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

HERRAMIENTA DE RESOLUCIÓN DE TRIANGULACIONES GEOMÉTRICAS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

JAVIER ULISES DÍAZ PALACIOS

PROFESOR GUÍA:
MARÍA CECILIA RIVARA ZÚÑIGA

MIEMBROS DE LA COMISIÓN:
GONZALO NAVARRO BADINO
JOSÉ MANUEL SAAVEDRA RONDO

SANTIAGO DE CHILE
2018

Resumen

La triangulación de Delaunay es una entidad geométrica con muchas aplicaciones en computación gráfica e ingeniería. Por lo general, su construcción es un problema difícil que a menudo viene acompañado con restricciones geométricas y de calidad. Para facilitar la experimentación de algoritmos relativos al problema de Delaunay, se presenta una herramienta con mejoras en simplicidad, desempeño y robustez frente a otras opciones existentes.

En primer lugar, se ofrece una implementación sólida del algoritmo de Delaunay incremental con intercambio de aristas, el cual es un método conocido que aborda el problema agregando cada punto de la triangulación secuencialmente. Esta implementación maneja las estructuras de datos de forma sencilla (triángulos con punteros a sus vecinos), por lo que es fácil de extender. Además, asegura que las triangulaciones siempre quedan bien definidas, anulando cualquier operación que invalide la triangulación de acuerdo con un criterio de robustez sobre los triángulos.

En segundo lugar, se implementa un algoritmo que adapta las triangulaciones para satisfacer restricciones en las aristas, el cual funciona por medio de una idea sencilla que reusa conceptos del algoritmo de Delaunay incremental. En cada iteración se localizan los triángulos que intersectan la arista que se quiere respetar y se intercambian las diagonales secuencialmente hasta que sea respetada.

Finalmente, el algoritmo Lepp-Centroide desarrollado por la profesora Rivara y coautores permite obtener una triangulación de buena calidad (ángulo mínimo acotado) por medio de la inserción de nuevos puntos. La implementación de esta memoria se comporta como ha sido establecido en estudios teóricos y prácticos previos, lo que significa que se consigue mejorar la calidad de las triangulaciones insertando pocos puntos y con restricciones geométricas menos fuertes.

Tabla de contenido

Índice de figuras

Índice de tablas

Índice de códigos

Introducción

La triangulación (partición de una región espacial en un conjunto de triángulos) es una herramienta muy importante para el modelamiento de objetos geométricos en computación gráfica y aplicaciones de ingeniería, ya sea con fines matemáticos o de visualización, porque aproxima bien las superficies utilizando polígonos simples.

Una clase particular de estas particiones que cumple con propiedades geométricas interesantes es la triangulación de Delaunay, la cual se describe a través de una restricción específica para todos los pares de triángulos contiguos que será abordada en detalle en la sección 1. El beneficio de esta entidad es asegurar que todos los polígonos del conjunto maximizan sus ángulos mínimos, una propiedad especialmente útil para la aritmética finita porque evita la aparición de ángulos pequeños que pueden entorpecer la precisión de los cálculos.

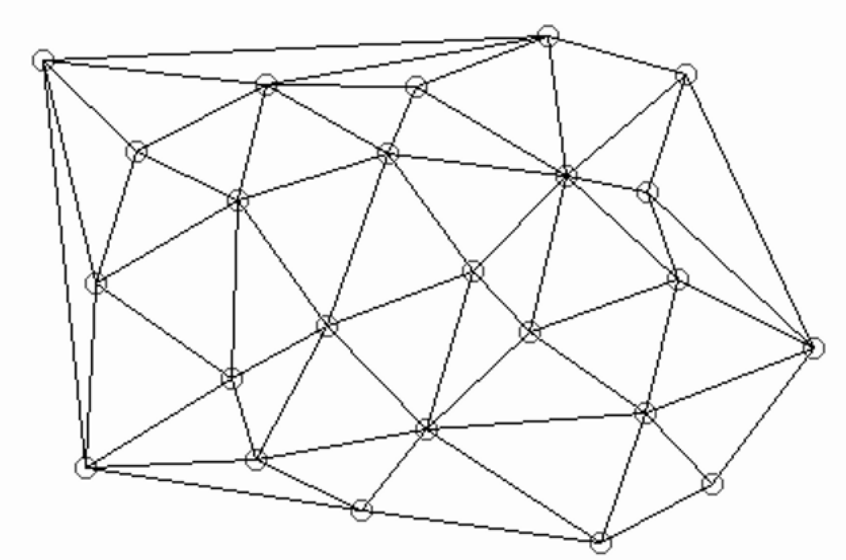


Figura 1: Triangulación de Delaunay de un conjunto de puntos

Existe mucha investigación alrededor del problema de Delaunay, que consiste en encontrar la triangulación de Delaunay para un conjunto determinado de puntos. Si bien esta investigación respalda soluciones robustas y eficientes para distintos escenarios y restricciones de tiempo, memoria y calidad, aún existe margen de mejora para las implementaciones de estos algoritmos cuando se trata de resolver el problema bajo un paradigma de simplicidad de diseño.

En consecuencia, se plantea una herramienta fácil de extender y que integra soluciones a algunos problemas típicos en el estado del arte como la restricción de aristas obligatorias, el refinamiento de la calidad de los triángulos, la imposición de bordes a la geometría y la optimización de los tiempos de ejecución.

El enfoque de esta implementación está en resolver de manera robusta algunos de los problemas frecuentes de las triangulaciones de Delaunay y, de esta manera, servir como base para la experimentación científica y la extensión de funcionalidades.

En general, se resuelven dos familias distintas de problemas:

1. **Construcción de triangulaciones de Delaunay con aristas restringidas**. Este problema consiste en agregar restricciones geométricas a la triangulación, imponiendo aristas que deben aparecer obligatoriamente en la geometría. Esto conlleva a una generalización de los datos de entrada, que pueden describirse como un Planar Straight-Line Graph o PSLG, es decir, un conjunto de puntos y aristas que describen una geometría (ver figura 2).

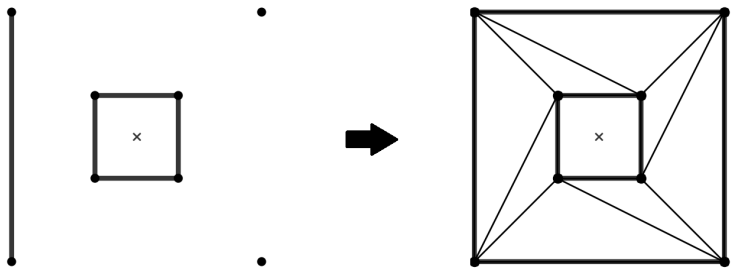


Figura 2: Triangulación de Delaunay restringida a partir de un PSLG

2. **Generación de triangulaciones de buena calidad**. Este problema consiste en obtener una triangulación τ_f adaptada a la geometría y tal que todo triángulo de τ_f tiene ángulo mínimo menor o igual que una tolerancia θ_{tol}. El objetivo es obtener una implementación simple y completa, además de validar en la práctica el algoritmo con geometrías con ángulos restrigidos interiores, los cuales no son considerados por implementaciones previas.

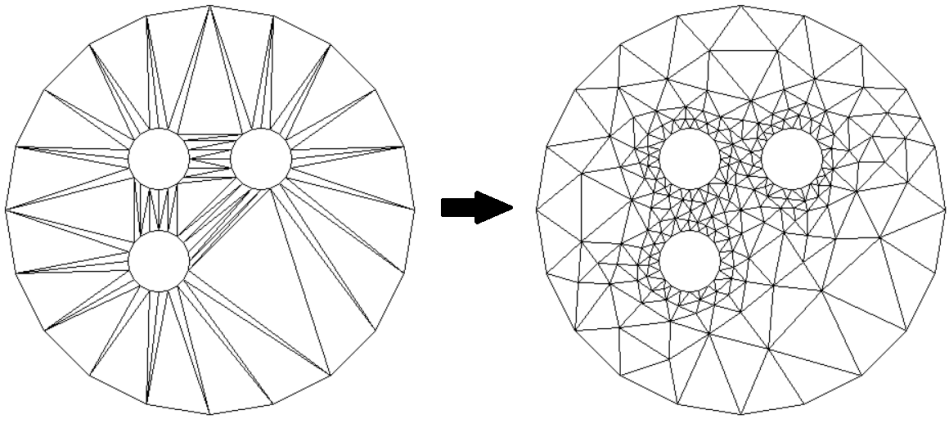


Figura 3: Mejoramiento de una triangulación de Delaunay

1. Aspectos generales de la implementación

La herramienta desarrollada resuelve los siguientes problemas:

1. Construcción de triangulación Delaunay de un conjunto de puntos.
2. Construcción de triangulación Delaunay restringida.
3. Refinamiento Lepp-Delaunay Centroide.

Esta implementación busca ser un desarrollo simple y completo en Python, en el que los aspectos más técnicos (estructuras de datos y operaciones geométricas) se vean simplificados por la selección del lenguaje de programación y el uso de técnicas sencillas.

1.1. Triangulación de Delaunay

Para comprender en su totalidad los desafíos y beneficios de construir una triangulación de Delaunay hace falta profundizar en su definición y sus propiedades matemáticas.

La triangulación de Delaunay satisface, por definición, la llamada *condición de Delaunay* que establece que para cualquier triángulo de la triangulación, el interior de su circuncírculo no contiene ninguno de los vértices de otro triángulo. Por supuesto, en el caso general esta condición se relaja cuando la arista interior es restringida.

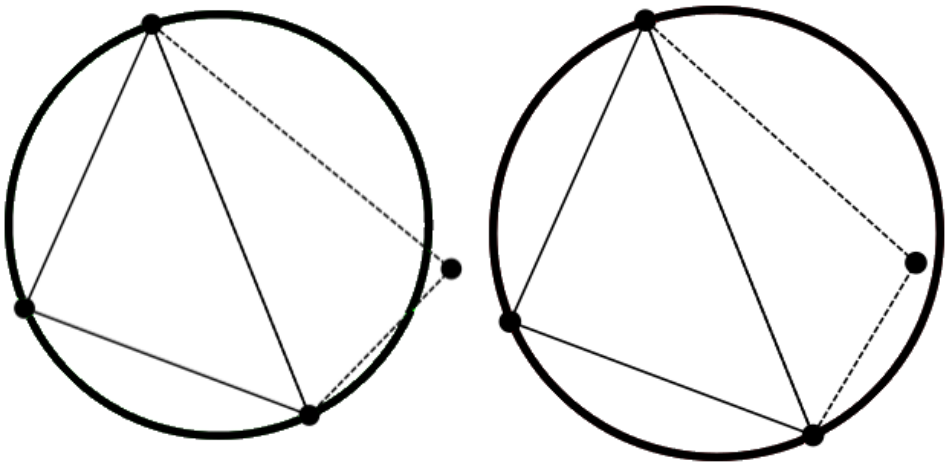


Figura 4: Condición de Delaunay

Matemáticamente se demuestra que la triangulación de Delaunay de un conjunto de puntos satisface las siguientes dos propiedades.

1. El borde exterior de la triangulación forma la envoltura convexa del conjunto de puntos, es decir, el polígono convexo mínimo que contiene los puntos. Esto permite construir consistentemente el borde de la geometría dada.

2. Se maximiza el ángulo interior mínimo de los triángulos, es decir, los triángulos son lo más equilátero posible para los datos dados.

1.2. Estructuras de datos

El objetivo de definir estructuras adecuadas a este problema es mantener información acerca de la vecindad de los triángulos contenidos en la triangulación y evitar la actualización de grandes cantidades de variables para operaciones que son, en esencia, simples.

En primer lugar, se define la estructura Point que representa a los puntos que serán vértices de la triangulación. Cada uno está formado por dos valores que indican coordenadas (x, y) y son almacenados una única vez en una lista al inicio del algoritmo, la cual es referenciada por cada triángulo.

La segunda estructura, llamada Triangle, contiene una lista con referencias a sus tres vértices, que son de tipo Point, y otra lista con referencias a los triángulos vecinos, del tipo Triangle. Además, esta estructura se inicializa con dos propiedades particulares: sus vértices mantienen una orientación en sentido antihorario (propiedad 1) y una correspondencia con sus vecinos, de manera que el triángulo vecino i-ésimo se opone al vértice i-ésimo (propiedad 2). Estas propiedades se respetan en la creación de cada triángulo y en cualquier otra operación que modifique sus valores. Por ejemplo, en la figura 5 sólo hay tres descripciones válidas del triángulo en cualquier momento del algoritmo: [A, B, C], [B, C, A] y [C, A, B].

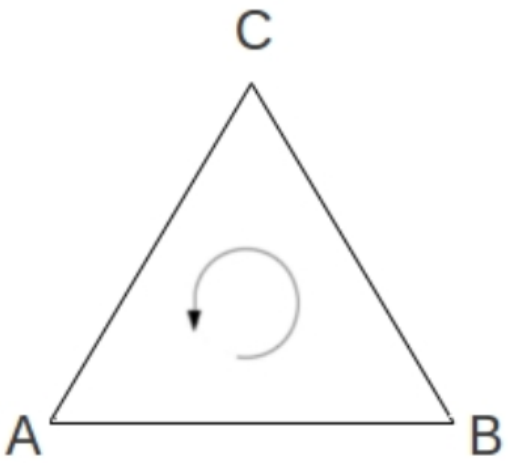


Figura 5: Orientación antihoraria de los triángulos.

La propiedad 1 (triángulos orientados) permite resolver el problema de robustez asociado al uso de aritmética finita en las complejas operaciones requeridas. Esto se logra verificando que dichas operaciones mantienen la orientación antihoraria de los triángulos. Si esta propiedad no se mantiene, se considera que la operación no es válida (lleva a un estado inconsistente) y se ignora.

La propiedad 2 facilita el manejo de las operaciones entre triángulos, ya que permite determinar en qué dirección se encuentra un triángulo vecino dado. De esta manera, basta con conocer el índice del vértice i para referirse al triángulo vecino por la arista opuesta $list\_of\_neighbors[i]$.

Además, la estructura Triangle almacena información adicional en las siguientes variables:

1. La variable *constraints* se relaciona con el proceso de restricción de aristas y contiene los índices a las aristas restringidas del triángulo. La consistencia entre vecinos, es decir, que un arista sea considerada restringida por ambos triángulos adyacentes, es asegurada por las operaciones que modifican la estructura de datos.

2. La variable *improvable* se relaciona con el proceso de refinamiento e indica que el triángulo no tiene ninguna restricción para ser mejorado, por lo que esto sólo depende de la condición de sus ángulos. Esta variable permite marcar aquellos triángulos cuyo refinamiento está prohibido por tener ángulos restringidos menores al ángulo de tolerencia, los cuales llevarían a un bucle si se intentaran refinar.

Finalmente, la estructura Triangulation almacena la triangulación resultante, es decir, es el contenedor de una lista con referencias a todos los triángulos sobre los que se ha computado la condición de Delaunay. Además, la estructura mantiene un puntero al conjunto de puntos que lo conforman.

2. Triangulación de Delaunay no restringida

Existen varios tipos de algoritmos capaces de generar una triangulación de Delaunay, los cuales cubren necesidades computacionales de diversa índole, como la simplicidad de su implementación o su adaptabilidad para ejecutarse en paralelo.

En particular, dos familias de algoritmos resuelven el problema eficientemente [9, 10]: los algoritmos de construcción incremental y los algoritmos de Divide and Conquer. Para esta implementación se escogió un algoritmo de construcción incremental basado en intercambio de aristas [8] porque maneja conceptos más sencillos que los de Divide and Conquer y otros de la misma familia. A cambio, el algoritmo seleccionado es más difícil de paralelizar, pero esto último no es parte del objetivo de la herramienta.

El algoritmo de construcción incremental basado en intecambio de aristas consiste en partir de una triangulación base que cumple la condición de Delaunay (normalmente un único triángulo que cubre todo el rango de puntos de entrada) e insertar secuencialmente cada punto. Para cada inserción de un punto p se localiza su triángulo contenedor (sección 2.2.1), se divide la triangulación en nuevos triángulos que surgen desde p (sección 2.2.2) y, finalmente, se corrigen los triángulos que dejaron de cumplir la condición de Delaunay por medio de una operación llamada *intercambio de aristas* (sección 2.2.3).

2.1. Inicialización

Antes de la inserción de puntos se realizan algunas tareas necesarias para asegurar que el algoritmo se comporta de manera eficiente y robusta.

En primer lugar, se permutan aleatoriamente los puntos de entrada, con el fin de evitar que el orden afecte a la eficiencia. Esto permite que cada inserción modifique en promedio sólo $O(1)$ triángulos [9].

Luego, se define el triángulo inicial de manera que contiene todos los puntos de la triangulación. Esto se implementa encontrando el rectángulo asociado a las coordenadas de borde

y construyendo un triángulo que lo contenga, el cual es ponderado por una constante lo bastante grande como para no influir en los circuncírculos de los triángulos que se van a generar.

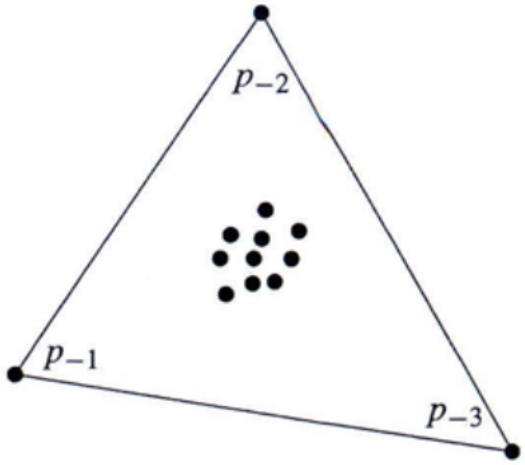


Figura 6: Triángulo inicial para un conjunto de puntos

2.2. Inserción de puntos

Como el algoritmo es incremental, se describen a continuación las operaciones necesarias para agregar cada punto del conjunto, siendo la triangulación inicial el triángulo cobertor descrito en la sección anterior.

2.2.1. Encontrar el triángulo contenedor del punto p

El primer paso del algoritmo consiste en encontrar cuál de los triángulos que pertenecen a la triangulación actual contiene el punto p a ser insertado. Para esto, se utiliza un algoritmo de *Jump and Walk* que, dado un punto q en un triángulo arbitrario, busca recursivamente al vecino que más se acerca a p a través del segmento $\overline{qp}$. Esto se logra por medio del algoritmo de orientación 2D (sección 2.3.2), de manera que el vecino más cercano es aquél que comparte una arista orientada a la derecha de p. Finalmente, el triángulo contiene al punto p si todas sus aristas están orientadas a la izquierda.

2.2.2. División de triángulos

Después de encontrar el triángulo, con vértices (p_i, p_j, p_k), se determina si el punto p está en su interior o yace sobre una de sus aristas. En el primer caso, se divide el triángulo en tres: (p_i, p_j, p), (p_j, p_k, p) y (p_k, p_i, p). En el segundo caso, se determina el triángulo vecino con el que comparte la arista, de vértices (p_j, p_r, p_k), y se dividen en cuatro triángulos: (p_i, p_j, p), (p_k, p_i, p), (p_j, p_r, p) y (p_r, p_k, p).

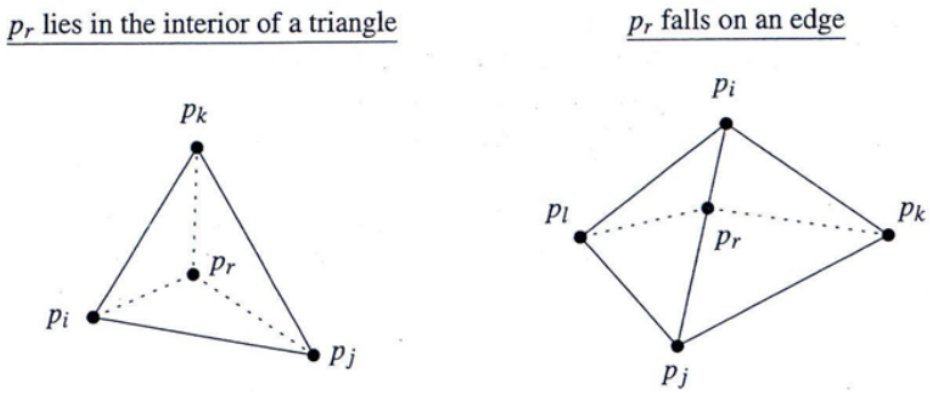


Figura 7: División de triángulos en tres o cuatro partes

Ambas divisiones se realizan construyendo un arreglo de triángulos con los vértices mencionados y luego se actualizan las referencias a los vecinos de cada triángulo implicado.

Es importante señalar que existe una diferencia fundamental entre los dos casos: dado que los triángulos mantienen su arreglo de vértices y vecinos en orientación antihoraria, la divisón en tres triángulos siempre es única y no depende de la forma en que se escogen los índices; por el contrario, la división en cuatro triángulos sí depende de la definición relativa de ambos triángulos, por lo que es necesario un tratamiento especial para describirla.

Sean N y M los triángulos implicados en el proceso de división en cuatro partes. Se definen los índices n y m como los vértices de N y M que se oponen a la arista compartida. Por ejemplo, el triángulo N de la figura 8 es descrito en memoria por la lista de vértices V_N = [v1, v2, v3] y tiene como índice n el 1, porque v2 (que se se opone a M) está en la posición 1de la lista. De este modo, el triángulo N puede ser redefinido como $V_N[1]$, $V_N[2]$ y $V_N[0]$ (se mantiene el orden antihorario) o, de forma más general, $V_N[n]$, $V_N[(n+1)\,\%3]$ y $V_N[(n+2)\,\%3]$.

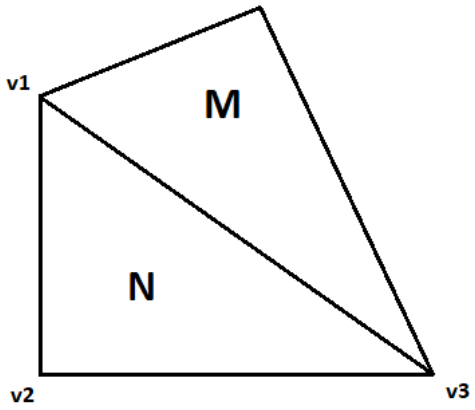


Figura 8: Notación para la división en cuatro partes

Esta nueva descripción de los triángulos permite que la actualización de las referencias tenga una descripción única, basada en n y m. Así, tras insertar el punto p, los cuatro triángulos nuevos se pueden describir con los siguientes vértices: $(V_N[n], V_N[(n+1)\,\%3], p)$, $(V_N[(n+2)\,\%3], V_N[n], p)$, $(V_M[m], V_M[(m+1)\,\%3], p)$ y $(V_M[(m+2)\,\%3], V_M[m], p)$

Finalmente, se agregan a la triangulación los nuevos triángulos generados. Cuando se realiza esta operación, se reutilizan los punteros de los antiguos triángulos, de manera que se asocian uno o dos de los triángulos nuevos al o a los triángulos de origen, según sea una división en tres o cuatro partes. De esta forma la lista de triángulos contiene exclusivamente los triángulos válidos y actualizados que conforman la triangulación y no es necesario borrar ningún puntero sobrante.

2.2.3. Legalización de aristas

Cada vez que se inserta un punto p a la triangulación se debe comprobar que se satisface la condición de Delaunay entre los nuevos triángulos y sus vecinos. Si algún par de triángulos no cumple la condición, se considera que su arista compartida es ilegal y debe ser legalizada.

Para esto, se comprueba que cada nuevo triángulo y su vecino exterior (esto es, aquel que se opone al punto insertado) satisfacen la condición de Delaunay (sección 2.3.2). En caso de que se determine que la arista que comparten es ilegal, se realiza un intercambio de arista, es decir, se reemplaza la arista ilegal por la otra arista diagonal del polígono. Luego, dado que esto modifica la condición de Delaunay en la vecindad, se legalizan recursivamente las

dos aristas exteriores E_1 y E_2 opuestas al punto p (ver figura 9).

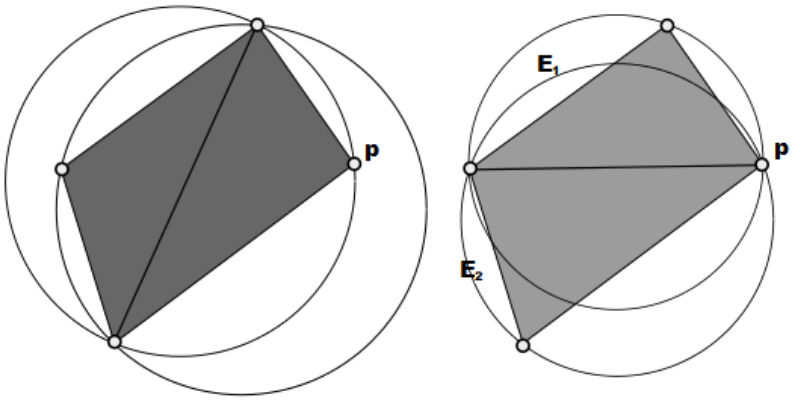


Figura 9: Intercambio de diagonal

El proceso de intercambio de diagonales es similar a la división de triángulos, en el sentido en el que se crean nuevas referencias a vértices y a vecinos que son superpuestas sobre los punteros de los triángulos originales y luego se actualizan las referencias de los vecinos de dichos triángulos. Además, se añade como restricción que la diagonal no debe ser intercambiada (la operación se cancela) en caso de que los triángulos resultantes no mantengan la orientación antihoraria de sus vértices, ya que se estaría obteniendo una situación ilegal de la triangulación introducida por un error de aproximación o, como se verá más adelante, por el propio algoritmo, cuando se operen aristas restringidas.

2.3. Operaciones geométricas

2.3.1. Cómo se asegura robustez

Un aspecto importante a considerar cuando se trabaja con algoritmos geométricos es la inexactitud debida a que se trabaja con aritmética finita, lo que puede generar problemas de robustez. Existen dos soluciones posibles para este problema: algoritmos de aritmética exacta o manejo de umbrales de error.

Este software emplea el método de umbrales de error, en otras palabras, utiliza un parámetro ϵ de error permisible, de manera que en situaciones de borde algunas condiciones geométricas pueden no ser completamente precisas, pero en compensación no se generan triangulaciones mal definidas.

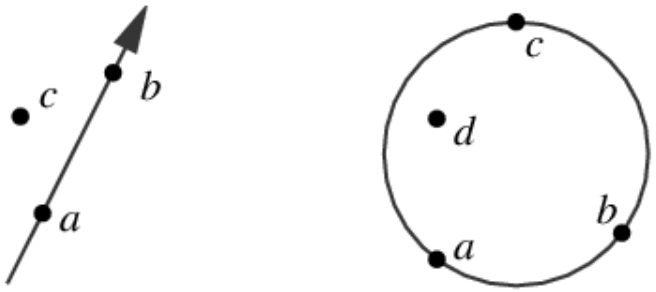
La implementación de este método permite relajar condiciones del estilo “igual a” usando un parámetro ϵ de error, de forma que una división en tres triángulos puede tornarse en una división en cuatro triángulos al considerar que el punto insertado está sobre una arista, cuando en realidad sólo es muy cercano a ella. Esto tiene como consecuencia que algunos casos límites pueden no llegar a cumplir la condición de Delaunay, pero a cambio se evita la construcción de triangulaciones inválidas.

2.3.2. Técnicas de orientación geométrica

El objetivo de esta sección es describir las técnicas utilizadas para realizar comprobaciones geométricas auxiliares para el proceso principal. Las dos técnicas descritas se basan en operaciones enunciadas por Shewchuk [4] que reducen el problema al cálculo de una determinante.

La primera técnica es el test de orientación 2D (*orient2D*), el cual recibe tres puntos y un valor ϵ como entrada. Los dos primeros puntos definen una recta orientada, de forma que el resultado de la operación consiste en determinar si el tercer punto está a la derecha, a la izquierda o sobre dicha recta, con error admitido ϵ (en la figura 10a, el punto c está a la izquierda de la recta ab). En el contexto del problema de Delaunay, este test es utilizado para determinar si un punto está contenido en un triángulo, lo que ocurre siempre que dicho punto está a la izquierda de cada una de las aristas del polígono gracias a la orientación antihoraria de los vértices.

La segunda técnica corresponde al test del círculo (*incircle*), el cual recibe cuatro puntos y un ϵ. Los tres primeros puntos corresponden a un triángulo y el resultado de la operación consiste en responder si el cuarto punto está dentro, sobre o fuera del circuncírculo formado por dicho triángulo, con error admitido ϵ (en la figura 10b, el punto d está dentro del circuncírculo del triángulo abc). Este test es utilizado para comprobar la legalidad de la arista que describe la condición de Delaunay.



(a) Test de orientación 2D (b) Test de círculo

Figura 10: Esquema de orientación de los test geométricos

Estos test se calculan empleando determinantes de 3x3 para la orientación 2D y de 4x4 para el circuncírculo [4], cuyo signo (considerando el uso del *epsilon* de error) verifica a que rango pertenece el punto.

2.4. Pseudocódigo del algoritmo Delaunay para un conjunto de puntos (algoritmo I)

Este algoritmo se basa en el procedimiento de inserción Delaunay descrito en el libro *Computational Geometry* [8].

Algoritmo 1: Triangulación de Delaunay no restringida

Input. A set P of n points
Output. A Delaunay triangulation of P

Let p_1, p_2 ***and*** p_3 ***be*** a suitable set of three points such that P is contained ***in*** the triangle $p\_1p\_2p\_3$
Initialize T as the triangulation consisting of the single triangle $p\_1p\_2p\_3$
Compute a random permutation $p_1, p_2, ...p_n$ of P
for each p ***in*** P ***do***
    InsertPoint(T, p)
end for
Discard p_1, p_2, p_3 with all their incident edges from T
return T

12

13 ***Function*** InsertPoint(T, p)

14 Find a triangle $p_ip_jp_k$ ***in*** T containing p_r.

15 ***if*** p_r lies ***in*** the interior of the triangle $p_ip_jp_k$ then

16 Add edges from p_r to the three vertices of $p_ip_jp_k$, thereby splitting $p_ip_jp_k$ into three triangles.

17 LegalizeEdge(p_r, p_ip_j, T)

18 LegalizeEdge(p_r, p_jp_k, T)

19 LegalizeEdge(p_r, p_kp_i, T)

20 ***else*** ($*$ p_r lies on an edge of $p_ip_jp_k$, say the edge p_ip_j*)

21 Add edges from p_r to p_k ***and*** to the third vertex p_i of the other triangle that is incident to p_ip_j into four triangles.

22 LegalizeEdge(p_r, p_ip_l, T)

23 LegalizeEdge(p_r, p_lp_j, T)

24 LegalizeEdge(p_r, p_jp_k, T)

25 LegalizeEdge(p_r, p_kp_i, T)

26 ***end if***

27

28 ***Function*** LegalizeEdge(T, p_r, p_ip_j)

29 ***if*** p_ip_j is illegal then

30 ***Let*** $p_ip_jp_k$ ***be*** the triangle adjacent to $p_rp_ip_j$ along p_ip_j

31 ($*$ Flip p_ip_j: $*$) Replace p_ip_j with p_rp_k

32 LegalizeEdge(p_r, p_ip_k, T)

33 LegalizeEdge(p_r, p_kp_j, T)

34 ***end if***

2.5. Resultados del algoritmo Delaunay para un conjunto de puntos (algoritmo I)

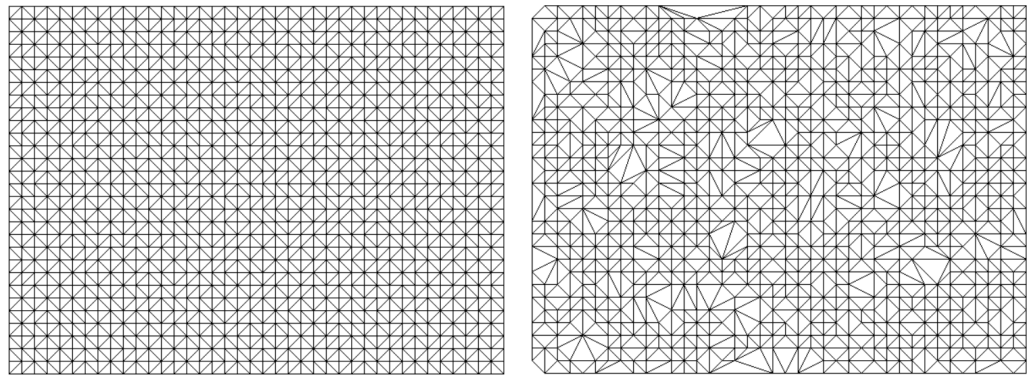
Para validar el funcionamiento del algoritmo I, que consiste en la generación incremental de una triangulación de Delaunay a partir de conjuntos de puntos, se comprobarán dos situaciones con parámetros aleatorios: conjunto de puntos seleccionados al azar de una grilla de control y conjunto de puntos en distribución aleatoria uniforme dentro de una región acotada.

2.5.1. Estructuras de datos consistentes

A continuación se comprobará empíricamente que el algoritmo I construye triangulaciones de Delaunay con estructuras de datos consistentes. Esto significa que:

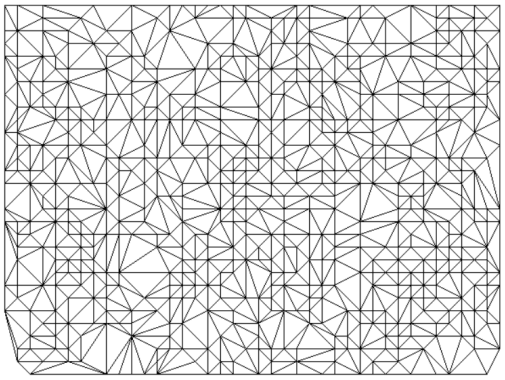
1. La triangulación resultante es una partición bien definida, sin intersecciones de aristas.
2. Todos los pares de triángulos cumplen con la condición de Delaunay.
3. La descripción de la vecindad de los triángulos está correctamente definida a través de una arista compartida.

Para este fin, se presentan algunos resultados de triangulaciones con una distribución de puntos sencilla y bien espaciada.

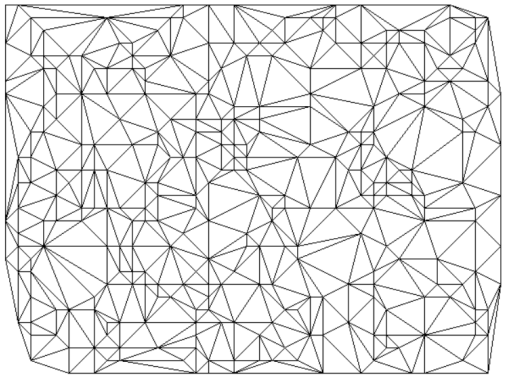


(a) 1200 puntos (Control)

(b) 900 puntos



(c) 600 puntos



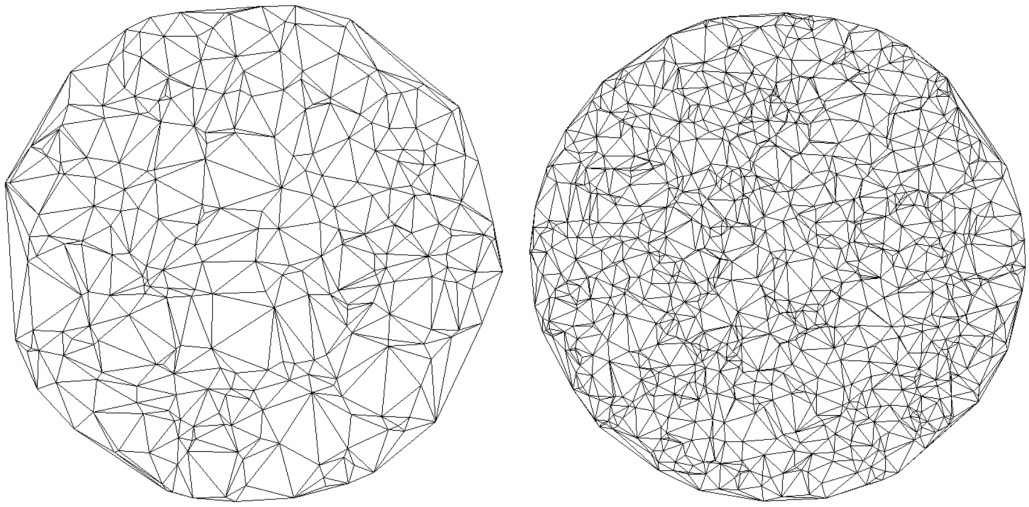
(d) 300 puntos

Figura 11: Puntos aleatorios en una grilla de 40x30

Se realizaron varias pruebas similares sobre conjuntos de puntos aleatorios y para cada caso la triangulación resultante cumplió con las condiciones exigidas.

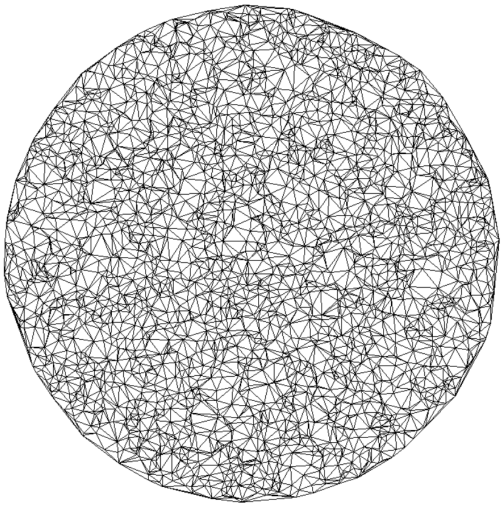
2.5.2. Manejo de aritmética con punto flotante

En esta sección se verificará que el algoritmo resuelve adecuadamente el problema de Delaunay con aritmética de punto flotante. Para esto, se empleará una distribución aleatoria de puntos con valor real y cotas en una región geométrica circular.

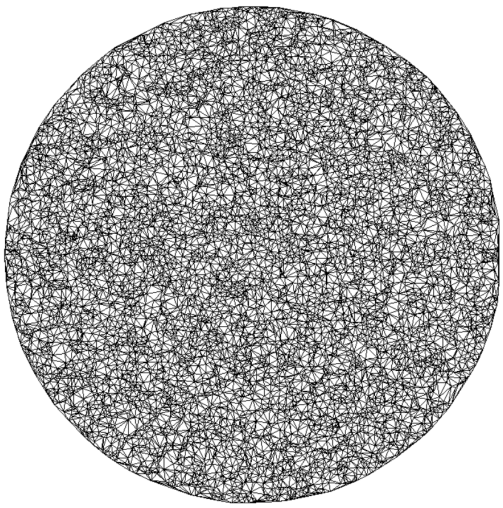


(a) 300 puntos

(b) 1k puntos



(c) 3k puntos



(d) 10k puntos

Figura 12: Puntos aleatorios en una región circular

Se realizaron decenas de pruebas para distintas cantidades de puntos aleatorios y en cada una de las pruebas el algoritmo generó una triangulación bien definida. Esto considera que las coordenadas podían tomar valores con punto flotante en cualquier posición de la región acotada, por lo que la evidencia empírica denota que las operaciones usadas en el algoritmo I son capaces de manejar dicha precisión.

3. Triangulación de Delaunay con aristas restringidas

El algoritmo Delaunay restringido maneja el problema anterior y añade la existencia de aristas restringidas, lo que significa que, además de insertar puntos a la triangulación, debe respetar la aparición de un conjunto de aristas incluso cuando se viole localmente la condición de Delaunay.

Así, el proceso de inserción de arista restringida tiene como objetivo adaptar la triangulación, sin agregar o quitar elementos, de forma que los triángulos describan naturalmente una arista dada.

La implementación de este algoritmo impone, sobre la estructura de datos, que la arista respetada debe mantenerse íntegra después de cualquier operación de inserción Delaunay, lo que implica que dicha arista no se puede reemplazar en un intercambio de diagonal y debe ser heredada correctamente cuando hay división de triángulos.

3.1. Preprocesos

Previo a la resolución del problema de inserción se realizan algunos procesos que simplifican el manejo de estructuras y validan los datos de entrada.

En primer lugar, se revisa que las aristas que deben ser respetadas no se intersecten entre sí. Si por el contrario, dos aristas se intersectan, se separa cada trozo como si fuera una operación diferente.

Luego, se revisa que los vértices de las aristas pertenezcan a la triangulación, ya que durante la inserción se asume su existencia. Debido a esto, cuando alguno de los vértices no está presente, se inserta como un punto más de la triangulación por medio del algoritmo de inserción Delaunay.

Finalmente, para simplificar el trabajo de inserción y reducir la cantidad de casos que se deben tratar, se revisa que no hayan puntos de la triangulación sobre la arista a insertar. Si, por ejemplo, al insertar la arista $\overline{p_1p_4}$ existen dos puntos p_2 y p_3 de la triangulación en

medio del trayecto de la arista, entonces se resuelven independientemente las inserciones de las aristas $\overline{p_1p_2}$, $\overline{p_2p_3}$ y $\overline{p_3p_4}$.

3.2. Inserción de aristas restringidas

Para cada subproblema de insertar una arista $\overline{p_ip_{i+1}}$ se computa el camino de triángulos que intersecta dicho segmento de manera ordenada (sección 3.2.1) y se recorre el camino intercambiando las diagonales de los triángulos consecutivos. Si el cuadrilátero formado por estos dos triángulos es cóncavo, entonces la operación de intercambio de arista produce una situación ilegal en la que los triángulos resultantes no se orientan en sentido antihorario, por lo que la operación se ignora.

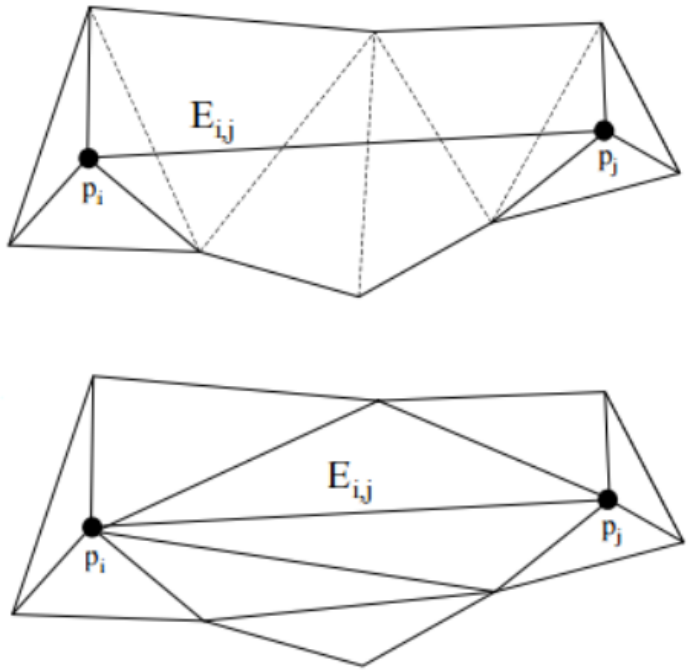


Figura 13: Proceso de respeto de la arista E_{ij}

La razón por la que pueden surgir casos en que un intercambio de arista produce una situación ilegal sobre la estructura de datos es porque la decisión de realizar dicha operación ya no depende de la condición de Delaunay, sino que se realiza en cada par de triángulos que se hayan en el camino. Esto incluye el caso del cuadrilátero cóncavo, para el cual la operación de intercambio de arista no tiene una definición establecida. Más aún, es necesario ignorar este caso para el correcto funcionamiento del algoritmo.

Finalmente, mientras no se genera la arista restringida, se repite el proceso computando el nuevo camino de triángulos intersectados e intercambiando sus aristas.

3.2.1. Encontrar triángulos intersectados por la arista restringida

Se empieza el algoritmo buscando el primer triángulo del conjunto que se intersecta con el segmento $\overline{p_i p_{i+1}}$. La búsqueda se realiza de la misma forma descrita en la sección 2.2.1, a través del algoritmo de Jump and Walk.

Luego, para encontrar el resto de triángulos, se recorren los vecinos del primero, se agrega a la lista aquel que intersecta al segmento y se repite el procedimiento para cada nuevo elemento agregado a la lista, hasta alcanzar el final del segmento.

Para determinar cómo se recorre el camino de triángulos en cada iteración se definen dos reglas:

1. El triángulo siguiente es el vecino del actual cuya arista compartida intersecta a la arista que se quiere respetar (ver figura 15).

2. El triángulo anterior queda marcado para que el algoritmo no regrese por el camino que ya recorrió.

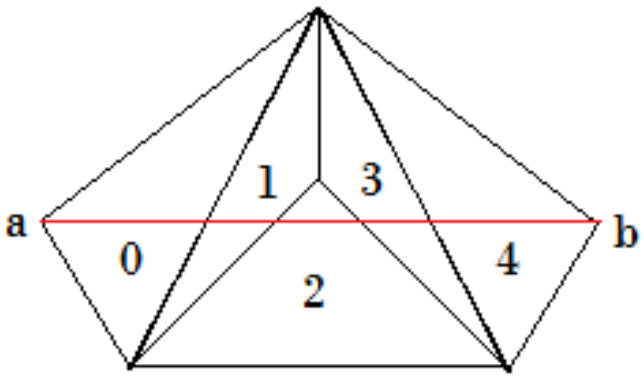


Figura 14: Orden para recorrer el camino de triángulos

Estas dos reglas aseguran que el algoritmo recorra el camino de triángulos de forma única y que alcance correctamente el extremo final.

3.3. Pseudocódigo del algoritmo Delaunay restringido (algoritmo II)

Algoritmo 2: Triangulación de Delaunay con aristas restringidas

Input. A set P of n points ***and*** a set E of m constrained edges

Output. A Delaunay triangulation of P with constrained edges ***in*** E

Let p_1, p_2 ***and*** p_3 ***be*** a suitable set of three points such that P is contained ***in*** the triangle $p\_1p\_2p\_3$.

Initialize T as the triangulation consisting of the single triangle $p\_1p\_2p\_3$

Compute a random permutation p_1,p_2,...,p_n of P.

for each p ***in*** P ***do***

    InsertPoint(T, p)

end for

for each e ***in*** E ***do***

    InsertEdge(T, e)

end for

Discard p_1, p_2 ***and*** p_3 with all their incident edges from T.

return T

Function InsertPoint(T, p)

    Find a triangle $p_ip_jp_k$ ***in*** T containing p_r.

    if p_r lies ***in*** the interior of the triangle $p_ip_jp_k$ then

        Add edges from p_r to the three vertices of $p_ip_jp_k$,

        thereby splitting $p_ip_jp_k$ into three triangles.

        LegalizeEdge(p_r, p_ip_j, T)

        LegalizeEdge(p_r, p_jp_k, T)

        LegalizeEdge(p_r, p_kp_i, T)

    else ($*$ p_r lies on an edge of $p_ip_jp_k$, say the edge p_ip_j*)

        Add edges from p_r to p_k ***and*** to the third vertex p_i of the

        other triangle that is incident to p_ip_j into four triangles.

LegalizeEdge(p_r, p_ip_l, T)

LegalizeEdge(p_r, p_lp_j, T)

LegalizeEdge(p_r, p_jp_k, T)

LegalizeEdge(p_r, p_kp_i, T)

end if

Function LegalizeEdge(T, p_r, p_ip_j)

if p_ip_j is illegal then

Let $p_ip_jp_k$ ***be*** the triangle adjacent to $p_rp_ip_j$ along p_ip_j

($*$ Flip p_ip_j: $*$) Replace p_ip_j with p_rp_k

LegalizeEdge(p_r, p_ip_k, T)

LegalizeEdge(p_r, p_kp_j, T)

end if

Function InsertEdge(T, e)

Let CP: p_1,p_2,...,p_n ***be*** the sorted set of all collinear points lying on the edge e.

for each p_i, p_{i+1} ***in*** CP ***do***

Find the path of triangles Tp: t_1,t_2,...,t_m intersecting the edge $\overline{p_ip_{i+1}}$.

while edge $\overline{p_ip_{i+1}}$ has not been built ***do***

for each t_j, t_{j+1} ***in*** Tp ***do***

Let p_1, p_2, p_3, p_4 ***be*** the points of the adjacent triangles t_j, t_{j+1} such that both are incident to $\overline{p_2p_3}$ (previously known as $\overline{p_ip_{i+1}}$.

($*$ Flip $\overline{p_2p_3}$: $*$) Replace $\overline{p_2p_3}$ with $\overline{p_1p_4}$ ***if*** possible

end for

end while

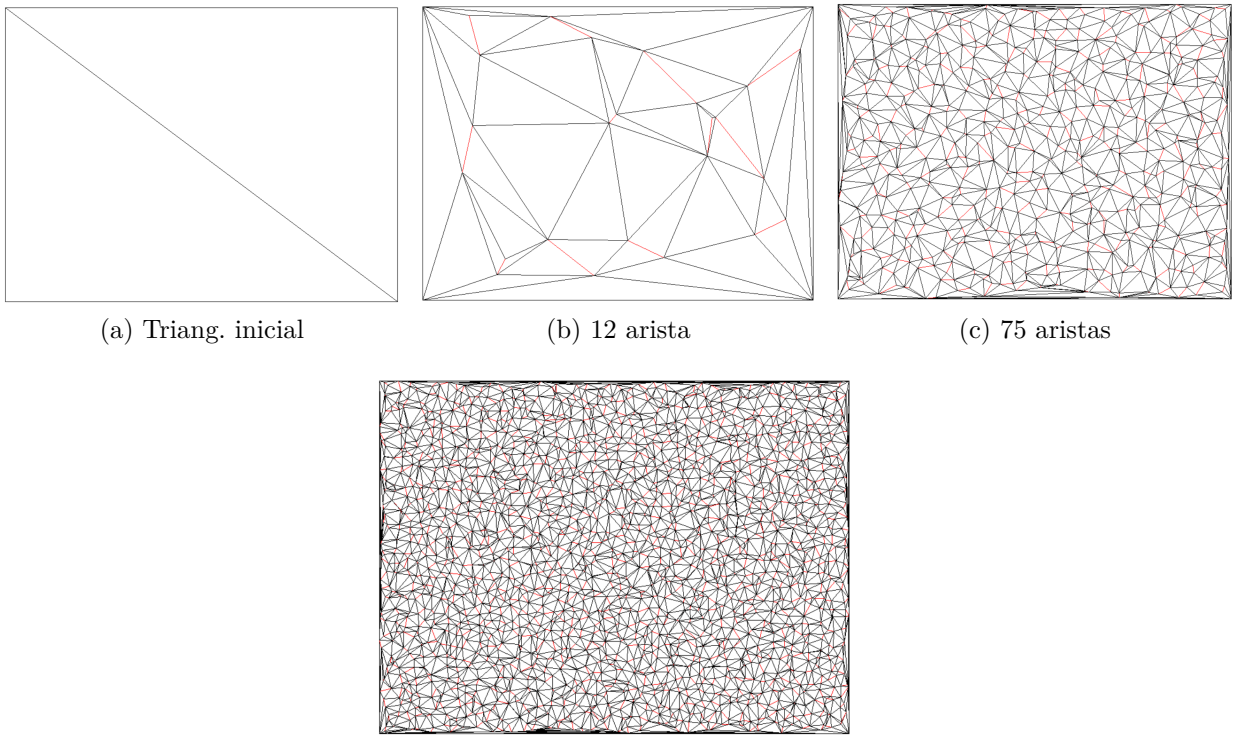
end for

3.4. Resultados del algoritmo Delaunay restringido (algoritmo II)

Para validar la implementación del algoritmo II, el cual consiste en la generación de una triangulación de Delaunay con aristas restringidas, se mostrará una serie de problemas test antes y después del proceso de respeto de aristas, los cuales comprobarán que la triangulación evoluciona de manera consistente en cada caso.

3.4.1. Geometrías con aristas restringidas sin intersecciones

A continuación se comprobará de forma empírica que la implementación del algoritmo II genera una triangulación válida y que las aristas son efectivamente respetadas. Para lograr esto, se plantea un problema sencillo en el que se distribuyen aristas aleatoriamente, pero de forma que no se intersecten entre si.



(a) Triang. inicial (b) 12 arista (c) 75 aristas

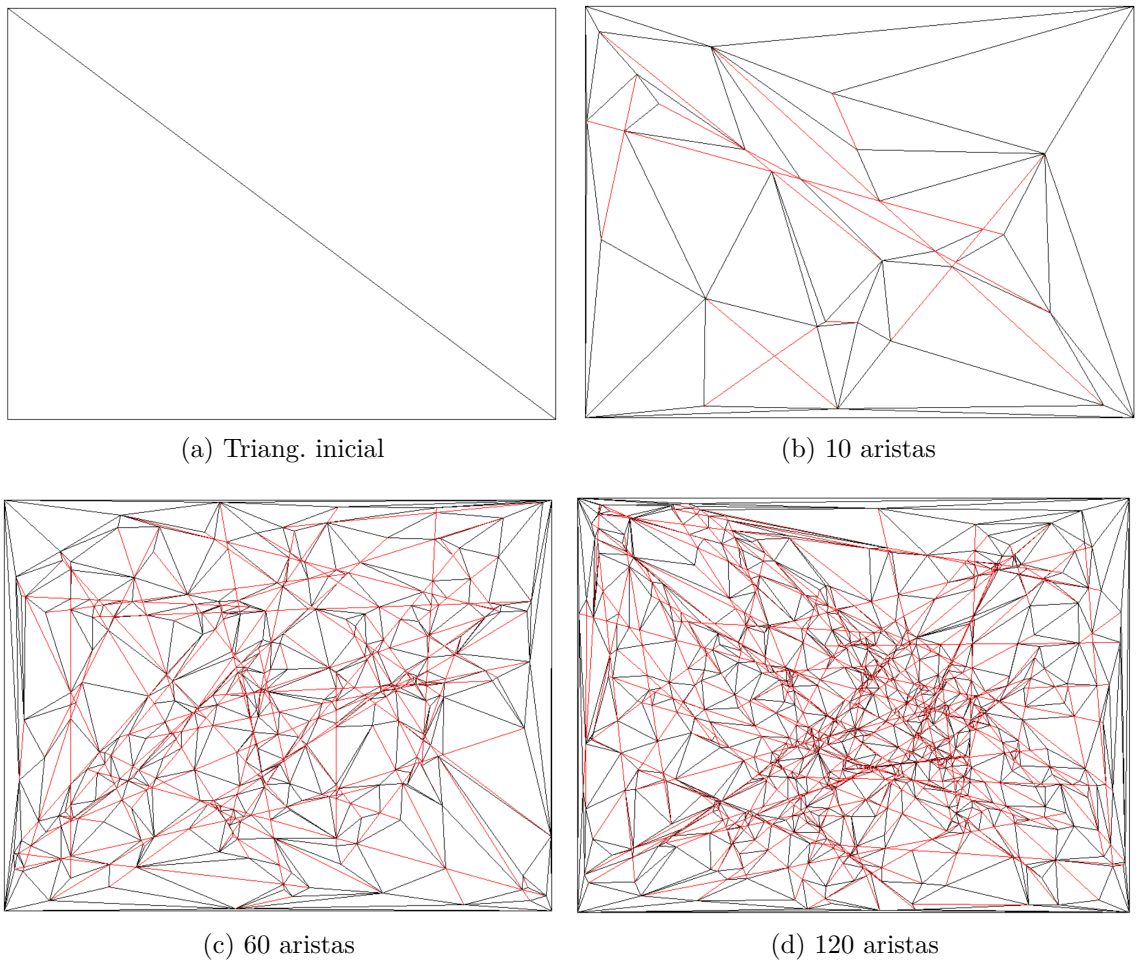
(d) 120 aristas

Figura 15: Aristas sin intersecciones

Las varias decenas de pruebas realizadas no mostraron ninguna triangulación cuya arista no pudiera ser respetada, bajo la condición dada de que no se permiten intersecciones.

3.4.2. Geometrías con aristas restringidas que pueden intersectarse

En esta sección se verificará empíricamente que el algoritmo también funciona cuando hay aristas restringidas intersectadas. Para esto, se plantea el mismo problema anterior de aristas distribuidas aleatoriamente, pero sin ninguna restricción para las intersecciones.



(a) Triang. inicial (b) 10 aristas

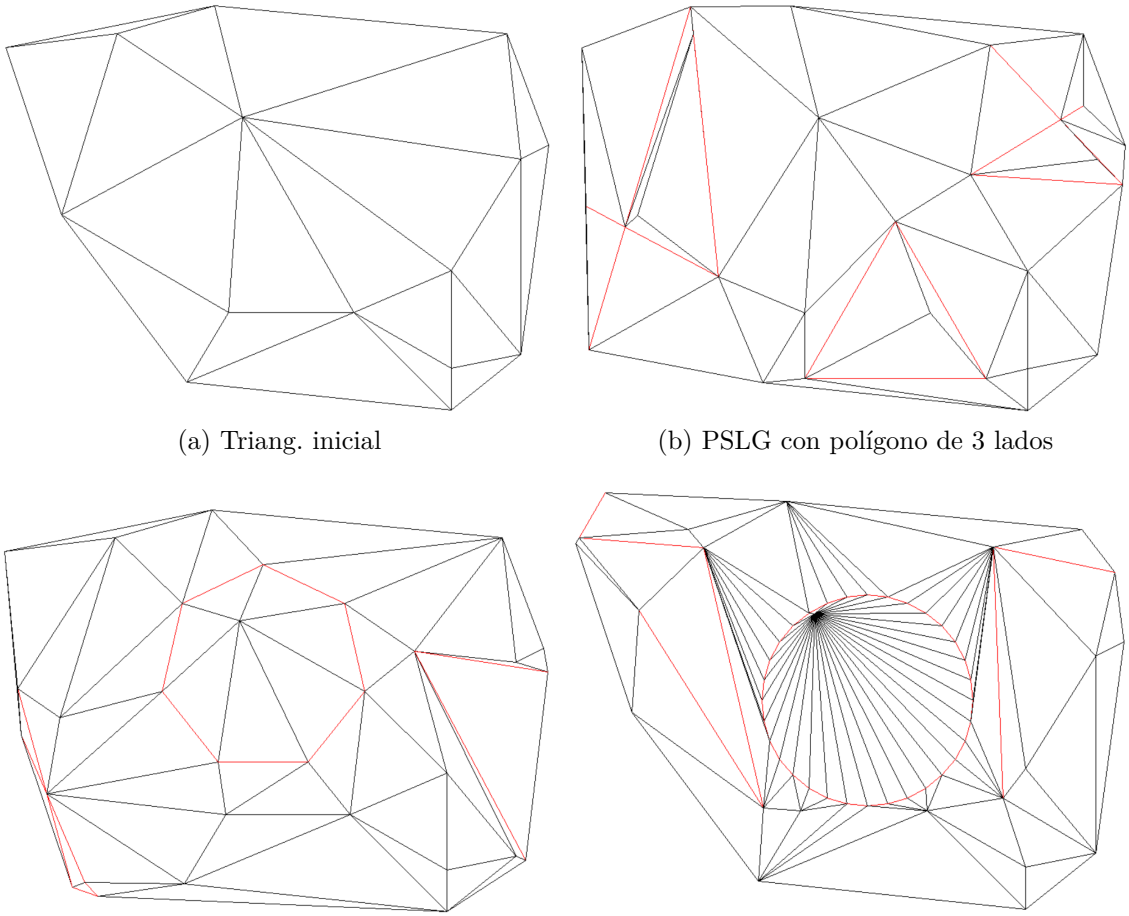
(c) 60 aristas (d) 120 aristas

Figura 16: Aristas que pueden intersectarse

Las triangulaciones generadas en pruebas con hasta cientos de aristas mostraron consistencia en sus estructuras de datos, vale decir, la restricción de las aristas fue correctamente heredada por los triángulos generados y bien compartida por los triángulos vecinos.

3.4.3. PSLGs generales

A continuación se estudiará de forma empírica el proceso de respeto de aristas aplicado a PSLGs. Para esto, se utilizarán conjuntos de puntos aleatorios y aristas que forman figuras abiertas y cerradas.



(a) Triang. inicial

(b) PSLG con polígono de 3 lados

(c) PSLG con polígono de 7 lados

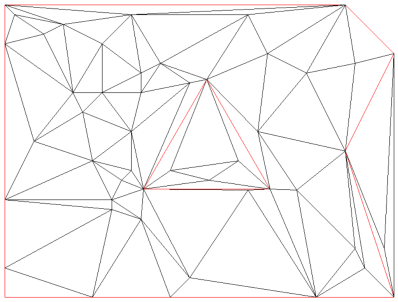
(d) PSLG con polígono de 32 lados

Figura 17: PSLGs con figuras abiertas y cerradas

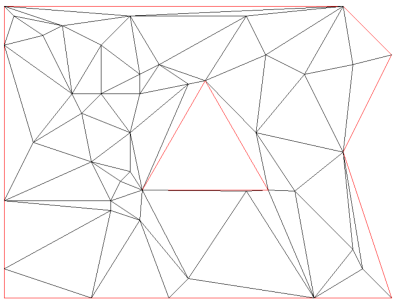
A través de decenas de pruebas con geometrías similares a la figura 17, se comprobaron las mismas condiciones de validación de las estructuras de datos que para el caso anterior, es decir, correcta herencia del estatus de arista restringida y consistencia de los datos entre triángulos vecinos.

3.4.4. PSLGs con borde definido

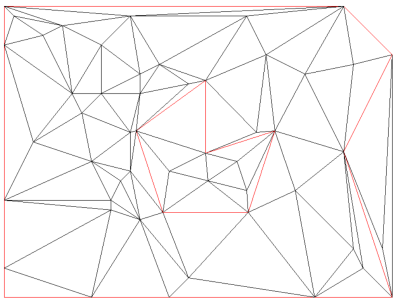
A continuación se muestran algunos resultados del algoritmo de eliminación de triángulos exteriores, el cual se encarga de descartar aquellos triángulos que están fuera del borde definido para el PSLG. Para esto, se emplearán PSLGs con aristas definiendo polígonos exteriores e interiores que actuarán como borde para la triangulación.



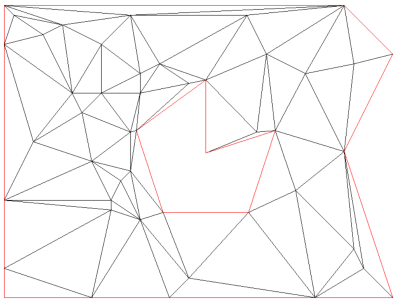
(a) Polígono de 3 lados sin respeto de borde



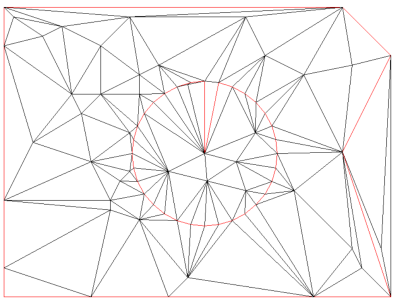
(b) Polígono de 3 lados con respeto de borde



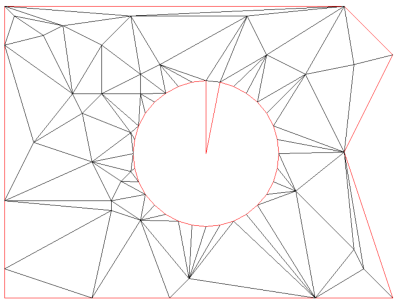
(c) Polígono acuñado de 5 lados sin respeto de borde



(d) Polígono acuñado de 5 lados con respeto de borde



(e) Polígono acuñado de 32 lados sin respeto de borde



(f) Polígono acuñado de 32 lados con respeto de borde

Figura 18: PSLGs con polígonos como borde

4. Algoritmo de refinamiento Lepp Centroide

El problema de construir una triangulación de Delaunay de buena calidad se basa en imponer a cada triángulo que su ángulo mínimo sea mayor o igual a un ángulo de tolerancia θ_{tol}, de forma que no se generen ángulos pequeños. La figura 20 ilustra la triangulación de Delaunay restringida de un PSLG que incluye un conjunto de triángulos de mala calidad debido a que la distribución de puntos es inadecuada.

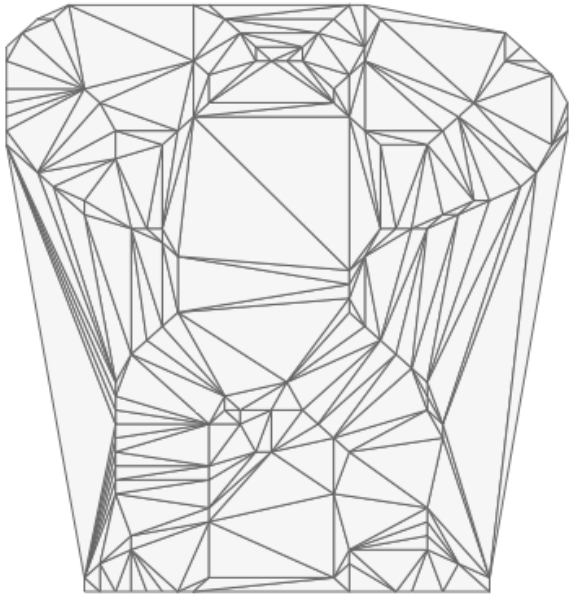


Figura 19: Triangulación donde predominan los triángulos de mala calidad

Los triángulos de mala calidad son indeseables porque no aproximan bien las geometrías, pero aparecen inevitablemente si la triangulación es limitada a un conjunto no representativo de puntos. Una solución para este problema sería agregar más puntos uniformemente en toda la geometría, pero esto es ineficiente tanto a nivel de uso de memoria como para los requerimientos de otras aplicaciones.

La forma de abordar esta situación es la siguiente: dada una triangulación de mala calidad y un parámetro de calidad θ_{tol} que se quiere alcanzar, se realiza un proceso de refinamiento en el que se insertan estratégicamente puntos con el algoritmo de Delaunay hasta respetar dicha restricción. De esta manera, se mejora la distribución de puntos sobre la geometría y,

en consecuencia, se obtienen triángulos de buena calidad.

El grupo de algoritmos Lepp-Delaunay resuelve eficientemente [5] el problema de refinamiento en tres pasos: la localización de los triángulos de mala calidad (sección 4.1), el mejoramiento de los triángulos a través del cálculo del Lepp y la inserción de un punto adecuado (sección 4.2) y la actualización de la estructura de datos (sección 4.3).

En la referencia [1] se ha demostrado que el algoritmo Lepp-Delaunay Centroide converge para $\theta_{tol} \leq 30°$ y en la práctica converge hasta los 35°. Además, los resultados de esta investigación respaldan que dicho algoritmo requiere de menos inserciones de triángulos que el algoritmo de Ruppert [11] para respetar el mismo parámetro de calidad θ_{tol} y no necesita ordenar los triángulos por calidad antes de procesarlos. Por las razones anteriores, esta herramienta implementa el algoritmo Lepp-Delaunay Centroide para el refinamiento de triangulaciones.

4.1. Encontrar los triángulos de mala calidad

Los triángulos de mala calidad se definen de forma que su ángulo mínimo es menor a un umbral θ_{tol} dado. Esto significa que ninguno de los triángulo de la triangulación puede poseer ángulos menores a θ_{tol}, a excepción de aquellos formados por aristas restringidas.

El algoritmo recorre toda la triangulación en búsqueda de triángulos de mala calidad y los almacena en una estructura S que se va actualizando a medida que se mejoran.

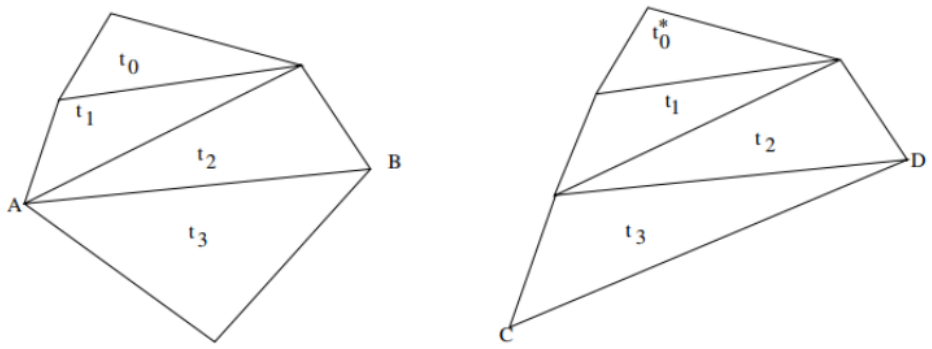
4.2. Mejorar los triángulos de mala calidad

El primer paso para mejorar un triángulo es encontrar su Lepp.

Para cualquier triángulo t_0, se define el *Longest Edge Propagation Path* de t_0 ($Lepp(t_0)$) como la lista ordenada de todos los triángulos $t_0, t_1, t_2, ..., t_{n-1}, t_n$, tal que t_i es triángulo vecino de t_{i-1} por la arista más larga de t_{i-1}, para $i = 1, 2, ..., n-1, n$ [6].

Como consecuencia de esta definición, el Lepp acaba cuando dos triángulos consecutivos comparten la misma arista más larga, o bien, se halla un borde de la triangulación.

Se definen entonces los triángulos terminales como los dos últimos triángulos del Lepp (ver figura 21a) o sólo el último si se termina en un borde (ver figura 21b) y, además, denotamos "arista terminal" a la arista compartida en el caso (a) o el borde en el caso (b).



(a) Lepp con arista terminal interior

(b) Lepp con arista terminal en el borde

Figura 20: Criterios de término del Lepp

Para mejorar el triángulo se necesita manejar tres casos de forma independiente:

1. Si la arista terminal es restringida, o bien, es un borde, se inserta Delaunay su punto medio, lo que se maneja dividiendo la arista en dos partes que conservan la restricción.

2. Si la segunda arista más larga de alguno de los triángulos terminales es restringida, entonces se inserta Delaunay el punto medio de dicha arista. Este criterio de inserción asegura que los triángulos de mala calidad mejoran más rápido en comparación a versiones anteriores del algoritmo Lepp-Delaunay [1, 2, 3, 6, 7].

3. Finalmente, si no se cumple ninguna de las condiciones anteriores, simplemente se inserta Delaunay el centroide del cuadrilátero formado por los triángulos terminales.

El algoritmo itera hasta que el triángulo inicial resulta mejorado, es decir, cumple con la condición de ángulo interior. Tras esto, se actualiza el conjunto S, se extrae un nuevo triángulo malo y se repite el algoritmo de mejoramiento hasta que el conjunto se queda sin elementos, o dicho de otra forma, se han mejorado todos los triángulos.

4.3. Actualización del conjunto S

Después de cada inserción Delaunay, se agregan nuevos triángulos a la triangulación y se modifican otros ya existentes. Debido a esto, el conjunto S de triángulos de mala calidad cambia y se debe actualizar en cada iteración del algoritmo.

La actualización se lleva a cabo simplemente agregando al conjunto S los triángulos modificados o generados por la inserción, si es que estos son de mala calidad. Por el contrario, no es necesario comprobar si alguno de estos triángulos ya pertenecía al conjunto S y resulta mejorado por la iteración, ya que el algoritmo de refinamiento se deshace de él en el momento en el que intenta mejorarlo.

Para determinar cuales son los triángulos modificados o generados por la inserción Delaunay se emplea una propiedad matemática que establece que estos elementos definen una cavidad con forma de estrella y centro en el punto insertado. Por lo tanto, el algoritmo verifica la calidad de los nuevos triángulos que convergen en el punto que se acaba de insertar.

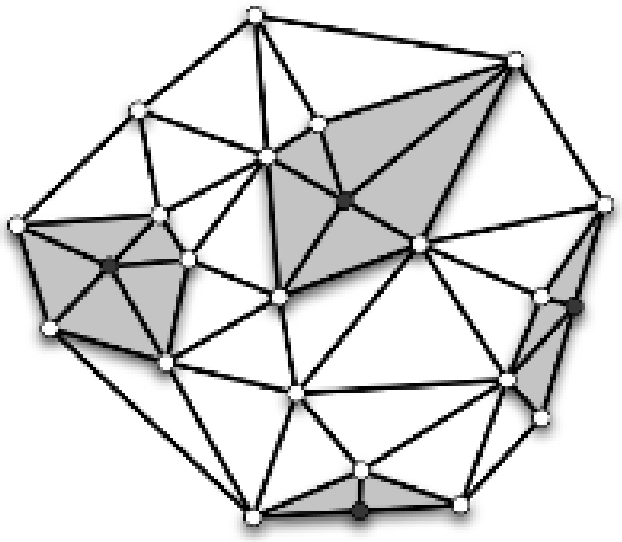


Figura 21: Cavidades formadas por la inserción Delaunay de puntos

4.4. Manejo de ángulos restringidos

Un caso particular que ocurre en las triangulaciones de Delaunay con aristas restringidas es la aparición de aristas intersectadas formando un ángulo pequeño, lo que, si bien no tiene ninguna interpretación especial para el algoritmo de respeto de aristas, presenta un problema para el algoritmo de refinamiento, ya que este ángulo está restringido por las aristas que lo conforman y no puede ser mejorado.

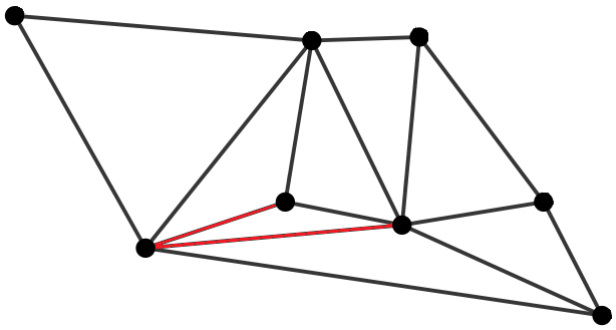


Figura 22: Ángulo restringido de mala calidad

Si el algoritmo III se aplicara sin ninguna modificación a un PSLG con ángulo restringido menor que θ_{tol}, entonces se quedaría en bucle intentando refinarlo. Esto se debe a que, por definición, un ángulo restringido no es mejorable, pero es posible controlar el tamaño deseado del triángulo que lo contiene. Para esto, es necesario añadir al algoritmo un método que maneje esta situación de forma particular.

Se define un parámetro δ que determina la longitud de arista máxima permitida para un ángulo restringido antes de romper forzosamente el bucle de refinamiento. De esta manera, un triángulo con ángulo restringido y lados menores a δ es marcado como no-refinable para el resto del algoritmo.

4.5. Pseudocódigo del algoritmo Lepp-Centroide (algoritmo III)

En esta memoria se implementa el algoritmo descrito y estudiado en la referencia [1], el cual ha sido extendido para manejar ángulos interiores restringidos menores que 30°.

Algoritmo 3: Refinamiento Lepp Centroide

Input. Triangulation T_i associated to PSLG data, tolerance angle θ_{tol} ***and*** maximum edge length δ.
Output. Refined triangulation T_f with angles $\geq \theta_{tol}$.

(∗ Find S set of bad quality triangles ∗)
for each t ***in*** T ***do***
    if CheckBadQuality(t, θ_{tol}, δ) then
        Add t to the set S
    end if
end for

for each t ***in*** S (***while*** $S \neq \emptyset$) ***do***
    while t is refinable ***and*** remains unrefined ***do***
        Use Lepp(t) to find terminal triangles t_1, t_2 (of smallest angles $\alpha_0(t1)$, $\alpha_0(t2)$) ***and*** terminal edge E.
        if E is constrained (this includes t_2 ***null***) then
            Set P as the midpoint of E
        else
            if there exists t (t_1 ***or*** t_2) such that $\alpha_0(t) < \theta_{tol}$ ***and*** second longest edge L is constrained then
                Set P as the midpoint of L
            else
                Set P as the centroid of the terminal triangles t_1, t_2
            end if
        end if

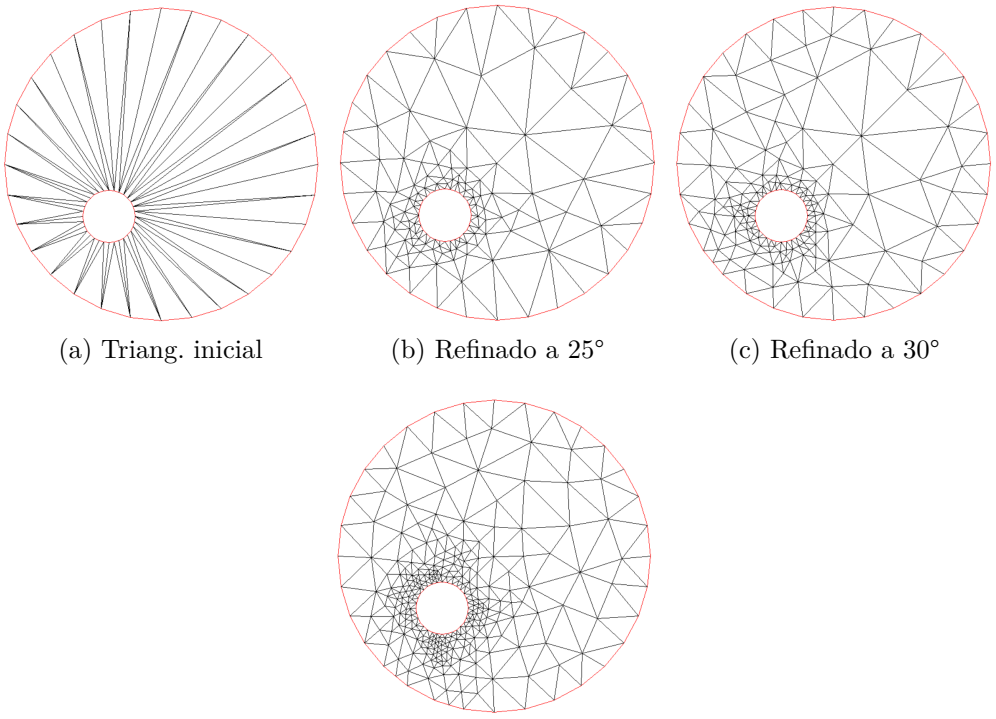
```
23          Perform Constrained Delaunay insertion of P
24
25          (* Update S *)
26          for each t that contains P do
27              if CheckBadQuality(t, θ_tol, δ) then
28                  Add t to the set S
29              end if
30          end for
31
32      end while
33  end for
34
35  Function CheckBadQuality(t, θ_tol, δ)
36      Let α_0(t) be the smallest angle of t
37      if t is refinable and α_0(t) < θ_tol then
38          Let L_1 and L_2 be the edges that make up the angle α_0(t)
39          if L_1 and L_2 are constrained (this includes null) and smaller than δ then
40              Set t as non−refinable
41              return false
42          return true
43      return false
```

4.6. Resultados del algoritmo Lepp-Centroide (algoritmo III)

Para comprobar algunas de las características del algoritmo III se realizaron pruebas con datos de problemas construidos para esta validación y problemas obtenidos de repositorios. A continuación, se presentan algunos de los resultados más representativos.

4.6.1. Refinamiento de PSLGs generales

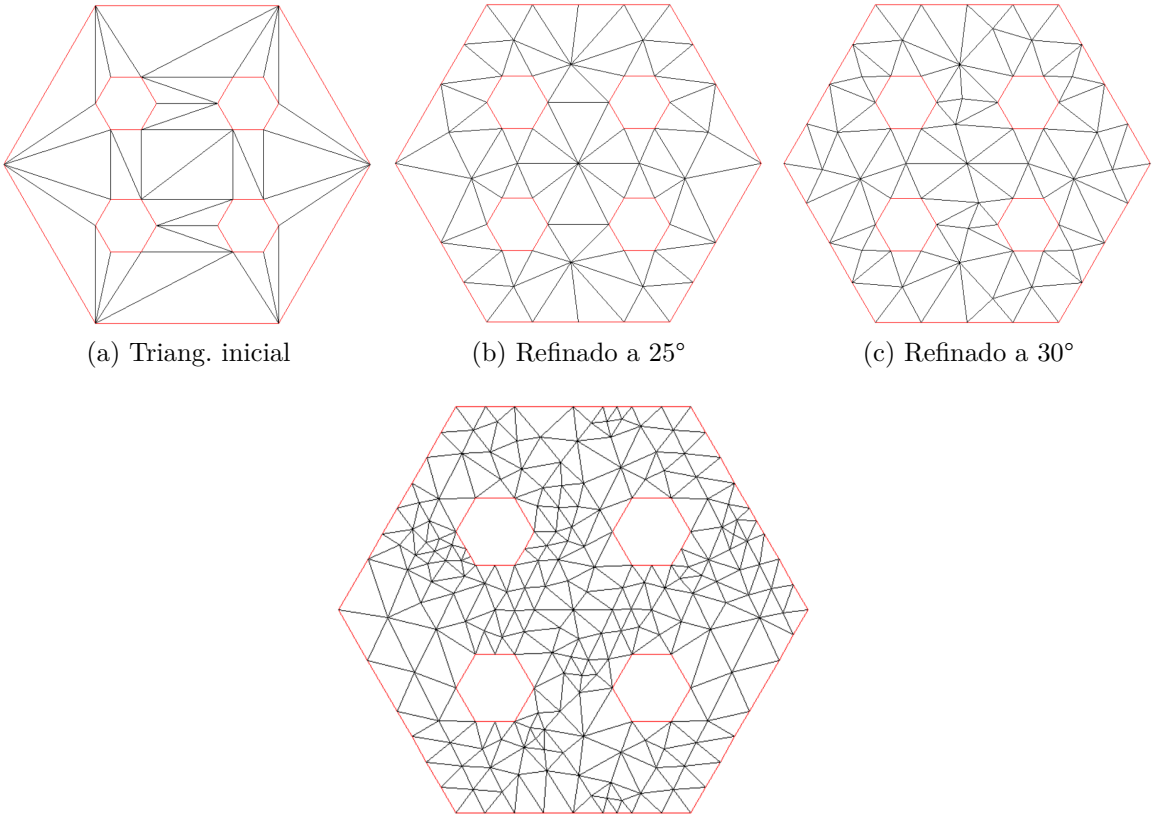
En esta sección se estudiará el comportamiento empírico del algoritmo III al generar triangulaciones de buena calidad, para ver si se correlaciona con lo establecido en estudios previos [1, 2, 5]. Para esto, se presentan algunos resultados del algoritmo III aplicado a datos de entrada como los usados en los algoritmos I y II, es decir, PSLGs generales con puntos, aristas restringidas y bordes definidos.



(a) Triang. inicial (b) Refinado a 25° (c) Refinado a 30°

(d) Refinado a 35°

Figura 23: Círculo aproximado con hueco circular



(a) Triang. inicial (b) Refinado a 25° (c) Refinado a 30°

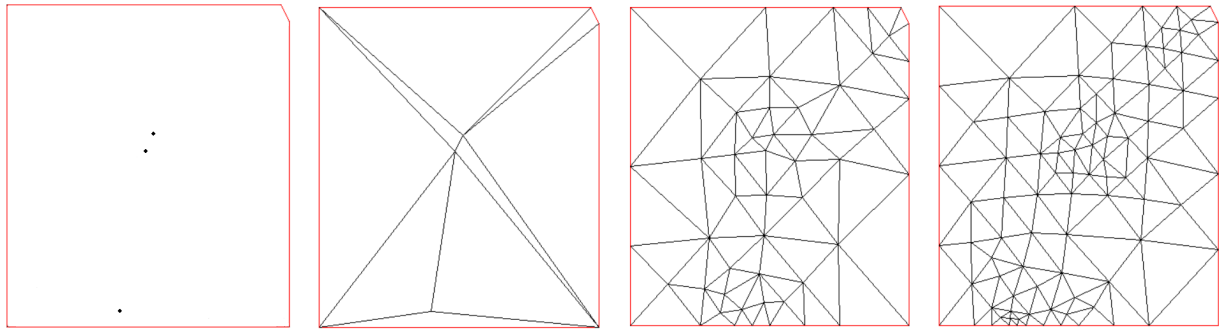
(d) Refinado a 35°

Figura 24: Hexágono con 4 huecos hexagonales

Se realizaron pruebas para geometrías similares a las de las figuras 23 y 24, corroborándose que en la práctica el algoritmo funciona consistentemente para ángulos de hasta 35°. Más aún, algunos de los resultados consiguieron alcanzar ángulos superiores, aunque no de forma sistemática.

4.6.2. Problemas test de geometrías conocidas

En esta sección se estudiará empíricamente el uso del algoritmo III en problemas test obtenidos de repositorios típicos para el testeo de algoritmos Delaunay.

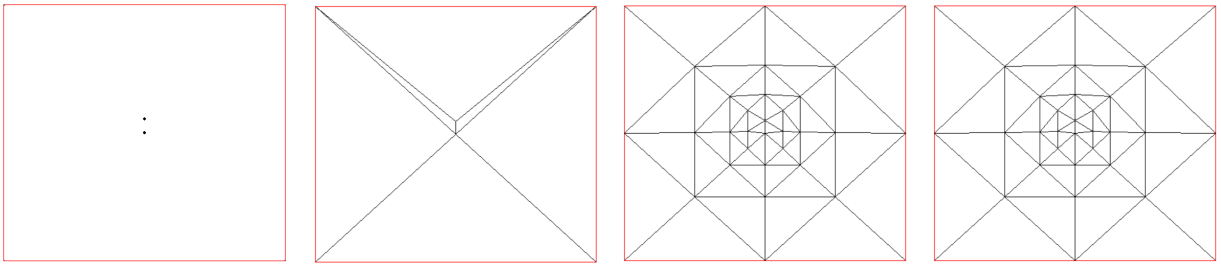


(a) Caso 1 PSLG de entrada

(b) Caso 1 Triang. inicial

(c) Caso 1 refinado a 30°

(d) Caso 1 refinado a 35°



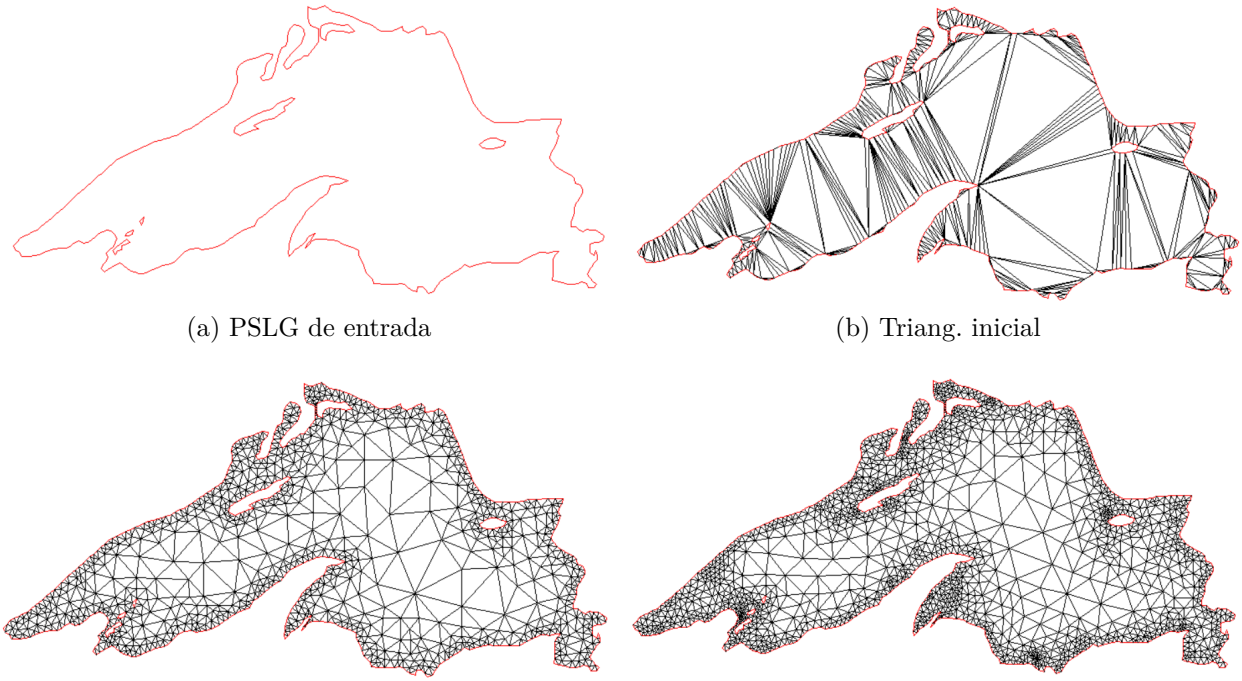
(e) Caso 2 PSLG de entrada

(f) Caso 2 Triang. inicial

(g) Caso 2 refinado a 30°

(h) Caso 2 refinado a 35°

Figura 25: *Needle Triangles Problem*



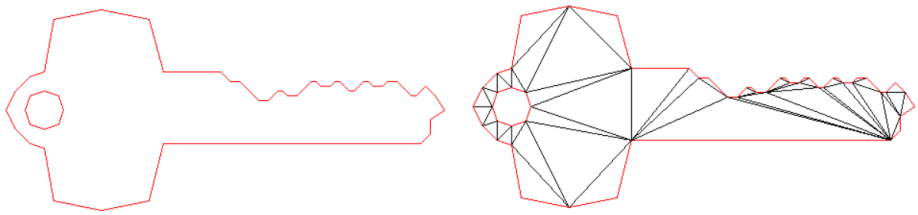
(a) PSLG de entrada

(b) Triang. inicial

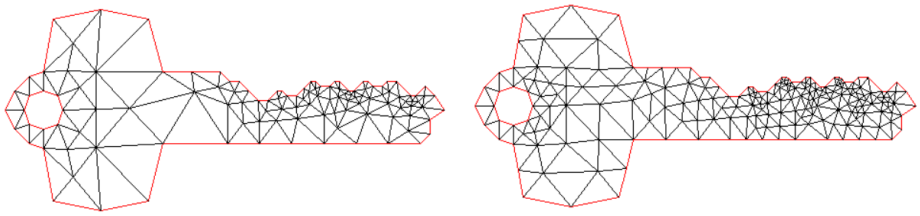
(c) Refinado a 30°

(d) Refinado a 35°

Figura 26: *Superior Lake*

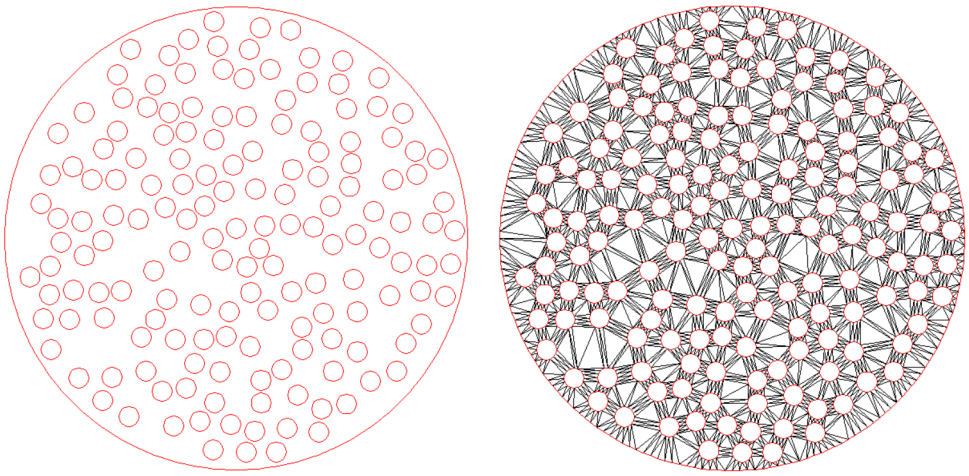


(a) PSLG de entrada (b) Triang. inicial

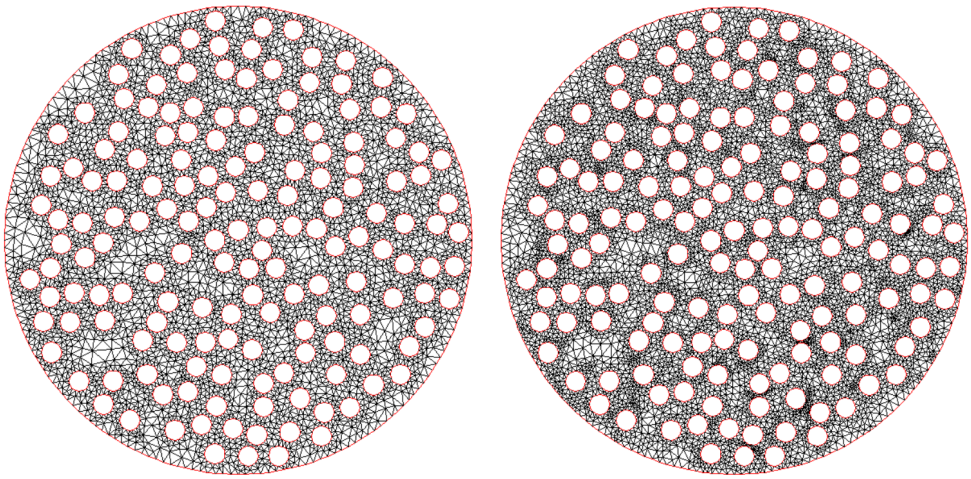


(c) Refinado a 30° (d) Refinado a 35°

Figura 27: *Key*



(a) PSLG de entrada (b) Triang. inicial



(c) Refinado a 30° (d) Refinado a 35°

Figura 28: *Neuss*

La tabla 1 compara los tamaños de las triangulaciones (número de triángulos) obtenidas por la implementación realizada en esta memoria y la tabla 2 de la referencia [1] para los casos mostrados en las figuras 25-28. A partir de esta comparación se concluye que la implementación actual maneja el mismo orden de inserciones de triángulos que lo mostrado en los estudios previos, con una pequeña diferencia entre los resultados que se atribuye a la aleatoriedad del orden de mejoramiento de los triángulos.

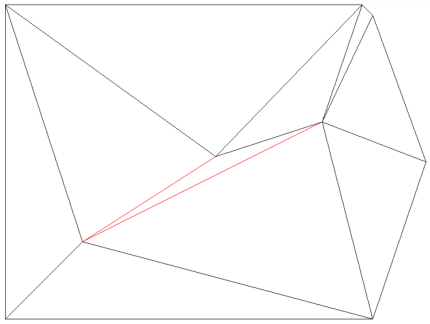
Problema	Sin refinar		Refinado a 30°		Refinado a 35°	
	Actual	Ref [1].	Actual	Ref [1]	Actual	Ref [1]
Needle-T	9	N/A	83	N/A	161	N/A
Needle-2T	6	N/A	54	54	54	54
Key	54	N/A	179	170	330	349
Neuss	3070	N/A	8385	8338	12691	12742
Sup. Lake	528	N/A	1840	1835	3117	3017

Tabla 1: Comparación del tamaño de las triangulaciones generadas por la herramienta con discusiones anteriores del algoritmo.

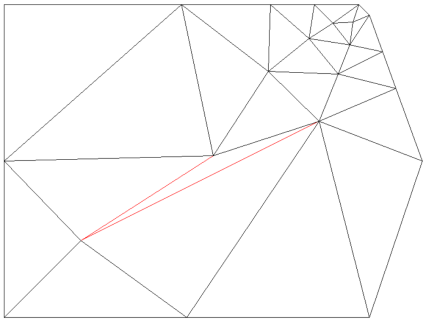
4.6.3. Geometrías con ángulos restringidos

Para verificar que el algoritmo III maneja correctamente el problema de los ángulos restringidos (ángulos formados por aristas restringidas), se probó la implementación con diversas geometrías seleccionadas especialmente para el caso. Cada prueba corroborará un patrón particular de distribución de los ángulos restringidos para diferentes valores del parámetro δ (longitud de arista máxima permitida para un ángulo restringido).

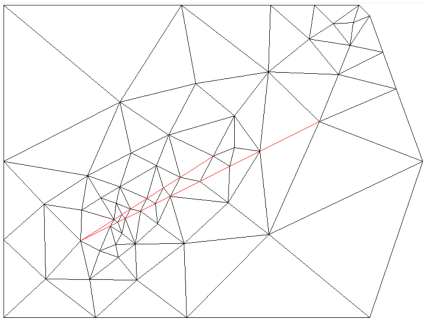
Para cada caso se realizaron varias iteraciones con distintos ángulos de tolerancia, aunque para mayor claridad se presentan a continuación los resultados para $\theta_{tol} = 30$ y algunos δ representativos.



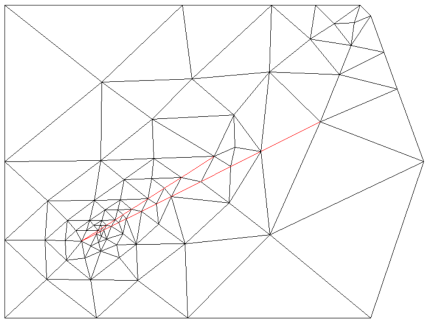
(a) Triang. inicial



(b) Refinado con $\delta = 1000$

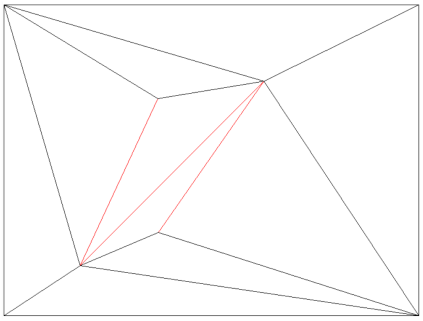


(c) Refinado con $\delta = 100$

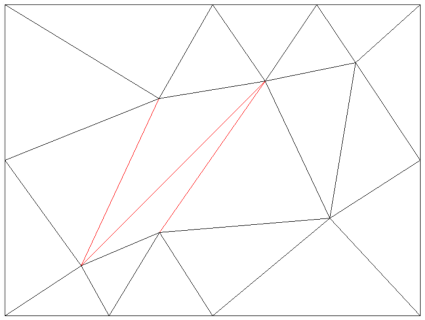


(d) Refinado con $\delta = 50$

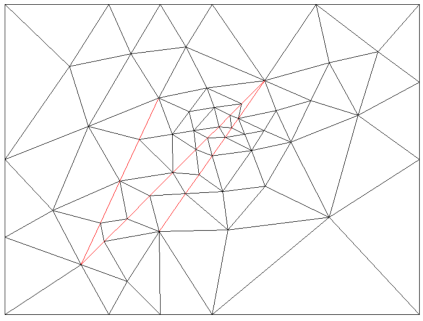
Figura 29: Ángulo simple



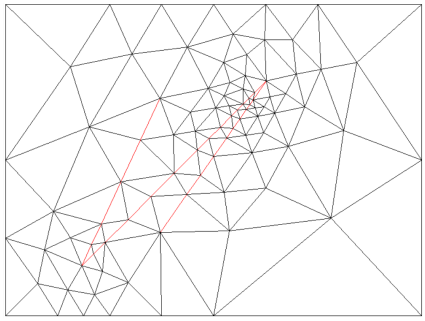
(a) Triang. inicial



(b) Refinado con $\delta = 1000$

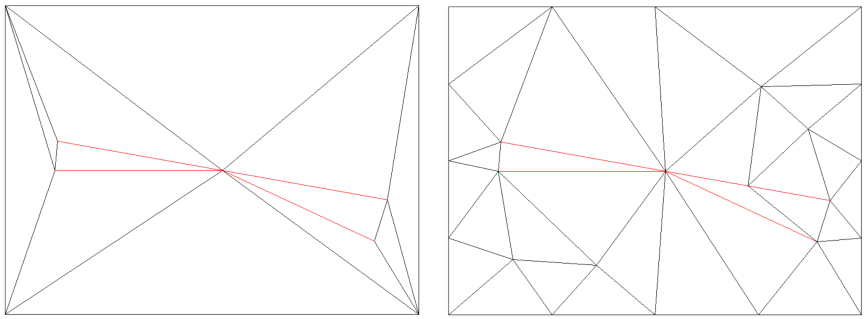


(c) Refinado con $\delta = 100$



(d) Refinado con $\delta = 50$

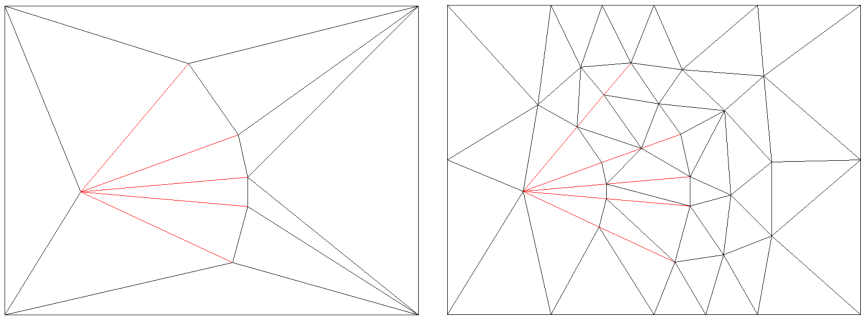
Figura 30: Ángulos adyacentes y opuestos



(a) Triang. inicial (b) Refinado con $\delta = 1000$

(c) Refinado con $\delta = 100$ (d) Refinado con $\delta = 50$

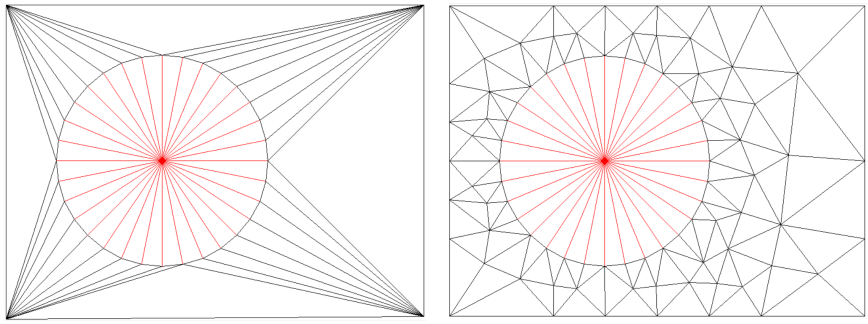
Figura 31: Ángulos opuestos por un vértice



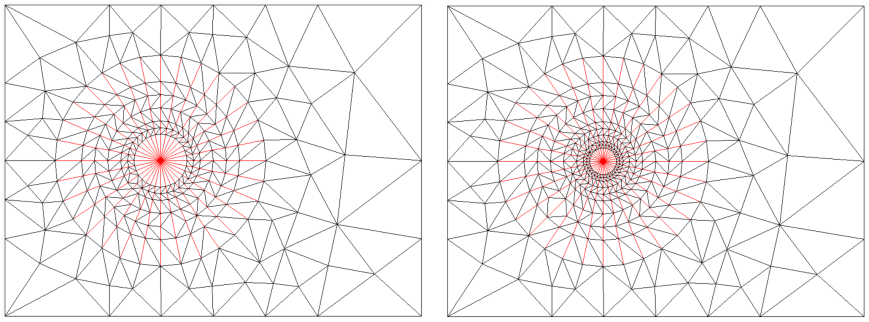
(a) Triang. inicial (b) Refinado con $\delta = 1000$

(c) Refinado con $\delta = 100$ (d) Refinado con $\delta = 50$

Figura 32: Ángulos concéntricos (arco)



(a) Triang. inicial (b) Refinado con $\delta = 200$



(c) Refinado con $\delta = 80$ (d) Refinado con $\delta = 40$

Figura 33: Ángulos concénctricos (círculo)

Cada una de estas pruebas (figuras 29-33) muestra un patrón particular de distribución de ángulos restringidos. Para estos casos, el resultado empírico permite concluir que la herramienta evita la propagación de ángulos restringidos (no mejorables), es decir, el número de triángulos con ángulo restringido no aumenta debido al refinamiento. Más aún, se puede disminuir el tamaño de dichos triángulos a costo de refinar más la triangulación.

5. Conclusión

La herramienta desarrollada en este trabajo es una solución con un nivel de diseño y gestión de los recursos en línea con los estándares de desarrollo de un proceso de ingeniería. En consecuencia la herramienta pueda ser empleada por ingenieros y científicos para la validación de nuevos algoritmos, los cuales permitirán que se amplíe el conjunto de problemas a abordar y se mejore el desempeño de software.

La herramienta propuesta implementa algoritmos bien respaldados en términos de eficiencia y robustez, en particular resolviendo problemas que otros algoritmos, o bien no manejan, o bien lo hacen bajo fuertes condiciones geométricas. A continuación, se listan las soluciones que los algoritmos implementados son capaces de entregar.

1. Respeto de aristas con múltiples intersecciones
2. Refinamiento consistente de ángulos hasta los 35°
3. Manejo de ángulos interiores restringidos de cualquier tamaño y varios tipos de distribuciones.
4. Construcción de triangulaciones de buenas calidad y tamaño óptimo para una tolerancia dada.

Además, esta herramienta entrega un valor agregado importante frente a otras implementaciones de algoritmos geométricos. La simplicidad de su diseño, tanto a nivel de estructuras de datos como de selección de algoritmos, implica que se hace más sencilla la tarea de intervenir y mejorar las soluciones existentes. Dicha extensibilidad se logra siguiendo un paradigma común en cada parte de la implementación: la subdivisión del problema en operaciones concisas. Esto permite que las nuevas ideas que surjan en futuras investigaciones puedan ser comprobadas en mucho menos tiempo.

Finalmente, una de las características que otras implementaciones no tienen es el manejo de datos de entrada más generales. La herramienta es capaz de trabajar con PSLGs que contengan cualquier combinación de puntos, aristas restringidas y bordes (internos y externos), incluyendo ángulos restringidos pequeños e interiores.

Bibliografía

[1] M.C. Rivara, P. Rodríguez-Moreno *Tuned terminal triangles centroid Delaunay algorithm for quality triangulation* Proceedings 27th International Meshing Roundtable, Albuquerque, NM, USA, October 2018, 17 pages.

[2] M.C. Rivara, C. Calderon. *Lepp Terminal Centroid Method for Quality Triangulation: A Study on a New Algorithm*. Department of Computer Science, University of Chile, 2008.

[3] M.C. Rivara. *New longest-edges algorithms for the refinement and/or improvement of unstructured triangulations* Int. J. Mumer. Meth. Engrg, 1997.

[4] J.R. Shewchuk, B.C. Brown. *Fast Segment Insertion and Incremental Construction of Constrained Delaunay Triangulations*. Department of Electrical Engineering and Computer Sciences, University of California, May 2015.

[5] C. Bedregal, M.c. Rivara. *New results on Lepp-Delaunay algorithms for quality triangulations.* Procedia Engineering, 2014.

[6] M.C. Rivara, N. Hitschfeld. *Lepp-delaunay algorithm: a robust tool for producing size-optimal quality triangulations.*

[7] R.B. Simpson, M.C. Rivara. *Geometrical mesh improvement properties of Delaunay terminal edge refinement, Technical report CS-2006-16.* The David Cheriton School of Computer Science, University of Waterloo, 2006.

[8] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. *Computational Geometry* Springer-Verlag Berlin Heidelberg, 1997.

[9] Leonidas J. Guibas, Donald E. Knuth; Micha Sharir. *Randomized incremental construction of Delaunay and Voronoi diagrams.* Algorithmica. 7: 381–413, June 1992.

[10] G. Leach. *Improving Worst-Case Optimal Delaunay Triangulation Algorithms.* Canadian Conference on Computational Geometry, June 1992.

[11] Shewchuk, J.R. *Delaunay refinement algorithms for triangular mesh generation.* Computational Geometry, 22(1-3), 21–74 (2002)