Universidad del Bío-Bío, Chile

Facultad de Ciencias Empresariales

Departamento de Sistemas de Información

Triangulación de Delaunay Restringida de objetos 2D con geometrías irregulares

Memoria presentada por Fabian Ernesto Castillo Cruces

para obtener el título de Ingeniero de Computación en Ejecución e Informática

Dirigida por Dr. Pedro Rodríguez Moreno

2017

Agradecimientos

Este trabajo no habría sido posible sin el apoyo y el estímulo de ...

A Sandra y Jorge, mis padres, los cuales fueron un apoyo incondicional, gracias por su confianza y afectuoso cuidados.

A Javier, mi hermano, ya que fue un amigo silencioso pero oportuno, el cual siempre estuvo dispuesto a tender una mano cuando lo necesite.

A Nicol, una gran mujer con alma bondadosa, tu aliento constante y tu absoluta fe en mi me dieron las fuerzas que necesitaba, te amo con locura.

A mis amigos por se una luz en este largo camino

A Pedro Rodríguez, por su apoyo, paciencia y disponibilidad durante todo este tiempo.

Este proyecto de título fue parcialmente financiado por el grupo de investigación Computación Ubicua, código GI 150115/EF.

Si para recobrar lo recobrado, debí perder primero lo perdido, si para

conseguir lo conseguido, tuve que soportar lo soportado, si para estar ahora enamorado, fue menester haber estado herido, tengo por bien sufrido lo sufrido, tengo por bien llorado lo llorado, porque después de todo he comprobado, que no se goza bien de lo gozado, sino después de haberlo padecido, porque después de todo he comprendido, que lo que el árbol tiene de florido, vive de lo que tiene sepultado(FRANCISCO LUIS BERNÁRDEZ).

Abstract

The importance of Triangulation algorithms of geometrical networks is due to the fact that there are multiple applications for them, such as cartographic maps models, creation of scaled 3D models and applications in the automotive industry. In general, they can be applied to any area where it is necessary to model complex objects or to make a study about it. This project contributes providing an informatics solution for the development of a software able to analyze and process images with a unique object, by means of filters which will provide, as output, the boundaries of it, in order to generate a series of dots representing the borders of the original object in the image. Once the dots have been obtained, the program will triangulate them maintaining the outline of the figure. In order to achieve the development of this solution, a search on triangulation methods was done with Professor Pedro Rodriguez. It was decided that the method best fitting this problem is the restricted Delaunay triangulation which generates a triangulation capable of maintaining the boundaries without inserting extra dots in the initial dots cloud. Then, the method to detect the boundaries was determined whish is concluding that, the optimal one for this problem is the Canny Edge Detection filter, one of the most widely used in this area, and consists of three phases: (a) obtaining of the gradient, (b) non-maximal suppression and (c) threshold hysteresis. After the methods to be used were determined, the development stage of the application was done using object oriented programming language (JAVA) to implement the canny algorithm because with the output generated

the initial cloud of dots was determined allowing to successfully generate a restricted Delaunay triangulation (it must be explicitly stated that this code is based on the Delaunay Triangulation provided by Professor Pedro Rodriguez). The result was a software which fully fulfills the requirements initially planned, with a minimalistic interface, because it is expected that it will be used as the basis for future informatics projects related to this subject.

Keywords — Triangulation, Delaunay, Canny Edge Detector, JAVA, Restricted Triangulation.

Resumen

La importancia de los algoritmos de triangulación de mallas geométricas radica en el hecho de que existen múltiples aplicaciones donde éstas se usan, tales como, modelos de mapas cartográficos, creación de modelos 3D a escala o aplicaciones en la industria automotriz. En general, se puede aplicar a cualquier área donde se necesite modelar un objeto complejo, o hacer un estudio sobre éste.

El objetivo de este proyecto es construir la Triangulación de Delaunay Restringida a partir de un conjunto de puntos en el espacio 2D que forman el contorno o borde de un objeto obtenido a partir de una imagen. Para obtener el borde de la figura, se aplica un proceso de filtrado que genera una serie de puntos que constituyen el borde del objeto. El programa puede también obtener el borde o contorno de hoyos presentes en el objeto a triangular. Por ejemplo, si tenemos un mapa de una isla, ésta puede tener lagos o lagunas, los cuales son considerados como espacios internos (hoyos) que no son triangulados. Una vez obtenidos los puntos el programa procede a triangular conservando el contorno de la figura.

Para concretar el desarrollo de esta solución se realiza una investigación junto al profesor Pedro Rodríguez de los métodos de triangulación. Se determinó que el método que más se adaptaba a ésta problemática es la Triangulación de Delaunay Restringida. Luego se determinó el método para la detección de bordes, concluyendo que el más óptimo para esta problemática es el *Filtro Canny Edge Detector*, uno de los más usados en esta temá-

tica, el cual consta de 3 fases: obtención del gradiente, supresión no máxima e hysterisis de umbral.

Palabras Claves — Triangulación, Delaunay, Canny Edge Detector, JAVA, Triangulación de Delaunay, Triangulación de Delaunay Restringida.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Introducción

Las mallas geométricas son herramientas fundamentales en ingeniería y en la ciencia, en particular en la modelación de fenómenos físicos a través de *Ecuaciones Diferenciales Parciales* (EDP). Una malla es un conjunto de caras poligonales (cuadriláteros, triángulos o tetraedros) que definen una superficie en el espacio tal que sí t_i y t_j son dos elementos de una malla T, entonces: $t_i \cap t_j$ es un vértice común, una arista común, una cara común o el conjunto vacío. Una malla tiene asociada un conjunto de elementos topológicos tales como: vértices, aristas y caras poligonales. Una malla se obtiene a partir de la discretización de un dominio Ω aplicando un algoritmo de triangulación. Si Ω es un conjunto de puntos, la triangulación obtenida es una partición de su cierre convexo en triángulos, tal que cada punto presente en nube sea un vértice de un triángulo. Por lo tanto a la triangulación sobre el dominio Ω en 2D se le denomina malla de triángulos, y en 3D malla de tetraedros.

Uno de los primeros algoritmos de generación de triangulaciones fue propuesto por el matemático ruso Boris Nikolaevich Delaunay [4]. Este algoritmo se conoce como *Triangulación de Delaunay* a través del cual se puede obtener una malla de triángulos, que a partir de un conjunto de puntos en el espacio conforman un polígono convexo y conexo, lo que implica que el resultado es un único conjunto de triángulos unidos de tal forma que

se puede ir de cualquiera de estos a otro en linea recta sin salir del polígono (ver Figura 1.1.a).

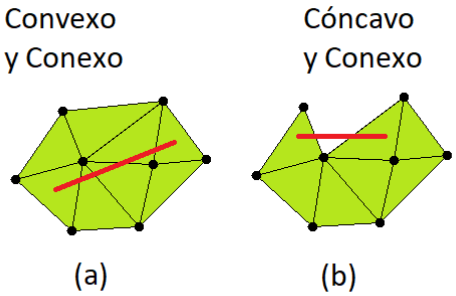

Figura 1.1: Polígonos convexo y cóncavo.

También se debe tener en cuenta que una triangulación es de Delaunay si cada triángulo presente en la triangulación cumple el *Test del Círculo*, lo que implica que el círculo circunscrito en cada triangulo solo debe contener los vértices que conforman el triángulo testeado, detallado en la Sección 2.1 (ver Figura 1.2).

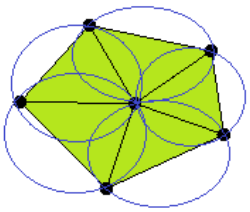

Figura 1.2: Test del Círculo.

Por otro lado se entiende Triangulación de Delaunay Restringida en este documento como a una Triangulación de Delaunay que concertaba los límites de la figura inicial a tratar. A menudo una Triangulación de Delaunay Restringida contiene triángulos en el

borde y espacios intermedios que no satisfacen la condición de Delaunay, además tiende a formar polígonos cóncavos (ver Figura 1.1.b). Por lo tanto, una Triangulación de Delaunay Restringida no suele ser una Triangulación de Delaunay en sí misma.

Es por esto que se construye un método para generar una Triangulación de Delaunay Restringida, que es aplicado a la nube de puntos generado, lo que permite analizar objetos irregulares altamente complejos, para ésto se utiliza como base un algoritmo de Triangulación de Delaunay proporcionada por el profesor Pedro Rodríguez Moreno. Por otro lado, debido a la importancia de estos datos, se pretende buscar un modo de guardarlos y exportarlos a otros programas que ocupan este tipo de mallas.

1.1. Definición del problema

Este trabajo esta enmarcado dentro del área de estudio e investigación del profesor Pedro Rodríguez Moreno quien se dedica al estudio y análisis de objetos complejos e irregulares utilizando mallas geométricas, en la actualidad no cuenta con un método que permita generar éstas mallas de una forma rápida, eficiente y automatizada para su posterior estudio.

Debido a la falta de un software de este tipo, el proceso se debe llevar a cabo de manera manual insertando cada punto de la figura e intentando conservar su contorno, ésto no asegura la fiabilidad del resultado final, ya que esta sujeto a errores humanos. Además incurre en un alto costo de tiempo, lo que da como resultado una figura inexacta y de poca utilidad.

En este proyecto se propone diseñar e implementar un software que permita capturar una figura geométrica irregular desde una imagen, mediante la aplicación de filtros digitales, ésto da como resultado una imagen retocada que contiene únicamente los bordes del objeto.

Una vez obtenida la imagen retocada, se procede a generar puntos que representaran la

figura inicial, para luego conformar con éstos una Triangulación de Delaunay Restringida que conserva el contorno y huecos presentes en la imagen.

1.2. Hipótesis

El proyecto mencionado plantea que es posible reducir considerablemente los tiempos para la construcción de mallas geométricas en objetos irregulares, ya que permitiría generar la nube de puntos de forma totalmente automatizada obteniendo la información requerida a través del procesamiento de imágenes, lo que aseguraría un alto nivel de fiabilidad y exactitud en la salida del algoritmo, por consiguiente los estudios serán más eficientes, claros y precisos.

1.3. Objetivos del proyecto

A continuación se presentan los objetivos que se pretenden alcanzar en esta memoria.

1.3.1. Objetivo general

El objetivo general de ésta memoria consiste en el desarrollo de un software en el lenguaje de programación JAVA que permita analizar y procesar imágenes para detectar los bordes internos y externos de ésta por medio de filtros. Con los bordes detectados el programa debe procesar la imagen retocada para identificar los puntos del contorno de la imagen y generar puntos en un plano cartesiano para proceder a generar una Triangulación de Delaunay Restringida que conserve la forma del objeto lo más cercano al original.

1.3.2. Objetivos específicos

- Investigar los diferentes algoritmos de tipo Delaunay que se han desarrollado para construir triangulaciones.
- Diseñar una estrategia general para obtener la Triangulación de Delaunay Restringida en un objeto 2D irregular.
- Implementar un algoritmo de Triangulación Delaunay Restringida.
- Procesar a nivel de pixel imágenes 2D que contengan un objeto con geometría irregular con el propósito de obtener el borde interno, externo y su representación poligonal.
- Implementar el software de triangulación utilizando un lenguaje orientado a objetos (JAVA) para obtener la Triangulación de Delaunay Restringida de los puntos del borde.

1.4. Alcances del proyecto

El software permite:

- Cargar y visualizar una imagen.
- Detectar los bordes de una figura presente en una imagen.
- Generar los vértices de la figura respetando bordes y hoyos presentes en ésta.
- Reducir la cantidad de puntos que se generan de las imágenes tratadas.
- Generar una Triangulación de Delaunay Restringida.
- Generar un informe de vértices que liste únicamente los vértices considerados en la triangulación.
- Guardar las mallas generadas en formato .m2d.
- Cargar mallas anteriormente generadas en formato .m2d.
- Guardar los vértices generados en formato .pto.
- Cargar vértices anteriormente generados en formato .pto.

1.5. Límites del proyecto

A continuación se listan los límites de este proyecto :

- Se utiliza un algoritmo de Triangulación Delaunay proporcionado por el profesor Pedro Rodríguez como base para implementar el algoritmo de Triangulación de Delaunay Restringida.
- Sólo se consideran imágenes de entrada en formato .jpg y .png.
- El proyecto sólo abarca modelos en 2D.
- No se considera el caso de segmentos de línea dentro del área del objeto.
- Los espacios interiores en la figura principal con área cercana a cero no serán considerados.
- Sólo consideran imágenes iniciales que contengan un único elemento en su interior.
- No se implementan algoritmos de refinamiento para eliminar triángulos de mala calidad.
- La reducción de puntos se hará de una única manera, eliminando n cantidad de puntos intermedios, definido por el usuario, de tal manera que por cada punto imprimido se salta n para imprimir el siguiente.

1.6. Estructura del documento

A continuación se listaran el resto de los capítulos presentes en este documento:

Capítulo 2: Esta dedicado a introducir conceptos fundamentales utilizados a lo largo del desarrollo del proyecto. Partiendo por la triangulación 2D, la calidad de una triangulación, Triangulación de Delaunay junto con sus propiedades y métodos de construcción. Luego se da paso a la Triangulación de Delaunay Restringida y Conforme para finalizar con la detección de bordes utilizando el algoritmo Canny Edge Detector y sus criterios.

Capítulo 3: Se introduce el ambiente de ingeniería de software a utilizar, mediante la explicación de la metodología de desarrollo seleccionada, se mencionan los beneficios de ésta. Luego se describen las técnicas y notaciones que se utilizan en el transcurso del proyecto, como modelos de diagramas de clases, casos de uso, especificaciones de casos de uso y diagramas de flujo de datos. Para finalizar con los estándares de la documentación.

Capítulo 4: Se presentan los requerimientos funcionales y no funcionales del sistema, junto con los casos de uso, sus diagramas y especificaciones. Luego se continúa con los diagramas de flujos de datos, Canny Edge Detector, generar lista de vértices, Triangulación de Delaunay Restringida y el diagrama de clases. Finalizando con las herramientas que se utilizan para el desarrollo del proyecto, a nivel hardware y software.

Capítulo 5: Se muestran los diseños de interfaz de usuario y la jerarquía con la que cumple el menú del sistema.

Capítulo 6: Se explica la implementación del sistema.

Capítulo 7: Se muestran las pruebas correspondientes del sistema.

Capítulo 8: Se exponen la conclusiones del proyecto, para finalizar mencionando posibles trabajos futuros relacionados con el tema de investigación.

Capítulo 2

Conceptos claves

2.1. Tringulación 2D

Una *Triangulación en Dos Dimensiones* sobre un conjunto de puntos, se define como una subdivisión del plano en caras triangulares cuyos vértices corresponden a los puntos del conjunto inicial.

Las triangulaciones también se conocen por el nombre de mallas de triángulos. En la Figura 2.1 se muestra un ejemplo de una triangulación sobre un plano.

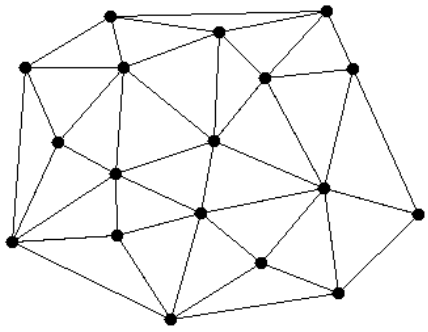

Figura 2.1: Triangulación sobre un plano 2D.

Se dice que una triangulación o malla de triángulos es válida si cumple con las siguientes

características:

- Todos los triángulos construidos poseen área mayor que cero.
- La intersección del interior de dos triángulos cualesquiera del conjunto, es siempre vacía.
- La intersección de dos triángulos cualesquiera del conjunto, corresponde a una arista común o a un vértice común.
- El conjunto define una superficie conexa, abierta o cerrada.

Las triangulaciones son utilizadas en aplicaciones como: modelación de terrenos [10], computación gráfica y animaciones, análisis de fenómenos físicos modelados mediante ecuaciones diferenciales y hasta en aplicaciones a macro escala como el estudio de la gran muralla cósmica [5], la super estructura más grande conocida por el hombre.

Un problema de particular interés para la geometría computacional y otras aplicaciones tales como hidrología [8], análisis de datos espaciales y el método de elementos finitos, es encontrar la triangulación de un conjunto de puntos sobre el plano que maximice el menor ángulo interno de los triángulos que la conforman.

2.2. Calidad de una triangulación

Una característica de las mallas utilizadas en elementos finitos, es que la densidad de triángulos en distintos sectores de la malla varía de acuerdo a la necesidad de precisión en los resultados que se requiera para dichos sectores. Generar una malla con más triángulos o con más vértices que la que se tiene inicialmente no es un proceso complicado. Un ejemplo trivial consistiría en elegir cualquier triángulo en un sector en que se requiera mayor densidad de la malla y reemplazarlo con los tres triángulos que forma la unión del incentro, el cual es el punto en el que se cortan las tres bisectrices de sus ángulos internos, que equidistan de los tres lados, por lo tanto, es el centro de la circunferencia inscrita en

el triángulo.

Sin embargo, la malla resultante de éste proceso puede producir resultados desastrosos en la aplicación que se utilice, debido a la forma dispar de los triángulos generados.

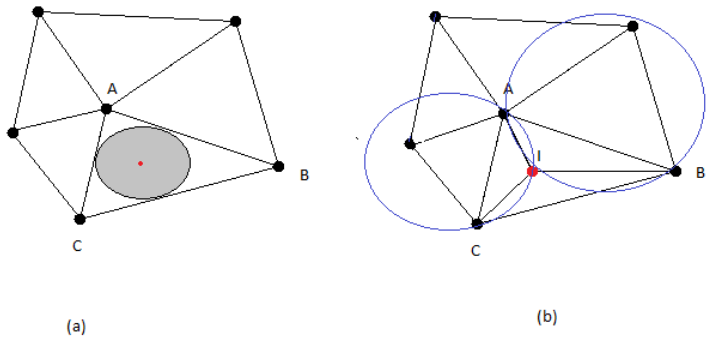

Figura 2.2: Insercíon de un punto.

En la Figura 2.2.a se inserta el punto correspondiente al incentro del triángulo ABC, para luego proceder a particionar el antiguo triángulo en tres nuevos triángulos ABI, BCI y CAI (ver Figura 2.2.b), los cuales son triángulos de mala calidad pues sus ángulos son muy agudos y los triángulos ABI y CAI no pasan el Test del Círculo. Para evitar mallas con triángulos indeseados, los algoritmos de refinamiento pueden usar las siguientes medidas de calidad:

- Evitar triángulos con ángulos interiores pequeños.
- Evitar los triángulos con una alta razón dada por la fórmula matemática:

$$R = \frac{Circunradio}{2 * Inradio} \tag{2.1}$$

Donde circunradio es el radio de la circunferencia circunscrita de un triángulo o circuncírculo, como se muestra en la Figura 2.3 donde el circunradio esta representado por R y pertenece al radio de la circunferencia circunscrita de ABC y el inradio es el radio de la circunferencia inscrita en el triángulo o incírculo. En este caso el inradio esta representado

por r y pertenece al radio de la circunferencia inscrita del triángulo ABC presente en la Figura 2.3.

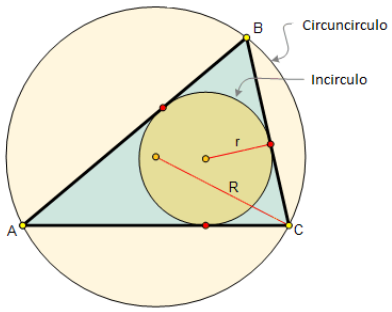

Figura 2.3: Circunradio, inradio.

Se considera un triángulo de mala calidad, como un triángulo presente en la triangulación que se desea refinar, de acuerdo a algún criterio de calidad.

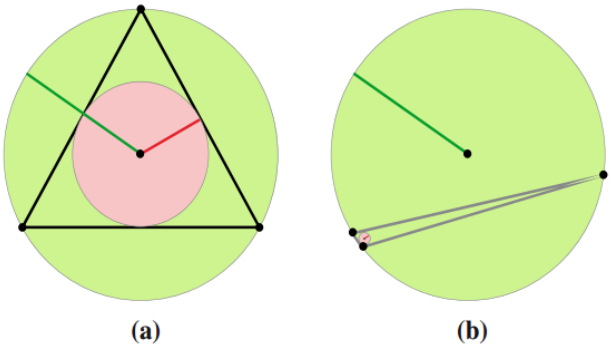

Figura 2.4: Calidad de triángulo.

En la Figura 2.4.a el triángulo presenta una razón de aspecto 1, dada por la Fórmula 2.1 y ángulos interiores iguales a 60º lo que indica que es un triángulo de calidad equilátero. Por otro lado el triángulo de la Figura 2.4.b presenta una razón de aspecto 5 y ángulo mínimo interior muy agudo. Ejemplo clásico de triángulo de mala calidad. Por lo tanto se busca refinar triángulos con una razón mayor que 1.

2.3. Triangulación de Delaunay

Dentro de la geometría computacional, se estudia la formación de triangulaciones sobre un conjunto de puntos P en un plano, el cual se define como una subdivisión plana máxima cuyos vértices corresponden a P. Por lo tanto se puede definir *Triangulación de Delaunay* como un conjunto de triángulos T de una triangulación del conjunto P de puntos sobre un plano. T es Delaunay sí y sólo sí cumple las propiedades mencionadas a continuación.

2.3.1. Propiedades

La Triangulación de Delaunay satisface las siguientes propiedades:

(a) La Triangulación de Delaunay maximiza el menor ángulo de la malla, es decir, el menor de los ángulos internos de los triángulos que la conforman.
(b) La frontera de la triangulación es la cerradura convexa de los puntos, en otras palabras, las aristas del borde de la triangulación forman un polígono convexo que contiene todos los demás puntos.
(c) Para cada triángulo presente en la triangulación se debe aprobar el *Test del Círculo.* (ver Definición 2.1), como se aprecia en la Figura 2.5.c.

Definición 2.1 *[Test del Circulo:] Dado un conjunto de puntos en el plano, tres puntos definen un triángulo de Delaunay sí el círculo que circunscribe a dicho triángulo no contiene ningún otro punto del conjunto de datos [1].*

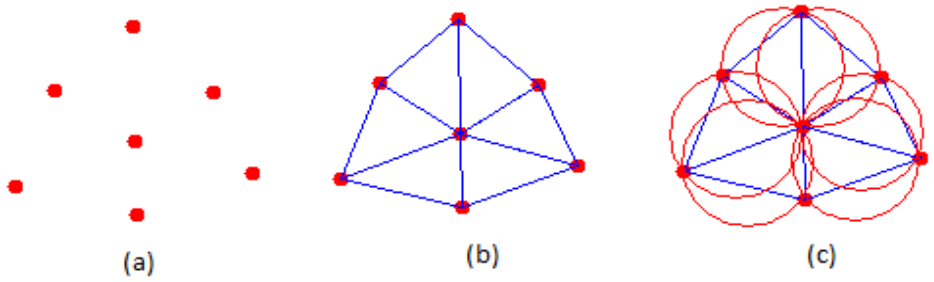

Figura 2.5: Test del Círculo satisfactorio sobre 7 vértices.

Por otro lado, es cierto que una Triangulación de Delaunay maximiza el ángulo mínimo sobre todas las triangulaciones de P, sin embargo, esto no necesariamente genera triángulos de una calidad esperada, por lo que los algoritmos de refinamiento realizan inserciones sucesivas de nuevos puntos, con la intención de que la inserción de éstos nuevos puntos destruya los triángulos de mala calidad y se generen nuevos triángulos con mejores ángulos interiores. Cuando la inserción de un nuevo punto en la triangulación se preocupa de mantener la condición Delaunay, será denominada *Inserción Delaunay.*

2.3.2. Métodos de construcción

Existen diferentes algoritmos para construir la Triangulación de Delaunay de un conjunto P de puntos dados. El algoritmo más simple permite *Delaunizar* una triangulación válida cualquiera de P, modificando sus aristas para obtener una Triangulación de Delaunay a través de un *Flip de Arista*, que es la operación de intercambiar una arista en una triangulación T reemplazando una arista interior ilegal E de T (ver Figura 2.7) por la otra diagonal del cuadrilatero Q que encierra la arista E, solamente sí Q es convexo.

Para decidir qué aristas deben modificarse se usa el criterio:

Definición 2.2 *[Arista Ilegal:] Sean t_1 y t_2 dos triángulos adyacentes que pertenecen a una triangulación, y sean p_i y p_j los vértices de la arista compartida por ambos triángulos. Se dice que la arista p_i p_j es ilegal si el círculo circunscrito de uno de los triángulos*

contiene en su interior un vértice del otro. Además, se cumple que sí t_1 *y* t_2 *forman un cuadrilátero convexo cuyos vértices no son co-circulares, entonces una y sólo una de las diagonales de dicho cuadrilátero es una Arista Ilegal.*

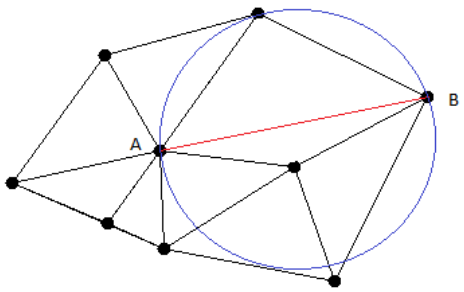

Figura 2.6: Arista AB Ilegal.

De acuerdo con la definición 2.2 en la Figura 2.6, la arista AB es una Arista Ilegal.

Los métodos de construcción de la Triangulación de Delaunay consisten en obtener primero una triangulación válida cualquiera del conjunto de puntos inicial, y eliminar todas las *Aristas Ilegales* de esa triangulación, pues por definición, la Triangulación de Delaunay no tiene Aristas Ilegales ya que al tenerlas no pasa el Test del Círculo. Para ello, se realiza Flip de Arista para el par de triángulos que contienen la Arista Ilegal, como se muestra en la Figura 2.7.

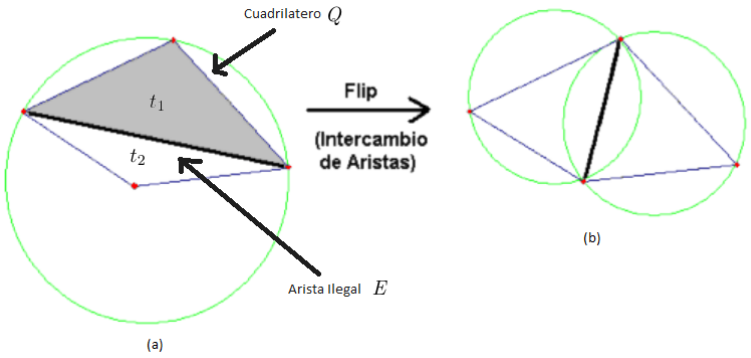

Figura 2.7: Flip de Arista

En el ejemplo de la Figura 2.7.a, el Test del Círculo falla, entonces el circuncírculo de *t1* contiene un vértice de algún triángulo adyacente *t2*, por lo tanto, la arista que comparten *t1* y *t2* es una Arista Ilegal, entonces debe ser reemplazada por la otra diagonal del cuadrilátero formado por ambos triángulos, como se muestra en la Figura 2.7.b.

A continuación se presenta el algoritmo en pseudocódigo de *Legalize Edge* y *Delaunay Triangulation* proporcionado por el profesor Pedro Rodríguez.

Algoritmo 1: LegalizeEdge(p_r, p_ip_j, T)

1.1 **if** *(p_ip_j is illegal)* **then**
1.2 Let $p_ip_jp_k$ be the triangle adjacent to $p_rp_ip_j$ along p_ip_j;
1.3 Replace p_ip_j with p_rp_k (/∗ Flip p_ip_j: ∗/);
1.4 LegalizeEdge(p_r, p_ip_k, T);
1.5 LegalizeEdge(p_r, p_kp_j, T);

En el algoritmo 1 se toma un triángulo, si éste contiene una Arista Ilegal, se busca el triángulo adyacente a ésta arista, a éstos triángulos se les aplica un Flip de Arista en la Arista Ilegal (ver Figura 2.7).b, reemplazando los antiguos triángulos por dos nuevos a los

cuales se les aplica el algoritmo 1 también.

Algoritmo 2: Delaunay_Triangulation(P)

2.1 Construct an initial triangle T that contains the set P. Let p_0, p and q the vertices of T;
2.2 **foreach** *(r=0 to n-1)* **do**
2.3 Locate the triangle $p_ip_jp_k \in T$ containing p_r.;
2.4 **if** *(p_r lies inside the triangle of $p_ip_jp_k$)* **then**
2.5 Add edges from p_r to the vertices of $p_ip_jp_k$ and subdivide $p_ip_jp_k$ into three new triangles;
2.6 LegalizeEdge(p_r, p_i p_j, T);
2.7 LegalizeEdge(p_r, p_j p_k, T);
2.8 LegalizeEdge(p_r, p_k p_i, T);
2.9 **else**
2.10 Add edges from p_r to p_i and to p_j, the third vertex of the other sharing $p_ip_k$and subdivide two triangles sharing p_ip_k into four new triangles;
2.11 LegalizeEdge(p_r, p_i p_l, T);
2.12 LegalizeEdge(p_r, p_l p_j, T);
2.13 LegalizeEdge(p_r, p_j p_k, T);
2.14 LegalizeEdge(p_r, p_k p_i, T);
2.15 Remove all edges containing vertices p and q;
2.16 return T;

El algoritmo 2 resuelve bien el problema de construir una Triangulación de Delaunay, la complejidad computacional de éste se cálcula en $\mathcal{O}(n \log n)$ para n puntos [2]. Éste algoritmo genera una triángulo inicial T que contiene la nube de puntos P, y es el primer triángulo de la triangulación T. Luego se comienza a recorrer los puntos del dominio P, si alguno de estos puntos se encuentra dentro del algún triangulo de T, éste triángulo

se subdivide en tres nuevos triángulos, uniendo los vértices del triángulo al punto que se encuentra dentro, para luego aplicar el algoritmo 1 a los tres nuevos triángulos de T. Si un punto P está en una arista de algún triángulo T, entonces se subdividen los triángulos que contengan esta arista, uniendo el punto al vértice opuesto de la arista que contiene el punto, para luego aplicar el algoritmo 1 a los nuevos triángulos, una vez terminado este proceso, se procede a eliminar las aristas que contengan el punto q o p pertenecientes al triángulo T. De ésta forma se logra construir una Triangulación de Delaunay.

2.4. Triangulacíon de Delaunay Restringida

En aplicaciones reales, con frecuencia se requiere construir una triangulación de un conjunto de vértices y aristas sobre el plano o PSLG (Planar Straight-Line Graph), manteniendo las aristas del conjunto inicial en la triangulación resultante. Una triangulación con éstas características se conoce como *Triangulación con Aristas Restringidas* o *Triangulación de Delaunay Restringida*, la cual es una malla de triángulos construida sobre un conjunto de puntos y aristas de G sobre el plano, es una Triangulación de Delaunay Restringida, sí para cada triángulo t de la malla se cumple que el circuncírculo de t no contiene un vértice de G en su interior, o el circuncírculo de t posee un vértice de G en su interior pero éste se encuentra en el lado opuesto de una arista restringida de t, actuando dicha arista como una muralla que bloquea la visión al resto de la malla, y por ende, a cualquier punto que esté en el interior del circuncírculo.

En el problema de triangular un polígono (que es un caso particular de PSLG), los bordes de éste también se consideran aristas restringidas. Es posible construir una Triangulación de Delaunay de un PSLG, relajando la condición de Delaunay, que exige que el circuncírculo de cualquier triángulo de la malla no contenga otro vértice en su interior [7].

En la Figura 2.8 se muestran algunos ejemplos de éste tipo de triangulación.

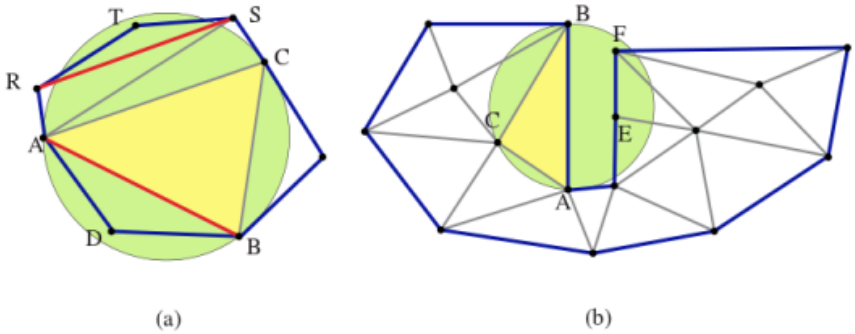

Figura 2.8: Triangulación de Delaunay Restringida [7].

Triangulaciones Delaunay Restringidas en ambos casos Figura 2.8.a y Figura 2.8.b, los segmentos rojos, azules y los vértices negros son elementos iniciales. Las aristas restringidas RS y AB en la Figura 2.8.a permiten ignorar los vértices T y D respectivamente. En la Figura 2.8.b el borde AB de la malla permite ignorar los vértices E y F.

2.5. Triangulación Delaunay Conforme

Una *Triangulación Delaunay Conforme* es una Triangulación Delaunay en la cual sus aristas restringidas están presentes en la triangulación como la unión de uno o más segmentos de recta.

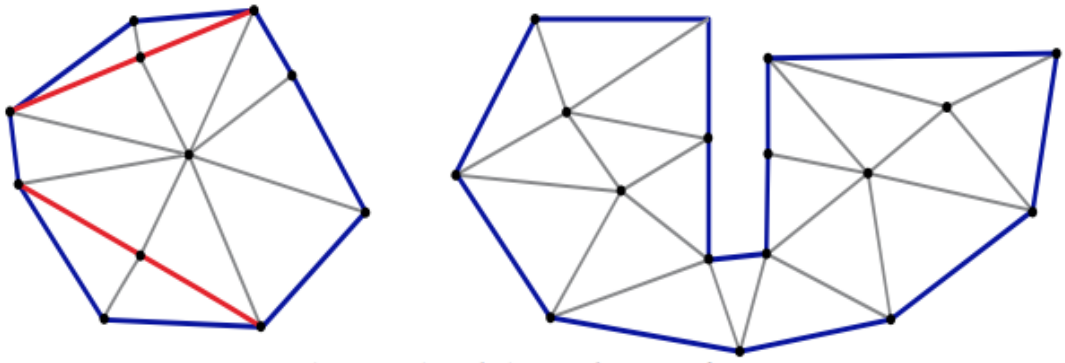

Figura 2.9: Triangulación Delaunay Conforme [7].

Las aristas restringidas y bordes de malla son los mismos que en la Figura 2.8, salvo que en este caso se encuentra dividida en aristas de menor tamaño. Las aristas RS y AB de la Figura 2.8.a y la arista AB de la Figura 2.8.b.

2.6. Detección de bordes algoritmo Canny Edge Detector

En el área de procesamiento de imágenes, la detección de los bordes de una imagen es de suma importancia y utilidad, pues facilita muchas tareas, entre ellas, el reconocimiento de objetos, la segmentación de regiones, entre otras. Se han desarrollado una variedad de algoritmos que ayudan a solucionar este inconveniente. El algoritmo de Canny es usado para detectar todos los bordes existentes en una imagen [11]. Este algoritmo es considerado como uno de los mejores métodos de detección de contornos y se basa en Máscaras de Convolución (ver Ecuación 2.5), las cuales determinan los coeficientes a aplicar sobre los puntos de una determinada área para realizar la operación matemática que consiste en una suma *pesada* de pixeles vecinos del pixel fuente. Esta operación se realiza de forma recursiva sobre todos los pixeles de la imagen. Las dimensiones de la matriz son normalmente impares e iguales, de forma que se pueda determinar un centro de una matriz cuadrada.

Las Máscaras de Convolución son el eje central de el *Filtro Gaussiano* y éste es el primer paso del algoritmo Canny Edge Detector. Éste Filtro entrega una imagen suavizada como resultado, como se muestra en la Figura 2.10.

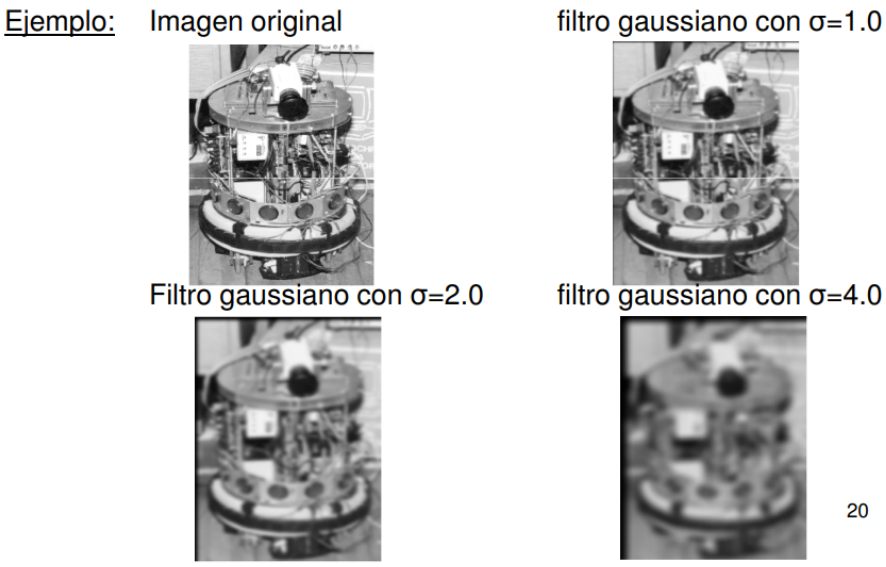

Figura 2.10: Filtro Gaussiano.

En 1D los pesos de la Máscara de Convolución asignados toman la forma de la Campana de Gauss (ver Figura 2.11.a) dada por la función 2.2, donde σ^2 es la varianza y representa el nivel de suavidad que se quiere lograr.

$$f(x) = e^{\frac{-x^2}{\sigma^2}} \tag{2.2}$$

Se debe tener en cuenta que a mayor varianza, la Campana de Gauss enancha por lo tanto se logra un mayor suavizado, de lo contrario, si la varianza es menor, la Campana de Gauss es mas estrecha, por lo tanto se logra un suavizado más tenue.

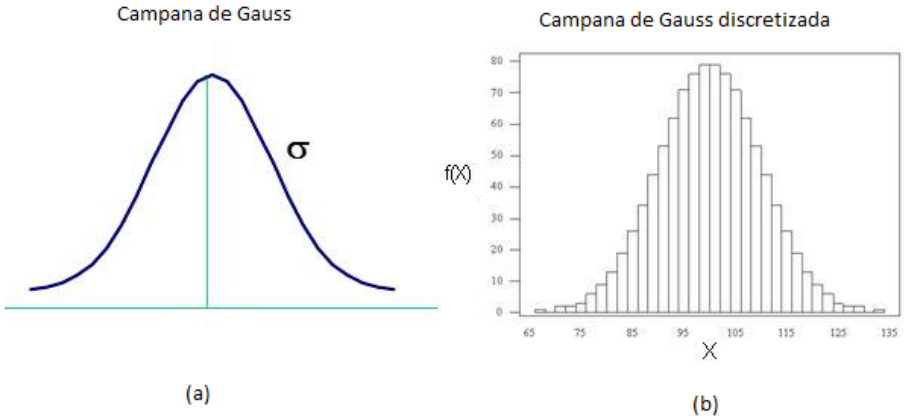

Figura 2.11: Campana de Gauss y Campana de Gauss discreta.

Para calcular los peso de la Mascara de Convolución 1D se debe calcular la función y discretizar los valores de la Campana de Gauss (ver Figura 2.11.b), o se puede usar Triángulo de Pascal 2.12, el cual forma la discretización de la Campana de Gauss, y se calcula comenzando con una tría de números ordenados de forma piramidal. Para agregar más filas, se necesita sumar los dos números superiores a los espacios que se desea agregar, conservando en los extremos un 1.

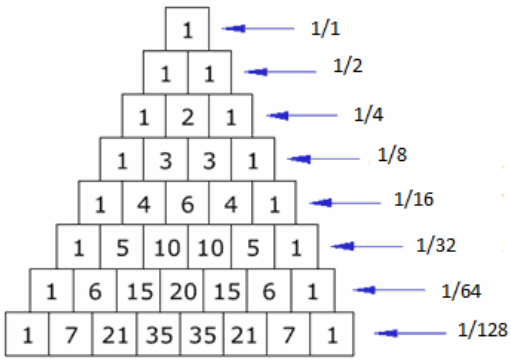

Figura 2.12: Triangulo de Pascal.

Por lo tanto una Mascara de Convolución de 3 en 1D es:

$$\frac{1}{4} = \begin{pmatrix} 1 & 2 & 1 \end{pmatrix} \tag{2.3}$$

Donde la matriz esta dada por el Triángulo de Pascal y la fracción es $\frac{\sigma}{4}$. El denominador de la fracción es calculada sumando los pesos de la fila correspondiente.

Para calcular una Máscara de Convolución en 2D se necesita multiplicar una de las filas del Triángulo de Pascal de manera horizontal y vertical como se muestra en la Ecuación 2.4.

$$\frac{1}{4} = \begin{pmatrix} 1 & 2 & 1 \end{pmatrix} X \frac{1}{4} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} = \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \tag{2.4}$$

A continuación se presenta un ejemplo de Máscara de Convolución 5X5 para Filtro Gaussiano con $\sigma = 1{,}0$.

$$\frac{1}{16} \begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix} \tag{2.5}$$

2.6.1. Pasos del algoritmo de Canny Edge Detector

El algoritmo de Canny consiste en tres grandes pasos [6]:

- ***Obtención de Gradiente:*** En éste paso se calcula la magnitud y orientación del vector gradiente en cada pixel.
- ***Supresión No Máxima:*** En éste paso se logra el adelgazamiento del ancho de los bordes, obtenidos con el gradiente, hasta lograr bordes de un pixel de ancho.

- ***Hystérisis de Umbral:*** En éste paso se aplica una función de Hystéresis basada en dos umbrales; con este proceso se pretende reducir la posibilidad de aparición de contornos falsos.

Algoritmo 3: Canny Edge Detector

3.1 **CannyEdgeDetector**
  inputs : $S \leftarrow G_\sigma * I$
  inputs : $[S_x S_y] \leftarrow \bigtriangledown(S)$
3.2 **foreach** *(Cada Pixel (i,j))* **do**
3.3 $|\bigtriangledown S(i,j)| \leftarrow \sqrt{S_x^2(i,j) + S_y^2(i,j)}$;
3.4 $S\Phi(i,j) \leftarrow ARCTAN\frac{S_x(i,j)}{S_y(i,j)}$;
3.5 Supresión();
3.6 Hysteresis();
3.7 **return** $|\bigtriangledown S|, S\phi$;

En el algoritmo 3.7 de complejidad computacional $\mathcal{O}(n)$ se muestra la generalización del algoritmo Canny Edge Detector, en el cual lo primero es la *Obtención de la Gradiente*, para ésto se realiza la aplicación de un Filtro Gaussiano a la imagen original con el objetivo de suavizarla y tratar de eliminar el posible ruido existente, entendido ruido como variación aleatoria de brillo o color en la imagen que no se corresponde con el objeto real capturado en ésta. Una vez que se suaviza la imagen, para cada pixel se obtiene la magnitud y módulo (orientación) del gradiente o imagen en escala de grises.

Algoritmo 4: Supresión de NO-Máximos

4.1 SupresionNoMaxima

inputs : $d_i \leftarrow [0°, 45°, 90°, 135°]$

4.2 **foreach** *(Cada Pixel (i,j))* **do**

4.3 $\widehat{d}_k \leftarrow ARGMÍN_{d_k} |d_k - S_\phi(i,j)|$;

4.4 **if** $|\bigtriangledown S(i,j)|$ *es menor que uno de sus dos vecinos en la dirección* $\widehat{d}_k$ **then**

4.5 $S_e(i,j) \leftarrow 0$;

4.6 **else**

4.7 $S_e(i,j) \leftarrow |\bigtriangledown S(i,j)|$;

4.8 **return** S_e;

En el algoritmo 4 de complejidad computacional $\mathcal{O}(n)$ se aplica la *Supresión No-Máximos* al resultado del gradiente, las dos imágenes generadas en el paso anterior sirven de entrada para generar una imagen con los bordes adelgazados. En el procedimiento se consideran cuatro direcciones identificadas por las orientaciones de 0º, 45º, 90º y 135º con respecto al eje horizontal. Para cada pixel se encuentra la dirección que mejor se aproxime a la dirección del ángulo de gradiente.

Algoritmo 5: Hysteresis

```
5.1  Hysteresis
     inputs : p ← Umbral de nivel alto
     inputs : q ← Umbral de nivel bajo
     inputs : L_visitados ← φ
     inputs : L_Bordes ← φ
5.2  foreach (Cada Pixel (i,j)) do
5.3      if (i,j)∄L_visitados y S_e(i,j) > p then
5.4          (k,l) ← (i,j);
5.5          B_ij ← φ;
5.6          while S_e(k,l) > q do
5.7              B_ij ← B_ij ∪ (k,l);
5.8              L_visitantes ← L_visitantes ∪ (k,l);
5.9              (k,l) ← Pixel adyacente a (k,l) en dirección perpendicular a
                  S_φ(k,l);
5.10         L_bordes ← ∪B_ij;
5.11 return L_bordes;
```

En el algoritmo 5 de complejidad computacional $\mathcal{O}(n^2)$ se aplica la *Hystérisis de Umbral* a la Supresión No Máxima, la imagen del paso anterior suele contener máximos locales creados por el ruido, para eliminar dicho ruido se usa la Hystéresis del Umbral. Se toma la imagen del paso anterior, tomando la orientación de los puntos de borde de la imagen y tomar dos umbrales, el primero más pequeño que el segundo. Para cada punto de la imagen se debe localizar el siguiente punto de borde no explorado que sea mayor al segundo umbral, y mejor que el primero. A partir de dicho punto seguir las cadenas de máximos locales.

2.6.2. Criterios del algoritmo de Canny Edge Detector

En 1986 John Canny identificó tres valores característicos importantes que un detector de bordes debe tener:

- *Tasa de error* - El detector debe devolver sólo los bordes, y debe encontrarlos a todos.
- *Localización* - La distancia entre los pixeles del borde encontrado y el real debe ser el mínimo posible.
- *Respuesta* - El detector sólo debe entregar una única respuesta para cada borde.

Capítulo 3

Ambiente de ingeniería de software

3.1. Metodología de desarrollo

Debido a la naturaleza particular del problema abordado se opta por desarrollar una metodología de tipo iterativo-incremental. En un desarrollo iterativo e incremental el proyecto se planifica en diverso bloques temporales llamado iteraciones, como se muestra en la Figura 3.1. En todas las iteraciones se repite un proceso similar para proporcionar un resultado completo sobre el producto final, de manera que el cliente pueda obtener los beneficios del proyecto de forma incremental. Cada iteración genera una evolución del producto a partir de los resultados generados en las iteraciones anteriores, añadiendo nuevos requisitos u objetivos[1].

[1]https://proyectosagiles.org

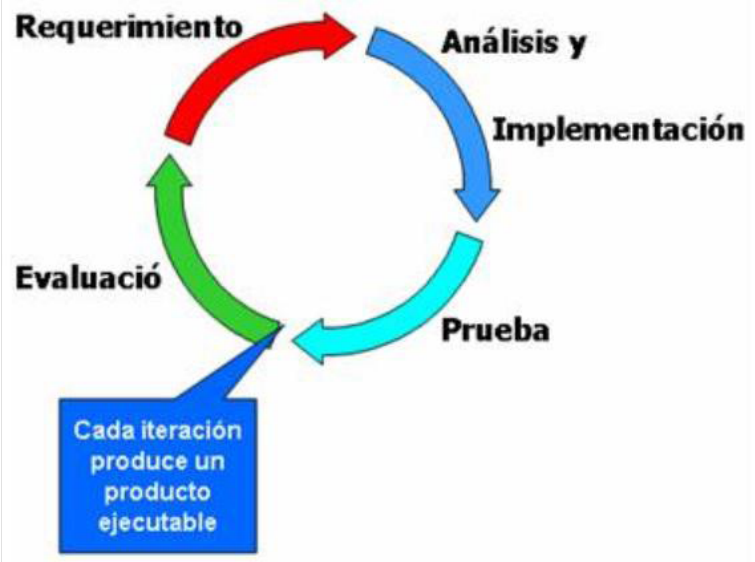

Figura 3.1: Metodología iterativa incremental.

Aplicando esta metodología, se piensa el desarrollo del sistema en tres iteraciones. Los objetivos y requerimientos específicos de cada iteración son definidos y modificados en la medida en que el tema es profundizado ya que se cuenta con una alta participación de todas las partes involucradas.

3.1.1. Beneficios de la metodología

- El cliente puede obtener resultados importantes ya desde las primeras iteraciones.
- Permite conocer el progreso real del proyecto desde las primeras iteraciones y extrapolar si su finalización es viable en la fecha prevista. El cliente puede decidir re priorizar los requisitos del proyecto, añadir nuevos equipos, cancelarlo, etc.
- El cliente puede comenzar el proyecto con requisitos de alto nivel, quizás no del todo completos, de manera que se vayan refinando en sucesivas iteraciones.
- El proyecto considera este tipo de método ya que el usuario presenta un alto grado de participación dentro del desarrollo, lo cual hace que éste sea más eficiente. Además

se utilizan iteraciones ya que se pueden hacer cambios de versiones, corrección de errores presentados, etc.

3.2. Técnicas y notación

A continuación se presentan los elementos técnicos que son utilizados en este documento para representar y acercar al lector a una mayor comprensión del sistema y sus especificaciones técnicas.

3.2.1. Modelo de notación de diagrama de clases

Los diagramas de clases son diagramas de estructura estática que muestran las clases del sistema y sus interrelaciones (incluyendo herencia, agregación, asociación, etc.). Los diagramas de clase son el pilar básico del modelado con UML, siendo utilizados tanto para mostrar lo que el sistema puede hacer (análisis), como para mostrar cómo puede ser construido (diseño).

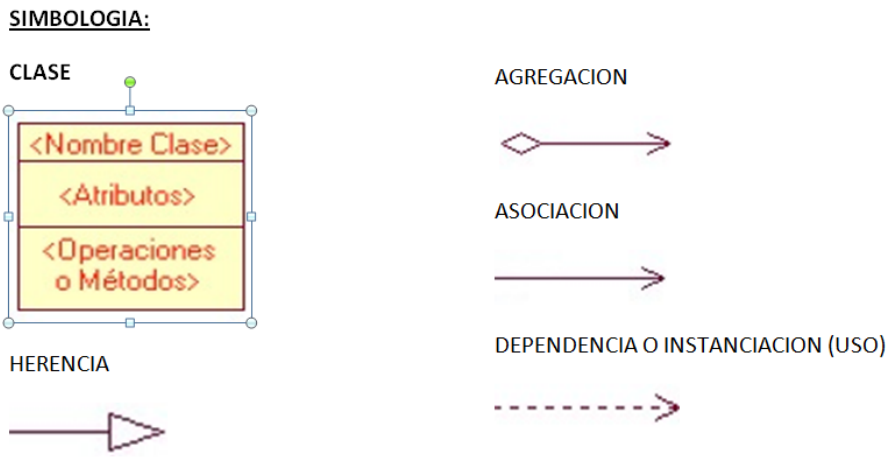

Figura 3.2: Simbología diagrama de clases.

- *Clase:* Es la unidad básica que encapsula toda la información de un objeto (un objeto es una instancia de una clase). Una clase es representada por un rectángulo que posee tres divisiones [9]:

  - *Nombre:* Contiene el nombre de la clase.
  - *Atributos:* Contiene los atributos (o variables de instancia) que caracterizan a la clase.
  - *Operaciones o Métodos:* Contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno.

- *Relaciones entre clases:* Ya definido el concepto de clase, es necesario explicar como se pueden interrelacionar dos o más clases:

  - *Herencia:* Indica que una subclase hereda los métodos y atributos especificados por una super-clase, por ende la sub-clase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la super-clase [9]:
  - *Composición:* Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye (el Objeto base se construye a partir del objeto incluido, es decir, es *parte/todo*).
  - *Agregación:* Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye (el objeto base utiliza al incluido para su funcionamiento).
  - *Asociación:* Permite asociar objetos que colaboran entre sí. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.
  - *Dependencia o instancia:* En la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase).

3.2.2. Modelo de notación de caso de uso

Diagrama utilizado para representar los puntos de interacción entre el sistema y sus respectivos usuarios. Utiliza la notación UML y su simbología se representa en la Figura 3.3 [2].

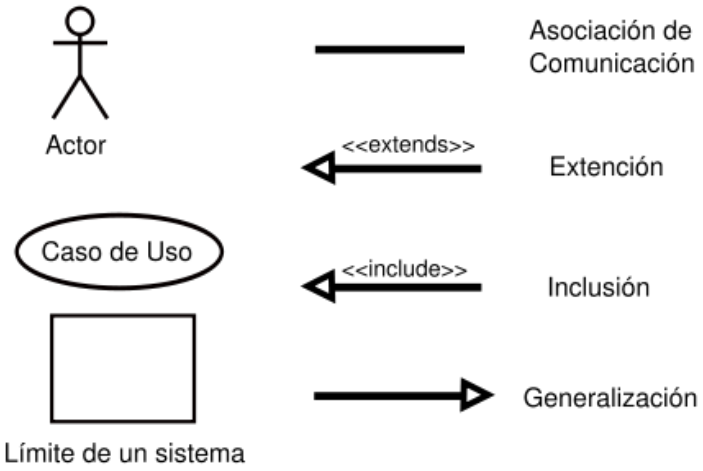

Figura 3.3: Simbología diagrama de casos de uso.

- *Usuario:* Cualquier sistema externo que interactúe con el nuestro (persona,maquina,..).
- *Clase:* Es una operación/tarea específica que se realiza tras una orden de algún agente externo, sea desde una petición de un actor o bien desde la invocación desde otro caso de uso. Se deben escribir los detalles aparte en un texto explicativo.
- *Asociación:* Marca una interacción entre dos elementos.
- *Include:* Indica que un caso de uso engloba la acción del otro.
- *Extend:* Indica que un caso de uso puede llevar a usar el caso extendido.
- *Generalización:* Indica que un caso de uso es un caso particular de uno más general (herencia).

[2]http://www.monografias.com/trabajos67/diagramas-uml/diagramas-uml2.shtml

3.2.3. Método de especificación de los casos de uso

A continuación se expone la plantilla que se usa para documentar cada caso de uso.

Nombre del caso de uso	
Campo caso de uso	***Descripción***
Actores	
Descripción del caso de uso	
Pre-Condición	
Post-Condición	
Asociación de caso de uso	
Resumen de entradas	
Resumen de salidas	
Indice de dificultad	
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
Curso alternativo de eventos	
Nota de Caso de uso	

Tabla 3.1: Tabla diagrama caso de uso.

- *Nombre de caso de uso:* Frase verbal activa que describe una tarea concreta.
- *Actores:* El actor inicia el caso de uso y todos los usuarios que participan en el mismo.
- *Descripción de caso de uso:* Breve descripción de sucesos de los resultados de terminación más probables.
- *Pre-Condición:* Condiciones que deben cumplirse previamente antes de ejecutar el caso de uso.
- *Post-Condición:* Condición en la que queda el software luego de la ejecución del caso de uso.
- *Asociaciones de casos de uso:* Casos de uso que interactúan con el que esta siendo definido.
- *Resumen de entrada:* Datos o archivos que participan el la ejecución del caso de uso.

- *Resumen de salida:* Datos o archivos que entrega el caso de uso luego de su ejecución.
- *Índice de dificultad:* Dificultad para implementar el caso de uso.
- *Curso normal de eventos:* Describe paso a paso la ejecución esperada del caso de uso.
- *Curso alternativo de eventos:* Describe eventos no satisfactorios en la ejecución de el caso de uso.
- *Nota caso de uso:* Nota de algo importante que no pueda ser descrito en los ítems anteriores.

3.2.4. Modelo de notación de diagramas de flujo de datos

Es una representación gráfica del flujo de datos a través de un sistema de información. Un diagrama de flujo de datos también se puede utilizar para la visualización de procesamiento de datos (diseño estructurado).

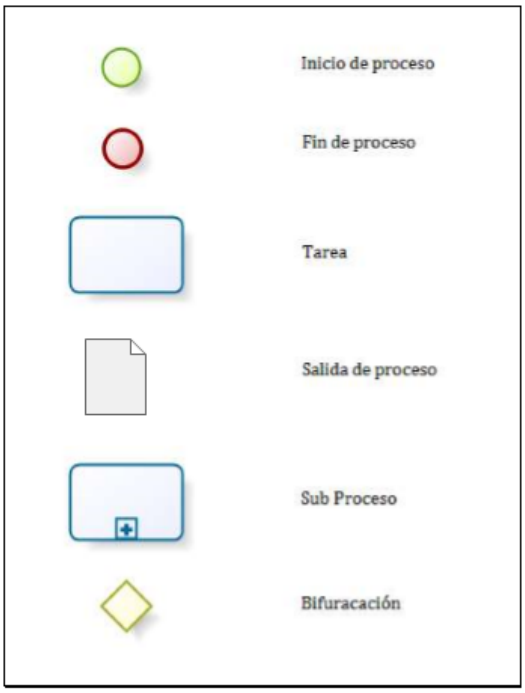

Figura 3.4: Simbología diagrama de flujos.

- *Inicio del proceso:* Marca el inicio del sistema.
- *Fin del proceso:* Marca el fin del sistema.
- *Tarea:* Representa los procesos o cálculos matemáticos que se llevan a cabo.
- *Salidas del proceso:* Representan los resultados que arrojan el proceso o alguna tarea.
- *Sub-procesos:* Con él se simbolizan los procesos que se ejecutan en paralelo a éste.
- *Bifurcación:* Con él se simbolizan los ciclos de condición o de iteración.

3.3. Estándares de documentación

- *Especificación de requerimientos:* Se modifica y adapta IEEE Software Requirements Specifications STD 830-1998 para ajustarlo al caso particular de esta memoria.
- *Bibliografía:* Se utiliza la norma APA.
- *Casos de uso:* Se usa como base ÏBM InfoSphere Master Data Management, Versión 11".

Capítulo 4

Análisis del sistema

4.1. Requerimientos del sistema

En las Tablas (4.1, 4.2) se expone la funcionalidad del software que se pretende generar.

4.1.1. Requerimientos Funcionales

En la siguiente tabla se listan de manera ordenada los Requerimientos Funcionales que debe cumplir el sistema.

ID	Nombre	Descripción
RF 01	Cargar imagen	El sistema debe poder cargar una imagen en formato jpg, png.
RF 02	Filtrar imagen	Una vez se encuentra cargada la imagen, el sistema debe permitir aplicar el filtro Canny Edge Detector mostrando el resultado en pantalla.
RF 03	Detección de bordes	El sistema debe detectar los bordes de una imagen (jpg, png) de una forma clara e inequívoca generando una nube de puntos que representan el contorno de la figura en la imagen.
RF 04	Guardar nube de puntos	Una vez generada la nube de puntos inicial el sistema debe permitir guardar en formato *.pto*.
RF 05	Cargar nube de puntos	El sistema debe permitir cargar una nube de puntos anteriormente guardada en formato *.pto*.
RF 06	Triangulación	El sistema debe triangular una nube de puntos conservando el contorno de la figura inicial en la imagen.
RF 07	Conservar espacios intermedios	El software debe permitir espacios intermedios en la figura, tal espacio debe ser conservados en la triangulación.
RF 08	Imagen de salida	El programa debe permitir guardar una copia de la imagen con los filtros aplicados.
RF 09	Informe de vértices	El sistema debe permitir generar reportes de todos los vértices generados en sentido contrario a las manecillas de reloj.
RF 10	Guardar malla en formato m2d	El sistema debe permitir guardar la malla geométrica generada en un archivo de extensión *.m2d*.
RF 11	Cargar malla en formato m2d	El sistema debe cargar un archivo de extensión *.m2d*.

Tabla 4.1: Requerimientos Funcionales RF01 - RF11.

4.1.2. Requerimientos No Funcionales del sistema

En la siguiente tabla se listan de manera ordenada los Requerimientos No Funcionales que debe cumplir el sistema.

ID	Nombre	Descripcion
RNF 01	Escalabilidad	El producto final debe facilitar iteraciones futuras para ampliar la funcionalidad de este.
RNF 02	Escalabilidad 3D	El sistema debe ser construido e implantado de tal manera que facilite implementar triangulación en 3D.
RNF 03	Código legible	Se debe estructurar el código de una manera consistente y predecible.
RNF 04	Documentación	El código debe estar debidamente documentado para su fácil comprensión en futuras iteraciones.
RNF 05	Aprendizaje	El sistema debe ser de fácil uso.
RNF 06	Usabilidad	El sistema debe ser intuitivo.
RNF 07	Facilidad de pruebas	La identificación de errores debe ser fácil en la etapa de pruebas y posteriores ejecuciones.

Tabla 4.2: Requerimientos No Funcionales RNF 01 - RNF 07.

4.2. Casos de uso

Esta sección se encarga de describir los casos de uso utilizados en el sistema.

"La parte más difícil de construir un sistema es precisamente saber qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los requerimientos técnicos detallados, incluyendo todas las interfaces con gente, máquinas, y otros sistemas."[3].

4.2.1. Diagrama casos de uso

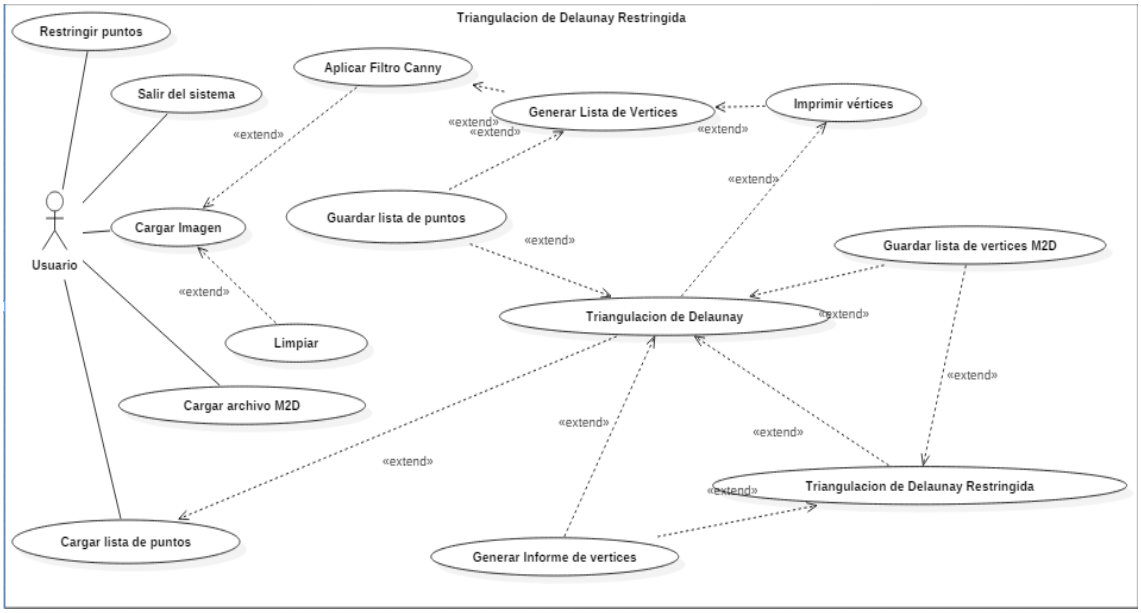

Figura 4.1: Diagrama de casos de uso.

4.2.2. Especificación de los casos de uso

En las siguientes tablas se detallan los casos de uso de manera ordenada.

1-Salir del sistema

1-Salir del sistema	
Campo caso de uso	***Descripción***
Actores	Usuario
Descripción del caso de uso	(File->close) El software se cierra completamente, si los datos no fueron guardados se perderán.
Pre-Condición	Este caso de uso no presenta Pre-Condición.
Post-Condición	Este caso de uso no presenta Post-Condición.
Asociación de caso de uso	
Resumen de entradas	
Resumen de salidas	
Indice de Dificultad	Baja.
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona File->Close.	2.- Se cierra el software.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.3: Diagrama caso de uso - Salir del sistema.

2-Cargar imagen

2-Cargar imagen	
Campo caso de uso	***Descripción***
Actores	Usuario
Descripción del caso de uso	(File->Load Image) El software abre una ventana para buscar un archivo con extension .jpg o .png para cargar una imagen.
Pre-Condición	Este caso de uso no presenta Pre-Condición.
Post-Condición	La imagen seleccionada es mostrada en pantalla.
Asociación de caso de uso	«Extend» hacia Aplicar filtros «Extend» hacia Limpiar imagen.
Resumen de entradas	
Resumen de salidas	Mostrar imagen en el panel.
Indice de Dificultad	Baja.
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona File->Load Image.	2.- Se abre una ventana (jfilechooser) para seleccionar una imagen (jpg, png).
3.- Selecciona una imagen a tratar	
4.- Acepta la imagen seleccionada	5.- Despliega la imagen seleccionada en el panel.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.4: Diagrama caso de uso - Cargar imagen.

3-Cargar lista de puntos

3-Cargar lista de puntos	
Campo caso de uso	***Descripción***
Actores	Usuario.
Descripción del caso de uso	(File->load Point) El software abre una ventana para buscar un archivo con extension *.pto* el cual contiene una lista de puntos, los cuales se imprimen en el panel.
Pre-Condición	
Post-Condición	El archivo *.pto* seleccionado se muestra en el panel.
Asociación de caso de uso	«Extend» hacia Tiangulación de Delaunay.
Resumen de entradas	Archivo *NombreImagen.pto*4.9.1
Resumen de salidas	Imprime en el panel los puntos de la lista.
Indice de Dificultad	Medio
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona File->Load Point.	2.- Se abre una ventana (jfilechooser) para seleccionar un archivo con extension *.pto*.
3.- Selecciona un archivo *.pto* a mostrar.	
4.- Acepta el archivo seleccionado.	5.- Despliega el archivo seleccionado en el panel.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.5: Diagrama caso de uso - Cargar lista de puntos.

4-Limpiar

4-Limpiar	
Campo caso de uso	***Descripción***
Actores	Usuario.
Descripción del caso de uso	El software reinicia el programa, el cual podrá ser usado inmediatamente.
Pre-Condición	
Post-Condición	El panel queda totalmente vacío.
Asociación de caso de uso	«Extend» de Cargar imagen.
Resumen de entradas	
Resumen de salidas	
Indice de Dificultad	Baja
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona File->Limpiar.	2.- Deja el panel vacío.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.6: Diagrama caso de uso - Limpiar imagen.

5-Aplicar filtros

5-Aplicar filtros	
Campo caso de uso	***Descripción***
Actores	Usuario.
Descripción del caso de uso	El software toma la imagen en el panel y aplica el algoritmo Canny Edge Detector, genera una nueva imagen la cual es guardada y desplegada en el panel remplazando al la imagen inicial.
Pre-Condición	El panel debe tener una imagen sin tratar.
Post-Condición	El panel debe mostrar la imagen tratada.
Asociación de caso de uso	«Extend» de Cargar imagen «Extend» hacia Generar lista de vértices.
Resumen de entradas	Imagen original a tratar
Resumen de salidas	Imagen tratada que resalta el contorno con pixeles blancos
Indice de Dificultad	Alta
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona Filtrar.	2.- Se guarda la imagen resultante.
	3.- Se muestra la imagen tratada.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.7: Diagrama caso de uso - Aplicar filtros.

6-Generar lista de vértices

6-Generar lista de vértices	
Campo caso de uso	***Descripción***
Actores	Usuario.
Descripción del caso de uso	El software debe tomar la imagen tratada en el panel y generar los vértices que representan el contorno de la imagen.
Pre-Condición	El panel debe tener una imagen tratada.
Post-Condición	El caso de uso no presenta Post-Condición.
Asociación de caso de uso	«Extend» de Aplicar filtros «Extend» hacia Imprimir informe de vértices. «Extend» hacia Imprimir vértices.
Resumen de entradas	Imagen tratada
Resumen de salidas	instancia con lista de vértices del contorno de la imagen
Indice de Dificultad	Alta
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona Generar Vértices.	2.- Se genera una lista de vértices dentro del software que representa el contorno del borde.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.8: Diagrama caso de uso - Generar lista de vértices.

7-Restringir puntos

7-Restringir puntos	
Campo caso de uso	***Descripción***
Actores	Usuario
Descripción del caso de uso	(Utilidad-> Restringir puntos) El software debe tomar la lista de vértices generado y los debe poder restringir en base a un dígito ingresado por el usuario, entre cada punto impreso, se debe permitir eliminar la cantidad de puntos indicados.
Pre-Condición	No haber impreso ya la nube de puntos.
Post-Condición	El caso de uso no presenta Post-Condición.
Asociación de caso de uso	Triangulación de Delaunay Restringida
Resumen de entradas	Lista de vértices
Resumen de salidas	
Indice de Dificultad	Media.
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.-Se presiona Utilidad->Restringir puntos	2.- Se restringe la cantidad de puntos .
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.9: Diagrama casos de uso - Restringir puntos.

8-Imprimir vértices

8-Imprimir vértices	
Campo caso de uso	***Descripción***
Actores	Usuario.
Descripción del caso de uso	El software debe permitir tomar la lista de vértices anteriormente generadas y debe poder imprimirlas en el panel.
Pre-Condición	Se debe haber generado la lista de vértices antes.
Post-Condición	El panel muestra los vértices del borde y huecos que representan la figura en la imagen inicial.
Asociación de caso de uso	«Extend» de Generar lista de vértices. «Extend» hacia Triangulación de Delaunay Restringida.
Resumen de entradas	Lista de vértices
Resumen de salidas	
Indice de Dificultad	medio
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona Edit->Imprimir vértices.	2.- Se muestra la lista de vértices generados.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.10: Diagrama caso de uso - Imprimir vértices.

9-Guardar lista de vértices m2d

9-Guardar lista de vértices m2d	
Campo caso de uso	***Descripción***
Actores	Usuario
Descripción del caso de uso	(File-> Sabe m2d) El software debe tomar una malla triangulada y la debe poder guardar en un archivo de extension m2d.
Pre-Condición	Se debe haber generado una Triangulación Delaunay previamente.
Post-Condición	El caso de uso no presenta Post-Condición.
Asociación de caso de uso	«Extend» de Triangulación de Delaunay «Extend» de Restringir Triangulación de Delaunay.
Resumen de entradas	
Resumen de salidas	Archivo *NombreImagen.M2D* 4.9.2.
Indice de Dificultad	Alta
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona File->Save M2D.	2.- Se guarda la malla geométrica anteriormente generada.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.11: Diagrama caso de uso - Guardar lista de vértices m2d.

10-Cargar lista de vértices m2d

10-Cargar lista de vértices m2d	
Campo caso de uso	***Descripción***
Actores	Usuario
Descripción del caso de uso	(File-> Load m2d) El software debe tomar una malla triangulada y debe permitir guardar en extension m2d e imprimir en el panel.
Pre-Condición	Se debe tener un archivo M2D
Post-Condición	El caso de uso no presenta Post-Condición.
Asociación de caso de uso	
Resumen de entradas	Archivo *NombreImagen.m2d* 4.9.2.
Resumen de salidas	
Indice de Dificultad	Alta
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona File->Save m2d.	2.- Se guarda la malla geométrica anteriormente generada.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.12: Diagrama caso de uso - Cargar lista de vértices m2d.

11-Guardar lista de puntos

11-Guardar lista de puntos	
Campo caso de uso	***Descripción***
Actores	Usuario
Descripción del caso de uso	(File-> Guardar puntos)El software debe tomar una lista de puntos generada y debe permitir guardar en una extencion.pto.
Pre-Condición	El caso de uso no presenta pre-Condición.
Post-Condición	El caso de uso no presenta Post-Condición.
Asociación de caso de uso	«Extend» de Restringir Triangulación de Delaunay. «Extend» de Generar lista de vértices.
Resumen de entradas	
Resumen de salidas	Archivo *NombreImagen.pto* 4.9.1.
Indice de Dificultad	Alta
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona File->Save point.	2.- Se guarda la lista de puntos anteriormente generada.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.13: Diagrama caso de uso - Guardar lista de puntos.

12-Triangulación de Delaunay

12-Triangulación de Delaunay	
Campo Caso de uso	***Descripción***
Actores	Usuario
Descripción del caso de uso	El software debe permitir triangular los vértices ya impresos en el panel.
Pre-Condición	Se debe haber generado una lista de vértices previamente.
Post-Condición	El panel muestra una triangulación de Delaunay Conforme.
Asociación de caso de uso	«Extend» de Cargar lista de vértices M2D «Extend» de Imprimir vértices «Extend» hacia Guardar lista de vértices M2D «Extend» hacia Restringir Triangulación de Delaunay
Resumen de entradas	Imagen tratada, lista de Vértices.
Resumen de salidas	Lista de vértices triangularizados.
Indice de Dificultad	Alta
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona botón triangulación Delaunay.	2.- Se debe permitir generar una Triangulación de Delaunay Restringida en la lista de vértices anteriormente generada.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.14: Diagrama caso de uso - Triangulación de Delaunay.

13-Restringir Triangulación de Delaunay

13- Restringir Triangulación de Delaunay	
Campo caso de uso	***Descripción***
Actores	Usuario
Descripción del caso de uso	El software debe permitir restringir los triángulos en la tringulación ya hecha.
Pre-Condición	Se debe haber generado una lista de vértices previamente.
Post-Condición	El panel muestra una Triangulación de Delaunay Restringida que conserva los bordes de la imagen inicial.
Asociación de caso de uso	«Extend» de Cargar lista de vértices M2D «Extend» de Triangulización de Delaunay «Extend» hacia Guardar lista de vertices M2D.
Resumen de entradas	Triangularización de Delaunay
Resumen de salidas	Triangularización Restringida
Indice de Dificultad	Alta
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
1.- Presiona Delaunay->triangulación.	2.- Se genera una Triangulacion de Delaunay Restringida en la lista de vértices anteriormente generada.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.15: Diagrama caso de uso - Restringir Triangulación de Delaunay.

14-Imprimir informe de vértices

14-Imprimir informe de vértices	
Campo caso de uso	***Descripción***
Actores	Usuario
Descripción del caso de uso	El software debe permitir tomar la lista de vértices de una Triangulación de Delaunay o una Triangulación de Delaunay Restringida y debe imprimirlas en un archivo *.txt*
Pre-Condición	Se debe haber generado una Triangulación de Delaunay o una Triangulación de Delaunay Restringida antes.
Post-Condición	El Caso de uso no presenta Post-Condición.
Asociación de caso de uso	«Extend» Triangulación de Delaunay Triangulación de Delaunay Restringida
Resumen de entradas	Lista de vértices
Resumen de salidas	Archivo con la información de los vértices presentes en la triangulación de extención .txt
Indice de Dificultad	Media.
Curso normal de eventos	
Acción del Actor	***Respuesta del sistema***
	1.- Se genera el informe al generar la lista de vértices en el paso anterior.
Curso alternativo de eventos	
Nota de caso de uso	

Tabla 4.16: Diagrama caso de uso - Imprimir informe de vértices.

4.2.3. Tabla de dificultad de casos de uso.

Tabla de dificultad de casos de uso		
Nivel de dificultad	***cantidad***	***Detalle***
Alta	4	Triangulación de Delaunay , Triangulación de Delaunay Restringida, Aplicar filtros, Generar lista de vértices
Media	5	Guardar lista de vértices m2d, Imprimir informe m2d, Cargar lista m2d, Imprimir vértices, Reducir puntos
Baja	5	Limpiar imagen, Cargar imagen, Salir del sistema, Cargar puntos, Guardar puntos.

Tabla 4.17: Tabla de dificultad caso de uso.

4.3. Diagrama de flujos de datos

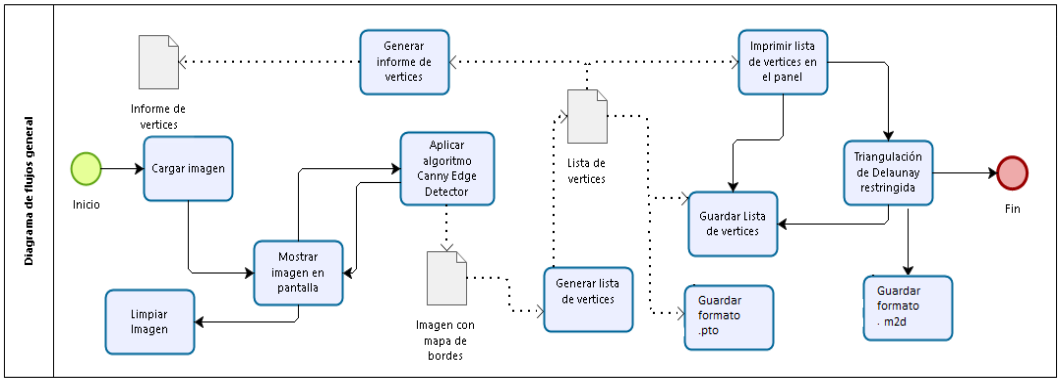

Figura 4.2: Diagrama de flujo general.

4.4. Diagrama de flujo Canny Edge Detector

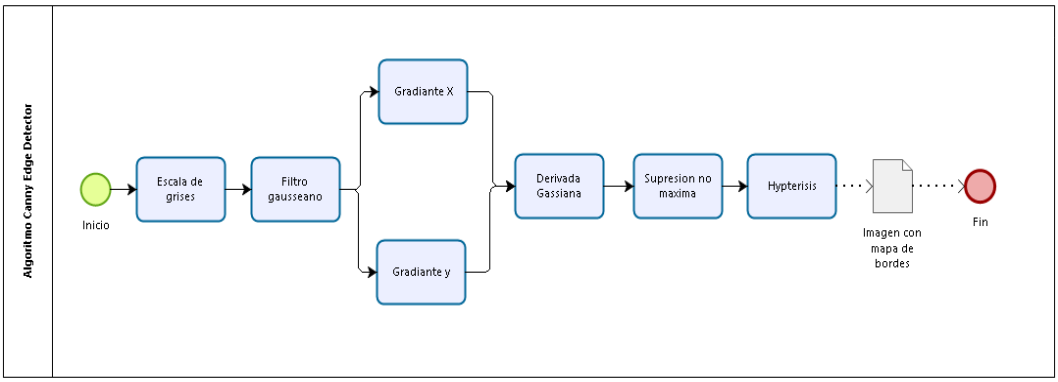

Figura 4.3: Diagrama de flujo Canny Edge Detector.

4.5. Diagrama de flujos generar lista de vértices

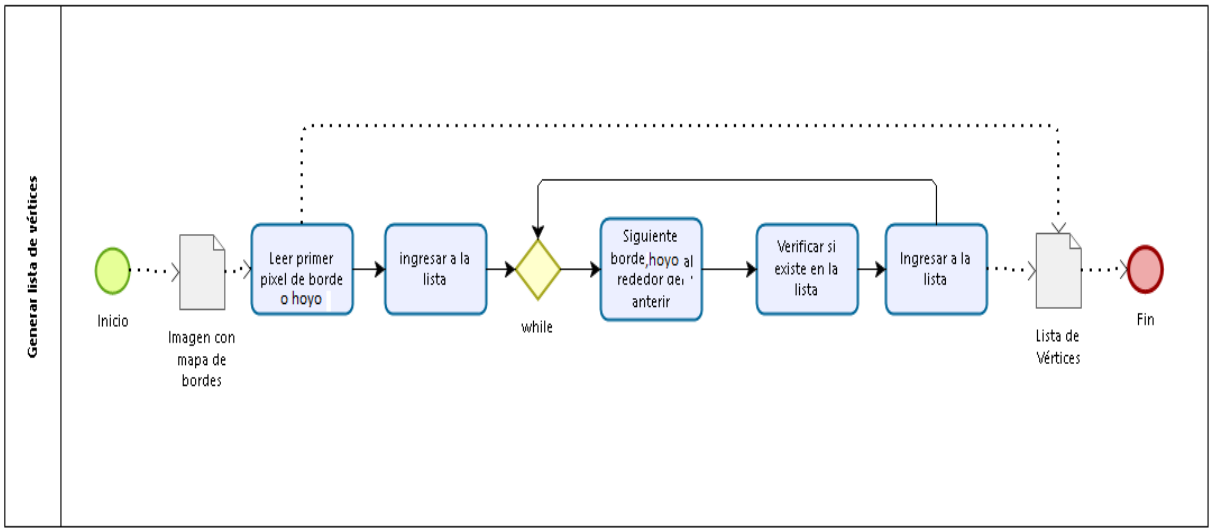

Figura 4.4: Diagrama de flujo generar lista de vértices.

4.6. Diagrama de flujos Triangulación de Delaunay Restringida

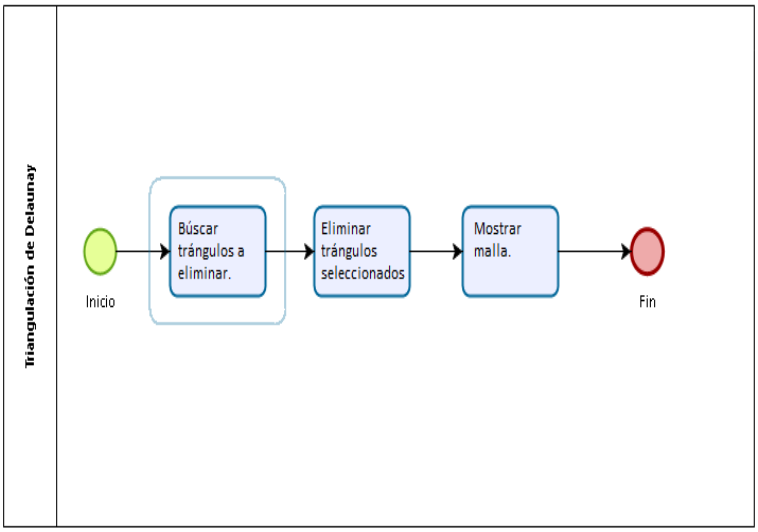

Figura 4.5: Diagrama de flujos Triangulación de Delaunay Restringida.

4.7. Diagrama de clases

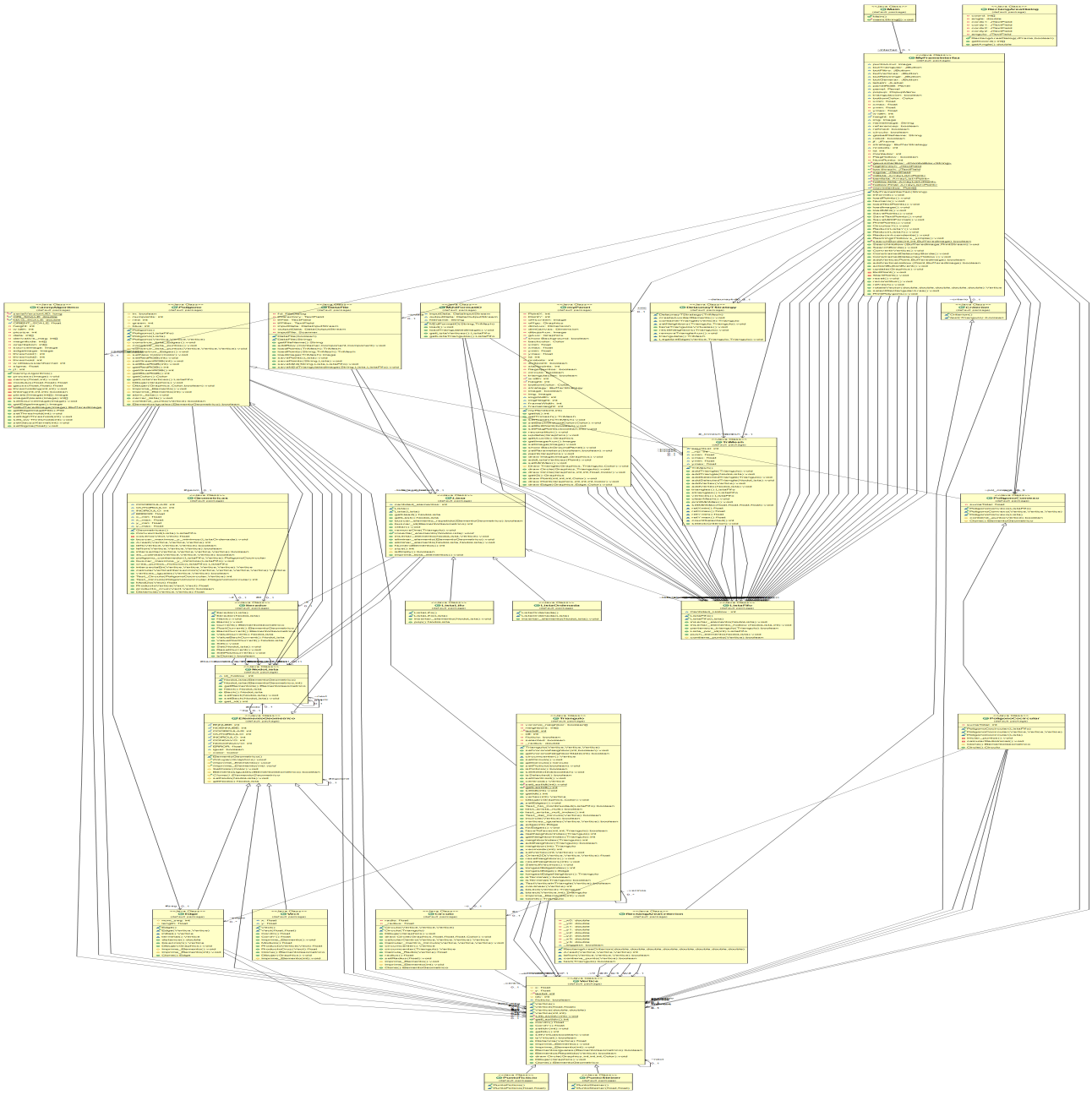

Figura 4.6: Diagrama de clases.

4.8. Herramientas a utilizar

A continuación se especifican el equipamiento a nivel de hardware y software requerido para la construcción del sistema en todas sus etapas.

4.8.1. Hardware

A continuación se especifican las herramientas a nivel de hardware que se utilizan para el desarrollo del sistema.

Herramientas tecnicas de hardware	
Hardware	***Especificación***
PC:	Notebook personal Lenovo G400s
Pantalla:	LED 14.0"(1366x768)
CPU:	Intel Core i3 3110M (2400 MHz)
RAM:	4GB DDR3 (1600 MHz)
Almacenamiento:	HDD 500GB (5400rpm)
Bateria:	4 celdas (41000 mWh)
Tarjetas de video:	NVIDIA GeForce GT 720M (2GB) Intel HD Graphics 4000 (Integrada)

Tabla 4.18: Herramientas tecnicas de hardware.

A continuación se especifican las herramientas a nivel de software que se utilizan para el desarrollo del sistema.

Herramientas tecnicas de hardware		
Tipo de software	***nombre***	***version***
Sistema operativo:	Windows 10 home single lenguage 64 bits	10.0
Ide JAVA:	Eclipse IDE for JAVA Developers	Neon.3 Release (4.6.3)
Herramientas de desarrollo JAVA:	JAVA SE Development Kit 8	1.8.0
Editor de texto LaTeX:	Textmaker	4.5
Manager de paquetes LaTeX:	MiKTeX Package Manager	2.9.6100
Gestor de proyectos:	Project 2013	
Modelado de software:	StartUML	2.7.0
Framework para diagramar:	Bizagi Process Modeler	3.1.0.011
Control de versiones:	GitKraken	1.9.3

Tabla 4.19: Herramientas tecnicas de hardware.

4.8.2. Softwares o componentes a utilizar

- *JAVA:* Es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. JAVA es, a partir de 2012, uno de los lenguajes de programación más populares en uso.
- *JAVA Development Kit(JDK):* Es un software que provee herramientas de desarrollo para la creación de programas en JAVA. Puede instalarse en una computadora local o en una unidad de red.

  En la unidad de red se pueden tener las herramientas distribuidas en varias computadoras y trabajar como una sola aplicación.

- *Eclipse:* Es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de JAVA llamado JAVA Development Toolkit (JDT) .
- *LaTeX:* Es un sistema de preparación de documentos para tipografía de alta calidad. Se utiliza con mayor frecuencia para documentos técnicos o científicos de tamaño medio a grande, pero puede utilizarse para casi cualquier forma de publicación [1].
- *Texmaker:* Es un editor libre de látex, moderno y multiplataforma para Linux, MacOSX y windows que integra muchas herramientas necesarias para desarrollar documentos con LaTeX, en una sola aplicación [2].

  Texmaker incluye soporte unicode, revisión ortográfica, auto completado, plegado de código y un visor de pdf incorporado con soporte de synctex y modo de vista continua.
- *MiKTeX:* Es una distribución TeX/LaTeX para Microsoft Windows que fue desarrollada por Christian Schenk.

  Las características más apreciables de MiKTeX son su habilidad de actualizarse por sí mismo descargando nuevas versiones de componentes y paquetes instalados previamente, y su fácil proceso de instalación [3].
- *Microsoft Project 2013:* Es una aplicación de gestión de proyectos usada por miles de empresas en todo el mundo. Microsoft Project 2013 ofrece todas las herramientas necesarias para planificar, gestionar y analizar proyectos empresariales.
- *StarUML:* Es un proyecto de código abierto para desarrollar una plataforma UML/MDA rápida, flexible, extensible, funcional y libremente disponible en la plata-

[1] https://www.latex-project.org/about/
[2] http://www.xm1math.net/texmaker/
[3] https://miktex.org/

forma Win32. El objetivo del proyecto StarUML es construir una herramienta de modelado de software y también una plataforma que sea un reemplazo convincente de herramientas comerciales UML como Rational Rose, Together y así sucesivamente [4].

- *Bizagi Process Modeler:* Es un freeware utilizado para diagramar, documentar y simular procesos usando la notación estándar BPMN [5].
- *Git Kraken:* Es un software de control de versiones, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente [6].
- *Balsamiq Mockups:* Es una maqueta de interfaz de usuario gráfica [7].

4.9. Formatos de archivos

Por la importancia que tendrán los datos generados por el software presentado en ésta investigación, es necesario buscar formas adecuadas de almacenamiento de éstos, a continuación se presentan los formatos en que guardaran los datos.

4.9.1. Formato de archivo .pto

En el formato .pto se listan los puntos de la figura completa recorriendo primeramente el borde externo de forma ordenada y en el sentido de la agujas del reloj para luego listar los huecos de la misma forma, como se muestra en la Figura 4.7.

[4] http://staruml.sourceforge.net/v1/about.php
[5] https://www.bizagi.com/es
[6] https://www.gitkraken.com/
[7] https://balsamiq.com/products/

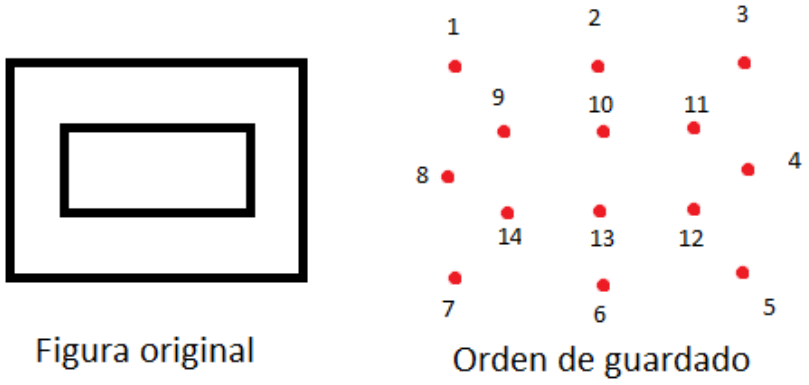

Figura 4.7: Formato .pto.

4.9.2. Formato de archivo .m2d

En el formato .m2d se guarda la información completa de la malla geométrica generada de forma ordenada y completa como se muestra a continuación:

- *Configuración:* Puede ser Delaunay o Delaunay Restringida según el tipo de malla que se este guardando.
- *Vértices generados:* Número de vértices presentes en la malla geométrica.
- *Triángulos generado:* Número de triángulos presentes en la malla geométrica.
- *Vértices:* Lista todos los vértices presentes en la malla dando primeramente un índice del vértices siento **v 1** el primero hasta **v n** el último, seguido de las coordenadas cartesianas del vértice de esta forma *v 1 7.56117 9.00671*.
- *Triángulos:* Lista todos los triángulos presentes en la malla dando primeramente un índice del triángulo, siendo **t 1** el primero hasta **t n** el último, seguido de los vértices que componen el triángulo de esta forma *t 1 64 5 38*.
- *Neighbors:* Lista todos las vecindades de los triángulos presentes en la malla, dando primeramente un índice de la vecindad siendo **n 1** el primero hasta **n n** el último, seguido de índices de triángulos vecinos de esta forma *n 1 7 182 2*.

Capítulo 5

Diseño del software: Interfaz

5.1. Interfaz de usuario

A continuación, en la Figura 5.1 se presenta el diseño general del software.

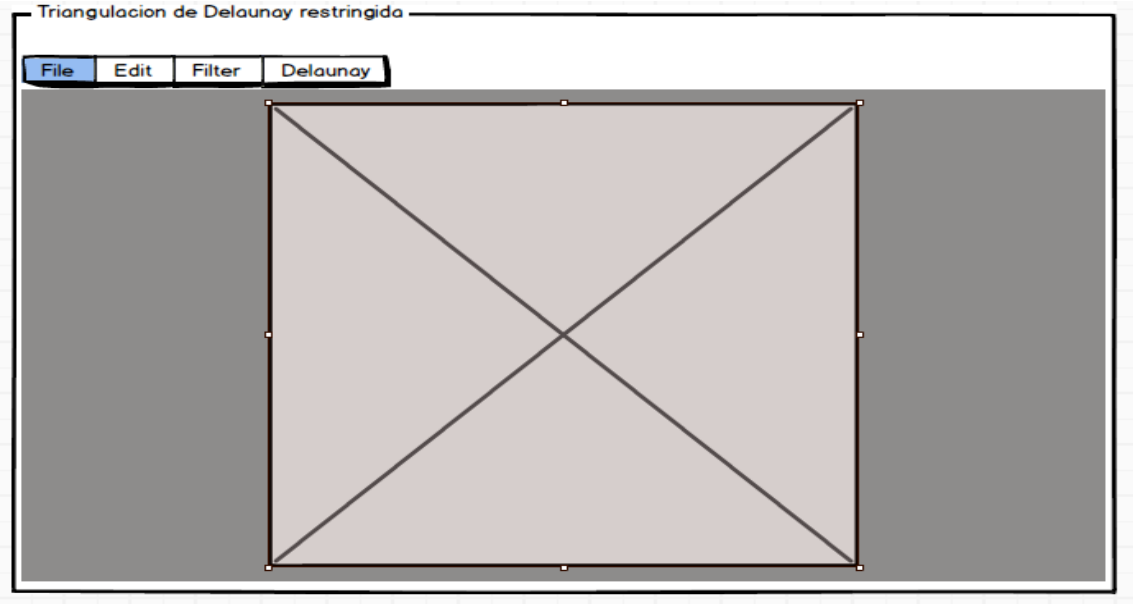

Figura 5.1: Diseño general.

5.2. Jerarquía de menú

A continuación se presenta la jerarquía de menú file.

Menu File	
File	–>
	Load Image
	Load Point
	Save Point
	Close

Tabla 5.1: Tabla jerarquía de menú file.

A continuación se presenta la jerarquía de menú edit.

Menu Edit	
Edit	–>
	Generar vértices
	Imprimir Vértice
	Generar Informe
	Clear Image

Tabla 5.2: Tabla jerarquía de menú edit.

A continuación se presenta la jerarquía de menú filter.

Menu Filter	
Filter	–>
	Generar Apply filter

Tabla 5.3: Tabla jerarquía de menú filter.

A continuación se presenta la jerarquía de menú Delaunay.

Menu Delaunay	
Delaunay	–>
	Generar Triangularización

Tabla 5.4: Tabla jerarquía de menú Delaunay.

Capítulo 6

Implementación

El primer objetivo a alcanzar es la creación de un panel en lenguaje JAVA, el cual permite leer una imagen y visualizarla, o cargar un archivo en formato *.m2d* o *.pto*.

Cuando se carga una imagen puede acceder a *Borrar imagen* o *Filtrar imagen*:

Borrar imagen: Permite eliminar la imagen cargada en caso de error al seleccionar la imagen que se desea usar.

Filtrar imagen: Procede a implementar el algoritmo *CannyEdgeDetector* mencionado en la Sección 6.1.1, el cual toma una imagen y arroja como salida una imagen tratada, la cual tiene el borde del objeto presente mediante pixeles blancos (RGB 255, 255, 255) y el resto de la imagen en negro (RGB 0,0,0), esto lo logra mediante la obtención gradiente (ver Pseudocódigo 3.7) para luego ejecutar un método de Supresión No Máxima (ver Pseudocódigo 4) la cual adelgaza los bordes logrando que éste quede de un grosor de un pixel, finalizando con la aplicación de Hystérisis de Umbral (ver Pseudocódigo 5) para asegurar y descartar posibles errores en la detección del borde. Una vez aplicado el algoritmo *CannyEdgeDetector* se muestra la imagen tratada para asegurar visualmente que los bordes se han detectado correctamente.

Al ejecutar algoritmo *CannyEdgeDetector* se habilita el botón *Generar Vértices*, el

cual aplica el método *SearchBorde()* (Ver Sección 6.1.2) que guarda la lista puntos del contorno de la figura. De inmediato se aplica el método *SearchHollow(BufferedImage img,PrintStream ps)* (Ver Sección 6.1.3), el cual recibe como parámetros el *BufferedImage* y *informe de vértices*, este método guarda los puntos de las cavidades presentes en la figura inicial.

Luego se habilita el botón *Dibujar Vértices*, el cual ejecuta el método *ConvertirVertice()* que elimina puntos innecesarios en la lista, además de pasar los puntos de *milista* al panel y los imprime en este mismo, se debe mencionar que antes de llegar a este punto se puede restringir la cantidad de puntos a imprimir con la opción *Restringir Puntos* presente en el menú *Utilidad->Restringir Puntos*.

Además con el método *loadTextPoints()* en el menú *Cargar Puntos*, cargar una nube de puntos previamente generadas por el software en formato *.pto* e imprimirla.

Una vez que los puntos están impresos en el panel se puede acceder a *Guardar Puntos* o *Triangulacion de Delaunay*:

Guardar Punto: El software permite guardar la nube de puntos generadas en una extensión *.pto*, mediante el método *SaveTextPoints()* presente en el menú *Cargar Punto*.

Triangulación de Delaunay: El método *Triangularizar()* toma la nube de puntos en el panel y genera una *Triangulación de Delaunay* dando como resultado una malla geométrica que se visualiza en el panel.

Una vez que la Triangulación de Delaunay esté impresa en el panel, se da la opción de restringir esta aplicando el método *ConstrainedDelaunayBorde()* 6.1.4 y *ConstrainedDelaunayHollow()* 6.1.5, eliminando los triángulos fuera del borde y los presentes en el interior de las cavidades respectivamente, generando una Triangulacion de Delaunay Restringida.

Por ultimo el software permite guardar las triangulaciones hechas en un formato m2d, el cual guarda tanto una Triangulacion de Delaunay como una Triangulación de Delaunay Restringida, permitiendo así exportar las mallas generadas con el software.

6.1. Métodos en java

A continuación se explican los métodos:

6.1.1. Método CannyEdgeDetector()

Para aplicar el algoritmo Canny Edge Detector se utilizaron los siguientes valores, como valores predeterminados:

- *Sigma (σ):* 1.0
- *Umbral superior:* 10
- *Umbral inferior:* 1
- *Máscara de Convolución 2D:* $3x3$

Una vez se aplica éste método, se genera la Máscara de Convolución de forma automática y si aplica el algoritmo con los valores anteriormente mencionados de la misma manera como se describe en la Sección 6.1.1.

6.1.2. Método SearchBorde()

Lee la imagen tratada resultante de la aplicación del algoritmo *CannyEdgeDetector* como *BufferedImage* para mantener una representación de una imagen en memoria, de modo que se puede acceder a cada pixel de la imagen por separado tratando a esta como una matriz de coordenadas X e Y.

Con la matriz de pixeles en la memoria se comienza a iterar desde el punto (0,0) hasta el largo y ancho de la imagen de modo que se consulta primeramente por el eje Y (debido al funcionamiento interno de *JAVA* con el plano cartesiano) hasta encontrar el primer pixel valido. Luego se comienza a iterar buscando el siguiente punto de borde alrededor del primer punto encontrado, para ésto se usa una lista de movimientos:

*static Point[] movimientos= /*primer anillo*/new Point(0,1), new Point(1,1), new*

*Point(1,0), new Point(1,-1), new Point(0,-1), /*segundo anillo*/new Point(-1,-1), new Point(-1, 0), new Point(-1,1), new Point(0,2), new Point(1,2), new Point(2,2), new Point(2,1), new Point(2,0), new Point(2,-1), new Point(2,-2), new Point(1,-2), new Point(0,-2), new Point(-1,-2), new Point(-2,-2), new Point(-2,-1), new Point(-2,0), new Point(-2,1), new Point(-2,2), new Point(-1,2), /*tercer anillo*/new Point(0,3),new Point(1,3),new Point(2,3),new Point(3,3),new Point(3,2),new Point(3,1),new Point(3,0), new Point(3,-1),new Point(3,-2),new Point(3,-3),new Point(2,-3),new Point(1,-3),new Point(0,-3),new Point(-1,-3),new Point(-2,-3), new Point(-3,-3),new Point(-3,-2),new Point(-3,-1),new Point(-3,0),new Point(-3,1),new Point(-3,2),new Point(-3,3),new Point(-3,-2),new Point(-3,-1);*

Siendo el primer anillo, los pixeles inmediatamente vecinos al último pixel encontrado, el segundo anillo son los segundo vecinos, y el tercer anillo los terceros vecinos. Se itera hasta volver al punto inicial, o primer punto válido encontrado.

6.1.3. Método SearchHollow()

SearchHollow() funciona de la misma forma que SearchBorde() 6.1.2, busca el primer pixel que sea borde, iterando desde el punto (0,0) hasta el largo y ancho de la imagen, de modo que se consulta primeramente por el eje Y pero con la condición de iterar por la lista de movimientos sólo si ese pixel no está en la lista del borde.

Una vez que encuentra un pixel candidato a pertenecer a un hueco, itera en la matriz de movimiento, pero no guarda los puntos encontrados hasta comprobar que los movimientos llevan a una figura cerrada, de lo contrario marca esos puntos como inservibles. El método SearchHollow() itera hasta recorrer todos los pixeles de la imagen llegando el punto (n,n) lo que indica que no existen mas puntos válidos.

6.1.4. Método ConstrainedDelaunayBorde()

Una vez la Triangulación de Delaunay esta impresa en el panel es posible restringir ésta. Para esto, primeramente se restringe los triángulos del borde, el método comienza a consultar cada triángulo presente en la triangulación y va seleccionando aquellos que sus tres vértices están presentes en la lista con los vértices del borde, una vez encontrado un triángulo que cumpla con esta restricción se procede a generar el centroide del triángulo candidato a eliminar, siendo el centroide el punto donde las tres medianas de un triángulo se interceptan, donde la mediana un segmento de recta que une un vértice son el punto medio de su lado apuesto (ver Figura 6.1), una vez el centroide es determinado, se proyecta en linea recta hacia un lado, contando cuantas veces éste choca con el borde de la figura inicial, si la cuenta termina en par, el triángulo es eliminado, de lo contrario se conserva.

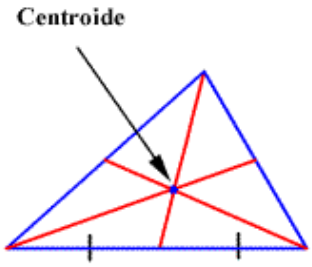

Figura 6.1: Centroide de un triángulo.

6.1.5. Método ConstrainedDelaunayHollow()

Éste método restringe los triángulos que están dentro de las cavidades de la figura inicial, para esto separa cada cavidad en figuras aislándolas, creando una lista para cada una de estas, luego se aplica el algoritmo 6.1.4 de forma inversa, esto quiere decir, que le eliminaran lo triángulos cuanto la cuenta sea impar y en el caso que sea par el triangulo se conserva.

Capítulo 7

Pruebas

Ya realizada la funcionalidad del software, éste se encuentra preparado para realizar sus pruebas correspondientes. Éste capítulo esta enfocado a estas pruebas, que pretenden verificar si lo realizado cumple con los Requerimientos mencionados en la Sección 4.1.

7.1. Interfaz principal

Figura 7.1: Interfaz inicial.

7.2. Menú

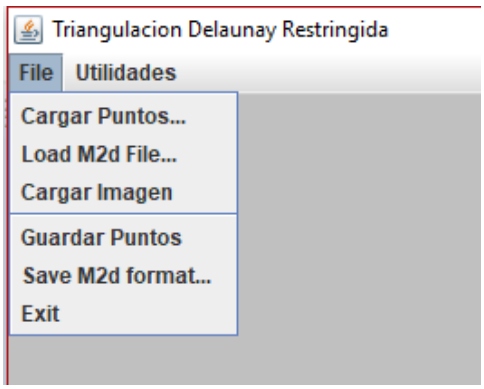

Figura 7.2: Menú file.

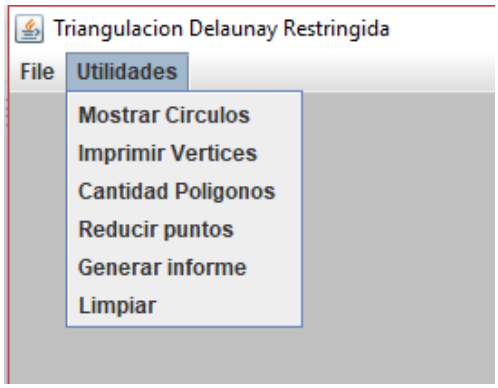

Figura 7.3: Menú utilidad.

7.3. Disposición de botones barra inferior

En esta sección se detalla la disposición de los botones en la barra inferior.

- *Filtrar:* Aplica el algoritmo CannyEdgeDetector.
- *Generar Vértices:* Aplica el algoritmo *SearchBorde();* e inmediatamente ejecuta *SearchHollow();*.
- *Dibujar Vértices:* Aplica el algoritmo *ConvertirVertice();*.

- *Triangulación Delaunay:* Aplica el algoritmo de Triangulación de Delaunay;
- *Restricción Delaunay:* Restringe la malla geométrica ya triangulada para eliminar bordes y cavidades.

El la Figura 7.4 se muestra la barra de botones completamente des-habilitadas, a la espera de que se cargue una imagen o una lista de puntos.

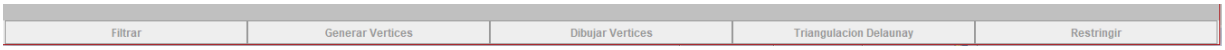

Figura 7.4: Disposición botones des-habilitados.

El la Figura 7.5 se habilita el botón *Filtrar* sólo cuando se ha leído una imagen.

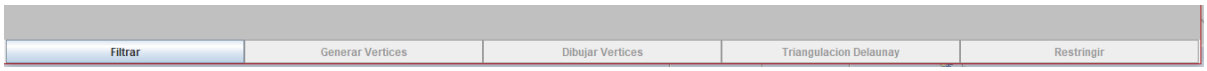

Figura 7.5: Disposición botón *Filtrar*.

El la Figura 7.6 se habilita el botón *Generar Vértices* cuando se ha filtrado una imagen y se bloquea el botón filtrar.

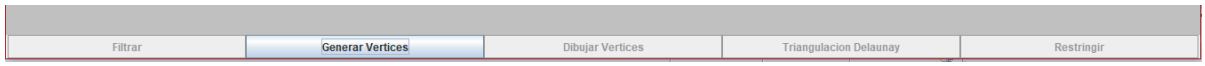

Figura 7.6: Disposición botón *Generar Vértices*.

El la Figura 7.7 se habilita el botón *Dibujar Vértices* cuando se ha generado una lista de vértices, y se bloquea el botón generar vértices.

Figura 7.7: Disposición botón *Dibujar Vértices*.

El la Figura 7.8 se habilita el botón *Triangulación de Delaunay* cuando dibujado la lista de vértices, y se bloquea el botón dibujar vértices. También puede habilitarse al cargar una lista de puntos.

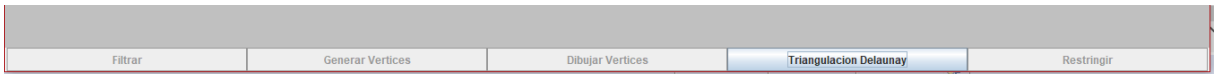

Figura 7.8: Disposición botón *Triangulación Delaunay.*

El la Figura 7.9 se habilita el botón *Restringir* cuando se ha generado malla triangulada en el panel, y se bloquea el botón Triangulacion de Delaunay. También puede habilitarse al cargar una lista de puntos.

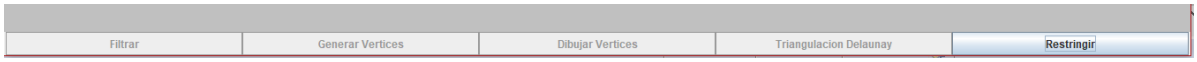

Figura 7.9: Disposición botón *Restringir.*

Se dispuso de los botones de esta manera para evitar errores al ejecutar el software, además de hacer el software muy intuitivo para ser utilizado sin ninguna complejidad.

7.4. Línea de Nazca Araña

A continuación se hace una prueba con una figura real que se encuentra localizada en Perú, la forma de Araña de las Líneas de Nazca. Ésta prueba restringe la cantidad de puntos con un intervalo de 5, lo que quiere decir que por cada punto impreso, se saltan 5 para imprimir el siguiente.

En la Figura 7.10 se muestra una representación digital de la Línea Araña de Nazca.

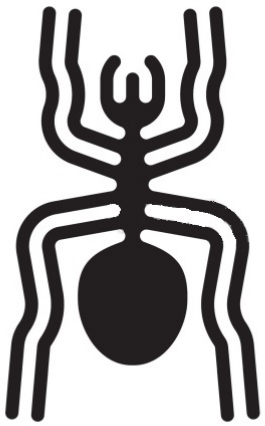

Figura 7.10: Imagen de la representación digital de la Línea de Nazca.

En la Figura 7.11 se muestra el resultado de la aplicación del algoritmo CannyEdge-Detector a la imagen de la Figura 7.10.

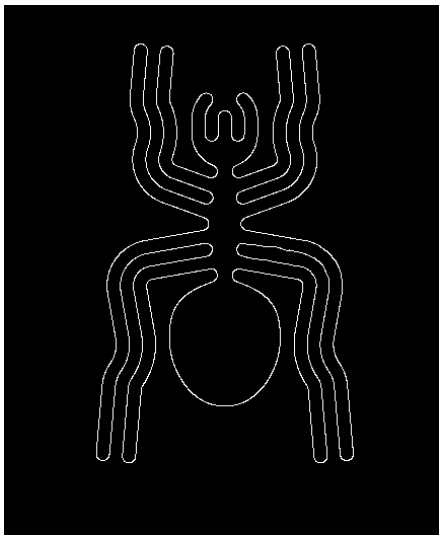

Figura 7.11: Imagen filtrada por el algoritmo CannyEdgeDetector.

En la Figura 7.12 se muestra el resultado de el algoritmo *SearghBorde()* y *SearchHollow()* los cuales se aplican sobre la imagen de la Figura 7.11.

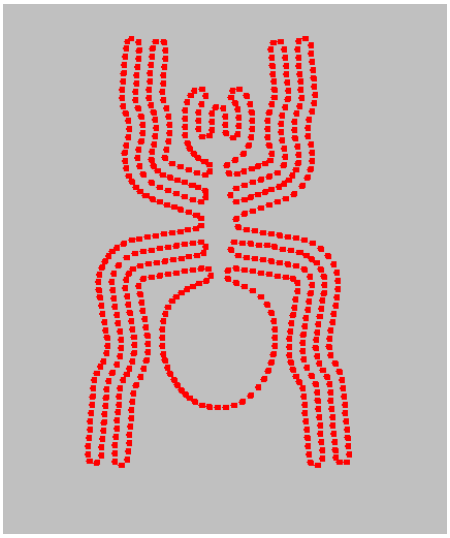

Figura 7.12: Vértices de la figura impresos en el panel.

En la Figura 7.13 se muestra la imagen de una Triangulación de Delaunay sobre la nube de punto generado, que se muestra en la imagen de la Figura 7.12.

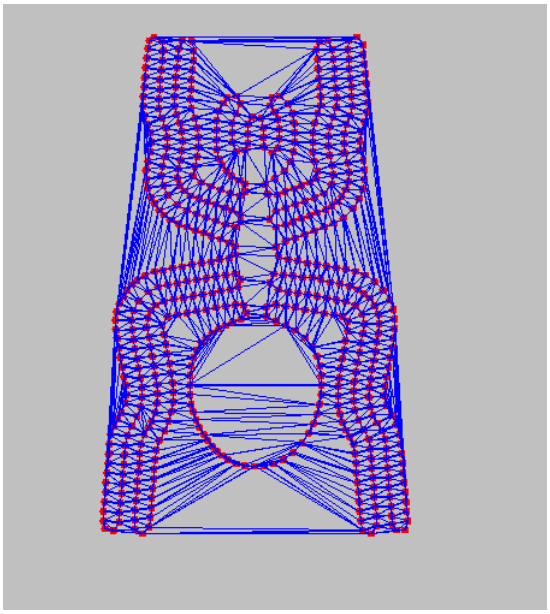

Figura 7.13: Triangulación de Delaunay sobre la nube de puntos generados.

Los datos resultantes son:

- Cantidad de *Puntos: 1162.*
- Cantidad de *Triángulos: 2299.*

En la Figura 7.14 se puede apreciar el resultado final del software, ya que se logra triangular, respetando el contorno de la figura tratada sin perder su forma, eliminando los triángulos que no están dentro del área o superficie del objeto. De esta forma se obtiene una Triangulación de Delaunay Restringida.

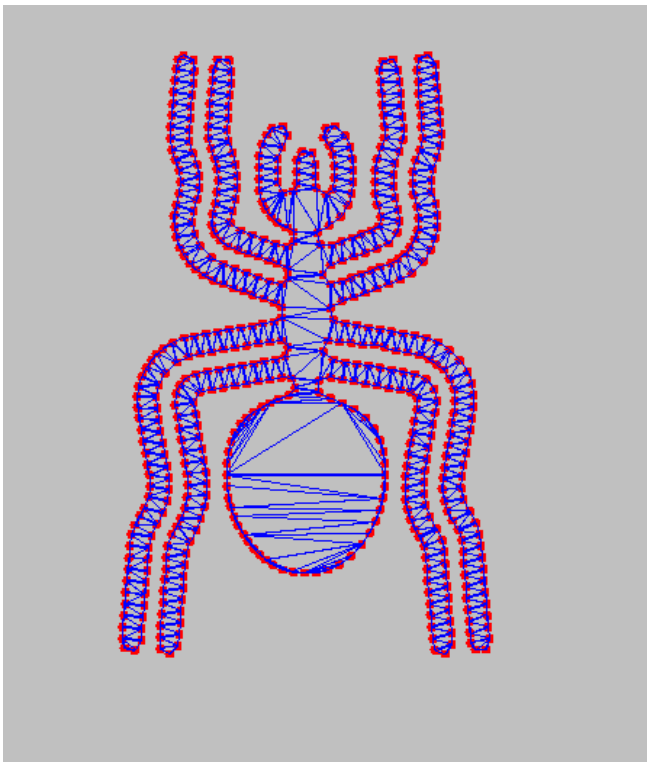

Figura 7.14: Resultado de la Restricción de la Triangulación de Delaunay.

Los datos resultantes luego de obtener la Triangulación de Delaunay Restringida son:

- Cantidad de *puntos: 1162.*
- Cantidad de *triángulos: 1227.*
- Cantidad de *triángulos eliminados: 1072.*

7.5. Prueba sobre el Mar Mediterráneo

En esta prueba se testea el Mar Mediterráneo, la dificultad de ésta imagen está en su alto grado de irregularidad. Para esta prueba se ocupa un mayor número de triángulos para conservar las irregularidades, por lo tanto se ocupa un intervalo de 2, lo que quiere decir que por cada punto impreso se saltan 2.

Figura 7.15: Mar Mediterráneo.

En la Figura 7.16 se muestra el resultado de la aplicación del algoritmo CannyEdge-Detector a la imagen de la Figura 7.15.

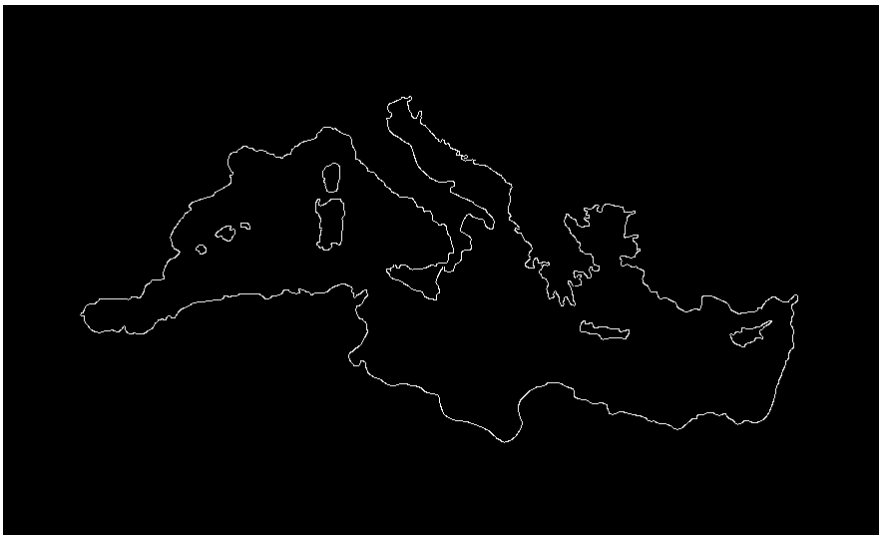

Figura 7.16: Imagen filtrada por el algoritmo CannyEdgeDetector

En la Figura 7.17 se muestra el resultado de el algoritmo *SearghBorde()* y *SearchHollow()* los cuales se aplican sobre la imagen de la Figura 7.16.

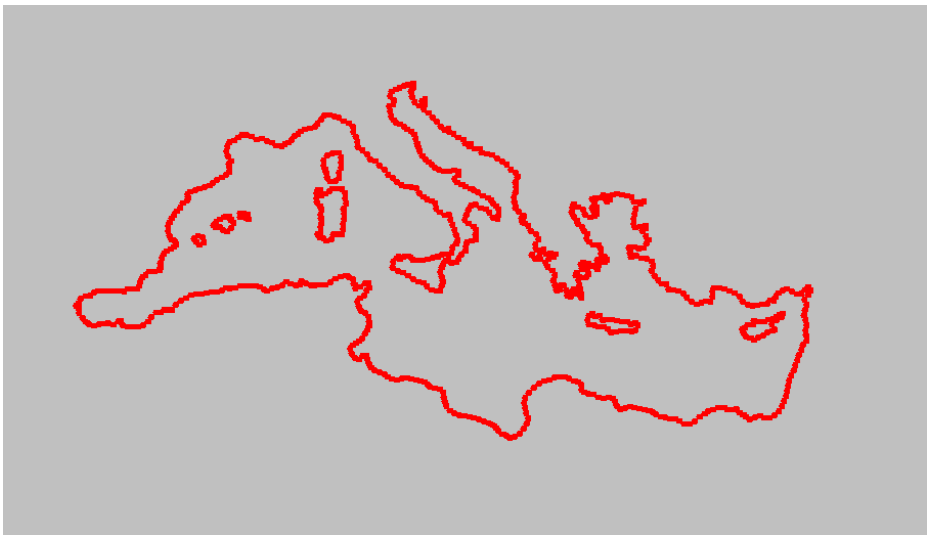

Figura 7.17: Vértices del Mar Mediterráneo son impresos en el panel.

En la Figura 7.18 se muestra una Triangulación de Delaunay sobre la nube de puntos generado que se muestra en la imagen de la Figura 7.17.

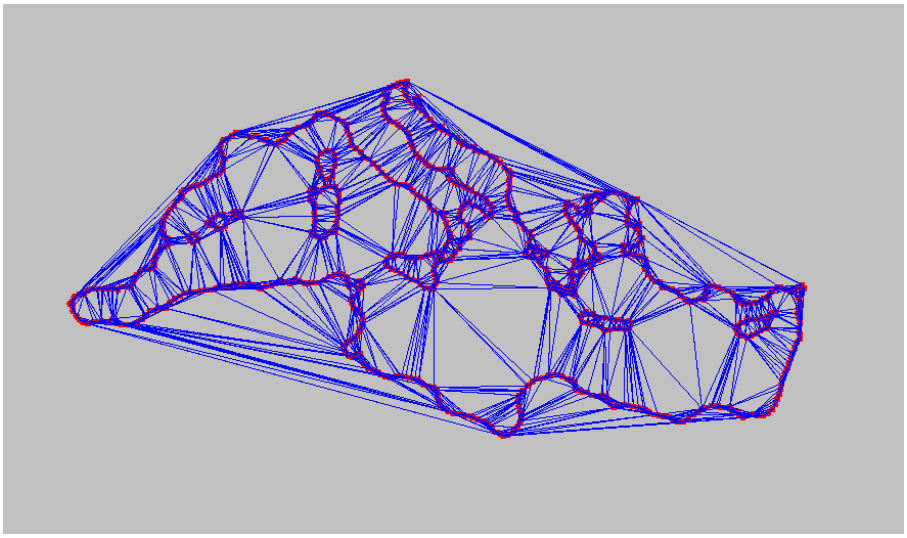

Figura 7.18: Triangulación de Delaunay sobre la nube de puntos generados.

Los datos resultantes son:

- Cantidad de *puntos: 1550.*
- Cantidad de *triángulos: 3073.*

En la Figura 7.19 se puede apreciar el resultado final de software, éste logra triangular respetando la figura deseada y respetando su contorno sin perder la forma.

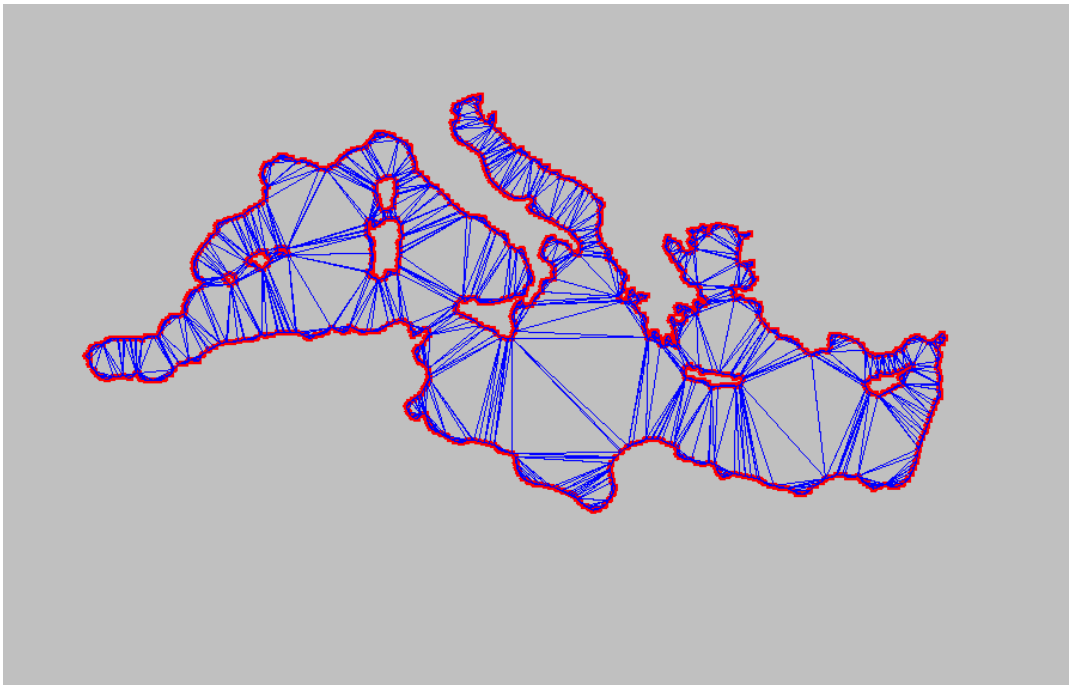

Figura 7.19: Resultado de la Restricción de la Triangulación de Delaunay.

Los datos resultantes luego de obtener la Triangulación de Delaunay Restringida son:

- Cantidad de *puntos: 1550.*
- Cantidad de *triángulos: 1611.*
- Cantidad de *triángulos eliminados: 1462.*

7.6. Prueba sobre Lago Superior

En esta prueba se testea el Lago Superior, ubicado entre Estados Unidos y Canadá, la dificultad de ésta imagen está en su alto grado de irregularidad y los cavidades presentas en esta, ya que están muy juntas en la esquina inferior izquierda como se observa en la Figura 7.20. Para esta prueba se ocupa un mayor número de triángulos para conservar las irregularidades, por lo tanto se ocupa un intervalo de 2, lo que quiere decir que por cada punto impreso se saltan 2.

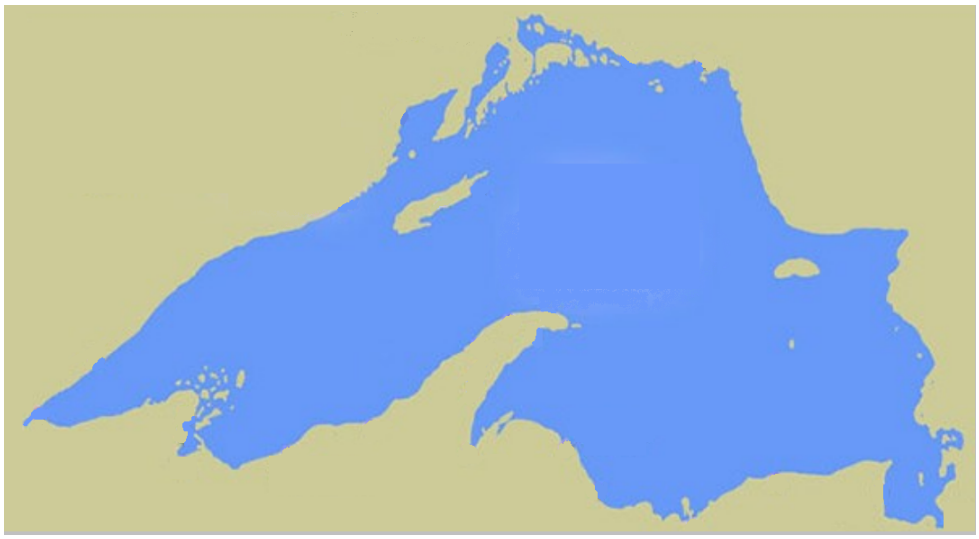

Figura 7.20: Lago Superior

En la Figura 7.21 se muestra el resultado de la aplicación del algoritmo CannyEdge-Detector a la imagen de la Figura 7.20.

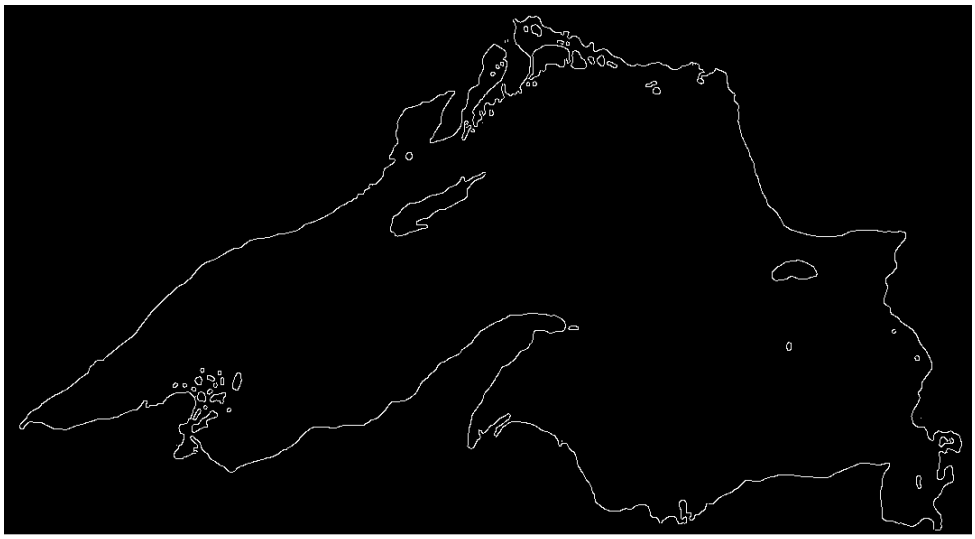

Figura 7.21: Imagen filtrada por el algoritmo CannyEdgeDetector

En la Figura 7.22 se muestra el resultado de el algoritmo *SearghBorde()* y *SearchHollow()* los cuales se aplican sobre la imagen de la Figura 7.21.

Figura 7.22: Vértices del Lago Superior son impresos en el panel.

En la Figura 7.23 se muestra una Triangulación de Delaunay sobre la nube de puntos generado que se muestra en la imagen de la Figura 7.22.

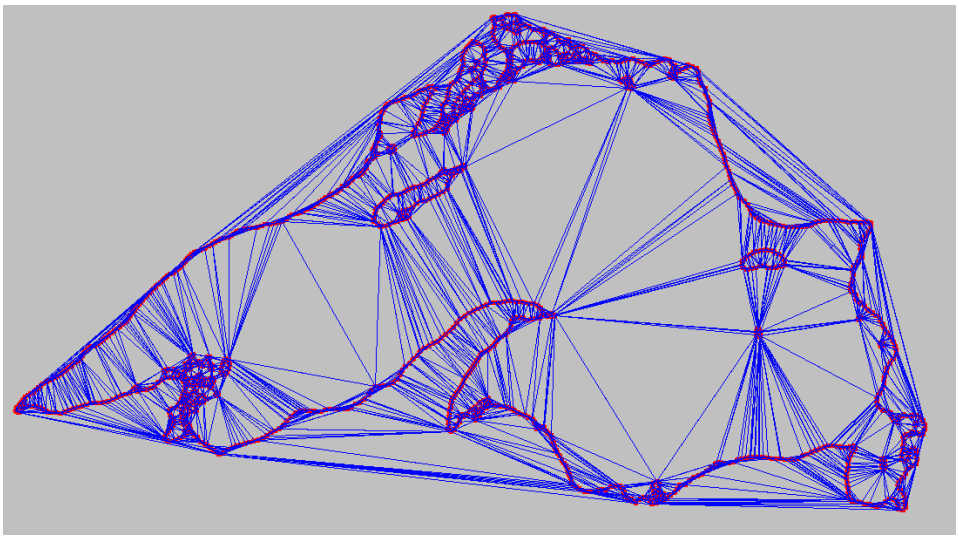

Figura 7.23: Triangulación de Delaunay sobre la nube de puntos generados.

Los datos resultantes son:

- Cantidad de *puntos: 2370.*
- Cantidad de *triángulos: 4714.*

En la Figura 7.24 se puede apreciar el resultado final de software, éste logra triangular respetando la figura deseada y respetando su contorno sin perder la forma.

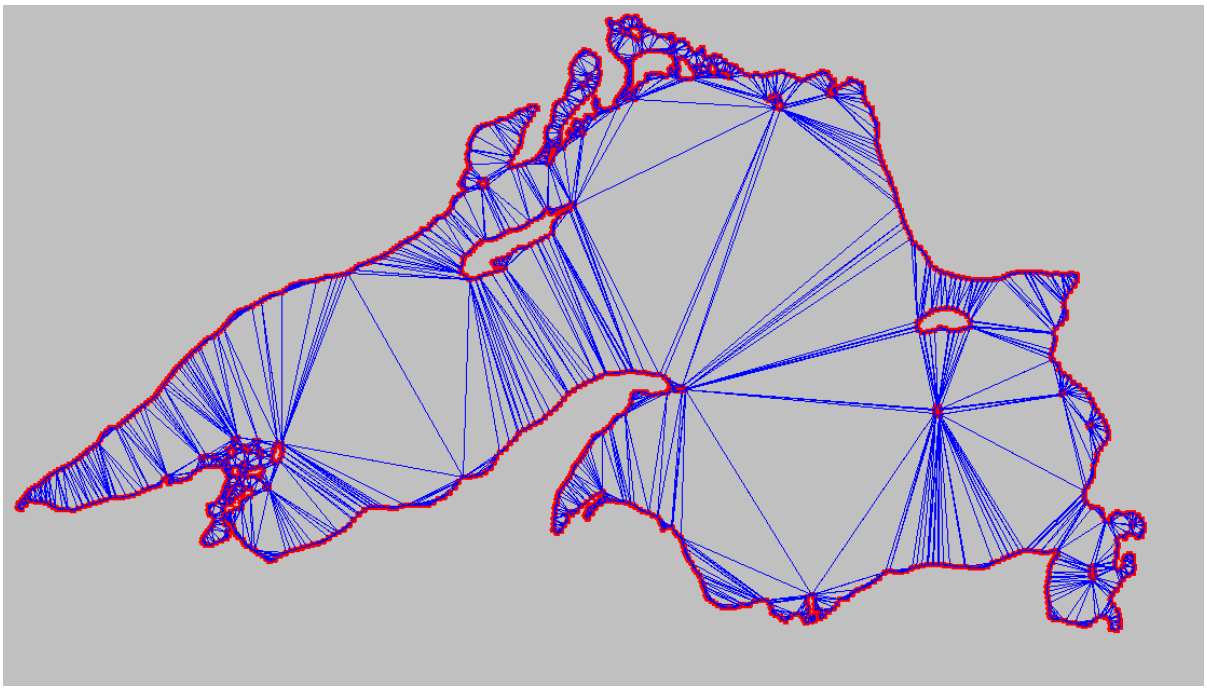

Figura 7.24: Resultado de la Restricción de la Triangulación de Delaunay.

Los datos resultantes luego de obtener la Triangulación de Delaunay Restringida son:

- Cantidad de *puntos: 2370.*
- Cantidad de *triángulos: 1979.*
- Cantidad de *triángulos eliminados: 2735.*

7.7. Prueba sobre Lago Budi

En esta prueba se testea el Lago Superior, ubicado al sur de Chile, la dificultad de ésta imagen está en su alto grado de irregularidad, superior a las otras pruebas como se observa en la Figura 7.25. Para esta prueba se ocupa un mayor número de triángulos para conservar las irregularidades, por lo que se ocupa un intervalo de 0, lo que quiere decir que se ocupan todos los puntos generados

Figura 7.25: Lago Budi

En la Figura 7.26 se muestra el resultado de la aplicación del algoritmo CannyEdge-Detector a la imagen de la Figura 7.25.

Figura 7.26: Imagen filtrada por el algoritmo CannyEdgeDetector.

En la Figura 7.27 se muestra el resultado de el algoritmo *SearghBorde()* y *SearchHollow()* los cuales se aplican sobre la imagen de la Figura 7.26.

Figura 7.27: Vértices del Lago Budi son impresos en el panel.

En la Figura 7.28 se muestra una Triangulación de Delaunay sobre la nube de puntos generado que se muestra en la imagen de la Figura 7.27.

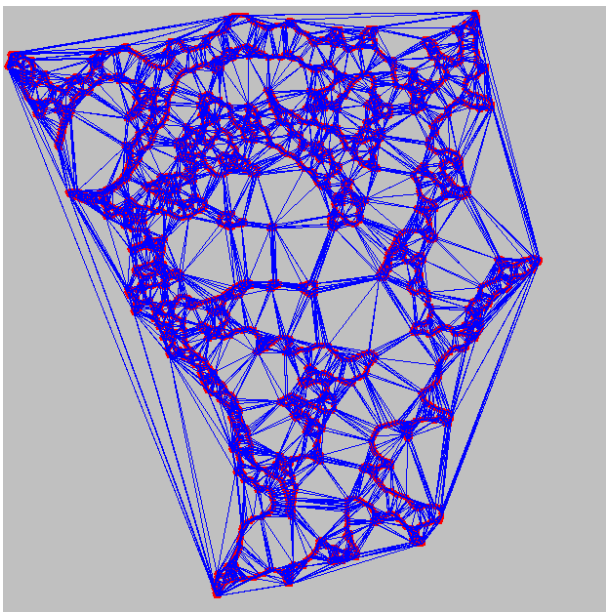

Figura 7.28: Triangulación de Delaunay sobre la nube de puntos generados.

Los datos resultantes son:

- Cantidad de *puntos: 6276.*
- Cantidad de *triángulos: 12533.*

En la Figura 7.29 se puede apreciar el resultado final de software, éste logra triangular respetando la figura deseada y respetando su contorno sin perder la forma.

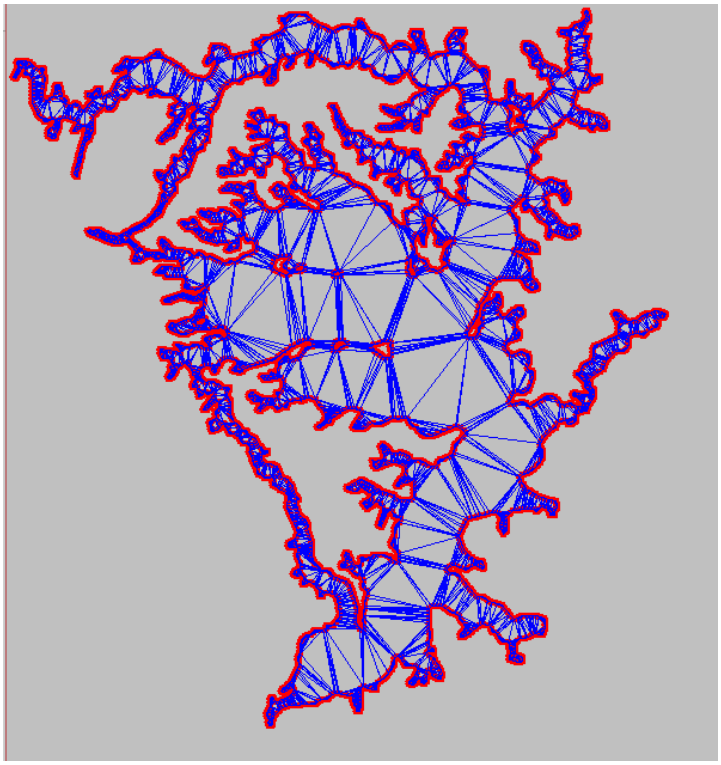

Figura 7.29: Resultado de la Restricción de la Triangulación de Delaunay.

Los datos resultantes luego de obtener la Triangulación de Delaunay Restringida son:

- Cantidad de *puntos: 6276.*
- Cantidad de *triángulos: 6424.*
- Cantidad de *triángulos eliminados: 6109.*

Capítulo 8

Conclusiones

8.1. Conclusiones

Durante el desarrollo de este informe se pretendió demostrar que la aplicación combinada del algoritmo Cannt Edge Detector y la Triangularización de Delaunay Restringida funcionan mejor que aplicarlos por separado, ya que la generación de una malla geométrica a partir de una imagen se da de una forma totalmente automatizada.

Otro punto muy importante a considerar es el diseño de la herramienta. La utilización de un buen diseño orientado a objetos junto con un lenguaje de programación robusto y enfocado a la orientación a objetos, permitió un desarrollo modular y extensible, al que fácilmente se le pueden agregar nuevos algoritmos, métodos de refinamiento y tipos de mallas (3D). Esto permite realizar fácilmente tanto la extensión de la herramienta para continuar perfeccionándola, como su utilización para otros problemas relacionados con la generación de mallas geométricas.

Al finalizar este trabajo de proyecto de título, se pudo demostrar la hipótesis planteada en un comienzo. Se consiguió reducir el tiempo de un trabajo tedioso y de mucho esfuerzo que conllevaba delimitar una figura de manera manual punto por punto, logrando

acotar un trabajo de muchas horas a un procedimiento simple que se puede ejecutar perfectamente en cosa de segundos con pequeñas variaciones de tiempo según la complejidad que pueda tener el objeto a tratar, además de asegurar una aproximación a la imagen real mucho más fiable que el antiguo procedimiento que estaba sujeto a errores humanos, por consiguiente los estudios futuros que pudiera hacerse sobre estas mallas serán más eficientes, claros y precisos. Por otro lado se consiguió implementar una Triangulación de Delaunay Restringida de manera exitosa pudiendo así generar mallas geométricas en base a la figura presente en la imagen, estas mallas que genera el software presentado en este trabajo, logra respetar los contornos de la figura y los huecos de manera precisa pudiendo así eliminar lo triángulos fuera de la figura y dentro de las cavidades con un alto grado de precisión.

Otro punto importante a considerar es la importancia que tendrán estos datos a futuro, por ende se implemento un método eficiente de almacenamiento. Para ésto el software puede almacenar nube de puntos (formato .pto), pero más importante aún puede almacenar mallas geométricas completas (formato .m2d), además de cargar y visualizar estas misma extensiones de archivos generados.

8.2. Trabajos futuros

Como ocurre en el desarrollo de todo software, las herramientas pueden ser mejoradas. La siguiente lista presenta una propuesta de los puntos a trabajar:

- Generar un software, usando éste trabajo de proyecto de título como base, para implementar una malla 2D $\frac{1}{2}$ para lograr modelos de mapas cartográficos.
- Generar un software, usando éste trabajo de proyecto de título como base, para implementar modelos 3D de sólidos irregulares.
- Continuar mejorando la eficiencia de tiempo de ejecución de los algoritmos.
- Implementar algoritmos de refinamiento de mallas.

Referencias

[1] Enrique Alarcón Álvarez. *Modelos matemáticos en ingeniería moderna*, tomo 74. CDCH UCV, 2000. Book.

[2] C Bradford Barber, David P Dobkin, y Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4): 469–483, 1996. Article.

[3] Frederick P Brooks. No silver bullet. *IEEE Computer*, 20(4): 10–19, 1987. Article.

[4] Siu-Wing Cheng, Tamal K Dey, y Jonathan Shewchuk. *Delaunay mesh generation*. CRC Press, 2012. Book.

[5] José Benjamín Ojeda Chong, Nancy Hitschfeld Kahler, Benjamín Bustos Cárdenas, y Johan Fanry. Algoritmo paralelo para vacíos poligonales en Triangulaciones de Delaunay. Article.

[6] Roberts Cross. Canny edge detector. *Algorithm Designs*, pág. 39. Article.

[7] Álvaro Martín Faúndez Reyes. Marco de experimentación para algoritmos de refinamiento de triangulaciones en 2D. Article, 2010.

[8] Thomas Leduc, Erwan Bocher, y Fernando González Cortés. Efficient constrained delaunay triangulation implementation in java for spatial hydrological analysis. En *Free and Open Source Software for Geospatial Conference-FOSS4G'2008*. 2008. Article.

[9] Juan Linares Mijail Leon, Manuel Vidal. Diagrama de clases. Article, 2013.

[10] José Enrique Priego de los Santos y María Joaquina Porres de la Haza. La Triangulación de Delaunay aplicada a los modelos digitales del terreno. En *X Congreso del Grupo de Métodos Cuantitativos, Sistemas de Información Geográfica y Teledetección.* 2002. Article.

[11] Jorge Valverde Rebaza. Detección de bordes mediante el algoritmo de canny. *Escuela Académico Profesional di Informática. Universidad Nacional de Trujillo*, 2007. Article.