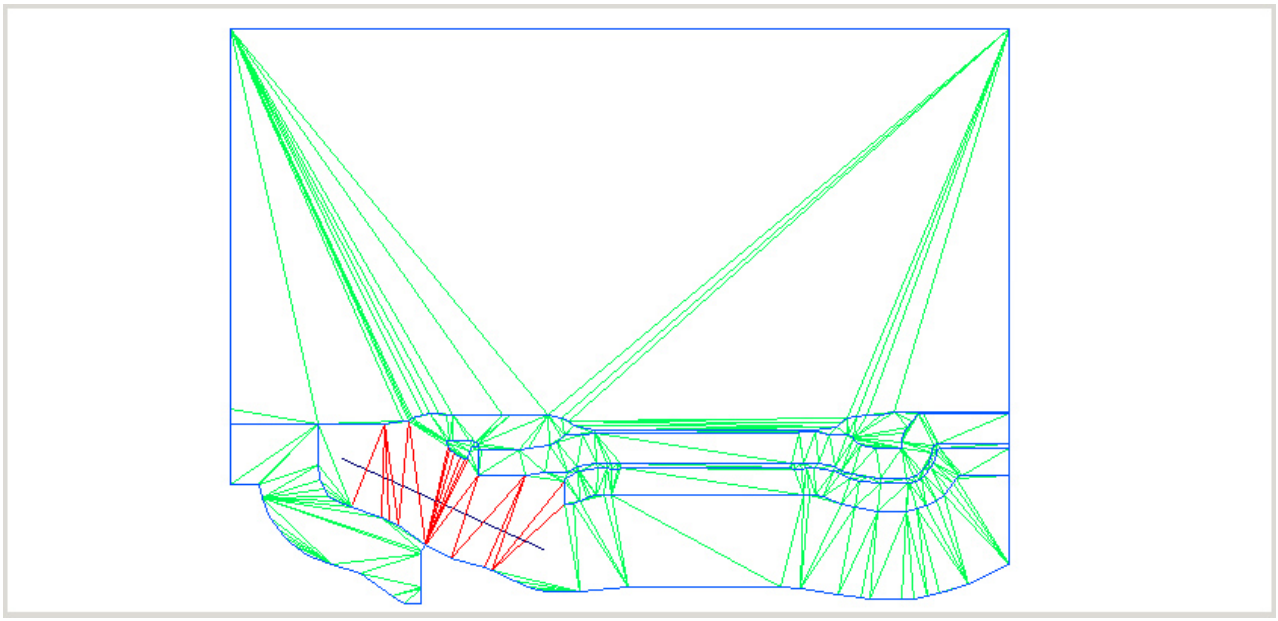

Figura 4.10: Región lineal seleccionada, todos los triángulos intersectados pertenecen a la región.

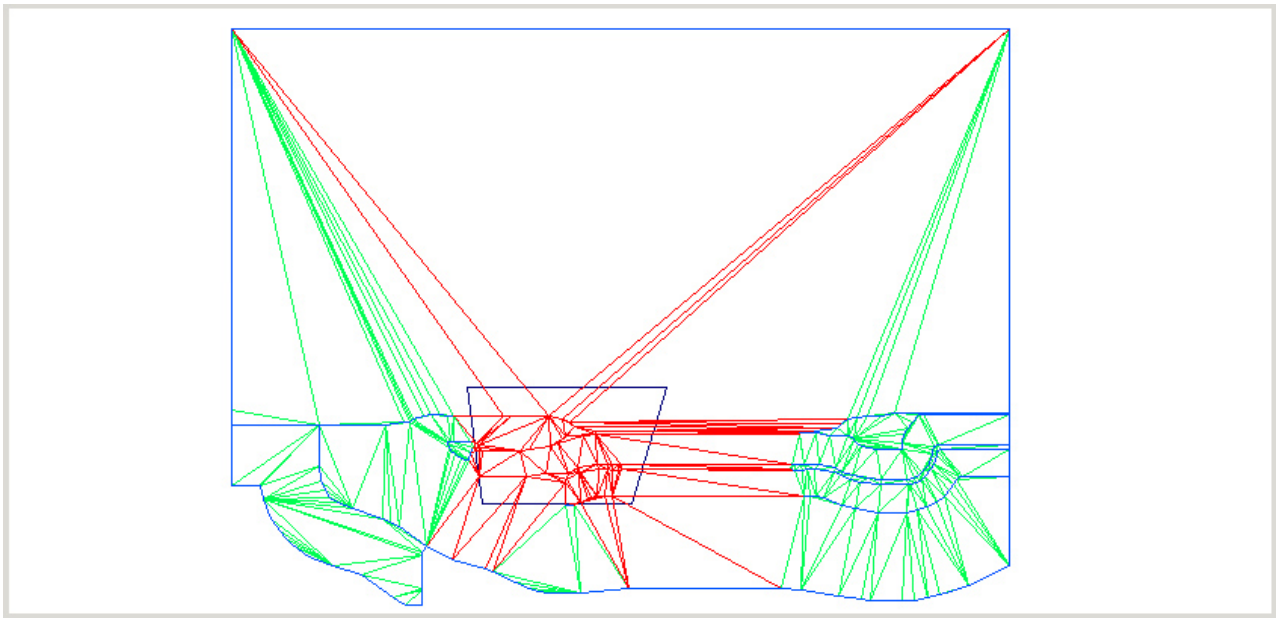

Figura 4.11: Región poligonal seleccionada, los triángulos
al interior del polígono son seleccionados.

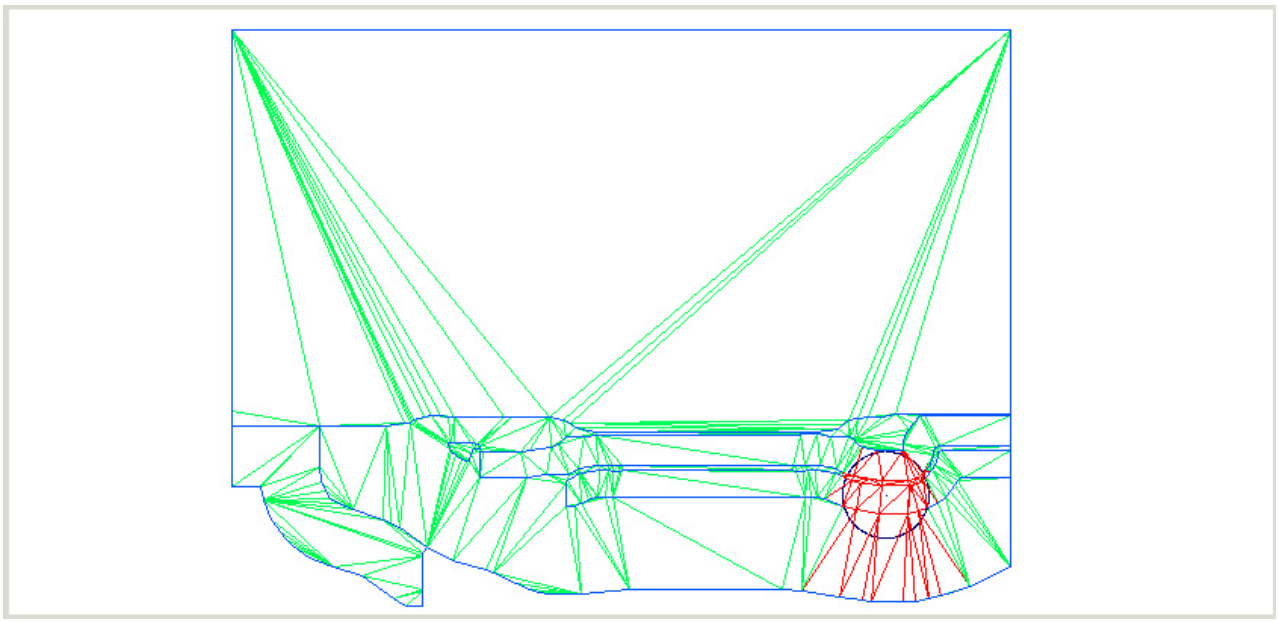

Figura 4.12: Región circunferencial seleccionada, los triángulos al interior son seleccionados.

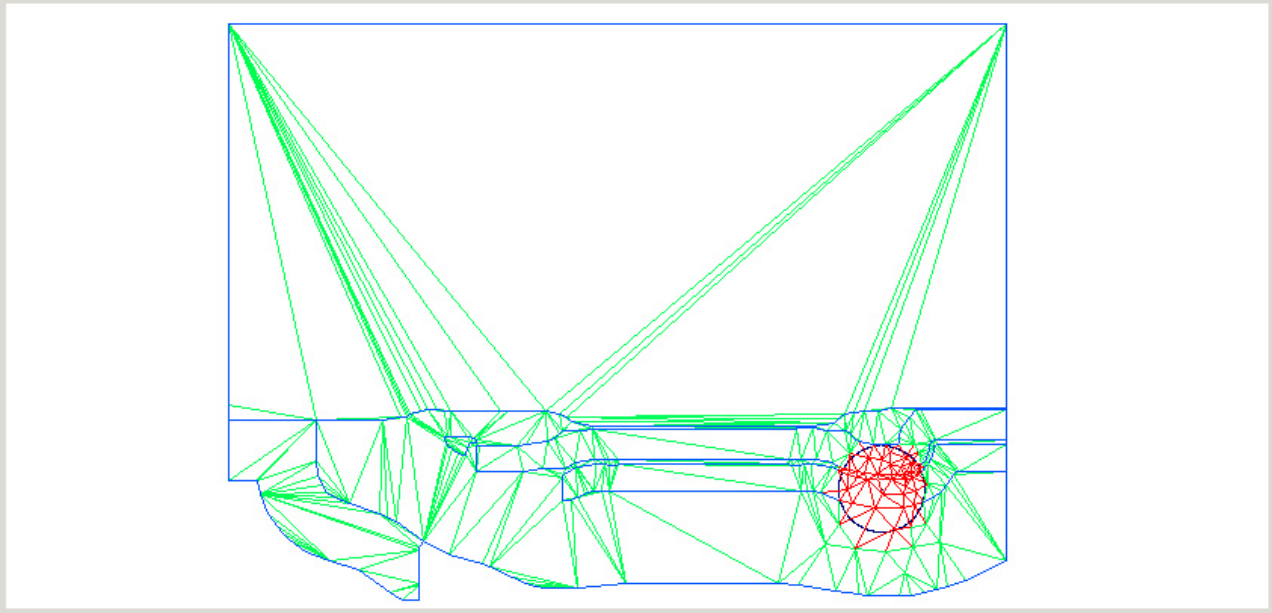

Figura 4.13: Malla refinada de acuerdo a la región seleccionada

El diagrama de clases está diseñado de tal manera que es sencillo agregar nuevas regiones a este set preestablecido, si se deseara en un futuro cercano.

5 Conclusión

En este capítulo se presentan las conclusiones finales de este trabajo. Se comenta adicionalmente de posible trabajo futuro sobre la aplicación.

5.1 Conclusiones

En esta memoria se abordó el problema de implementar un visualizador interactivo de mallas geométricas en dos dimensiones. Para esto, se continuó el desarrollo de la aplicación escrita por Javier Irarrázabal[1]. A la aplicación se le incluyeron los siguientes cambios o extensiones:

1. Rediseño de las clases de la aplicación.
2. Mejoramiento completo de la interfaz gráfica de la aplicación.
3. Incorporación de nuevos algoritmos de refinamiento de mallas geométricas.
4. Incorporación de regiones como concepto dentro de la aplicación, y modificación de los algoritmos de refinamiento ya existentes para su uso con regiones limitadas de la malla.
5. Modificación de las clases que tienen que ver con los criterios son usados dentro de los algoritmos de refinamiento, para aumentar su extensibilidad a nuevos criterios.

Como conclusión y resultado final de este trabajo, se tiene una aplicación computacional que es capaz de:

- Visualizar un PSLG o una malla geométrica en dos dimensiones. Capacidad de *zoom* dentro de la malla.

- Cargar un PSLG o malla geométrica desde un archivo.
- Guardar la malla en un archivo.
- Generar una malla Delaunay inicial a partir un PSLG.
- Realizar operaciones de refinamiento sobre mallas geométricas, usando diferentes algoritmos, mientras se mantiene la propiedad de malla Delaunay.
- Definir regiones de refinamiento, de manera rápida y sencilla, las que luego son usadas por las operaciones de refinamiento para limitar su dominio.
- Definir criterios de refinamiento, para decidir cuando un triángulo es aceptable dentro de una malla geométrica, los que luego pueden ser usados por los distintos algoritmos disponibles para las operaciones de refinamiento.

El rediseño de la aplicación fue realizado aplicando herramientas UML como diagramas de clases para así obtener un buen diseño orientado a objetos considerando el uso de patrones de diseño en las partes necesarias. El rediseño se concentró principalmente en garantizar un buen grado de extensibilidad en todos los elementos claves del programa, los algoritmos de refinamiento, la definición de regiones, y la definición de criterios. Otro de los puntos importantes a considerar fue el encapsulamiento de las clases relacionadas con la interfaz gráfica, con el resto de la aplicación, así es relativamente sencillo cambiar sólo la interfaz gráfica (o incluso cambiarla completamente) sin tener que modificar nada de la sección del manejo de mallas en sí. Este tipo de diseño facilita cualquier futuro desarrollo sobre la aplicación.

Se reconstruyó la interfaz de usuario desde cero, usando elementos propios de GTK, para la interfaz en sí, y OpenGL, para la visualizador de la malla, que está

embebido dentro de la ventana de la interfaz. Esto produce como resultado una interfaz muy poderosa, pero a la vez amigable y sencilla de usar.

Todos estos elementos previamente descritos, producen como resultado una herramienta para visualizar y manejar mallas geométricas en dos dimensiones, que es poderosa, extensible, flexible y simple de usar.

5.2 Trabajo Futuro

Existen varias direcciones en donde este trabajo puede seguir desarrollándose, entre los problemas que se pueden abordar se encuentran:

Lectura y escritura en formatos distintos al actualmente soportado. Tener la capacidad de importar mallas desde visualizador y formatos externos da más flexibilidad a la aplicación.

Cargar mallas a partir de imágenes. El objetivo es poder generar una malla coherente con una imagen o foto de la realidad. Esta aplicación puede ser muy útil en campos como medicina, por ejemplo.

Regiones no contiguas. En este momento las regiones que se pueden definir son solo contiguas, sería interesante definir un tipo de región que fuese una composición de regiones simples.

Agregar otros tipos de algoritmos. Es posible por ejemplo, incluir algoritmos de de-refinamiento o suavizado a esta herramienta, para hacerla mucho más completa.

Reescribir LibMesh. LibMesh tiene algunas desventajas, como por ejemplo ser pobremente documentada, y no poseer algunos métodos que pueden ser deseables para ciertas aplicaciones. El objetivo en un trabajo futuro sería generar una

biblioteca LibMesh *opensource*, en la cual se pudieran agregar las principales características de este visualizador, y así tener una gran librería que no solo almacene y maneje mallas en dos dimensiones, sino que también tenga diferentes algoritmos de refinamiento, suavizado; posea manejo de diferentes regiones y criterios.

6. Referencias

[1] Irarrázabal Galleguillos J. Interfaz Gráfica para Algoritmos de Generación de Mallas 2D, Memoria para optar al título de Ingeniero Civil en Computación, 2004.

[2] Bastarrica MC, Hitschfeld-Kahler N. Designing a Product Family of Meshing Tools, Advances in Engineering Software, 37(2006)1-10.

[3] Shewchuk J. R., Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, Applied Computational Geometry: Towards Geometric Engineering, volume 1148 of Lecture Notes in Computer Science, pages 203-222, Springer-Verlag, Berlin, May 1996.

[4] Rivara MC, Hitschfeld N, Simpson RB. Terminal-edges delaunay (small-angle-based) algorithm for the quality triangulation problem. CAD 33(2001) 263-77.

[5] Rivara MC, Hitschfeld-Kahler N. LEPP-Delaunay Algorithm. A Robust Tool For Producing Size-Optimal Quality Triangulations. Departamento de Ciencias de la Computación. Universidad de Chile.

[6] Hitschfeld-Kahler N. Apuntes Curso CC60Q - Seminario de Geometría Computacional. Departamento de Ciencias de la Computación. Universidad de Chile. URL:http://www.dcc.uchile.cl/~cc60q

[7] Sitio Oficial de GTK+. URL: http://www.gtk.org

[8] Sitio Oficial de GTKmm. URL: http://www.gtkmm.org

[9] Sitio Oficial de GTKGlExtmm. URL: http://www.k-3d.org/gtkglext/Main_Page