**FACULTAD DE INFORMÁTICA**
UNIVERSIDAD POLITÉCNICA DE MADRID

# UNIVERSIDAD POLITÉCNICA DE MADRID

FACULTAD DE INFORMÁTICA

TRABAJO FIN DE CARRERA

## *"TRIANGULACIÓN DE PESO MÍNIMO Y PROGRAMACIÓN DINÁMICA"*

AUTOR: Caio Borges Leal da Silva
TUTOR: Gregorio Hernández Peñalver

# ÍNDICE

# 1 INTRODUCCIÓN

Una triangulación de un conjunto de puntos es un conjunto finito de aristas, trazadas en el espacio, de forma que se trace el mayor número de aristas posibles sin que se crucen entre sí. Como consecuencia, todas las caras (faces) del grafo son triángulos (Tan, 1993). En otras palabras, dado un conjunto S de puntos en el plano, una triangulación de S es un grafo geométrico maximal sin cortes sobre S.

El peso de una triangulación es la suma de los pesos de todas sus aristas, donde normalmente (y en este proyecto), se trata de la distancia euclidea entre sus dos vértices (Loera, 2009).

Existen diversos algoritmos para obtener triangulaciones de un conjunto de puntos, donde cada uno explora, maximiza o minimiza alguna característica especial de la triangulación. Por ejemplo, maximizar el ángulo más pequeño de todos los triángulos resultantes (algoritmo de Delaunay).

Una triangulación de peso mínimo (MWT) es aquella donde la suma de los pesos de sus aristas es la mínima posible. Si se quiere minimizar la longitud de la máxima arista de una triangulación, entonces obtenemos una triangulación MLT (Tan, 1993).

Se utilizan distintos algoritmos de triangulación para resolver problemas de la vida real, tales como: modelización de terrenos mediante estructura poliédricas, cálculo de distancia mínima (por ejemplo, el GPS), modelos de zona de influencias, etc. Además, la triangulación mínima de polígonos, junto con el algoritmos de coloreado de grafos, sirve de soporte para resolver el problema del número mínimo de guardias y cámaras en un recinto.

Debido a que el cálculo de la MWT es un problema NP-Duro (Loera, 2009), se puede dividir la triangulación en pequeñas piezas independientes, y aplicar, sobre cada pieza, un algoritmo que resuelva el problema (esta estrategia se llama “programación dinámica”).

En este proyecto, estudiaremos distintas estrategias de programación dinámica para obtener el resultado exacto y/o una aproximación de las triangulaciones MWT para cualquier nube de puntos.

Los resultados obtenidos a través de la descomposición de la nube en piezas menores y posterior aplicación de algoritmos polinómicos para MWT, nos devuelve una aproximación a la triangulación de peso mínimo. No obstante, obtenemos el resultado exacto de la triangulación MLT, por tratarse de un problema polinómico.

En resumen, en este proyecto estudiaremos las distintas estrategias, utilizando la programación dinámica, para calcular las triangulaciones MWT y MLT.

También estudiaremos las principales características de las triangulaciones en general y la triangulación de Delaunay.

Este documento se encuentra organizado de la siguiente forma:

- En la primera parte, describiremos los objetivos del proyecto y el plan que se ha llevado a cabo para su finalización.
- Luego, explicaremos con detalles los algoritmos y las estrategias que se ha utilizado e implementado.
- Haremos una presentación de las funcionalidades del programa creado para la visualización de los algoritmos y de cómo utilizarlo.
- Después se mostrará una introducción del diseño del programa.
- Por último, se mostraran las pruebas y compararemos los resultados obtenidos.

# 2 OBJETIVOS Y ALCANCE

## 2.1 OBJETIVOS

Los objetivos, tanto de la parte teórica como la parte funcional del programa, son los siguientes:

1. Entender los conceptos y características básicas de las triangulaciones.
2. Implementar distintas estrategias para obtener la triangulación de peso mínimo (MWT), la triangulación MLT y los algoritmos auxiliares.
3. Comparar los resultados obtenidos en función de los pesos de las triangulaciones y del tiempo en obtener los resultados.
4. Estudiar e implementar las distintas formas de obtener la MWT y MLT de un polígono.
5. Estudiar e implementar diferentes algoritmos ya existentes para obtener triangulaciones (incremental, Greedy y Delaunay).
6. Desarrollar una aplicación para visualizar los algoritmos implementados y los pasos que ejecutan, dando soporte, además, a comparaciones de diferentes estrategias y/o algoritmos.

Para ser más concreto, los algoritmos que se han implementado durante el desarrollo de este proyecto han sido:

- Cierre convexo: obtiene el cierre convexo de una nube de puntos.
- MST (árbol generador mínimo): obtiene el árbol de peso mínimo de una nube.

- Poligonización monótona: obtiene una poligonización a partir de una nube de puntos.
- Triangulación de Greedy.
- Triangulación de Delaunay.
- RNG: grafo de vecindad relativa.
- Triangulación incremental.
- Calculo de aristas ligeras.

Las estrategias implementadas para llegar a un resultado aproximado o exacto (en algunos casos) de una MWT o MLT son las siguientes:

- **Estrategia 1:** Consiste en calcular el cierre convexo y el polígono monótono de la nube de puntos. El resultado consiste en obtener polígonos menores (piezas), donde se puedan aplicar el algoritmo de triangulación de polígonos.
- **Estrategia 2:** Calcula el árbol generador mínimo de una triangulación de Greedy, y en conjunto con el cierre convexo, genera piezas menores, que pueden contener ramas. Al aplicar el algoritmo de triangulación de polígonos en cada una de las piezas resultantes, se obtiene una aproximación a la MWT.
- **Estrategia 3**: Parecida a la estrategia 2, con la diferencia de que se calcula el árbol mínimo de una nube de puntos, y no de la triangulación de Greedy.
- **Estrategia 4**: Calcula el cierre convexo y el RNG de una nube de puntos. El resultado es la división de la nube en pequeñas piezas, donde se calcula la MLT. Es la única estrategia implementada que devuelve la triangulación mínima exacta (Tan, 1993).
- **Estrategia 5:** Se obtiene todas las aristas ligeras de una nube de puntos. Y para todas las piezas que no son triángulos, se aplica el algoritmo de triangulación de polígonos.

Así como las estrategias, se ha utilizado la técnica de programación dinámica para el cálculo de las triangulaciones MWT y MLT, que se explicará más adelante.

Por último, se ha implementado distintas formas de obtener todas las triangulaciones (por fuerza bruta) a partir de una nube de puntos. Se explicarán con más detalles en su apartado, pero podemos adelantar que la conclusión que se ha obtenido es que está muy lejos de ser una manera eficiente de encontrar la MWT.

## 2.2 ALCANCE DEL PROYECTO

Después del desarrollo del proyecto, se obtienen dos productos entregables: el programa desarrollado y la documentación del proyecto.

El programa desarrollado se trata de una aplicación Windows en Java, y su correspondiente código fuente. La documentación (este documento), tiene como fin

explicar los objetivos, desarrollos y conclusiones obtenidas del estudio de las triangulaciones y de las implementaciones realizadas.

Como se comentará posteriormente, la aplicación desarrollada consta de 4 perspectivas diferentes, donde se pueden ejecutar distintos algoritmos: para nube de puntos y para polígonos.

El usuario interactúa con la aplicación de forma dinámica. Las perspectivas proporcionan la capacidad de ejecutar y visualizar los pasos de los algoritmos y estrategias implementadas, es decir, los algoritmos y estrategias comentados en el apartado anterior.

Dos de estas perspectivas se relacionan a nubes de puntos en un espacio 2D. La primera de ellas, implementa algoritmos generales de las nubes de puntos. La explicación y demonstraciones de estos algoritmos se pueden encontrar en los artículos referenciados en este proyecto, por lo que las demonstraciones de ellos no son objetivos de este proyecto.

Para el desarrollo de la aplicación, se ha utilizado como base los códigos fuentes de proyectos similares proporcionados por el tutor del proyecto. De estos códigos fuentes se pudo aprovechar la implementación de algunos algoritmos basado en nubes de puntos, haciendo una adaptación para encajar con los requisitos de la nueva aplicación. La interfaz gráfica, sin embargo, es completamente nueva y original.

La eficiencia de los algoritmos no es objetivo de la aplicación, debido a que se debe enseñar de forma gráfica los pasos de las ejecuciones. Y eso provoca una perdida más que considerable en el rendimiento.

Como consecuencia, algunos algoritmos pueden estar limitados por la memoria RAM disponible en el ordenador, ya que la implementación de los elementos de la visualización gráfica consume mucho espacio en memoria. Por tanto, el programa no es robusto en respecto a esta propiedad, ya que puede dejar de funcionar sin previo aviso.

El desarrollo del proyecto se ha realizado de forma individual con la ayuda y tutela de Gregorio Hernández. Y como ya se ha comentado anteriormente, se trata de una extensión de la asignatura “Sistemas Informáticos”, del quinto curso de la carrera superior.

Para el desarrollo del proyecto, todos los recursos disponibles estaban al alcance. Es decir, el portátil personal con un sistema operativo Windows, Java y Eclipse (entorno de programación y de pruebas) y la ayuda del tutor, ofreciendo soporte y documentación necesaria.

El único punto que ha podido causar problemas, ha sido el tiempo, ya que durante un poco más de la mitad del desarrollo del proyecto, he empezado a trabajar a jornada completa, teniendo poco tiempo disponible para dedicarme al proyecto. Por esa causa, durante el periodo de formación en el trabajo, dejé este proyecto en stand-by por tres meses.

# 3 PLANIFICACIÓN

En este apartado se identificarán y se explicarán las tareas que se han realizado para alcanzar todos los objetivos del proyecto. Las tareas se agrupan de forma que estén enfocadas en uno o más objetivo descrito anteriormente. Las tareas se enumerarán a continuación:

- Primera reunión
- Estudio de triangulaciones
    - Estudio de la teoría general de grafos
    - Estudio de los algoritmos de triangulaciones de peso mínimo (MWT)
    - Estudio de Java Swing: librería gráfica
- Diseño de la aplicación
    - Analizar y crear documento de requisitos
    - Diseño de alto nivel
    - Revisión de requisitos y diseño
- Implementación de la aplicación
    - Implementar las estructuras básicas
    - Implementar la interfaz gráfica
    - Implementar algoritmos
- Validación y pruebas
    - Pruebas unitarias
    - Integración entre interfaz e estructuras básicas
    - Pruebas y validación de los algoritmos
- Conclusiones
- Documentación final

# 4 PARTE TEÓRICA - TRIANGULACIONES

## 4.1 INTRODUCCIÓN A TRIANGULACIONES

Una triangulación de un conjunto de puntos es un conjunto finito aristas, trazadas en el espacio, de forma que haya el mayor número de aristas posibles sin que se cruzen entre sí. Como consecuencia, todas las caras (faces) del grafo son triángulos (Tan, 1993). En

otras palabras, dado un conjunto S de puntos en el plano, una triangulación de S es un grafo geométrico maximal sin cortes sobre S.

Las triangulaciones de nubes de puntos son aplicables a todos los campos del conocimiento y de la vida real en los que se requiera determinar interacciones y propiedades entre los elementos de un conjunto, siempre y cuando dichos elementos se puedan representar como puntos en un plano. Algunos ejemplos podrían ser (Moreno, 2007):

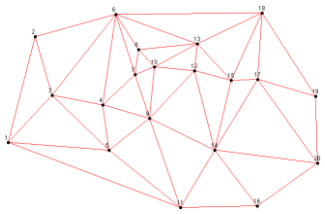
- **Modelación de terrenos mediante superficies poliédricas**: Para representar un terreno en un mapa tridimensional se han de tomar medidas de la altura del terreno, pero sólo podrá hacerse en determinados puntos (es imposible hacer mediciones en los infinitos puntos de un terreno).
  Una forma sencilla y bastante realista de aproximar los puntos que no se han podido medir sería representar los puntos que se han medido como una nube de puntos en el plano, sin tener en cuenta la medida de la altura de cada punto. Después se triangula esa nube de puntos y posteriormente se elevan esos puntos a la altura real a la que se midió cada uno. Dado que las triangulaciones generan una gran cantidad de aristas, el resultado de añadir altura a los puntos de la triangulación será una superficie poliédrica en tres dimensiones que puede servir de aproximación al terreno que se trataba de medir.
- **Cálculo de estructuras geométricas**: Ciertas triangulaciones (como la de Delaunay) tienen la propiedad de ser la base para calcular grafos y árboles. Esto quiere decir que, una vez calculada la triangulación de un conjunto de puntos, basta realizar una serie de sencillas operaciones sobre sus aristas para ir obteniendo estructuras con características interesantes que los hacen muy aplicables:
  1. *Cálculo de distancias mínimas:* Descartando aristas puede obtenerse el grafo del vecino más próximo, donde cada punto se conecta con el más cercano. Es muy útil para casos en los que se quiere conocer la distancia mínima entre dos objetos cualquiera (aviones, ciudades...) que se puedan modelizar como puntos en un plano.
  2. *Modelos de zonas de influencia:* A partir de la triangulación de Delaunay puede obtenerse el diagrama de Voronoi, que básicamente divide el plano en zonas según su cercanía a cada punto. Así, se pueden modelizar centros de abastecimiento, de servicios o de almacenamiento (tiendas, hospitales, naves industriales) como puntos y calcular su diagrama de Voronoi para averiguar cuál es el punto que más zona abarca o cómo colocar un nuevo punto más cercano o alejado de otros.

La triangulación es un objeto de estudio de la Geometría Computacional. La Geometría Computacional estudia el diseño y análisis de algoritmos eficientes para resolver problemas geométricos (Peñalver, 2006).

### 4.1.1 TRIANGULACIÓN DE UNA NUBE DE PUNTOS

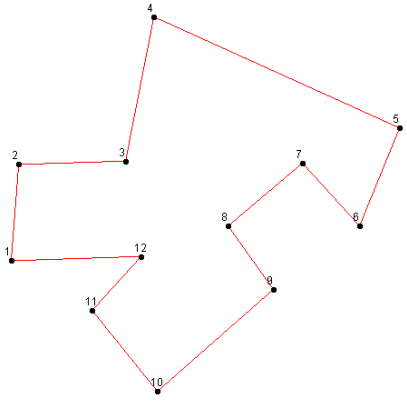
Dado un conjunto S de puntos en el plano, una triangulación de S es una descomposición de la envolvente convexa de S en triángulos cuyos vértices son los de S y tal que cada par de triángulos de la descomposición tiene sus interiores disjuntos (Peñalver, 2006).

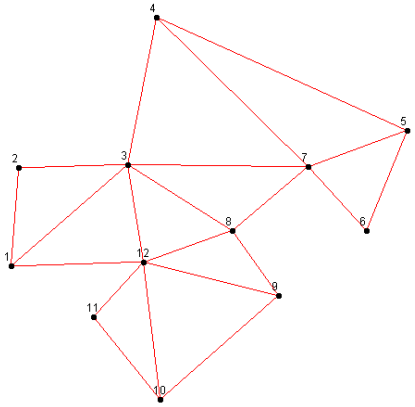
Dado un conjunto S de puntos en el plano, una triangulación de S es un grafo geométrico maximal sin cortes sobre S. Análogamente se define una triangulación de una nube de puntos del plano como una partición del cierre convexo en triángulos.



### 4.1.2 TRIANGULACIÓN DE UN POLÍGONO

En geometría, la triangulación de un polígono o área poligonal es una partición de dicha área en un conjunto de triángulos.




De manera más precisa, una triangulación es una división del área en un conjunto de triángulos que cumplen las siguientes condiciones:

- La unión de todos los triángulos es igual al polígono original
- Los vértices de los triángulos son vértices del polígono original
- Cualquier pareja de triángulos es disjunta o comparte únicamente un vértice o un lado

### 4.1.3 TRIANGULACIÓN DE PESO MÍNIMO (MWT)

La triangulación de peso mínimo (MWT) es una triangulación completa con el menor peso posible. El problema de computar la MWT de un conjunto de puntos en el plano ha recibido una atención importante, especialmente cuando el peso es la distancia euclidea entre las aristas.

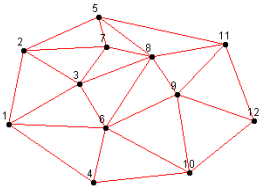
Se trata de un problema con complejidad NP-duro, e intentar encontrar una la triangulación MWT por fuerza bruta está fuera de cuestión: un grafo con 10 puntos puede tener hasta 2000 triangulaciones distintas.

La triangulación de Greedy y la triangulación de Delaunay se aproximan, pero no devuelve el resultado exacto de una MWT (Tan, 1993).

### 4.1.4 FÓRMULA DE EULER

La fórmula de Euler nos dice que hay una relación directa en la triangulación entre el número de vértices, el número de aristas y el número de regiones de la triangulación (Peñalver, 2006).

n: número de vértices
q: número de aristas
r: número de regiones



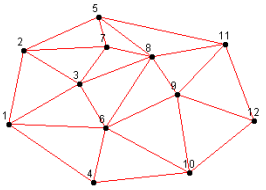

n = 12
q = 26
r = 16

La relación es la siguiente: $n - q + r = 2$

### 4.1.5 OBTENER NÚMERO DE TRIÁNGULOS Y ARISTAS

El número de triángulos y aristas de una triangulación depende de las posiciones de los vértices en el plano (Peñalver, 2006).

n: número de vértices
q: número de aristas
r: número de regiones
k: vértices en el cierre convexo




n = 12
q = 26
r = 16
k = 7

El número de triángulos en una triangulación se obtiene por la siguiente fórmula:

$$t = 2n - k - 2$$

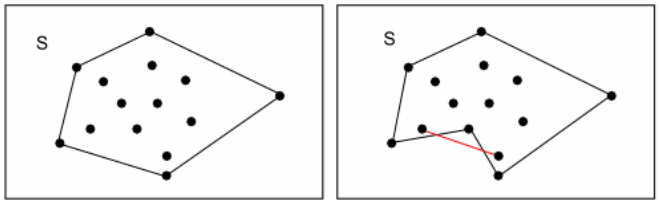
El número de aristas de una triangulación:

$$q = 3n - k - 3$$

## 4.2 ALGORITMOS GENERALES

### 4.2.1 CIERRE CONVEXO

Un conjunto de puntos en un plano, los cuales determinan a un polígono, se denomina convexo si cualquiera de los segmentos generados a partir de las uniones de dos puntos pertenecientes al conjunto, se encuentra contenido en su totalidad dentro del polígono.




En la primera se muestra como el conjunto S de puntos es encerrado por un cierre convexo. En cambio, en la segunda figura se ve como no se cumple la propiedad para considerar al polígono convexo

Dado un conjunto de puntos en el plano, se llama cierre convexo al menor de los conjuntos de puntos que lo contiene (Torres, 2009).

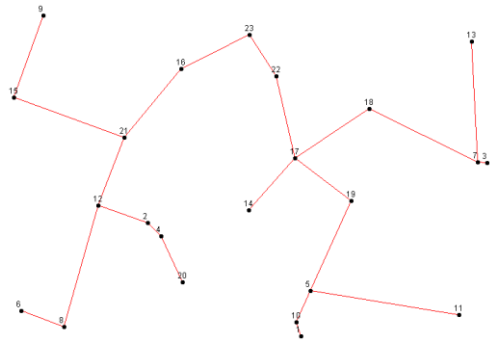
Para obtener el cierre convexo de una nube de puntos, se ha implementado el Algoritmo de Melkman. Dicho algoritmo utiliza una lista de vértices, donde solo se puede insertar o eliminar los puntos que se encuentran al principio o al final de la lista.

Se puede obtener una descripción del funcionamiento del algoritmo de forma exhaustiva en el siguiente enlace: http://w3.impa.br/~baier/Melkman/gcmelkma.html; O directamente desde la página del departamento DMA de la FI-UPM:

http://www.dma.fi.upm.es/docencia/trabajosfindecarrera/programas/geometriacomputacional/LeeMelSkl/..%5CLeeMelSkl%5CMelkman.html

### 4.2.2 ÁRBOL GENERADOR MÍNIMO (MST)

Un árbol es un grafo conexo sin ciclos. El árbol generador mínimo de G, MST(G), es un árbol generador con el menor peso posible (Peñalver, Árboles generadores mínimos).



1. En general, no es único
2. La arista de mayor peso de un ciclo no pertenece a MST(G)
3. La arista de menor peso de un corte pertenece a MST(G).
    a. Un conjunto C de aristas de G es un corte si existe una partición (V1 y V2) de V tal que C contiene todas las aristas con un extremo en V1 y otro en V2.

Existen diferentes algoritmos para obtener el árbol de peso mínimo. Y además, el árbol no es único. El algoritmo elegido en el desarrollo de este proyecto ha sido el algoritmo

de Prim. La idea básica consiste en añadir, en cada paso, una arista de peso mínimo a un árbol previamente construido:

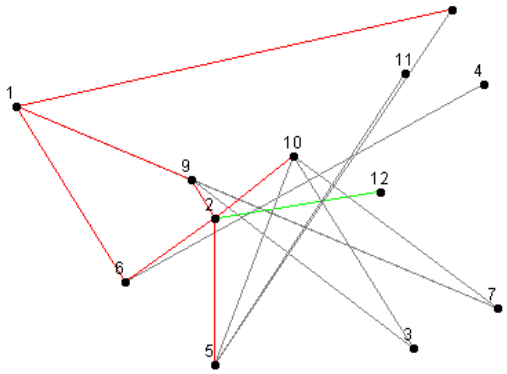
- **Paso 1:** Se elige un vértice u de G y se considera el árbol S={u}
- **Paso 2:** Se considera la arista e de mínimo peso que une un vértice de S y un vértice que no es de S, y se hace S = S + e
- **Paso 3:** Si el nº de aristas de T es n-1 el algoritmo termina. En caso contrario, vuelve al paso 2

*Fuente: http://www.dma.fi.upm.es/gregorio/grafos/PrimKruskal/prim/prim.html*

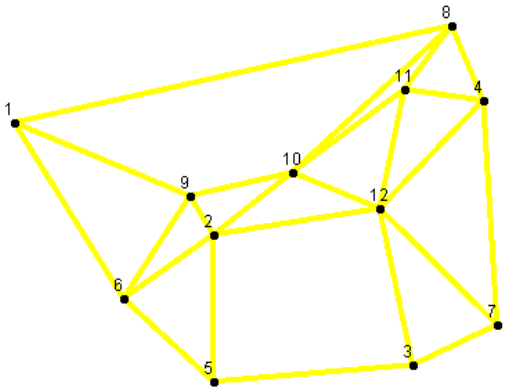
### 4.2.3 ARISTAS LIGERAS

Una arista es ligera si su peso es el mínimo entre todas las demás aristas del grafo que corte con ella. Todas las aristas del cierre convexo de una nube de puntos son ligeras, ya que ninguna otra arista las corta.

A continuación vemos un ejemplo gráfico del cálculo de una arista ligera en una nube de puntos. La imagen que aparece más a la derecha son todas las aristas ligeras de una nube de puntos.




La arista (2,12), que aparece en verde, es ligera, ya que todas las aristas que la cortan (en color gris) no son más pequeñas




Todas las aristas ligeras de la nube de puntos

Se podría pensar que todas las aristas ligeras de una nube de puntos generasen una triangulación. Pero se puede ver en la imagen anterior que eso no es del todo cierto. Ya que pueden generar huecos al obtener todas aristas ligeras de la nube de puntos.

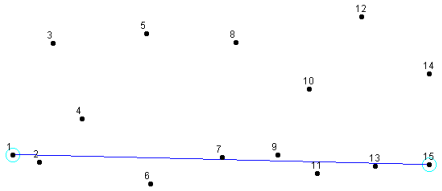
### 4.2.4 POLIGONIZACIÓN DE NUBE DE PUNTOS

Una cadena poligonal es una sucesión finita de segmentos de recta en la que el origen de un segmento coincide con el extremo final del anterior. Un polígono simple es una cadena poligonal cerrada simple con tres o más segmentos.

Un polígono cuyos vértices coincidan con un conjunto finito de puntos C se dice que es una poligonización del conjunto C (Abellanas, vol 11 (2008)).

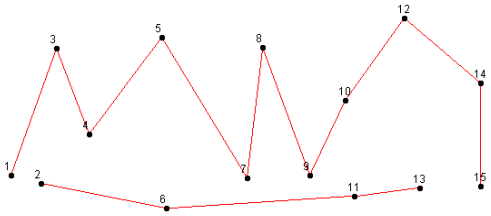
Dado un conjunto de puntos, se ordenan los puntos en función de la abscisa (de menor a mayor). En el caso de coincidir algún punto con la misma abscisa, se ordena en función de la ordenada.

Una vez que el conjunto de puntos están ordenados, obtenemos dos puntos: $P_1$ y $P_2$ (primero y último punto del conjunto). Dichos puntos formarán el segmento de reta intermediaria.




Con el resto de los puntos se hacen dos conjuntos: $C_{inf}$ y $C_{sub}$. El primero contiene los puntos que están encima de la recta intermediaria, y los demás, los que están abajo y alineados con la recta intermediaria, se sitúan en el segundo conjunto.

El siguiente paso es ordenar los puntos de los dos conjuntos. Cada conjunto de puntos representa las aristas del polígono monótono




Las aristas de la poligonización son las que generan cada subconjunto ($C_{inf}$ y $C_{sub}$). El último paso es unir los dos conjuntos: se crean dos nuevas aristas, la primera contiene el primer punto de cada conjunto, y la segunda, el último punto de cada conjunto.

A continuación, se describirán los casos especiales que hay que tenerlos en cuenta durante la ejecución del algoritmo:

- **Alineaciones en abscisa:** Al ordenar el conjunto de puntos, puede dar el caso de que al menos dos puntos tengan el mismo valor de la abscisa. En estos casos, el orden de los puntos vienen determinado por el valor de la ordenada.
- **Alineaciones con la recta intermedia:** Si un punto se encuentra alineado con la recta intermediaría, pasará a formar parte del subconjunto de los puntos inferiores. En caso de que no haya ningún punto en la parte superior de la recta, los puntos que se encuentran alineados con la recta intermedia (incluso los puntos extremos) pasarán a formar parte del conjunto de los puntos superiores.

El pseudocódigo del algoritmo es el siguiente:

*puntos ordenados = lista de puntos ordenados en función de x*

*p1 = primer elementos de puntos ordenados*
*pn = último elemento de puntos ordenados*

*Reta intermedia = segmento de recta entre punto p1 y pn*

*Puntos arriba = puntos que se encuentran por encima de la recta intermedia*
*Puntos abajo = puntos que se encuentran por debajo o alineados a la recta intermedia*

***For** i=0 **hasta** (tamaño de puntos arriba - 1)*
*arista = nueva arista (puntos arriba[i], puntos arriba[i+1])*
*añadir arista a aristas*

***For** i=0 **hasta** (tamaño de puntos abajo - 1)*
*arista = nueva arista (puntos abajo[i], puntos abajo[i+1])*
*añadir arista a aristas*

*arista = nueva arista (puntos abajo[1], puntos arriba[1])*
*Añadir arista a aristas*
*arista = nueva arista (puntos abajo[n], puntos arriba[n])*
*Añadir arista a aristas*

### 4.2.5 GRAFO DE VECINDAD RELATIVA (RNG)

Un grafo de vecindad relativa (*relative neighborhood graph*) de un conjunto P de *n* nodos en el plano, denominado RNG(P), es un grafo cuyo conjunto de vértices es P y existe una arista enlazando dos puntos *p* y *q* de S si:

$$Distancia(p,q) \leq min_{r \in S\{p,q\}}\{max\{Distancia(p,q), Distancia(q,r)\}\}$$

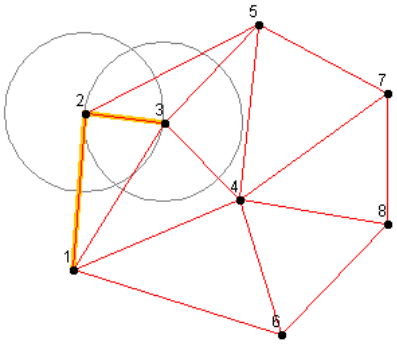
Esta desigualdad determina una región "prohibida" para el punto *r* si *p* y *q* son adyacentes en el grafo de RNG(P). Esta región es formada por la intersección de los círculos de centro *p* y *q*, y de radio Distancia (p,q).

Fuente: http://www.ime.usp.br/~cris/aulas/11_2_331/listas/l6.pdf

Otra definición de RNG, todavía más clara, y que hemos utilizado para implementar la funcionalidad es la siguiente:

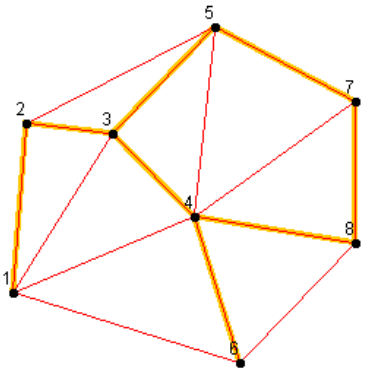
Consiste en todas las aristas *uv* de una nube de puntos, tal que no exista un vértice *w* con aristas *uw* y *wv* que satisfaga la siguiente condición (Marques, 2005):

$$uw < uv \; y \; wv < uv$$




Es decir, la arista pertenece al grafo solo si no existe otro vértice en la intersección de las circunferencias de centro *u* y *v*, y radio *uv*.

La arista (2,3) pertenece al grafo RNG ya que no hay otro punto en la intersección de las dos circunferencias.
El algoritmo que hemos implementado parte de un triangulación de Delaunay y comprueba si cada arista cumple la condición anterior.



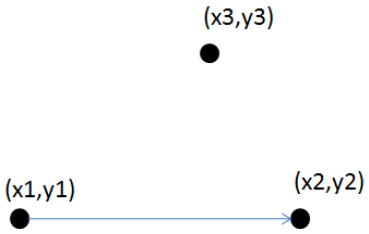

Este algoritmo permite encontrar el grafo RNG en complejidad linear una vez que se obtiene la triangulación de Delaunay (depende del número de aristas de la triangulación).

El grafo RNG es un subgrafo del grafo de Gabriel. El grafo de Gabriel, a su vez, es un subgrafo de la triangulación de Delaunay (Claus, 2004).

## 4.3 FUNCIONES AUXILIARES

### 4.3.1 ALINEACIÓN DE PUNTOS Y SENTIDO DEL TRIÁNGULO

Para comprobar el sentido del triángulo (contrario o a favor de las agujas del reloj), se ha implementado una función que recibe 3 puntos como parámetros. Los dos primeros indican la dirección del segmento de recta a estudiar, y tercer punto determina si el triángulo está orientado positiva o negativamente.




En la imagen arriba, queremos determinar la orientación del punto C en relación al segmento de recta ((x1,y1),(x2,y2)). Para ello, calculamos el determinante de la siguiente matriz:

$$\begin{pmatrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ x3 & y3 & 1 \end{pmatrix}$$

Si el determinante de la matriz es positivo, significa que la orientación del triángulo ((x1,y1), (x2,y2), (x3,y3)) es positiva, es decir, el giro es contrario al movimiento de las agujas del reloj. Si el determinante es negativo, el movimiento es a favor al movimiento de las agujas del reloj. Si el determinante es 0, los puntos están alineados.

### 4.3.2 DISTANCIA ENTRE DOS PUNTOS

En nuestra implementación, hemos utilizado la distancia euclidiana para calcular la distancia entre dos puntos. Java implementa la función: *Point2D.Double.distance(x1, y1, x2, y2).*

### 4.3.3 ORIENTACIÓN DE POLÍGONOS

Algunos algoritmos que se ha implementado necesitan, en algunos momentos, que los polígonos estén orientados positivamente. La forma de orientar un polígono es la siguiente:

- El algoritmo recibe una lista de puntos que determina el polígono.
- Obtener el punto que está más a la izquierda $P_1$.
- Reordenar la lista de puntos para que el punto más a la izquierda sea el primer elemento de la lista.
- Obtener los dos vértices adjuntos $P_2$ y $P_n$ (el segundo y el último de la lista).
- Obtener la orientación del triángulo ($P_1$, $P_2$, $P_n$).
- Si la orientación es negativa, hay que reordenar la lista de puntos tal que el segundo elemento de lista anterior ($P_2$) sea el último, y el último elemento de la lista anterior ($P_n$) sea el segundo.

## 4.4 ALGORITMOS DE TRIANGULACIÓN DE NUBES DE PUNTOS

### 4.4.1 TRIANGULACIÓN DE GREEDY

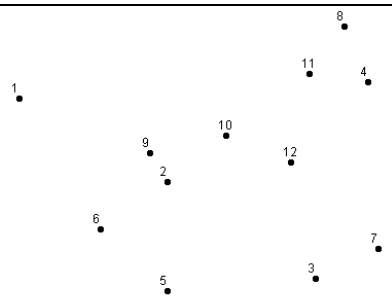
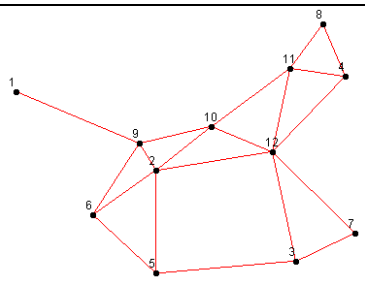
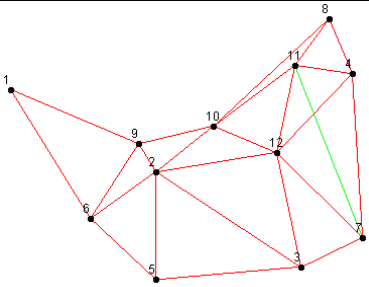
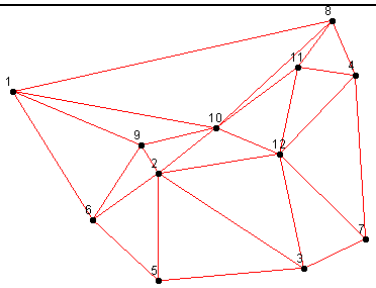
El nombre de Voraz que recibe esta triangulación se debe a su método de construcción; las aristas se van añadiendo al grafo una a una en orden creciente de tamaño, con la salvedad de que una arista no puede cortar a otra más corta, ya añadida al grafo (Loera, 2009).

El algoritmo recibe una lista de puntos como entrada. El pseudocódigo del algoritmo es el siguiente:

***Para cada punto en puntos***
*arista = crear arista con todos los demas puntos.*

*eliminar punto de la lista de puntos.*
*insertar ordenadamente, por tamanyo, arista en aristas.*
***Fin bucle***

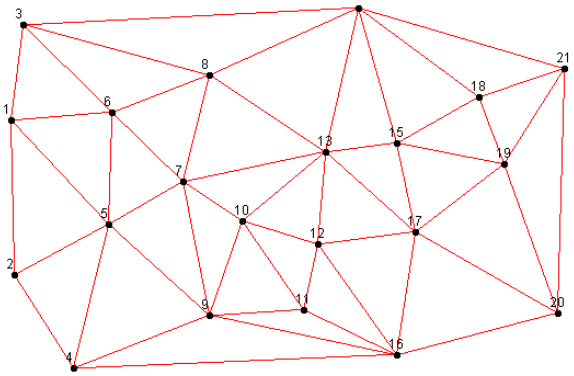
***Mientras aristas no es una lista vacia***
*arista = obtener primer elemento de aristas.*
*verificar si hay alguna arista en aristas finales que se cruce con la arista actual.*
***si no se cruza***
*insertar arista en aristas finales*
***Fin Bucle***

<table>
<tr>
<td>  <br>Nube de puntos</td>
<td>  <br>Ejecutando el algoritmo de Greedy</td>
</tr>
<tr>
<td>  <br>Encontrada una arista que se cruza. No se inserta en la triangulación</td>
<td>  <br>Resultado del algoritmo</td>
</tr>
</table>

Para el correcto funcionamiento del algoritmo, hay que tener en cuenta algunos casos especiales. Por ejemplo, una arista no se cruza con la otra si el punto de intersección entre ambas es un vértice de las dos aristas, es decir, ambas aristas tienen un vértice en común. En caso contrario, las aristas se cruzan.

### 4.4.2 TRIANGULACIÓN DE DELAUNAY

Es una de las triangulaciones más interesantes por ser aplicable para la resolución de multitud de problemas aparentemente sin relación entre sí, debido a sus propiedades geométricas, y por contar con algoritmos bastante eficientes para su cálculo.




La triangulación de Delaunay es considerada uno de las triangulaciones "más uniformes", es decir, es la que más se acerca una triangulación regular, asegurando que los ángulos del interior de los triángulos sean lo más grandes posibles, y que la longitud de los lados de los triángulos sea mínima y que la triangulación formada sea única.

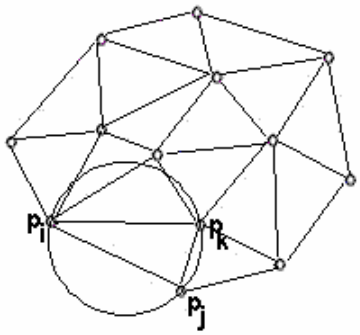
Las propiedades principales de la triangulación de Delaunay son (Loera, 2009):

- Es la triangulación que minimiza tanto el ángulo máximo como el radio de las circunferencias de sus triángulos.
- Tres puntos de la nube de puntos son vértices de un mismo triángulo de la triangulación de Delaunay si y sólo si puede trazarse un círculo cuyo contorno contenga esos tres puntos y no contenga otros puntos de la nube en su interior.
- Dos puntos de la nube definen una arista de la triangulación si y sólo si es posible trazar un círculo cuyo contorno contenga a esos dos puntos pero en su interior no contenga ningún otro punto de la nube.

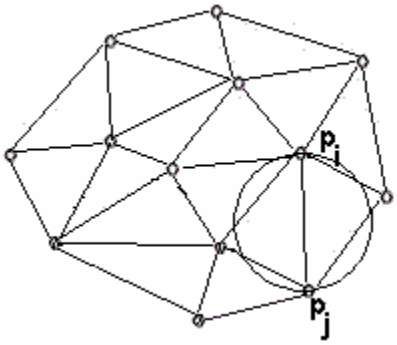
#### *4.4.2.1 CARACTERIZACIÓN FORMAL (DELAUNAY)*

Sea $P = \{p_1, p_2, \dots, p_n\}$ un conjunto de puntos en el plano, una triangulación de Delaunay de P cumplirá las siguientes propiedades:

- **Propiedad 1:** Tres puntos Pi, Pj y Pk son vértices de la misma cara de la triangulación de Delaunay de P, si y solamente si, el círculo que pasa por los puntos Pi, Pj y Pk no contiene puntos de P en su interior.

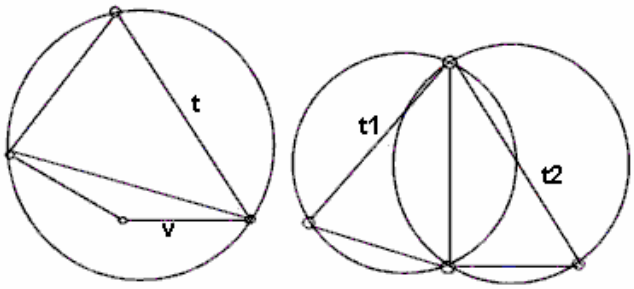


- **Propiedad 2:** Dos puntos Pi, Pj pertenecientes a P forman un lado de la triangulación de Delaunay de P, si y solamente si, existe un círculo Pi, Pj en su circunferencia y no contiene en su interior ningún punto de P.




Las dos propiedades anteriormente descritas son resumidas por una tercera, "Propiedad del Círculo Vacío", que caracteriza la triangulación de Delaunay:

- **Propiedad 3:** Propiedad del círculo vacío.
  Sea P un conjunto de puntos en el plano y T una triangulación de P, T es una triangulación Delaunay de P, si y solamente si la circunferencia circunscrita de cualquier triángulo de T no contiene puntos de P.




No cumple la condición de Delaunay

Sí cumple la condición de Delaunay

Se puede generalizar y afirmar que una triangulación T de un conjunto P de puntos en el plano es una triangulación Delaunay, si y solamente si, todas las aristas son legales.

Se llama arista legal, a toda arista de una triangulación que pertenece a dos triángulos tal que la circunferencia circunscrita a uno de los triángulos no contiene al punto restante que pertenece al otro triángulo.

Una arista ilegal de una triangulación es la arista que pertenece a dos triangulos tales que forman un cuadrilátero conexo y si se intercambia dicha arista por la diagonal del cuadrilátero mejora el vector de ángulos. A esta operación de consiste en sustituir una diagonal por la otra en un cuadrilátero se le denomina de "flips".

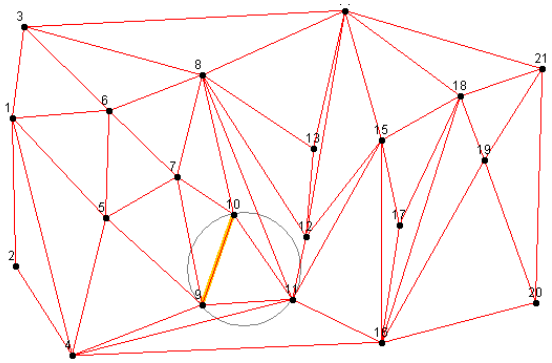
Por tanto, sea cual sea la triangulación de puntos inicial que se tenga, siempre se podrá obtener la triangulación de Delaunay equivalente por medio de operaciones de "flips".

Fuente: http://www.dma.fi.upm.es/mabellanas/bigdelone/Memoria.pdf

De hecho, el algoritmo que hemos implementado parte de una triangulación incremental, y comprueba si las aristas cumplen la última condición.

El algoritmo recibe una lista de puntos como entrada. El pseudocódigo del algoritmo es el siguiente:

```
Obtener triangulación incremental
do {
  cambio = false
  for i=0..tam_aristas
  a = aristas[i]
  ilegal = comprobarArista(a)
  IF es ilegal
    flip
   cambio = true
} while (cambio)
```



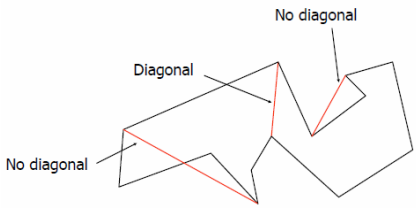

## 4.5 ALGORITMOS DE TRIANGULACIÓN DE POLÍGONOS

En geometría, la triangulación de un polígono o área poligonal es una partición de dicha área en un conjunto de triángulos.

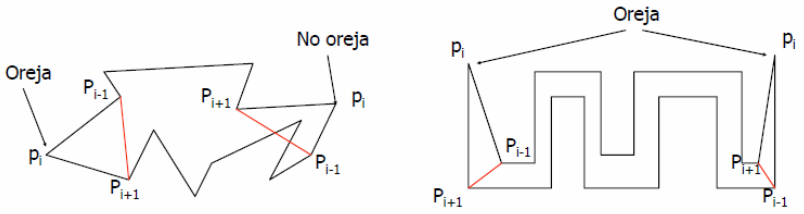
### 4.5.1 TRIANGULACIÓN POR RECORTADO DE OREJAS

Se trata de un algoritmo para obtener una triangulación de un polígono, de complejidad $O(n^3)$, desarrollado por Meisters, en 1975. El algoritmo se base en buscar las orejas del polígono. Las orejas que se encuentran comprenden la triangulación.

Una diagonal es un segmento que está totalmente incluida en el polígono y que une dos vértices no consecutivos:



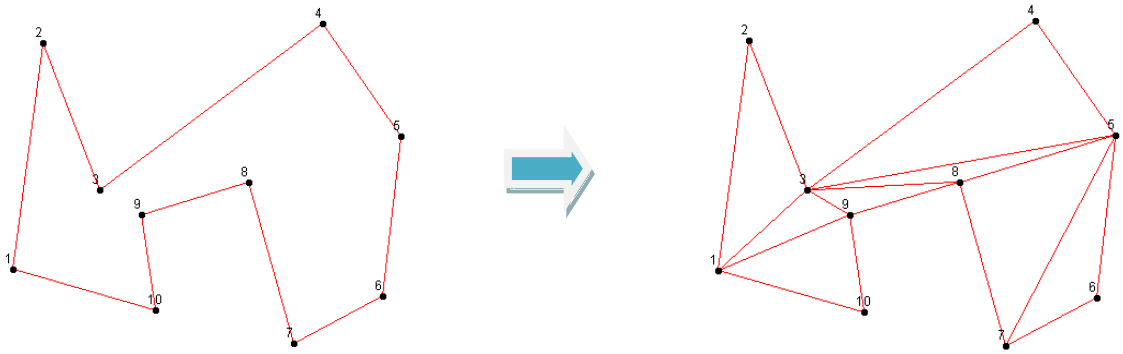

Un vértice $P_i$ se llama oreja si el segmento $(P_{i-1}, P_{i+1})$ es una diagonal del polígono:




El algoritmo se base en la búsqueda de orejas. Cuando encuentra una, la arista creada con sus puntos adyacentes se añade a la triangulación. Se elimina el nodo oreja del polígono y sigue buscando otras orejas, hasta que el polígono tenga solamente 3 puntos.

Fuente: http://www.dccia.ua.es/dccia/inf/asignaturas/RG/pdf/intro-triang-polig.pdf

En la imagen a continuación, vemos el resultado de aplicar el algoritmo sobre un determinado polígono:

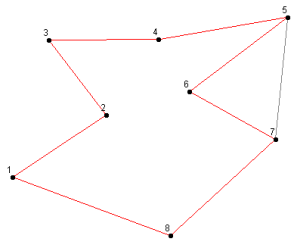


**Preposición:** *Las dos orejas de Meister*. Todo polígono de n ≥ 4 vértices tiene al menos dos orejas no superpuestas.

### 4.5.2 MWT DE UN POLÍGONO

Podemos construir la triangulación MWT de un polígono utilizando programación dinámica a través de un algoritmo de complejidad cúbica (Gilbert y Klincsek).

**Preposición:** Sea $P = \{p_1, p_2, \dots, p_n\}$ un conjunto de vértices de un polígono P orientado negativamente (a favor de las agujas del reloj). Entonces existe un algoritmo de complejidad O($n^3$) que calcula la triangulación de P que minimiza la suma de la longitud euclidiana de las aristas interiores (Tan, 1993).




Como el polígono P puede que no sea necesariamente convexo, la longitud de una arista inválida (una arista que tiene algún punto en el interior del polígono) es infinito.

$$d = \begin{cases} d(p_i, p_j) & si\ i = j+1\ o\ \overline{p_i p_j}\ es\ interior \\ \infty & en\ caso\ contrario \end{cases}$$

Llamaremos de optcost(i,j) la longitud mínima de una triangulación del subpolígono cuyos puntos son $p_i, p_{i+1}, .., p_j$. La idea principal del algoritmo, sabiendo que los puntos que están dentro de un polígono simple, es que la siguiente igualdad se mantiene:

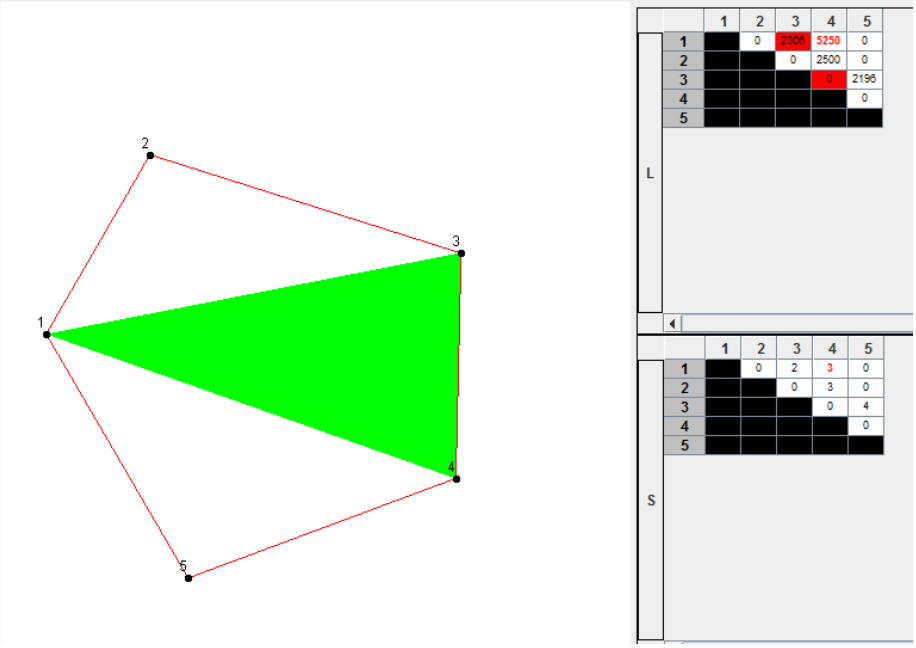
$$optcost(i,j) = d(p_i, p_j) + \min_{i<m<j} \big(optcost(i,m) + optcost(m,j)\big)$$

El pseudocódigo del algoritmo es el siguiente:

```
/* Initialize */
for i := 1 to n-1 do
   L[i,i+1] = 0
/* Calculate L[...] and S[...]:
for l := 2 to n-1 do
   for i := 1 to n-l do
      j := i+l
      L[i,j] = infinity
      if(isValidEdge(i,j))
         for k := i+1 to j-1 do
            x := L[i,k] + L[k,j] + Circumference(i,j,k)
            if x < L[i,j] then
               L[i,j] := x
               S[i,j] := k
         od
      fi
   od
od
```

Donde "L" es una matriz que guarda la los pesos de la triangulación. Se actualiza el valor de la matriz si el resultado parcial de la triangulación estudiada es menor que el resultado que aparece en la posición correspondiente en la matriz.

La matriz "*S*" guarda la información de las aristas de la triangulación. Se actualiza cuando se actualiza la matriz *"L"*. En la imagen siguiente podemos ver un paso del algoritmo donde se actualizan las dos matrices:

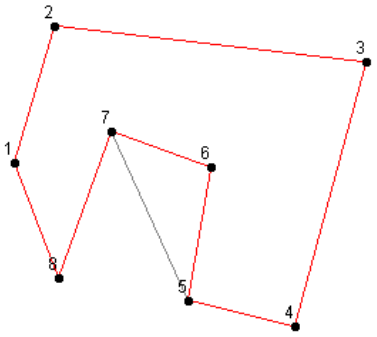



Donde i = 1, j = 4 y k = 3. Vemos que tanto la matriz L como la matriz S actualizan el valor en la posición correspondiente.

La función "*IsValidEdge*" comprueba si la arista es válida. Es decir, si no es una arista exterior o que no se cruce con ninguna otra arista. Primero comprueba que la arista no se cruza con otra cualquiera. Si se cruza, la arista es inválida. Si no se cruza, hay que comprobar si la arista es una arista exterior de un polígono cóncavo.

Se quiere comprobar si la arista $\overline{p_i p_j}$ es arista exterior (suponiendo que el polígono está orientado positivamente). Para ello, se obtiene la orientación del triángulo $t_1 = (p_{i-1},\ p_i, p_j)$ y la orientación del triángulo $t_2 = (p_j,\ p_{i+1}, p_i)$. La arista es inválida si:

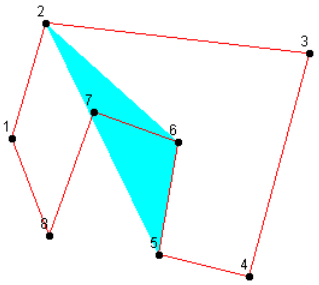
- Las dos orientaciones son iguales a cero.
- O si la orientación de $t_1$ es mayor que cero y la de $t_2$ es menor que cero.
- Si la orientación de $t_1$ es menor que cero y la de $t_2$ es mayor que cero, la arista es válida.
- Por último, si no se cumple ninguna condición anterior, la arista es ilegal si la orientación del triángulo $t_3 = (p_{i-1},\ p_i, p_{i+1})$ es negativa.



En la imagen a la derecha, vemos que la arista (5, 7) es inválida, ya que la orientación del triángulo $t_1 = (4, 5{,}7)$ es negativa, la orientación del triángulo $t_2 = (7, 6, 5)$ es negativa y la orientación del triángulo $t_3 = (4, 5{,}6)$ es negativa.

Se cumple la última condición expuesta anteriormente. Por tanto, la arista es ilegal.

La función "Circunference" calcula la longitud del triangulo (i, j, k). Además de comprobar que la arista (i, j) sea válida, hay que comprobar si las demás aristas del triángulo son válidas:

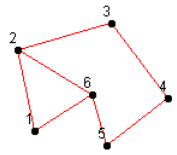


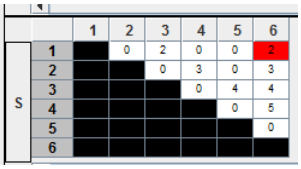
Cuando i=2 y j=6, la arista (i, j) es válida, pero el triángulo (i, j, k) no es válido (en azul en la imagen).

En estos casos, no hay que actualizar las matrices L y S, ya que daría resultados erróneos por estar teniendo en cuenta triángulos inválidos de la triangulación de un polígono.

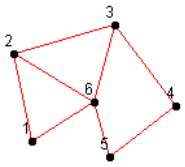
Una vez que termina el algoritmo, la información de la triangulación se encuentra en la matriz *S*. La información se obtiene de forma recursiva. Se obtiene el elemento que se encuentra en la posición S[*1*, *n]* = *k* de la matriz. Obtenemos dos nuevas aristas de la triangulación: (1, k) y (k, n).

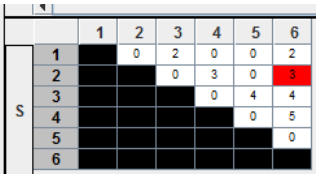
Ahora, vamos a la posición S[1, k] y S[k, n] para obtener las siguientes aristas. Seguimos este mismo paso hasta obtener la triangulación:



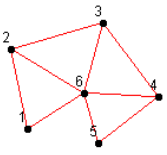


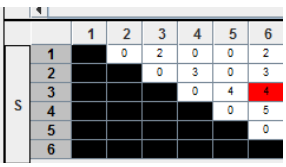
| S | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | ■ | 0 | 2 | 0 | 0 | 2 |
| 2 | ■ | ■ | 0 | 3 | 0 | 3 |
| 3 | ■ | ■ | ■ | 0 | 4 | 4 |
| 4 | ■ | ■ | ■ | ■ | 0 | 5 |
| 5 | ■ | ■ | ■ | ■ | ■ | 0 |
| 6 | ■ | ■ | ■ | ■ | ■ | ■ |





| S | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | ■ | 0 | 2 | 0 | 0 | 2 |
| 2 | ■ | ■ | 0 | 3 | 0 | 3 |
| 3 | ■ | ■ | ■ | 0 | 4 | 4 |
| 4 | ■ | ■ | ■ | ■ | 0 | 5 |
| 5 | ■ | ■ | ■ | ■ | ■ | 0 |
| 6 | ■ | ■ | ■ | ■ | ■ | ■ |





| S | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | ■ | 0 | 2 | 0 | 0 | 2 |
| 2 | ■ | ■ | 0 | 3 | 0 | 3 |
| 3 | ■ | ■ | ■ | 0 | 4 | 4 |
| 4 | ■ | ■ | ■ | ■ | 0 | 5 |
| 5 | ■ | ■ | ■ | ■ | ■ | 0 |
| 6 | ■ | ■ | ■ | ■ | ■ | ■ |

Este algoritmo fue inventado por Gilbert. El algoritmo descrito tiene una complejidad O($n^2$) si se aplica sobre un polígono convexo. Nadie sabe todavía si se puede mejorar el espacio utilizado en memoria o el tiempo de ejecución del algoritmo.

Casos especiales:

- **Alineaciones:** Las aristas y los triángulos se consideraran no válidos caso hayan alineaciones, ya que en este caso, va a haber al menos una arista que se cruce con alguna arista del triángulo.
- **Arista no válida:** Se pone el valor de la posición correspondiente en L a infinito y no se actualiza el valor correspondiente de la matriz S.
- **Triángulo inválido:** Alguna arista del triángulo no es válida, y por tanto, no se actualizan los valores en ambas matrices.
- **Triángulo válido pero no actualiza las matrices:** El valor de la celda L[i, j] anterior es menor que el nuevo valor calculado. Por lo tanto, no se actualizan las matrices L y S.

Fuentes: (Tan, 1993) y (Kunzweb, 2004).

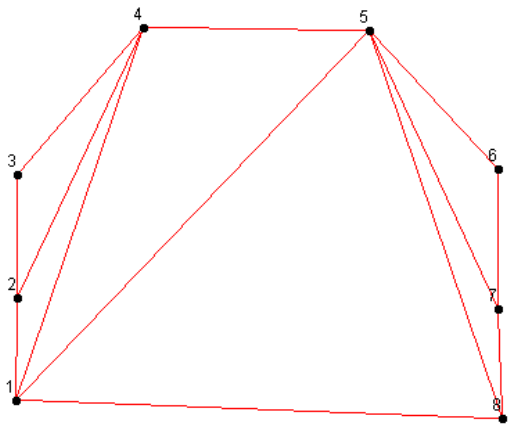
### 4.5.3 MLT DE UN POLÍGONO

Una triangulación que de un polígono que minimiza la longitud de la arista más grande se llama "*minmax length triangulation":* MLT.

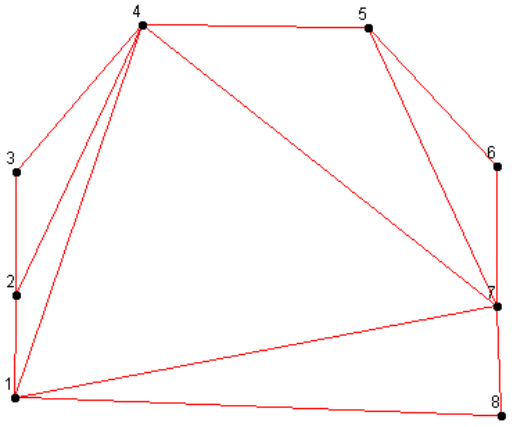
Para encontrar la triangulación *minmax* de un polígono, basta modificar una función del algoritmo presentado en el apartado anterior (el de calcular la triangulación de peso mínimo de un polígono).

Se modifica la función *Circunference(i,j,k).* En vez de devolver el perímetro del triángulo, devolverá la longitud de la mayor arista.

Eso garantiza que se elija siempre las triangulaciones parciales que minimice la mayor de las longitudes de las aristas de las triangulaciones. Además, las triangulaciones MWT y MLT del mismo polígono no tienen por qué ser iguales:




MWT
Peso = 2519,75




MLT
Peso = 2535,46

En este ejemplo, se puede ver que ni la triangulación ni el peso son iguales al comparar los dos algoritmos.

## 4.6 ESTRATEGIAS Y PROGRAMACIÓN DINÁMICA

La programación dinámica es una técnica empleada para resolver problemas de optimización mediante una secuencia de decisiones, basándose en el principio de optimalidad de Bellman: "*cualquier subsecuencia de decisiones de una secuencia óptima de decisiones que resuelve un problema también debe ser óptima respecto al subproblema que resuelve*".

- Supongamos que un problema se resuelve tras tomar una secuencia $d_1, d_2, d_3, \ldots, d_n$ de decisiones.
- Si hay d opciones posibles para cada una de las decisiones, una técnica de fuerza bruta u otra técnica exploraría las soluciones óptimas de los subconjuntos.
- La técnica de programación dinámica evita explorar todas las secuencias posibles por medio de la resolución de subproblemas de tamaño creciente y almacenamientos en una tabla de las soluciones óptimas de esos subproblemas para facilitar la solución de los problemas más grandes.

Fuente: http://www.lsi.upc.edu/~iea/transpas/4_dinamica/

Podemos utilizar la programación dinámica para intentar llegar a soluciones aproximadas de las triangulaciones MWT y MLT de una nube de puntos.

Las estrategias de triangulación son las combinaciones de diferentes métodos para obtener una aproximación de las triangulaciones MWT y MLT. En este proyecto, se ha implementado 4 estrategias que nos devuelven el valor aproximado de la triangulación MWT y una estrategia que devuelve la triangulación MLT.

Esta última estrategia, que devuelve la triangulación MLT, es la única entre todas las estrategias que no se trata de una aproximación, y sí del valor exacto (Tan, 1993).

### 4.6.1 ESTRATEGIAS MWT

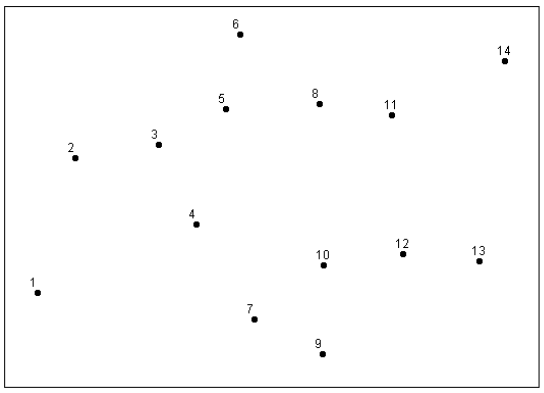
Hemos implementado 4 estrategias que calculan el valor aproximado de las triangulaciones MWT de una nube de puntos. En este aparatado explicaremos de forma detallada cada una de ellas.

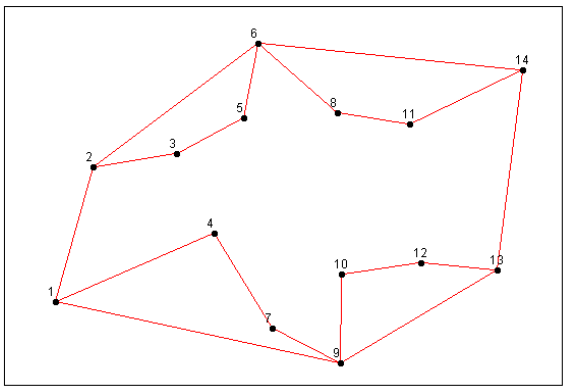
#### *4.6.1.1 ESTRATEGIA 1*

Como ya se ha dicho anteriormente, las estrategias son acciones que se ejecutan en secuencia para encontrar una aproximación de la triangulación de peso mínimo. La primera estrategia ejecuta los siguientes pasos:

- Calcular el cierre convexo de una nube de puntos.
- Poligonizar la nube de puntos utilizando el algoritmo de poligonización monótona.
- Calcular las regiones encontradas.
- Para cada región, aplicar el algoritmo MWT.

El cierre convexo y la poligonización se obtienen ejecutando los algoritmos correspondientes explicados en apartados anteriores. Una vez ejecutado los dos algoritmos, tenemos conjuntos de vértices, que representan diferentes polígonos: CC y Pol:




En la imagen de la izquierda vemos la nube de puntos, que contiene 14 nodos. En la imagen de la derecha, el resultado de aplicar los dos primeros pasos de la estrategias: poligonización y cierre convexo.

Visualmente, podemos ver que hay 5 regiones: la primera es la región que genera la poligonización, y las demás son los huecos que generan el cierre convexo y la poligonización. No obstante, encontrar las regiones de forma automática no es una tarea trivial.

Los dos subconjuntos de nodos (CC y Pol) que representan los dos polígonos son:

*CC = {6, 14, 18, 9, 1, 2,}*
*Pol = {1, 2, 3, 5, 6, 8, 11, 14, 13, 12, 10, 9, 7, 4}*

Para encontrar las subregiones, se hace lo siguiente:

- Recorrer los vértices de Pol.
- Encontrar una secuencia de al menos 3 vértices consecutivos de manera que solo el primero y el último vértice de la secuencia estén en CC.
- Repetir el paso anterior hasta que se encuentren todas las subregiones.

Al iniciar una nueva secuencia, se salta un vértice si el siguiente vértice de la secuencia también pertenece a CC. Es decir, que la secuencia sea solamente de dos vértices. Estos casos se dan cuando hay aristas en comunes entre la poligonización y el cierre convexo.

La secuencia siempre debe empezar por un vértice que pertenece a CC. En algunos casos, conviene ordenar los puntos de Pol para facilitar la búsqueda. El conjunto CC no necesita ser ordenado, ya que en cada interacción, solo se comprueba si el siguiente vértice esta o no en CC.

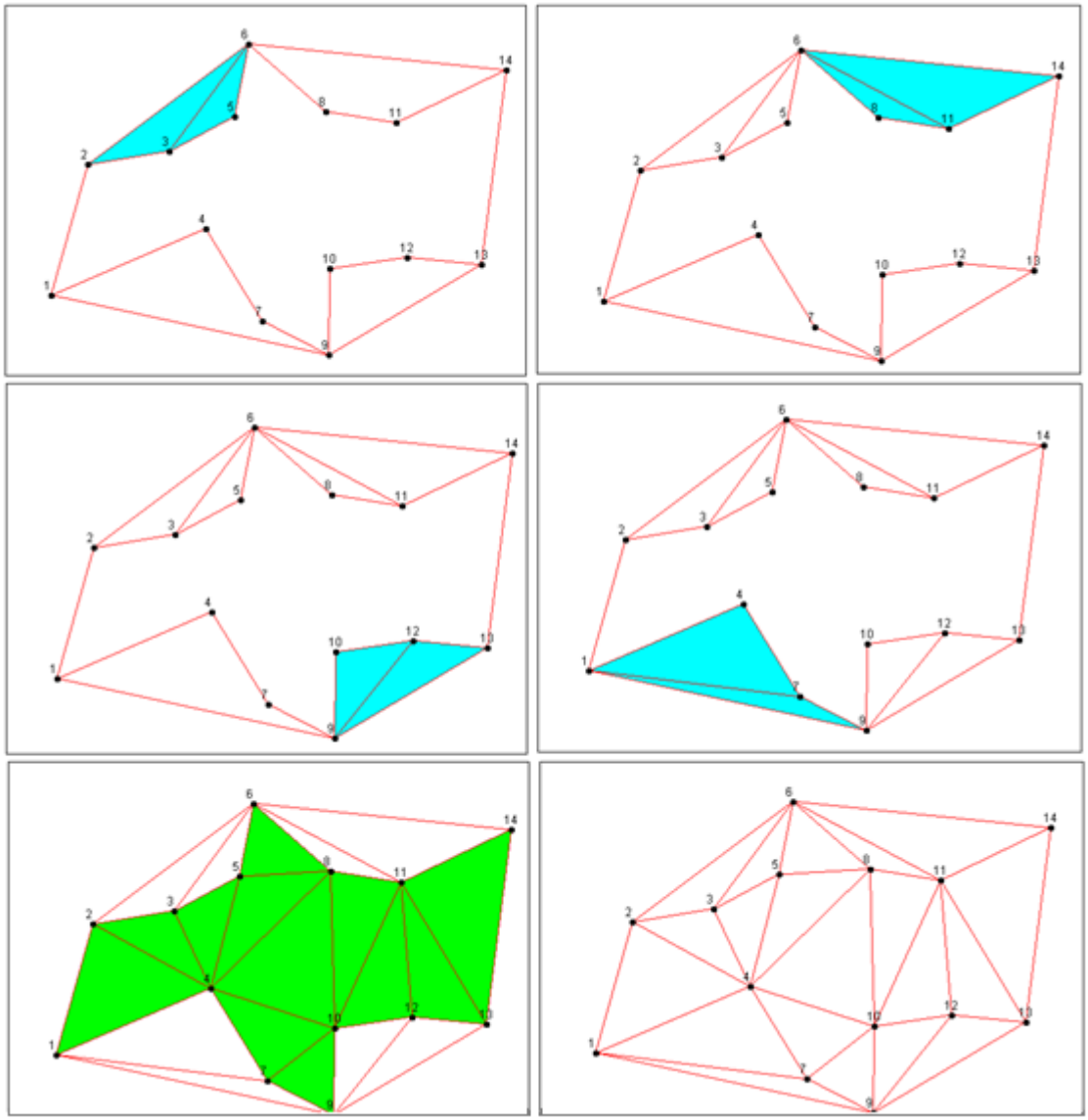
Volviendo a nuestro ejemplo anterior, al iniciar la primera secuencia, se saltaría el nodo 1 de Pol, ya que el siguiente nodo (el nodo 2), también se encuentra en el cierre convexo (CC).

El nodo 2 es válido para el principio de la secuencia. Se recorre el conjunto Pol hasta que encuentre un nodo que esté en CC. Los nodos 3 y 5 no están en CC y se añaden a la secuencia. El siguiente nodo, nodo 6, pertenece a CC. Se añade a la secuencia y, por tanto, se ha encontrado una nueva subregión: *{2, 3, 5 y 6}*.

Al final, se encuentran 4 subregiones:

- subregión 1 = {2, 3, 5, 6}
- subregión 2 = {6, 8, 11, 14}
- subregión 3 = {13, 12, 10, 9}
- subregión 4 = {9, 7, 4, 1}

Teniendo en cuenta la región generada por el polígono, se encuentran 5 regiones. El siguiente paso es aplicar MWT en cada región encontrada. Este paso es bastante sencillo, ya que el algoritmo MWT ya ha sido implementado anteriormente. El resultado es el siguiente:



#### *4.6.1.2 ESTRATEGIA 2*

La segunda estrategia para encontrar una aproximación de la triangulación de peso mínimo se basa en los siguientes pasos:

- Calcular el cierre convexo de la nube de puntos.
- Calcular la triangulación de Greedy.
- Calcular el árbol generador mínimo (MST) de la triangulación de Greedy.
- Unir las aristas del cierre y del MST.
- Obtener las ramas.
- Se formarán regiones. Obtener las regiones que se forman sin tener en cuenta las ramas generadas por el árbol mínimo.
- Asociar las ramas a las regiones encontradas.
- Para cada región encontrada, ejecutar el algoritmo de MWT.

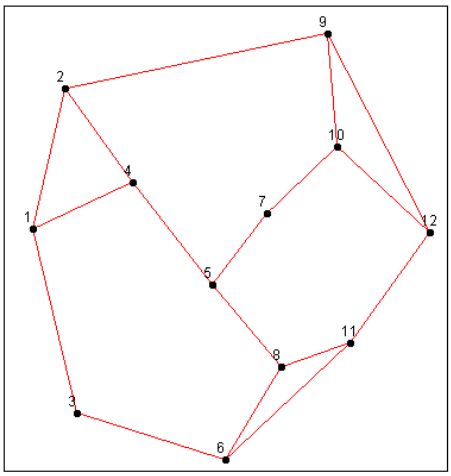
El cierre convexo, la triangulación de Greedy y el MST se calculan utilizando las funciones explicadas en anteriores apartados. La gran dificultad de este algoritmo reside en dos partes: la primera es calcular las regiones, y la segunda es calcular la MWT teniendo en cuenta las ramas.

Una vez obtenido el cierre convexo y el MST, hay que identificar las ramas y eliminarlas del grafo para poder calcular las regiones. El algoritmo para identificar las aristas que pertenecen a rama es el siguiente:

*Mientras exista vértices con solo 1 vecino:*
*v = vértice que solo tiene 1 vecino*
*a = arista que contiene el vértice v*
*eliminar arista a y vértice v del grafo*

*Verificar se la arista a pertenece a alguna rama anteriormente encontrada*
*Si pertenece:*
*rama = añadir a*
*Si no*
*rama = nueva rama con a*

El algoritmo para obtener las regiones es bastante más complejo que el algoritmo de la estrategia 1. Para poder explicarlo mejor, vamos a utilizar un ejemplo para ilustrar los pasos:




En este ejemplo, podemos ver la unión de las aristas del cierre convexo y del MST de la triangulación de Greedy. Como se puede observar, todas las aristas del cierre convexo participan en solamente 1 región, y por tanto, serán el punto de partida para calcular las regiones.

Se ordenan los puntos del cierre convexo. El primer elemento del conjunto es el punto que se encuentra más a la izquierda. La ordenación se hace en el sentido negativo, es decir, en siguiendo el sentido de las agujas del reloj. En nuestro ejemplo, la ordenación es la siguiente: *V = {1, 2, 9, 12, 11, 6, 3}*

Como mucho, pueden existir 7 regiones en el polígono, ya que hay 7 aristas en el cierre convexo. No obstante, las aristas consecutivas pueden pertenecer a una misma región,

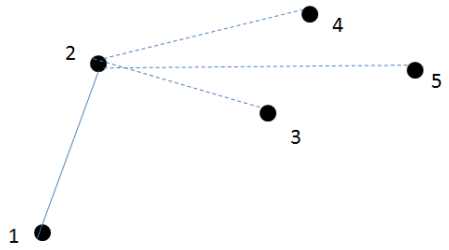
incluso a solo una. Es decir, el rango de posibles regiones de de 1 a N, con N igual al número de aristas en el cierre convexo.

Empezamos a ejecutar el algoritmo con la primera arista: (1,2). El punto a analizar es el vértice 2. Hay que analizar cuál de los vecinos del nodo 2 es el siguiente. Los nodos vecinos de 2 son los nodos 9 y 4. Para obtener el siguiente nodo de la región, se analizan las orientaciones de todos los vecinos en relación a la arista anterior, en este caso, en relación a la arista (1,2).

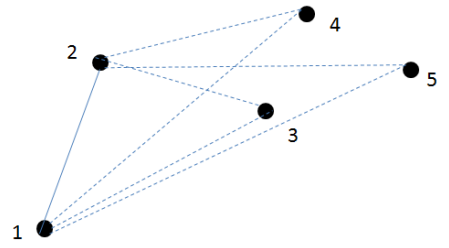
El método que se utiliza, para verificar cual de los nodos vecinos es el siguiente, tiene 3 ramificaciones:

1. La orientación de todos los vecinos en relación a la arista es negativa
2. No todos las orientaciones son negativas.
3. Todas las orientaciones son positivas.

Para entender el **primer caso**, estudiaremos una situación donde existen 3 vecinos del nodo a estudiar, y todos ellos se encuentran por debajo de la arista:




La arista a estudiar es la (1,2) y el nodo a estudiar es el nodo 2. Los vecinos de 2 son los nodos 3, 4, y 5. Para encontrar el siguiente nodo, tenemos que trazar las siguientes aristas: (4,1), (5,1) y (3,1).

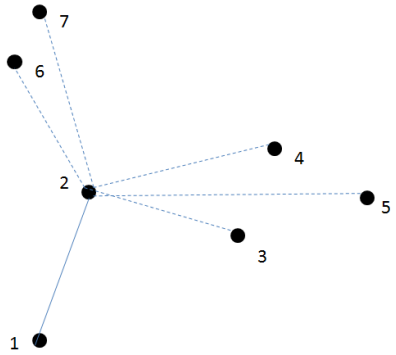



Se descartan los nodos cuyas aristas obtenidas anteriormente se crucen con alguna de las posibles nuevas aristas. Es decir, en nuestro ejemplo, tenemos dos conjuntos de aristas: las aristas (4,1), (5,1) y (3,1), y el conjunto de las posibles aristas, (2,4), (2,5) y (2,3). La arista (4,1) se cruza con la arista (2,5) y (2,3), por tanto, se descarta el nodo 4.

Todavía quedan 2 nodos posibles: el 5 y el 3. Para verificar cual de los dos es el siguiente, se analizan los triángulos (1,2,3) y (1,2,5). Se elige el triángulo que está

dentro de todos los demás triángulos. En este caso, el triángulo (1,2,3) está dentro del triángulo (1,2,5). **El punto elegido es el punto 3**.

El **segundo caso** a estudiar es cuando algunos de los vecinos del nodo actual tienen orientación positiva y otros tienen orientación negativa:

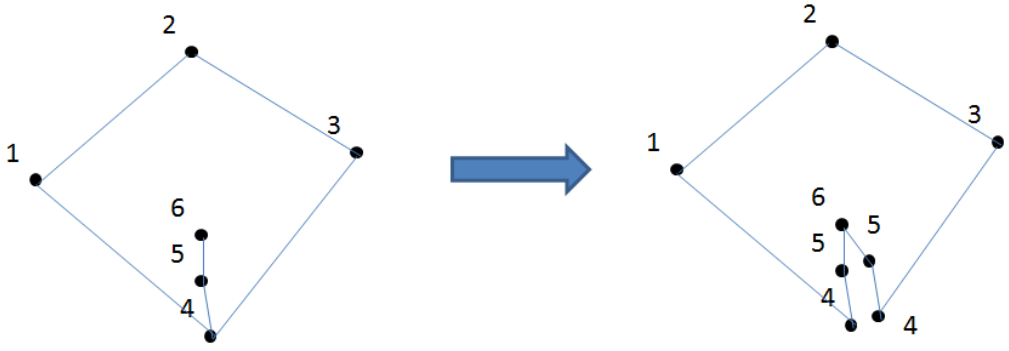



En este caso, se descartan los puntos que tienen la orientación positiva, y se ejecuta el mismo método descrito anteriormente, cuando todos los vecinos tienen orientación negativa.

El **último caso**, si todos los nodos vecinos tienen orientación positiva, el método a utilizar es justo el contrario que el primer caso, es decir, se elige el nodo que cruza con todas las aristas posibles o que contiene en su interior todos los triángulos.

Con este método, se pueden obtener todas las regiones.

Una vez obtenidas todas las regiones, el siguiente paso es asociar las ramas a las regiones. No se trata de un paso trivial, ya que los puntos de las ramas tienen que multiplicarse. Por ejemplo:




Los puntos duplicados (5 y 4) son, en la realidad, los mismos puntos, es decir, tienen las mismas coordenadas. Multiplicar los puntos de las ramas garantiza que el polígono siga teniendo una orientación negativa y se represente por una secuencia de puntos.

Para obtener los puntos de las ramas multiplicados y orientados negativamente, se ha implementado un algoritmo recursivo que hace una búsqueda en profundidad, analizando las orientaciones de los vecinos del nodo actual a estudiar. El algoritmo recibe una rama como parámetro de entrada, además de una lista auxiliar que tiene los siguientes puntos a estudiar, y el punto anterior estudiado. Al principio, el punto anterior es el punto de la rama que pertenece al polígono. Y la lista auxiliar solo tiene un punto, que es el primer nodo de la rama. El pseudocódigo es el siguiente:

```
p = primer punto de la lista auxiliar (punto a estudiar)
lista auxliar = eliminar p

salida = anyadir p

vecinos = obtener vecinos de p (excepto el punto anterior)
vecinos = ordenar puntos (orientados negativamente)

Mientras no estudie todos los vecinos
  siguiente = siguiente punto vecino a estudiar
  lista auxiliar = anyadir siguiente punto a estudiar
  LLAMADA RECURSIVA a la funcion para el siguiente punto a estudiar
  salida = anyadir p
```

Al final de la ejecución, en nuestro ejemplo, tenemos la secuencia de puntos: $P = \{4, 5, 6, 5, 4\}$.

El método también funciona cuando las ramas son más complejas, es decir, hayan más nodos, de de un mismo nodo, salga más de una ramificación.

Por fin ya se tienen las regiones con sus ramas. El último paso es aplicar MWT a cada región encontrada. Pero debido al hecho de que las regiones tienen "ramas" (nodos repetidos con la misma coordenada), hay que hacer algunas modificaciones del algoritmo MWT:

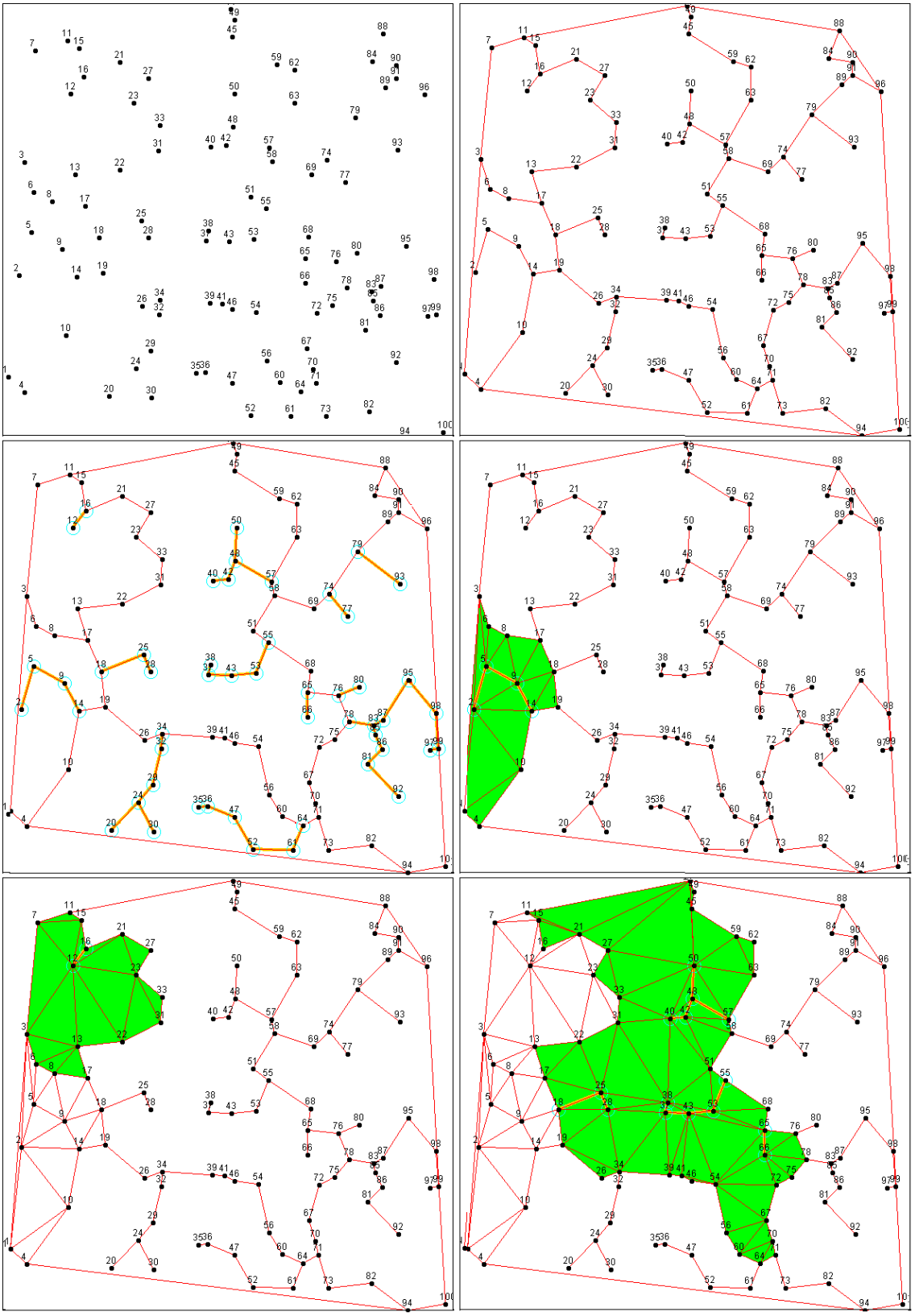
- **Comprobar si la arista es válida:** Si algún punto de la arista (i,j) pertenece a una rama, la arista es inválida solamente si se cruza con alguna arista existente.
- **Comprobar si la arista es externa al polígono:** Si uno de los puntos a estudiar es el inicio de la rama, no se tienen en cuenta los puntos de la rama para obtener el siguiente punto y el anterior en la lista de puntos.
- **Comprobar si el triángulo es válido:** Además de comprobar que las aristas son válidas, hay que comprobar que no haya ninguna rama en el interior del triángulo.

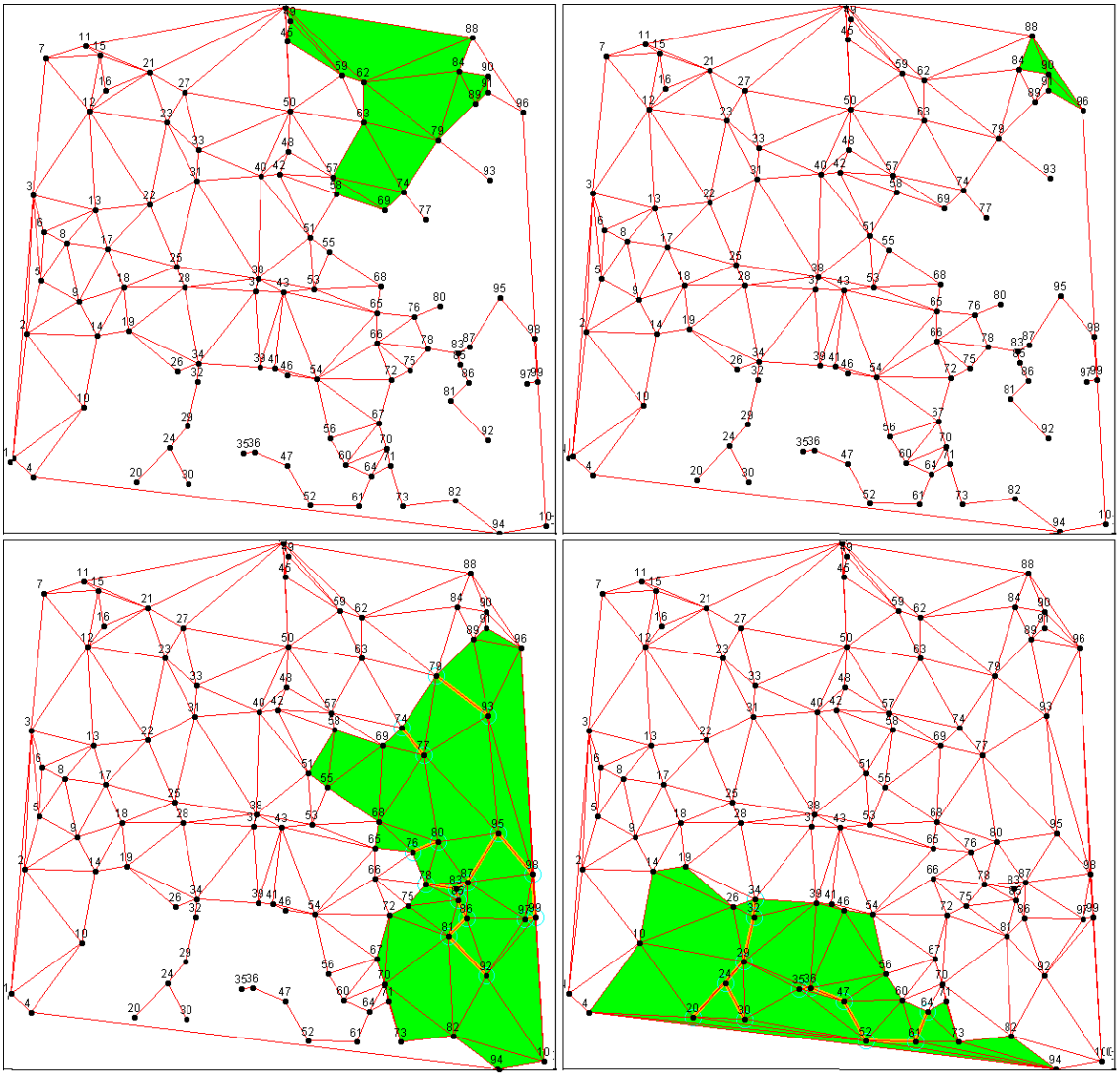
Además, aparece otro problema en la ejecución del algoritmo. Por tener puntos multiplicados, las matrices L y S tienen distintos índices que representan los mismos

puntos. Lo que se hace en estos casos, es acceder siempre a solo una columna que correspondan los índices multiplicados. Por ejemplo, la secuencia de puntos del polígono anterior es: *P = {1, 2, 3, 4, 5, 6, 5, 4}*. Los índices 4 y 8 corresponden al punto 4, y por lo tanto, corresponderán a una solo columna. Lo mismo pasa con los índices 5 y 7, que corresponden al punto 5.

En algunos casos especiales, cuando se dan casos de alineaciones por causa de las ramas, o cuando la complejidad de la rama es muy alta, el algoritmo falla y no llega a ejecutar de forma satisfactoria. Pero en la mayoría de los casos, el algoritmo ejecuta con éxito.

Un ejemplo de la ejecución del algoritmo es la siguiente:





#### *4.6.1.3 ESTRATEGIA 3*

La estrategia 3, en general, es muy parecida a la estrategia anterior. La única diferencia es que se calcula el árbol generador mínimo de la nube de puntos, y no de la triangulación de Greedy. Por lo tanto, los pasos a ejecutar son:

1. Calcular el cierre convexo de la nube de puntos.
2. Calcular la triangulación de Greedy.
3. Calcular el árbol generador mínimo (MST) de la nube de puntos.
4. Unir las aristas del cierre y del MST.
5. Obtener las ramas.
6. Se formaran regiones. Obtener las regiones que se forman sin tener en cuenta las ramas generadas por el árbol mínimo.
7. Asociar las ramas a las regiones encontradas.
8. Para cada región encontrada, ejecutar el algoritmo de MWT.

Como se puede ver, la única diferencia en relación en la estrategia anterior es el paso 3.

#### *4.6.1.4 ESTRATEGIA 5*

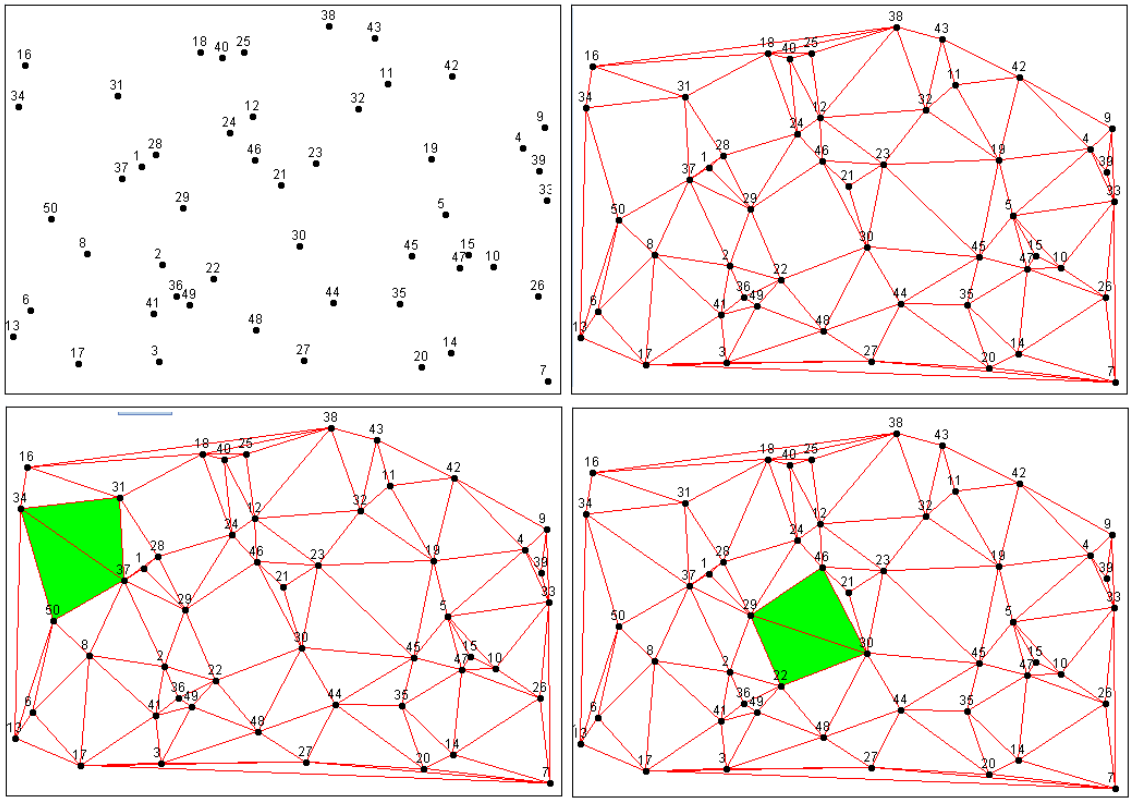
La quinta estrategia implementada se base en las aristas ligeras de la nube de punto. Se dice que una arista es arista ligera si no existe otra arista que la cruce y que sea de menor longitud. Por tanto, los pasos de la estrategia son los siguientes:

- Obtener todas las aristas ligeras de la nube de puntos.
- Para todas las regiones que no sean triángulos, aplicar el algoritmo de triangulación MWT de polígonos.

Como ya se ha dicho anteriormente, el cálculo de todas las aristas ligeras de una nube de puntos no genera ninguna rama, es decir, todas las regiones son o triángulos u otros polígonos.

Si las aristas ligeras de una nube de puntos forman una triangulación, entonces la triangulación de Greedy de la nube de puntos es una triangulación MWT (Loera, 2009).

A continuación vemos la secuencia de pasos de la triangulación de una nube de puntos utilizando la quinta estrategia:



### 4.6.2 ESTRATEGIA MLT

A continuación, explicaremos los detalles de la única estrategia que devuelve una triangulación MLT.

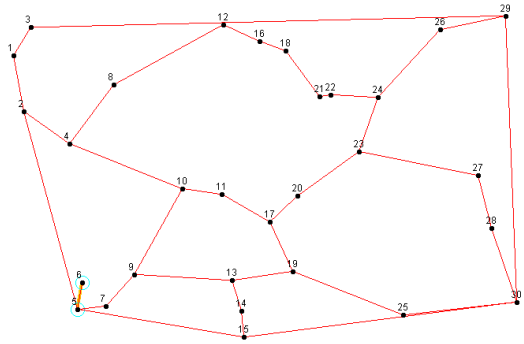
#### *4.6.2.1 ESTRATEGIA 4*

A diferencia de las demás estrategias comentadas en el apartado anterior, la estrategia 4 es la única que no devuelve una aproximación, sino el resultado exacto. Por tanto, la complejidad deja de ser NP-hard, como las triangulaciones MWT, y pasa a ser de un algoritmo de tiempo polinomial (Tan, 1993).

La estrategia ejecuta las siguientes operaciones:

- Cálculo del cierre convexo
- Cálculo del grafo RNG.
- Juntar el cierre convexo y el grafo RNG
- Triangular (MLT) las regiones que se obtienen.

El algoritmo se basa en la idea de que las aristas del cierre convexo y del grafo RNG pertenecen a la triangulación MLT de la nube de puntos (Tan, 1993). Al juntar las aristas del cierre convexo y del grafo RNG, obtenemos un grafo donde podemos obtener regiones menores.



Sin embargo, las regiones que se obtienen no siempre son regiones simples, ya que algunas de ellas pueden contener ramas. Por tanto, se aplica el algoritmo modificado para triangulación MLT de regiones con ramas (parecido a lo que se hizo en las estrategias 2 y 3).

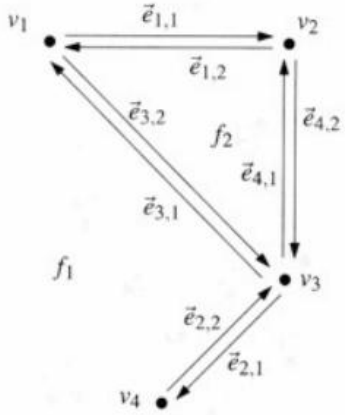
Se ha tenido que implementar otra forma de obtener las regiones, debido a que pueden aparecer regiones interiores, donde las aristas del cierre convexo no sirvan como soporte para buscar todas las regiones.

Para ello, se ha implementado la estructura DCEL. Para implementar dicha estructura, se ha utilizado el siguiente libro como base de información: (Mark de Berg, 2008).

Dicha estructura nos permite obtener fácilmente las regiones del grafo.

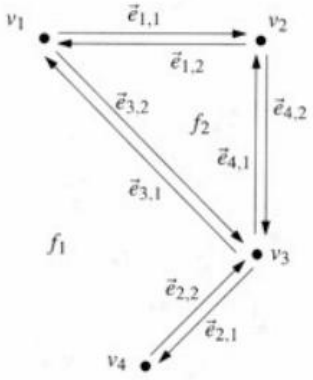
No es objetivo del proyecto explicar con detalles las propiedades y características de la estructura, así que nos centraremos en las propiedades básicas:

- Cada arista del grafo es representada dos veces, es decir, representando sus dos sentidos.
- A cada una de las aristas orientadas las denominaremos semi-aristas. La relación entre dos semi-aristas que se correspondan se llama arista gemela.
- Para cada vértice, guardamos las coordenadas del vértice y una de las aristas que inciden en él:




| Vertex | Coordinates | IncidentEdge |
|---|---|---|
| $v_1$ | $(0,4)$ | $\vec{e}_{1,1}$ |
| $v_2$ | $(2,4)$ | $\vec{e}_{4,2}$ |
| $v_3$ | $(2,2)$ | $\vec{e}_{2,1}$ |
| $v_4$ | $(1,1)$ | $\vec{e}_{2,2}$ |

- Para cada cara, guardamos una de las aristas de la componente exterior de la cara y una de las aristas de la componente interior de la cara:

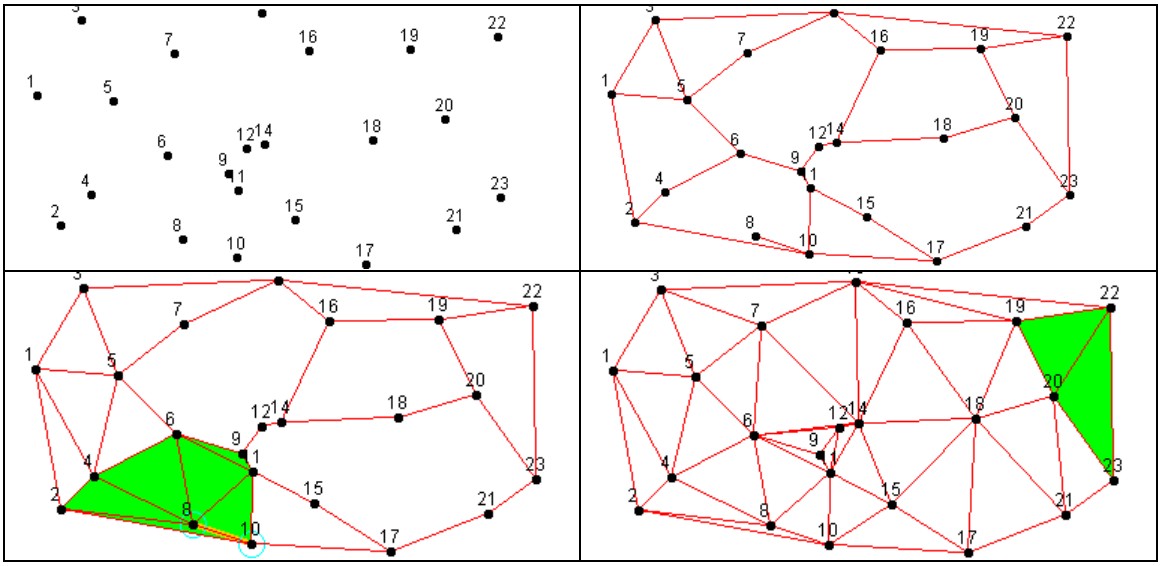



| Face | OuterComponent | InnerComponents |
|---|---|---|
| $f_1$ | **nil** | $\vec{e}_{1,1}$ |
| $f_2$ | $\vec{e}_{4,1}$ | **nil** |

- Para cada semi-arista, se guarda el vértice origen, la arista gemela, la cara a la arista incidente de la arista, la arista siguiente y la arista anterior.

Por tanto, los dos algoritmos que se ha implementado para manejar la estructura DCEL fueron el de crear la estructura a partir de un conjunto de aristas y el de obtener las regiones. Y para eso, se ha utilizada una librería JAVA de código libre.

Lo que queda del algoritmo es similar a las demás estrategias: triangular cada una de las regiones obtenidas. Algunos pasos del algoritmo pueden ser vistos en la siguiente secuencia de imágenes:



# 5 REQUISITOS, DISEÑO Y PRUEBAS DE LA APLICACIÓN

En este apartado definiremos los requisitos y diseños de la aplicación, además de realizar pruebas y comprobar el correcto funcionamiento del programa.

## 5.1 REQUISITOS DE DISEÑO

### 5.1.1 INTRODUCCIÓN

El documento de especificación de requisitos (ERS) proporciona una descripción clara e inequívoca de lo que ha de ser el sistema, incluyendo criterios precisos para evaluar el producto terminado, de forma que garanticen que hace lo que se supone que tiene que hacer.

Especificamos a continuación en lenguaje natural del sistema a realizar:

Se trata en crear una aplicación que sea capaz de visualizar grafos y de calcular algunos algoritmos de triangulación. Se trata de una aplicación gráfica que interactúa con el usuario. Los algoritmos pueden ejecutarse de forma directa o paso a paso.

### 5.1.2 METODOLOGÍA DE DESARROLLO

El método de desarrollo elegido será el método incremental, utilizando dos ciclos. Las fases de cada ciclo son los siguientes:

- Estudio del problema
- Requisitos
- Diseño de Alto Nivel

- Diseño de Bajo Nivel e Implementación
- Pruebas Unitarias y de Sistema

El segundo ciclo empieza cuando termine la fase de pruebas y corrección del primer ciclo. Los requisitos pueden cambiar, y por lo tanto, los diseños e implementación. En principio, en el segundo ciclo solo se desarrollarán más requisitos funcionales.

### 5.1.3 REQUISITOS A NIVEL DE SISTEMA

En esta sección se enumerarán los requisitos que deben cumplir el sistema final. Los requisitos a nivel del sistema se dividen en dos partes: Requisitos funcionales (funciones principales) y Requisitos de la interfaz gráfica (funciones de la misma).

#### *5.1.3.1 REQUISITOS DE LA INTERFAZ GRÁFICA*

Los requisitos de la interfaz gráfica son las funciones que facilitan el usuario a alcanzar sus objetivos deseado, es decir, que permita la interacción del mismo con las funciones del sistema. Las funciones de la interfaz gráfica que debe implementarse son las siguientes:

1. **Nuevo Proyecto:**
   Limpiar las pantallas, borrar vértices y aristas de todas las perspectivas.
2. **Nueva nube de puntos:**
   Borrar los nodos y las aristas de la perspectiva de la nube de puntos.
3. **Nuevo polígono:**
   Borrar los nodos y las aristas de la perspectiva de la triangulación de polígonos.
4. **Nueva estrategia:**
   Borrar los nodos y las aristas de la perspectiva de las estrategias.
5. **Nuevo "todas triangulaciones":**
   Borrar los nodos y las aristas de la perspectiva de todas las triangulaciones.
6. **Abrir proyecto:**
   Abrir proyecto existente, cargando nodos en todas las perspectivas.
7. **Guardar proyecto:**
   Guardar el proyecto actual.
8. **Añadir vértice:**
   Añadir un nuevo vértice en la nube de vértices. Si hay algún algoritmo activo, lo recalcula teniendo en cuenta el nuevo vértice.
9. **Borrar vértice:**
   Borrar un vértice. Si hay algún algoritmo activo, lo recalcula teniendo en cuenta la nueva nube de vértices.
10. **Mover vértice:**
    Mover un vértice de la nube de grafos. Si hay algún algoritmo activo, lo recalcula teniendo en cuenta la nueva posición del vértice.
11. **Añadir vértices aleatorios:**

Añadir de forma aleatoria la cantidad de vértices que pide el usuario. Se borran los vértices ya existentes.

12. **Ejecutar algoritmo:**
    Seleccionar y ejecutar uno de los algoritmos implementado.
13. **Ejecución paso a paso:**
    Seleccionar y ejecutar los algoritmos implementados de forma que se pueda ver la cada paso de la ejecución.
14. **Mostrar coordenadas:**
    En todo el momento, el usuario quiere saber la localización del ratón. Si hay algún algoritmo activo, lo recalcula teniendo en cuenta el nuevo vértice.
15. **Crear Polígono:**
    El usuario desea crear un polígono para, posteriormente, ejecutar paso a paso el algoritmo de MWT.
16. **Cerrar polígono:**
    Cerrar el polígono actual y empezar la triangulación.
17. **Mostrar estado y peso de la triangulación:**
    Mostrar el estado del paso actual y el la suma de los pesos de todas las aristas que aparecen en la pantalla.
18. **Copiar y pegar puntos:**
    Copiar los nodos de una perspectiva y pegarlos en otra o en la misma perspectiva.
19. **Mostrar aristas ligeras:**
    Mostrar las aristas ligeras de la nube de puntos o del polígono
20. **Mostrar pseudocódigo del algoritmo:**
    Mostrar el pseudocódigo y los pasos del algoritmo en ejecución.

#### *5.1.3.2 REQUISITOS FUNCIONALES DEL SISTEMA*

Los requisitos funcionales se tratan de funciones que debe realizar el sistema, y que proveen resultados o beneficios para el usuario. Como el objetivo principal del sistema es el cálculo de triangulaciones, los requisitos funcionales son:

##### 5.1.3.2.1 ALGORITMOS GENERALES:

1. **Poligonizar:**
   Poligonizar una nube de puntos.
2. **MST:**
   Calcular el árbol de peso mínimo (MST) a partir de una nube de puntos o de un grafo.
3. **Cierre convexo:**
   Se trata de calcular el cierre convexo dada una nube de puntos.
4. **Triangulación de Greedy o voraz:**
   Calcular la triangulación de Greedy dada una nube de puntos.

5. **Triangulación de Delaunay:**
   Calcular la triangulación de Delaunay dada una nube de puntos.
6. **RNG (Grafo de vecindad relativa):**
   Calcular el grafo RNG dada una nube de puntos.

5.1.3.2.2 ALGORITMOS DE POLÍGONOS

1. **MWT de polígonos:**
   Se debe calcular una trangulación de peso mínimo de un polígono.
2. **MLT de polígonos:**
   Se debe calcular una trangulación de peso mínima longitud de arista de un polígono.
3. **Triangulación EAR de polígonos:**
   Se debe calcular una trangulación EAR de un polígono.

5.1.3.2.3 ALGORITMOS DE ESTRATEGIAS

1. **Primera estrategia de triangulación:**
   Algoritmo para triangular un conjunto de puntos utilizando la primera estrategia definida en los objetivos del proyecto
2. **Segunda estrategia de triangulación:**
   Algoritmo para triangular un conjunto de puntos utilizando la segunda estrategia definida en los objetivos del proyecto
3. **Tercera estrategia de triangulación:**
   Algoritmo para triangular un conjunto de puntos utilizando la tercera estrategia definida en los objetivos del proyecto
4. **Cuarta estrategia de triangulación:**
   Algoritmo para triangular un conjunto de puntos utilizando la cuarta estrategia definida en los objetivos del proyecto
5. **Quinta estrategia de triangulación:**
   Algoritmo para triangular un conjunto de puntos utilizando la quinta estrategia definida en los objetivos del proyecto

5.1.3.2.4 ALGORITMOS DE TODAS LAS TRIANGULACIONES

1. **Calcular todas las triangulaciones por fuerza bruta:**
   Se implementará 4 algoritmos que calculan todas las triangulaciones por fuerza bruta.

### 5.1.4 CICLOS Y DIVISIÓN DE FUNCIONALIDADES

En este apartado se va a enumerar las funcionalidades del sistema que se implementarán en cada una de las fases.

#### *5.1.4.1 REQUERIMIENTOS CICLO 1*

En el primer ciclo se implementarán todas las funcionalidades de la interfaz gráfica. Es decir, este módulo estará completo al finalizar el ciclo. También se implementarán algunos algoritmos sencillos de teoría de grafos.

1. Todas funcionalidades de la interfaz gráfica.
2. Poligonizar
3. Cierre Convexo
4. Algoritmo de Greedy o voraz.

#### *5.1.4.2 REQUERIMIENTOS CICLO 2*

Se implementarán todas las funcionalidades que aparecen en los requisitos y que no fueron implementadas en el primer ciclo.

### 5.1.5 RESTRICCIONES DE DISEÑO E IMPLEMENTACIÓN

Para el diseño de alto nivel, se utilizará la notación UML. Y la codificación se hará en el lenguaje Java. Se utilizará la librería Java Swing para implementar las funcionalidades de la interfaz gráfica.

## 5.2 DISEÑO DE ALTO NIVEL

En la fase de Diseño de Alto Nivel de un ciclo de desarrollo se investiga sobre el problema, sobre los conceptos relacionados con el subconjunto de casos de uso que se esté tratando.

Por tanto, en este apartado, vamos presentar una descripción mucho más detallada a respecto del funcionamiento interno de cada requisito funcional, así como el diagrama de clases de la aplicación.

### 5.2.1 DESCRIPCIÓN DE LOS CASOS DE USO DE LA INTERFAZ GRÁFICA

En este apartado se describirán con detalles todos los pasos que debe realizar la interfaz para cumplir los objetivos de los casos de uso.

#### *5.2.1.1 NUEVO PROYECTO*

- **Nombre:** Nuevo proyecto.
- **Precondición**: No está en el medio de una acción de ejecución paso a paso.
- **Flujo de eventos:** El usuario desea empezar un nuevo proyecto. El sistema borra todos los vértices y aristas actuales en todas las perspectivas, y borra la pantalla.
- **Postcondición:** Se borran las aristas y vértices de todas las perspectivas.

#### *5.2.1.2 NUEVA NUBE DE PUNTOS*

- **Nombre:** Nueva nube de puntos

- **Precondición**: No está en el medio de una acción de ejecución paso a paso.
- **Flujo de eventos:** El usuario desea empezar una nueva nube de puntos. El sistema borra todos los vértices y aristas actuales en la perspectiva correspondiente, y borra la pantalla.
- **Postcondición:** Se borran las aristas y vértices de la perspectiva correspondiente.

#### *5.2.1.3 NUEVO POLÍGONO*

- **Nombre:** Nuevo polígono
- **Precondición**: No está en el medio de una acción de ejecución paso a paso.
- **Flujo de eventos:** El usuario desea empezar un nuevo polígono. El sistema borra todos los vértices y aristas actuales en la perspectiva correspondiente, y borra la pantalla.
- **Postcondición:** Se borran las aristas y vértices de la perspectiva correspondiente.

#### *5.2.1.4 NUEVA ESTRATEGIA*

- **Nombre:** Nueva estrategia
- **Precondición**: No está en el medio de una acción de ejecución paso a paso.
- **Flujo de eventos:** El usuario desea empezar una nueva estrategia. El sistema borra todos los vértices y aristas actuales en la perspectiva correspondiente, y borra la pantalla.
- **Postcondición:** Se borran las aristas y vértices de la perspectiva correspondiente.

#### *5.2.1.5 NUEVO "TODAS LAS TRIANGULACIONES"*

- **Nombre:** Nuevo "todas las triangulaciones"
- **Precondición**: No está en el medio de una acción de ejecución paso a paso.
- **Flujo de eventos:** El usuario desea empezar un nuevo "todas las triangulaciones". El sistema borra todos los vértices y aristas actuales en la perspectiva correspondiente, y borra la pantalla.
- **Postcondición:** Se borran las aristas y vértices de la perspectiva correspondiente.

#### *5.2.1.6 ABRIR PROYECTO*

- **Nombre:** Abrir proyecto
- **Precondición**: No está en el medio de una acción de ejecución paso a paso.
- **Flujo de eventos:** El usuario desea abrir un nuevo proyecto cargado previamente. El sistema borra todas las aristas y vértices de las perspectivas y carga el fichero dado por el usuario, que contiene las posiciones de los puntos.

- **Postcondición:** Se borran las aristas y vértices de la perspectiva correspondiente y se cargan los vértices definidos en el fichero de entrada.

#### *5.2.1.7 GUARDAR PROYECTO*

- **Nombre:** Guardar proyecto
- **Precondición**: No está en el medio de una acción de ejecución paso a paso.
- **Flujo de eventos:** El usuario desea guardar el proyecto (todos los nodos de todas las perspectivas) en un fichero.
- **Postcondición:** Guarda las posiciones de los nodos en cada perspectiva en un fichero.

#### *5.2.1.8 AÑADIR VÉRTICE*

- **Nombre:** Añadir vértice.
- **Precondición:** No se está en medio de una ejecución paso a paso.
- **Flujo de eventos:** El usuario desea gestionar un vértice y pulsa en un la posición de la pantalla. El sistema localiza las coordenadas. El sistema crea un nuevo vértice y lo enseña en pantalla junto a los demás. Si hay un algoritmo activo, se debe recalcular teniendo en cuenta en nuevo vértice.
- **Caminos alternativos:** El vértice añadido se encuentra a menos de 3 pixeles de distancia a otro vértice ya existente. En este caso, la función termina con error.
- **Postcondición:** Se añade un nuevo vértice a la nube de vértices.
- **Atributos:** Coordenadas.

#### *5.2.1.9 BORRAR VÉRTICE*

- **Nombre:** Borrar vértice.
- **Precondición:** Existe el vértice. No se está en el medio de una ejecución paso a paso.
- **Flujo de eventos:** El usuario desea borrar un vértice de la nube de vértices. El vértice es borrado de la nube.
- **Caminos alternativos:** Si hay algún algoritmo activo, se recalcula teniendo en cuenta la nueva nube de vértices.
- **Postcondición:** Se borra el vértice de la nube de vértices.
- **Atributos:** Vértice a borrar.

#### *5.2.1.10 MOVER VÉRTICE*

- **Nombre:** Mover vértice.
- **Precondición:** Vértice existe y no se está en el medio de una ejecución paso a paso.
- **Flujo de eventos:** El usuario desea mover un vértice ya existente. El usuario selecciona el vértice y mantiene el ratón pulsado. Entonces, empieza a moverlo. El sistema captura los movimientos del ratón y enseña a cada momento la nueva

posición del vértice. El usuario elige la nueva posición y libera el ratón. El sistema crea el nuevo punto y lo añade a los vértices actuales. Se recalcula el algoritmo actual activo.

- **Caminos alternativos:** Caso ya exista un vértice a menos de 3 pixeles de distancia del nuevo vértice, se borra el vértice y termina la función con error.
- **Postcondición**: El vértice seleccionado se encuentra en una nueva posición.
- **Atributos:** Vértice seleccionado. Nueva posición.

#### *5.2.1.11 AÑADIR VÉRTICES ALEATORIOS*

- **Nombre:** Añadir vértices aleatorios.
- **Precondición:** No se está en el medio de una ejecución paso a paso.
- **Flujo de eventos:** El usuario indica cuantos nodos quiere insertar. El sistema borra los vértices y aristas actuales. Para todos los nuevos nodos a insertar, el sistema calcula de forma aleatoria sus posiciones (x,y).
- **Caminos alternativos:** Si el número elegido por el usuario es menor que 1, la función termina con error. Casos ya exista un nodo que tenga la misma posición, se vuelve a recalcular las coordenadas.
- **Postcondición:** Se añaden nodos aleatorios en el sistema.
- **Atributos:** Número de nodos aleatorios a insertar

#### *5.2.1.12 EJECUTAR ALGORITMO*

- **Nombre:** Ejecutar Algoritmo.
- **Precondición:** No se está en el medio de una ejecución paso a paso.
- **Flujo de eventos:** El usuario selecciona el algoritmo a ejecutar. El sistema detecta el tipo de algoritmo. Envía la lista de puntos a la función del sistema que calcule dicho algoritmo y recibe la lista de nuevas aristas como retorno de dicha función. El sistema enseña por pantalla las aristas resultantes de la ejecución.
- **Caminos alternativos**: Si el número de vértices es menor que 3, la función no se ejecuta y termina con error.
- **Postcondición:** Se enseña en pantalla el resultado del algoritmo seleccionado. Se muestra en pantalla (en el lado inferior izquierdo el pseudocódigo del algoritmo ejecutado) .
- **Atributos:** Puntos y algoritmo a ejecutar.

#### *5.2.1.13 EJECUCIÓN PASO A PASO*

- **Nombre:** Ejecución paso a paso
- **Precondición:** La nube de vértices tiene más que 2 vértices.
- **Flujo de eventos:** El usuario selecciona el algoritmo que quiere que se ejecute paso a paso. El sistema deshabilita todos los demás botones y menús del sistema, excepto los relativos a la ejecución paso a paso. El usuario pude avanzar y retroceder en la ejecución. Cuando la ejecución termina, se habilitarán los

botones nuevamente. Se enseñará en pantalla el resultado de la ejecución final del algoritmo.

- **Caminos alternativos:** En cualquier momento el usuario desea parar la ejecución del algoritmo. Se detiene la ejecución y no se enseña en pantalla el resultado final del algoritmo seleccionado.
- **Postcondición:** Se enseña la ejecución del algoritmo paso a paso, y una vez que termine, el resultado final del mismo. Se muestra en pantalla (en el lado inferior izquierdo el pseudocódigo del algoritmo ejecutado).

#### *5.2.1.14 MOSTRAR COORDENADAS*

- **Nombre:** Mostrar Coordenadas.
- **Flujo de eventos:** En todo el momento, el usuario desea saber las coordenadas que se encuentra el ratón, para poder utilizarlo. El sistema debe capturar, en todo el momento, la posición actual del ratón y enseñar en pantalla la posición actual.
- **Postcondición:** Se enseña en pantalla las coordenadas actuales del ratón.
- **Atributos:** Coordenadas.

#### *5.2.1.15 CREAR POLÍGONO*

- **Nombre:** Crear polígono.
- **Flujo de eventos:** El usuario desea crear un polígono para, posteriormente, ejecutar paso a paso el algoritmo de MWT, MT o EAR. Para ello, el usuario pulsará en pantalla para crear el polígono. Cuando haya creado el polígono, lo cerrará pulsando en un botón.
- **Postcondición:** Se enseña en pantalla las coordenadas actuales del ratón.
- **Atributos:** Coordenadas.

#### *5.2.1.16 CERRAR POLÍGONO*

- **Nombre:** Cerrar polígono
- **Flujo de eventos:** El usuario desea empezar el algoritmo de triangular polígonos. Para ello, tiene que cerrar el polígono actual. El usuario pulsa el botón cerrar polígono, y una ventana emergente aparece indicando que elija el algoritmo que quiere ejecutar.
- **Caminos alternativos:** Alguna arista del polígono se cruza con otra o no hay suficientes puntos. En ambos casos, el sistema avisa el usuario con un mensaje de error.
- **Postcondición:** Se cierra el polígono y empieza la ejecución del algoritmo elegido por el usuario.

#### *5.2.1.17 MOSTRAR ESTADO Y PESO DE LA TRIANGULACIÓN*

- **Nombre:** Mostrar estado y peso de la triangulación

- **Flujo de eventos:** Durante cualquier paso de la ejecución paso a paso o durante la finalización de cualquier algoritmo, el usuario desea saber el estado de la ejecución actual y el la suma de los pesos de las aristas de la triangulación.
- **Postcondición:** El usuario puede ver el estado de la ejecución y la suma de los pesos de las aristas.

#### *5.2.1.18 COPIAR Y PEGAR PUNTOS*

- **Nombre:** Copiar y pegar puntos
- **Precondición:** No se está en el medio de una ejecución paso a paso.
- **Flujo de eventos:** El usuario desea copiar los puntos de una determinada perspectiva y pegarlos en otra. Para ello, pulsa en el menú "editar -> copiar", cambia la perspectiva, pulsa en el botón derecho "editar -> pegar". El sistema copia los puntos de una perspectiva a otra, borrando los puntos que ya estaban en la perspectiva de destino.
- **Postcondición:** Se copian los puntos de una perspectiva a otra, borrando los puntos que ya estaban en la perspectiva de destino.

#### *5.2.1.19 MOSTRAR ARISTAS LIGERAS*

- **Nombre:** Mostrar aristas ligeras
- **Flujo de eventos:** El usuario, en cualquier momento, desea ver las aristas ligeras de una triangulación o de un polígono. Para ello, solo tiene que "checkear" la opción correspondiente en la barra de herramientas. El sistema calcula las aristas ligeras y las enseña en pantalla, en color amarillo y de un mayor grosor, y por debajo de todo que esté en pantalla.
- **Postcondición:** El sistema calcula las aristas ligeras de una nube de puntos o de un polígono, enseñándola en color amarillo y de un mayor grosor.

#### *5.2.1.20 MOSTRAR PSEUDOCÓDIGO DEL ALGORITMO*

- **Nombre:** Mostrar pseudocódigo del algoritmo
- **Precondición:** Debe estar en el medio de una ejecución paso a paso o después de una ejecución directa
- **Flujo de eventos:** El usuario desea ver el pseudocódigo del algoritmo en ejecución y en que parte del código nos encontramos cuando estamos en el medio de una ejecución paso a paso.
- **Postcondición:** Si se trata de una ejecución paso a paso, se muestra el pseudocódigo del algoritmo actual y en paso del algoritmo estamos ejecutando. Si se trata de una ejecución directa, solamente muestra el pseudocódigo del algoritmo.

### 5.2.2 DESCRIPCIÓN DE LOS CASOS DE USO FUNCIONALES DEL SISTEMA

En este apartado se describirán con detalles todos los pasos que debe realizar el sistema para implementar los requisitos definidos en apartados anteriores.

#### *5.2.2.1 POLIGONIZAR*

- **Nombre:** Poligonizar
- **Flujo de eventos:** El usuario desea ejecutar el algoritmo de poligonización. El sistema utiliza el método de poligonización monótona para poligonizar la nube de puntos.
- **Camino alternativo:** Si el número de vértices de la nube es menor que 3, la función termina con error. Si todos los puntos del sistema están alineados, no se garantiza que la función ejecute con éxito. Caso sea una ejecución paso a paso, se guardarán las acciones que se crean convenientes.
- **Postcondición:** Se ejecuta el algoritmo de poligonización monótona y se muestra el resultado en pantalla.

#### *5.2.2.2 MST*

- **Nombre:** MST
- **Flujo de eventos:** El usuario desea ejecutar el algoritmo de MST. El sistema calcula el MST de la nube de vértices, dado sus aristas.
- **Caminos alternativos:** Si el número de vértices de la nube es menor que 3, la función termina con error. Si todos los puntos del sistema están alineados, no se garantiza que la función ejecute con éxito. Caso sea una ejecución paso a paso, se guardarán las acciones que se crean convenientes. Caso el conjunto de aristas de entrada sea vacío, se calcula todas las posibles aristas de la nube y se ejecuta normalmente el algoritmo.
- **Postcondición:** Se ejecuta el algoritmo de cálculo del MST.

#### *5.2.2.3 CIERRE CONVEXO*

- **Nombre:** Cierre Convexo
- **Flujo de eventos:** El usuario desea ejecutar el algoritmo de cierre convexo. El sistema calcula el cierre convexo de la nube de puntos actual.
- **Caminos alternativos:** Si el número de vértices de la nube es menor que 3, la función termina con error. Si todos los puntos del sistema están alineados, no se garantiza que la función ejecute con éxito. Caso sea una ejecución paso a paso, se guardarán las acciones que se crean convenientes.
- **Postcondición:** Se ejecuta el algoritmo de cierre convexo y se muestra el resultado en pantalla.

#### *5.2.2.4 TRIANGULACIÓN DE GREEDY O VORAZ*

- **Nombre:** Triangulación de Greedy
- **Flujo de eventos:** El usuario desea ejecutar el algoritmo de triangulación de Greedy. El sistema calcula la triangulación de la nube de puntos actual.
- **Caminos alternativos:** Si el número de vértices de la nube es menor que 3, la función termina con error. Si todos los puntos del sistema están alineados, no se garantiza que la función ejecute con éxito. Caso sea una ejecución paso a paso, se guardarán las acciones que se crean convenientes.
- **Postcondición:** Se ejecuta el algoritmo de triangulación de Greedy y se muestra el resultado en pantalla.

#### *5.2.2.5 TRIANGULACIÓN DE DELAUNAY*

- **Nombre:** Triangulación de Delaunay
- **Flujo de eventos:** El usuario desea ejecutar el algoritmo de triangulación de Delaunay. El sistema calcula la triangulación de la nube de puntos actual.
- **Caminos alternativos:** Si el número de vértices de la nube es menor que 3, la función termina con error. Si todos los puntos del sistema están alineados, no se garantiza que la función ejecute con éxito. Caso sea una ejecución paso a paso, se guardarán las acciones que se crean convenientes.
- **Postcondición:** Se ejecuta el algoritmo de triangulación de Delaunay y se muestra el resultado en pantalla.

#### *5.2.2.6 RNG (GRAFO DE VECINDAD RELATIVA)*

- **Nombre:** RNG
- **Flujo de eventos:** El usuario desea ejecutar el algoritmo que calcula el grafo RNG de vecindad relativa. El sistema calcula el grafo nube de puntos actual a partir de la triangulación de Delaunay.
- **Caminos alternativos:** Si el número de vértices de la nube es menor que 3, la función termina con error. Si todos los puntos del sistema están alineados, no se garantiza que la función ejecute con éxito. Caso sea una ejecución paso a paso, se guardarán las acciones que se crean convenientes.
- **Postcondición:** Se ejecuta el algoritmo del grafo RNG y se muestra el resultado en pantalla.

#### *5.2.2.7 MWT DE POLÍGONOS*

- **Nombre:** MWT
- **Flujo de eventos:** El usuario desea cerrar el polígono y visualizar los pasos del algoritmo que calcula la triangulación MWT.
- **Caminos alternativos:** Alguna arista se cruza o no hay menos de 3 puntos en pantalla. En ambos casos, el sistema indica al usuario que no puede ejecutar la operación.

- **Postcondición:** El sistema muestra los pasos de la ejecución y el resultado al usuario.

#### *5.2.2.8 MLT DE POLÍGONOS*

- **Nombre:** MLT
- **Flujo de eventos:** El usuario desea cerrar el polígono y visualizar los pasos del algoritmo que calcula la triangulación MLT.
- **Caminos alternativos:** Alguna arista se cruza o no hay menos de 3 puntos en pantalla. En ambos casos, el sistema indica al usuario que no puede ejecutar la operación.
- **Postcondición:** El sistema muestra los pasos de la ejecución y el resultado al usuario.

#### *5.2.2.9 TRIANGULACIÓN EAR DE POLÍGONOS*

- **Nombre:** EAR
- **Flujo de eventos:** El usuario desea cerrar el polígono y visualizar los pasos del algoritmo que calcula la triangulación EAR.
- **Caminos alternativos:** Alguna arista se cruza o no hay menos de 3 puntos en pantalla. En ambos casos, el sistema indica al usuario que no puede ejecutar la operación.
- **Postcondición:** El sistema muestra los pasos de la ejecución y el resultado al usuario.

#### *5.2.2.10 PRIMERA ESTRATEGIA DE TRIANGULACIÓN*

- **Nombre:** Estrategia 1
- **Flujo de eventos:** El usuario desea ejecutar la primera estrategia de triangulación. La estrategia consta de los siguientes pasos: poligonizar una nube de puntos; calcular el cierre convexo; juntar las aristas obtenidas en los pasos anteriores; obtener las subregiones; calcular MWT de cada subregión.
- **Caminos alternativos:** Si el número de vértices de la nube es menor que 3, la función termina con error. Si todos los puntos del sistema están alineados, no se garantiza que la función ejecute con éxito.
- **Postcondición:** Se ejecuta la estrategia de triangulación y se muestra el resultado en pantalla.

#### *5.2.2.11 SEGUNDA ESTRATEGIA DE TRIANGULACIÓN*

- **Nombre:** Estrategia 2
- **Flujo de eventos:** El usuario desea ejecutar la segunda estrategia de triangulación. La estrategia consta de los siguientes pasos: calcular el árbol generador mínimo de la nube de puntos; calcular el cierre convexo; juntar las

aristas obtenidas en los pasos anteriores; encontrar las ramas; obtener las subregiones; calcular MWT de cada subregión.

- **Caminos alternativos:** Si el número de vértices de la nube es menor que 3, la función termina con error. Si todos los puntos del sistema están alineados, no se garantiza que la función ejecute con éxito.
- **Postcondición:** Se ejecuta la estrategia de triangulación y se muestra el resultado en pantalla.

#### *5.2.2.12 TERCERA ESTRATEGIA DE TRIANGULACIÓN*

- **Nombre:** Estrategia 3
- **Flujo de eventos:** El usuario desea ejecutar la tercera estrategia de triangulación. La estrategia consta de los siguientes pasos: calcular la triangulación de greedy de la triangulación de greedy; calcular el árbol generador mínimo de la nube de puntos; calcular el cierre convexo; juntar las aristas obtenidas en los pasos anteriores; encontrar las ramas; obtener las subregiones; calcular MWT de cada subregión.
- **Caminos alternativos:** Si el número de vértices de la nube es menor que 3, la función termina con error. Si todos los puntos del sistema están alineados, no se garantiza que la función ejecute con éxito.
- **Postcondición:** Se ejecuta la estrategia de triangulación y se muestra el resultado en pantalla.

#### *5.2.2.13 CUARTA ESTRATEGIA DE TRIANGULACIÓN*

- **Nombre:** Estrategia 4
- **Flujo de eventos:** El usuario desea ejecutar la cuarta estrategia de triangulación. La estrategia consta de los siguientes pasos: calcular el cierre convexo, calcular el grafo RNG, juntar los dos pasos anteriores. Para cara región que forma (con posibles ramas), ejecutar el algoritmo para calcular la triangulación MLT de polígonos.
- **Caminos alternativos:** Si el número de vértices de la nube es menor que 3, la función termina con error. Si todos los puntos del sistema están alineados, no se garantiza que la función ejecute con éxito.
- **Postcondición:** Se ejecuta la estrategia de triangulación y se muestra el resultado en pantalla.

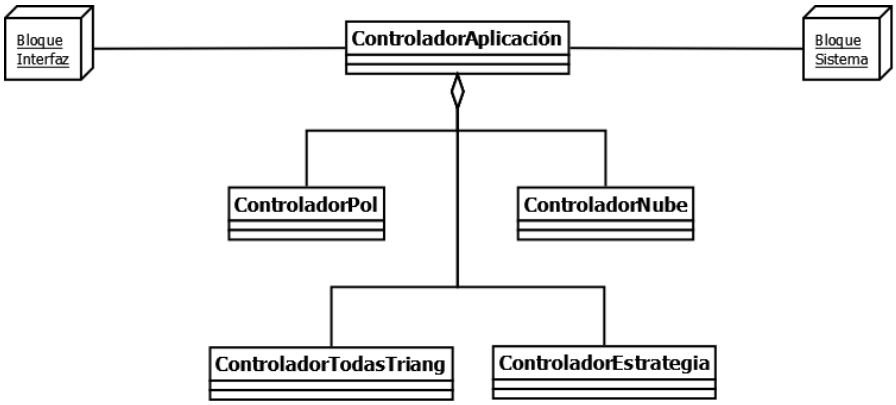
#### *5.2.2.14 QUINTA ESTRATEGIA DE TRIANGULACIÓN*

- **Nombre:** Estrategia 5
- **Flujo de eventos:** El usuario desea ejecutar la cuarta estrategia de triangulación. La estrategia consta de los siguientes pasos: calcular las aristas ligeras. Para cada región que se forma que no sea una triangulación, aplicar el algoritmo MWT de polígonos.

- **Caminos alternativos:** Si el número de vértices de la nube es menor que 3, la función termina con error. Si todos los puntos del sistema están alineados, no se garantiza que la función ejecute con éxito.
- **Postcondición:** Se ejecuta la estrategia de triangulación y se muestra el resultado en pantalla.

### 5.2.3 DIAGRAMAS DE CLASES

El sistema posee tres bloques diferentes: bloque de la interfaz gráfica, controladores, y bloque del sistema. Los controladores funcionan como una fachada entre los dos bloques, haciendo con que la programación del bloque del sistema sea independiente de la programación del bloque de la interfaz gráfica:




#### *5.2.3.1 CONTROLADORES*

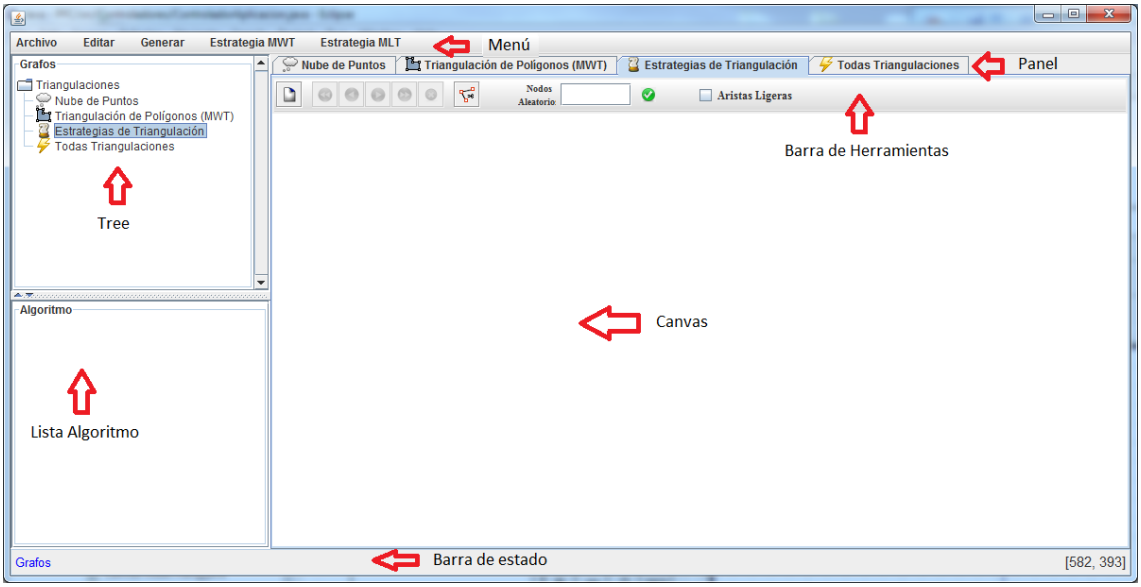
Existe 1 controlador para cada perspectiva diferente, además del controlador de la aplicación, que realiza operaciones generales de la aplicación, donde se implican todas las perspectivas.

El controlador de la aplicación es creado por el objeto Ventana, y se encarga de crear los objetos de la interfaz gráfica, los demás controladores, y de sincronizarlos.

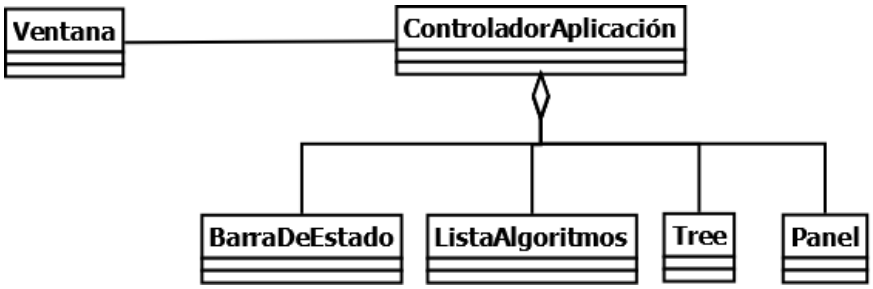
Cada controlador específico tiene todas las variables que controla el estado de su perspectiva. Los estados se cambien con acciones provenientes de la interfaz gráfica. Además, es capaz de comunicar con los objetos del sistema para delegar las tareas.

#### *5.2.3.2 BLOQUE DE LA INTERFAZ GRÁFICA*

Los principales objetos de la interfaz gráfica son: tree, lista algoritmos, canvas, barra de herramientas, menú y panel.

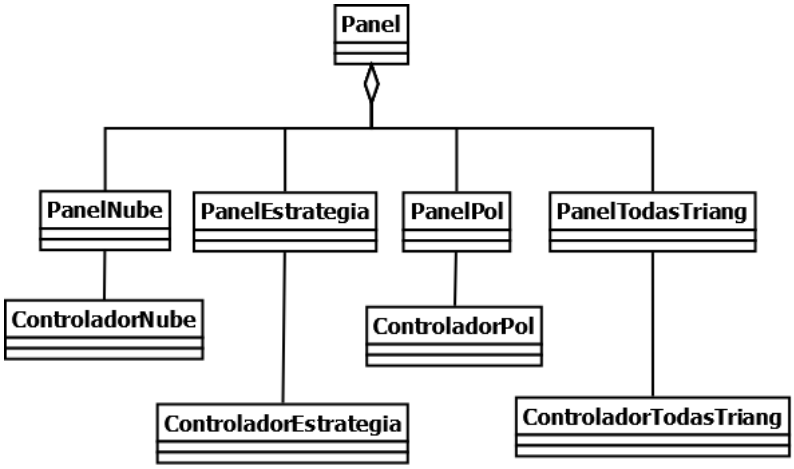



La relación dentre los objetos, el controlador de la aplicación y la ventana se puede ver en la siguiente estructura:

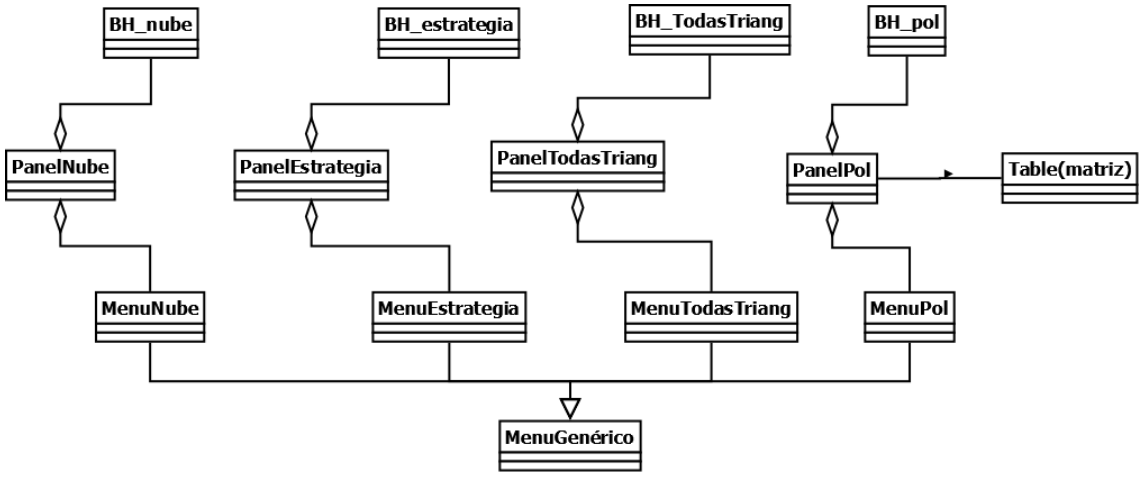



### *5.2.3.3 PANELES Y CANVAS*

Existe un canvas distinto para cada perspectiva. Todos ellos, implementan las interfaces de "mouseAction y mouseMoveAction". Los paneles comunican dichos eventos al controlador correspondientes:

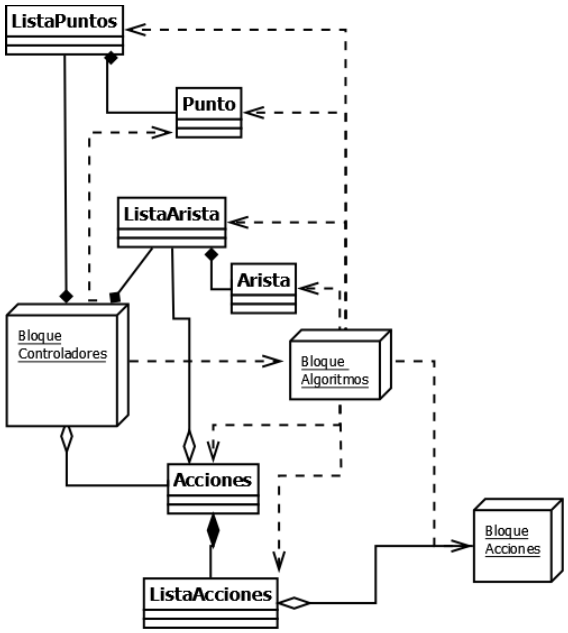


Además, cada panel posee su propia barra de herramienta y su propio menú. El panel de polígonos, además, contiene dos matrices: L y S.




### *5.2.3.4 BLOQUE DEL SISTEMA*

En este aparatado mostraremos la relación entre los elementos del sistema y los controladores:




Cada controlador específico contiene estas relaciones con los puntos, lista de puntos, aristas y listas de aristas.

Además, podemos ver dos bloques distintos: los de algoritmo y los de acciones.

Las acciones son elementos que se pintan en pantalla y que se crean durante la ejecución de los algoritmos paso a paso.

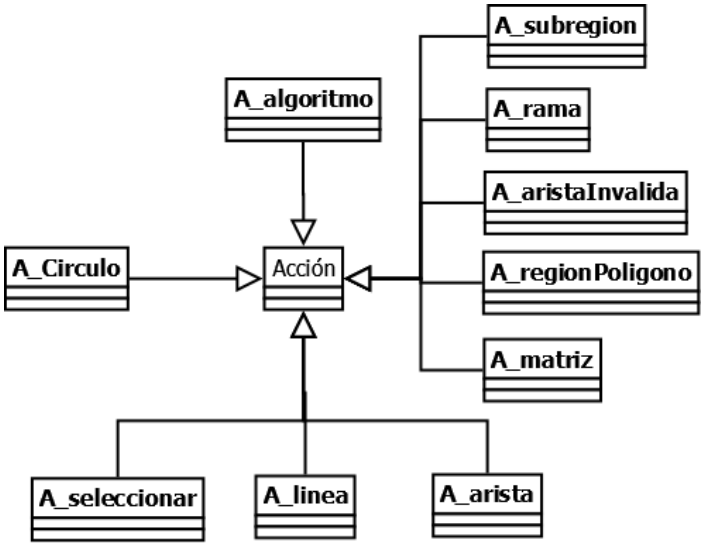
Para cada algoritmo que se puede ejecutar, se ha creado una clase estática para implementar su código

### *5.2.3.5 BLOQUE DE ACCIONES*

Las acciones son objetos que se crean durante la ejecución paso a paso, y se guardan en listas de acciones, y las listas de acciones se guardan en el vector de acciones.

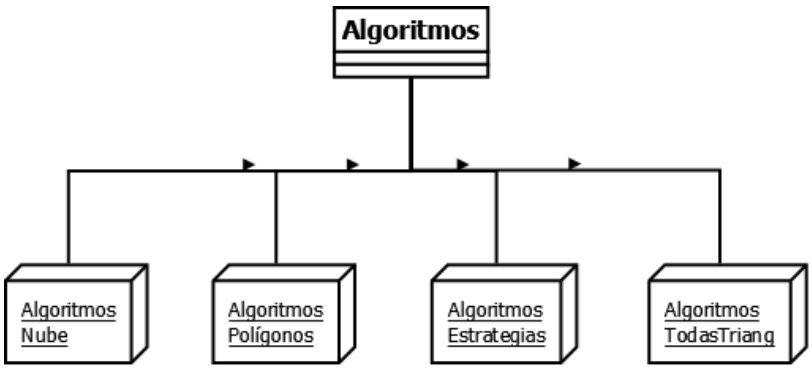
En la ejecución paso a paso, se obtiene las listas de acciones de la posición correspondiente del vector de acciones. Luego, obtenemos cada acción de la lista y la pintamos en pantalla.

Es decir, en una ejecución paso a paso, se ejecuta todo el algoritmo, guardando las acciones en las listas, y las listas en el vector, y vamos recorriendo el vector y las listas para pintar los pasos en pantalla.



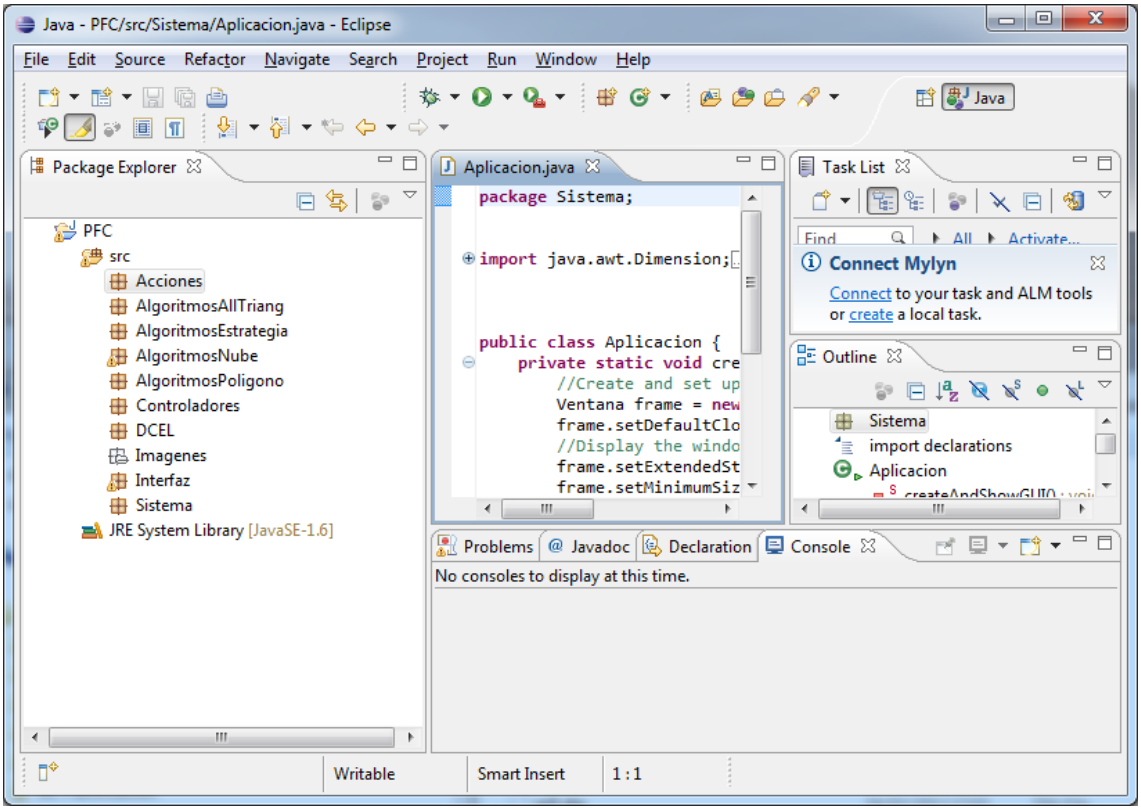

#### *5.2.3.6 BLOQUE DE ALGORITMOS*

Los algoritmos se dividen en bloques menores. Cada bloque menor corresponde a un conjunto de algoritmos distintos: algoritmos de nubes, polígonos, estrategias y todas triangulaciones. Cada bloque menor agrupa los algoritmos correspondientes a cada perspectiva.



#### *5.2.3.7 ORGANIZACIÓN DEL PROYECTO JAVA*

En la siguiente imagen, vemos como se ha organizado las distintas clases dentro del proyecto JAVA:




## 5.3 PRUEBAS

En este aparatado, vamos a realizar las pruebas unitarias de los módulos y las pruebas de integración.

### 5.3.1 PRUEBAS UNITARIAS

El objetivo de las pruebas unitarias es el de comprobar el correcto funcionamiento de la mayoría de las operaciones de cada módulo. Una vez terminada la codificación de los mismos, éstos deberán pasar las pruebas unitarias diseñadas de forma individualizada para cada uno de ellos.

Cabe destacar, sin embargo, que el hecho de que las pruebas unitarias se superen, no garantizan el correcto funcionamiento del proyecto, ya que estas pruebas están diseñadas para comprobar el funcionamiento de los distintos procedimientos implementados en cada uno de ellos, y no su correcta integración con el resto de los módulos del programa.

Para el diseño de las pruebas, se utilizará dos estrategias. La primera serán las de pruebas automáticas para aquellos módulos que lo permitan. Otros módulos, en cambio, serán comprobados de manera manual, y documentándolo en forma de checklist.

#### *5.3.1.1 VENTANA*

La ventana realiza dos operaciones: la primera es la de crear los objetos de la interfaz gráfica, y la segunda es posicionarlos en la ventana (de manera gráfica). Las dos pruebas que se realizan son de forma manual:

- **Prueba 1:** Comprobar que se crea bien los objetos de la interfaz gráfica.
- **Éxito**: Sí.

- **Prueba 2:** Comprobar que los objetos de la ventana están bien posicionados.
- **Éxito:** Sí.

#### *5.3.1.2 MENÚS*

Las pruebas del menú consisten en comprobar que se crean los elementos del menú, que se activan cuando son pulsados y que las funciones de habilitar y deshabilitar funcionan correctamente.

- **Prueba 1:** Comprobar en la interfaz gráfica si se crea el menú y todos los elementos.
- **Éxito:** Sí.

- **Prueba 2**: Comprobar que la función 'ActionListener' está activada y funciona de manera correcta cuando se produce algún evento.
- **Éxito:** Sí.

- **Prueba 3**: Comprobar que se deshabilita y se habilita el menú de manera correcta cuando indicado.
- **Éxito:** Sí.

#### *5.3.1.3 BARRAS DE HERRAMIENTAS*

Las pruebas de la barra de herramientas consisten en comprobar que se crean los elementos, que se activan cuando son pulsados y que las funciones de habilitar y deshabilitar funcionan correctamente.

- **Prueba 1**: Comprobar en la interfaz gráfica si se crea la barra y todos los elementos.
- **Éxito:** Sí.

- **Prueba 2:** Comprobar que la función 'ActionListener' está activada y funciona de manera correcta cuando se produce algún evento.
- **Éxito**: Sí.

- **Prueba 3:** Comprobar que se deshabilita y se habilita el menú de manera correcta cuando indicado.
- **Éxito:** Sí.

#### *5.3.1.4 BARRA INFERIOR*

La primera prueba consiste en verificar que el objeto funciona correctamente. Y la segunda en que las funcionalidades de cambiar el estado y las coordenadas funcionan.

- **Prueba 1:** Verificar que se crea correctamente el objeto.
- **Éxito:** Sí.

- **Prueba 2:** Se comprueba que el estado y las coordenadas funcionan.
- **Éxito**: Sí.

#### *5.3.1.5 RENDER*

Las pruebas de la clase Render sirven para comprobar que determinadas celdas de las matrices L y S se destacan más que las demás.

- **Prueba** 1: Verificar que se pinta de rojo el fondo de dos celdas determinadas.
- **Éxito:** Sí.

- **Prueba 2:** Verificar que el texto da una determinada celda se pinta de rojo.
- **Éxito:** Sí.

#### *5.3.1.6 CANVAS (PANELES)*

Las pruebas del canvas también se hacen de forma manual. Se prueba que se crea bien el objeto, que captura los eventos del ratón, y las demás funciones.

- **Prueba 1:** Verificar si se crea el objeto con éxito. Verifica si se activan las interfaces de eventos y si se cambia el cursor.
- **Éxito:** Sí.

- **Prueba 2:** Verificar si la función 'paint' se realiza con éxito después de ejecutar un algoritmo.
- **Éxito:** Sí.

- **Prueba 3**: Comprobar si las funciones de las interfaces MouseListener y MouseMotionListener se activan y funcionan de manera adecuada.

- **Éxito:** Sí.

- **Prueba 4:** Verificar las demás funcionalidades de apoyo: cambiar cursor, habilitar y deshabilitar los eventos del ratón.
- **Éxito:** Sí.

#### *5.3.1.7 TABLA (MATRIZ)*

Consiste en comprobar que las tablas de las matrices L y S se crean y se actualizan correctamente durante la ejecución paso a paso del algoritmo MWT y MLT.

- **Prueba 1:** Comprobar que las matrices se crean correctamente (dimensión de la matriz) la inicio de la ejecución paso a paso.
- **Éxito:** Sí.

- **Prueba 2**: Verificar que la Matriz S se actualiza correctamente.
- **Éxito:** Sí.

- **Prueba 3**: Verificar que la Matriz L se actualiza correctamente.
- **Éxito:** Sí.

#### *5.3.1.8 TREE*

Comprobaremos que se crean los 4 elementos referentes a las 4 perspectivas, además de comprobar que la perspectiva cambia cuando pulsamos dos veces en algún elemento del árbol y se selecciona el elemento del árbol adecuado cuando se cambia de perspectiva.

- **Prueba 1:** Comprobar que se crea bien el árbol, poniendo los nombres y los iconos correspondientes en cada nodo.
- **Éxito:** Sí.

- **Prueba 2**: Verificar que cambia de perspectiva cuando pulsamos dos veces en todos los nodos del árbol.
- **Éxito:** Sí.

- **Prueba 3**: Comprobar que la selección dentro del árbol cambia cuando cambiamos de perspectiva.
- **Éxito:** Sí.

#### *5.3.1.9 CONTROLADOR APLICACIÓN*

Comprobaremos que las funciones de carácter general funcionan, además de crear bien los objetos y organizarlos.

- **Prueba 1:** Comprobaremos la funcionalidad de la función abrir y guardar proyecto. Guardamos un proyecto con varios puntos en las perspectivas, cerramos la aplicación y abrir el proyecto guardado anteriormente.
- **Éxito:** Sí.

- **Prueba 2**: Verificaremos la funcionalidad de la función "nuevo proyecto".
- **Éxito:** Sí.

- **Prueba 3**: Comprobaremos que se crean todos elementos de la interfaz gráfica y que se los pasa a la ventana.
- **Éxito:** Sí.

- **Prueba 4**: Comprobaremos que se crean todos paneles y menús específicos.
- **Éxito:** Sí.

- **Prueba 5:** Comprobaremos la funcionalidad de copiar y pegar puntos de una perspectiva a otra
- **Éxito:** Sí.

#### *5.3.1.10 CONTROLADORES ESPECÍFICOS*

Comprobaremos el correcto funcionamiento de estos controladores

- **Prueba 1:** Verificar que en la función creadora y en la función 'setParámetros', se inicializan las variables de control.
- **Éxito:** Sí.

- **Prueba 2**: Comprobar que la función setCoordenadas funciona y se comunica de manera correcta con la barra inferior.
- **Éxito:** Sí.

- **Prueba 3:** Comprobar el flujo de información en la función Pintar: Si es ejecución paso a paso, comprueba si se enseñan todos los elementos (acciones, vértices y aristas). Si no es ejecución paso a paso, verifica que primero se pintan las aristas y luego los nodos.
- **Éxito:** Sí.

- **Prueba 4:** Verificar si la función añadir funciona correctamente, comprobando que se pinta en nuevo nodo en pantalla.
- **Éxito:** Sí.

- **Prueba 5**: Verifica que las funciones referentes al caso de uso 'Mover nodo', esto es, mouse pressed, dragged y releasead funcionan de manera correcta. El nodo se mueve, o vuelve al mismo sitio caso no pueda moverlo.
- **Éxito:** Sí.

- **Prueba 6**: Analizar que se borran los nodos al ejecutar la función 'Nueva plantilla' y que no se pintan ningún nodo en pantalla.
- **Éxito**: Sí.

- **Prueba 7**: Analizar que se generan N nodos aleatorios de manera satisfactoria.
- **Éxito:** Sí.

- **Prueba 8:** Verificar que el flujo de eventos en las ejecuciones paso a paso y normales funcionan correctamente.
- **Éxito:** Sí.

- **Prueba 9**: Comprobar que se cambia el cursor dependiendo de su posición en pantalla.
- **Éxito:** Sí.

#### *5.3.1.11 PUNTO*

Se hace una prueba automática y se comprueba que los resultados son los esperados. Se comprueban las funciones de creación, de obtener elementos, clone y equals, distancia y alineación.

El programa de pruebas se encuentra en la carpeta PruebaUnitaría, en el directorio del código fuente. La salida del programa es la siguiente:

*¿x=1 e y=2? True*
*¿p=copia? true true*
*Distancia entre (0,0) y (1,2) = 2.23606797749979*
*Estan alineados p, p2 y p3? true*

#### *5.3.1.12 ARISTA*

Se hace una prueba automática para verificar el correcto funcionamiento de las funcionalidades del creador, de los gets, de la función 'se Cruzan' y de las funciones clone y equals.

El programa de pruebas se encuentra en la carpeta PruebaUnitaría, en el directorio del código fuente. La salida del programa es la siguiente:

*x1=0 y1=0*
*x2=1 y2=1*

*¿a=copia? True*
*¿a cruza con copia? True*
*¿a cruza con nueva? true*

#### *5.3.1.13 LISTA PUNTOS*

En la siguiente prueba, se comprueba las funcionalidades básicas de la lista de puntos: añadir y eliminar puntos, verificar si el punto está en la lista y obtener puntos.

El programa de pruebas se encuentra en la carpeta PruebaUnitaría, en el directorio del código fuente. La salida del programa es la siguiente:

*Crear objeto*
*tam lista=0*
*Insertar dos elementos*
*tam lista=2*
*Dar primer punto*
*x1=0 y1=0*
*¿Contiene elemento (100,100)? False*
*¿Contiene elemento (20,20)? True*
*Eliminar segundo elemento: (20,20)*
*¿Contiene elemento (20,20)? False*

#### *5.3.1.14 LISTA ARISTAS*

Se comprueba las funcionalidades básicas de la lista: insertar, eliminar y obtener elementos, y tamaño de la lista.

El programa de pruebas se encuentra en la carpeta PruebaUnitaría, en el directorio del código fuente. La salida del programa es la siguiente:

*Crear objeto*
*tam lista=0*
*Insertar dos elementos*
*tam lista=2*
*Obtener primer elemento y compararlo con a1 y a2*
*¿a=a1? True*
*¿a=a2? False*
*Eliminar primer elemento*
*tam lista=1*

#### *5.3.1.15 ALGORITMO*

Consiste en verificar si el control de tipos de algoritmos funciona correctamente.

- **Prueba 1:** Comprobar si el control de tipos de algoritmos funciona correctamente.
- **Éxito**: Sí.

#### *5.3.1.16 LISTA ACCIÓN*

Se prueba las funcionalidades básicas de las listas, además de comprobar que se pintan todos los elementos en pantallas.

- **Prueba 1:** Comprobar las funcionalidades básicas de lista: crear, insertar y obtener elemento, vaciar lista y tamaño de la lista
- **Éxito:** Sí.

- **Prueba 2:** Dada una posición de la lista, verificar que todos los elementos se pintan bien en pantalla.
- **Éxito:** Sí.

### 5.3.2 PRUEBAS DE INTEGRACIÓN

El objetivo de las pruebas de integración es el de comprobar el correcto funcionamiento cuando se utilizan simultáneamente dos o más módulos. Se realizarán las pruebas de integración tras haber pasado las pruebas unitarias. Asegurándonos por tanto que los módulos funcionan correctamente cuando trabajan de manera independiente.

Para realizar las pruebas de integración se hará uso de la interfaz ya que en este momento está desarrollada al completo.

Las pruebas se realizan por niveles. En el primer nivel se hace la integración de la ventana con los demás objetos de la interfaz gráfica. En el segundo nivel, se hace la integración de la interfaz gráfica con el motor. El tercer nivel corresponde a la integración de todo la anterior con los objetos básicos del sistema. Y por último, con los algoritmos.

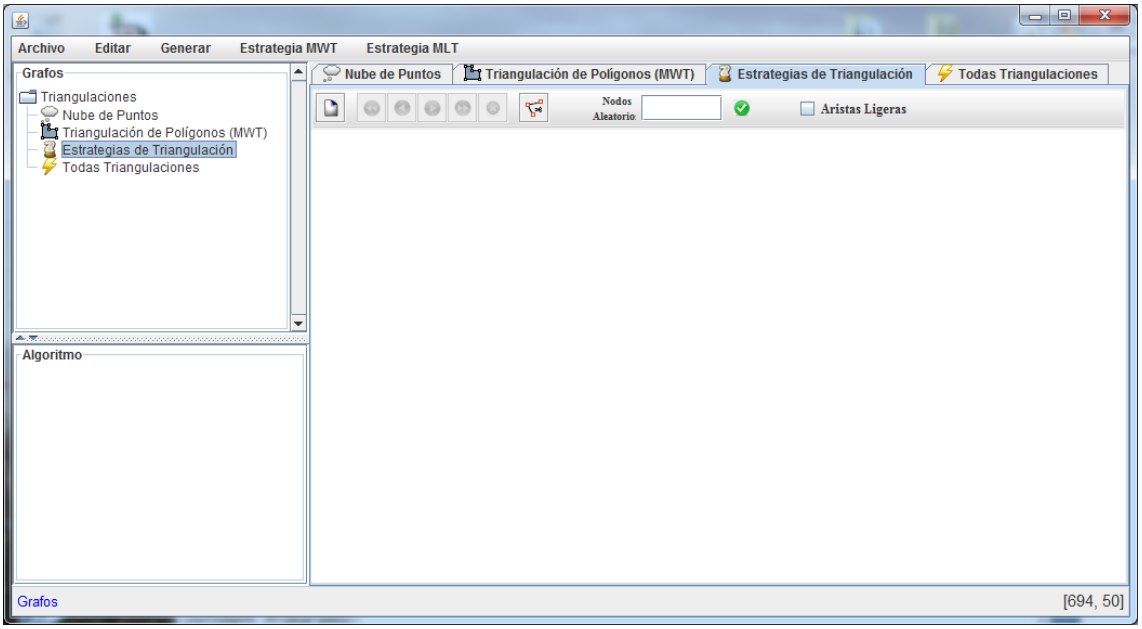
#### *5.3.2.1 PRUEBA NIVEL 1 – INTERFAZ GRÁFICA*

Consiste en integrar la ventana con los demás objetos de la interfaz gráfica: barra inferior, menú, barra de herramientas, tablas y canvas.

Probaremos:

1. La ventana creará todos los objetos.
2. Se añadirán los objetos a la interfaz gráfica.
3. Se comprobará que los objetos están bien posicionados.

**Resultado:** Positivo.




#### *5.3.2.2 PRUEBA NIVEL 2 – INTERFAZ CON CONTROLADORES*

Se basa en integrar los objetos de la interfaz gráfica con los controladores (fachada entre sistema e interfaz).

Probaremos:

1. Comprobar que no hay errores de compilación.
2. Verificar que todos los objetos se crean y tienen visibilidad hacia el motor.
3. Verificar si el motor tiene visibilidad a todos los objetos.
4. Ejecutar funciones del motor desde los objetos de la interfaz gráficas.
5. Ejecutar funciones de los objetos de la interfaz gráfica desde el motor. Por ejemplo, una de las pruebas sería: verificar que deshabilita la barra de herramientas.

**Resultado:** Positivo.

#### *5.3.2.3 PRUEBA NIVEL 3 – INTERFAZ Y CONTROLADORES CON ELEMENTOS BÁSICOS DEL SISTEMA*

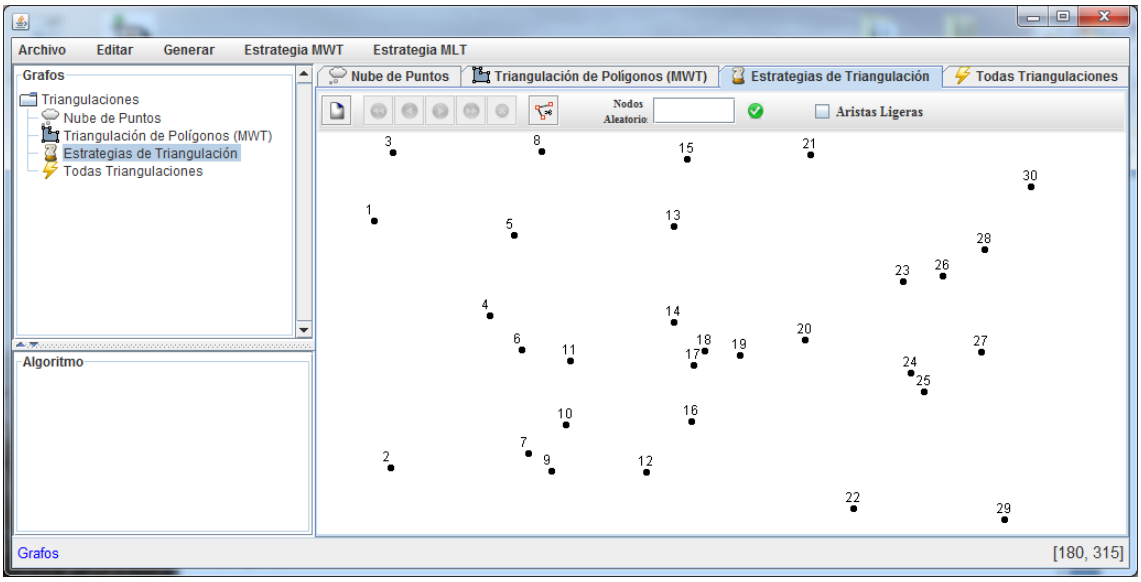
Se trata de juntar algunas funcionalidades del sistema con la interfaz gráfica. Se probarán los casos de uso de la interfaz gráfica:

Probaremos:

1. Se probará que no hay errores de compilación.
2. Se comprobará que se añade un nodo correctamente desde la interfaz gráfica.
3. Comprobará que se puede eliminar un nodo.

4. Verificará que el nodo vuelve a la posición original si no se puede mover lo a la nueva posición.
5. Se comprobará que se puede mover un nodo caso el destino sea correcto.
6. Analizará el funcionamiento de la función “crear nueva plantilla”.
7. Comprobará que se cambia el cursor cuando el mismo se encuentra encima de un nodo.
8. Comprobar que deshabilita la barra de herramientas y el menú al borrar un nodo.
9. Verificar que se puede crear un polígono para la posterior ejecución de MWT.

**Resultado:** Positivo.



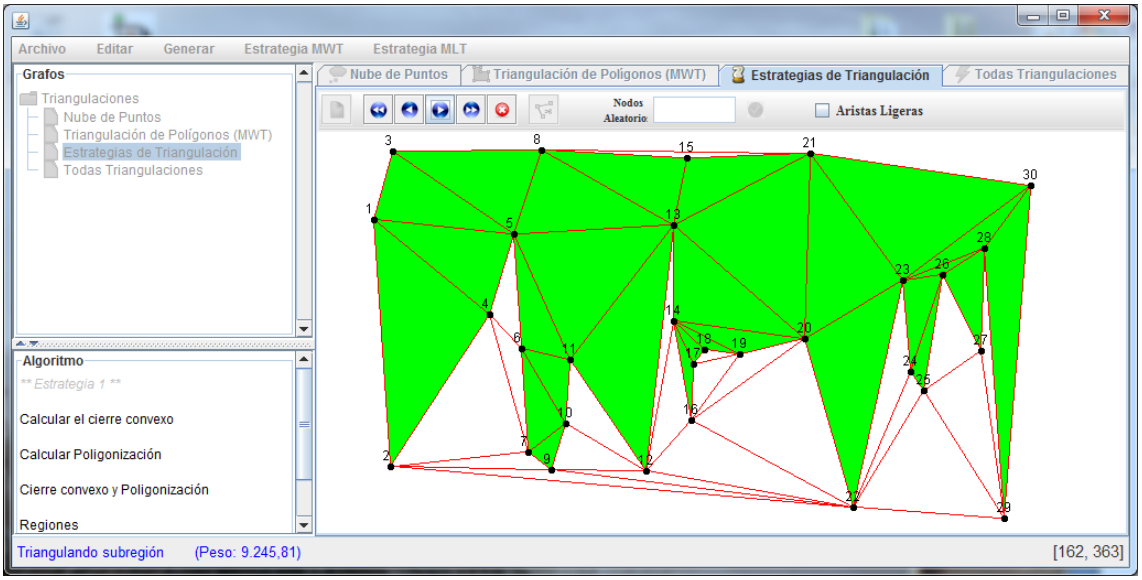

### *5.3.2.4 PRUEBA NIVEL 4 – INTERFAZ INTEGRAR LOS ALGORITMOS*

Una vez integrada la interfaz gráfica con los objetos básicos del sistema, hay que comprobar que es posible ejecutar los algoritmos.

Probaremos:

1. Comprobar que se puede ejecutar el algoritmo de triangulación de Greedy si hay más de 2 nodos no alineados.
2. Comprobar que se puede ejecutar el algoritmo de poligonización si hay más de 2 nodos no alineados.
3. Comprobar que se puede ejecutar el algoritmo de MST si hay más de 2 nodos no alineados.
4. Comprobar que se puede ejecutar el algoritmo de cierre convexo si hay más de 2 nodos no alineados.
5. Comprobar que se puede ejecutar los 4 algoritmos anteriores paso a paso.

6. Verificar que la estrategia 1 se calcula las regiones y se ejecuta de forma satisfactoria.
7. Verificar que la estrategia 2 se calcula las regiones, calcula las ramas y se ejecuta de forma satisfactoria.
8. Verificar que la estrategia 3 se calcula las regiones, calcula las ramas y se ejecuta de forma satisfactoria.
9. Verificar que el menú se deshabilita completamente y la barra de herramientas parcialmente si se ejecuta un algoritmo paso a paso.



# 6 MANUAL DE LA APLICACIÓN

En este apartado describiremos las funcionalidades de la aplicación, describiendo como podemos acceder a ellas y como interactuar con las diferentes perspectivas que nos proporciona.

## 6.1 REQUISITOS DE LA APLICACIÓN

Se trata de una aplicación Windows en Java. Por tanto, se puede ejecutar la aplicación desde cualquier sistema operativo que soporte una máquina virtual de Java previamente instalada en la máquina.

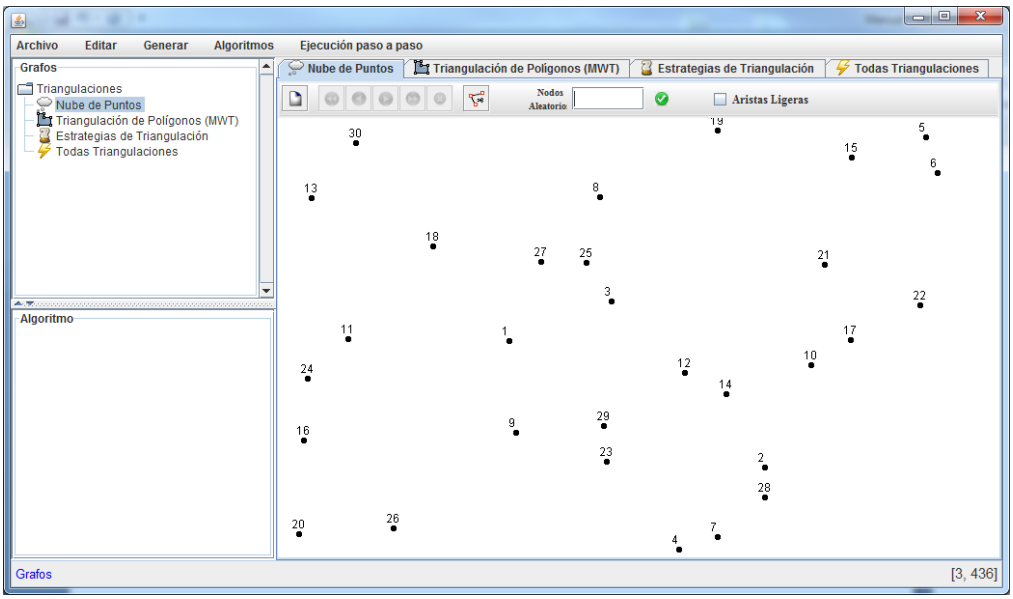
Además, se recomienda tener más de 1 GB de memoria RAM disponible, debido a que las ejecuciones paso a paso utilizan mucha memoria para guardar las informaciones y acciones de cada paso realizado durante la ejecución de los algoritmos.

## 6.2 PRESENTACIÓN Y FUNCIONALIDADES GENERALES

La aplicación dispone de 4 perspectivas diferentes, que proporcionan distintas funcionalidades:

- Algoritmos que se ejecuten sobre nubes de puntos
- Triangulación de polígonos
- Estrategias y programación dinámica
- Todas las triangulaciones

La imagen de la pantalla inicial se puede ver a continuación:



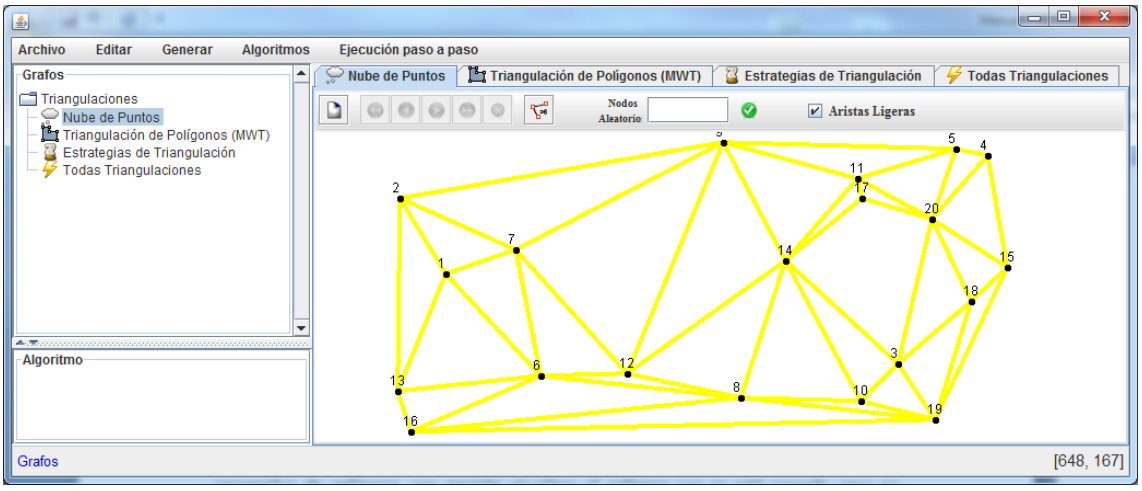

Los diferentes elementos de la interfaz gráfica si dividen en la siguiente forma:

1. Panel de perspectivas

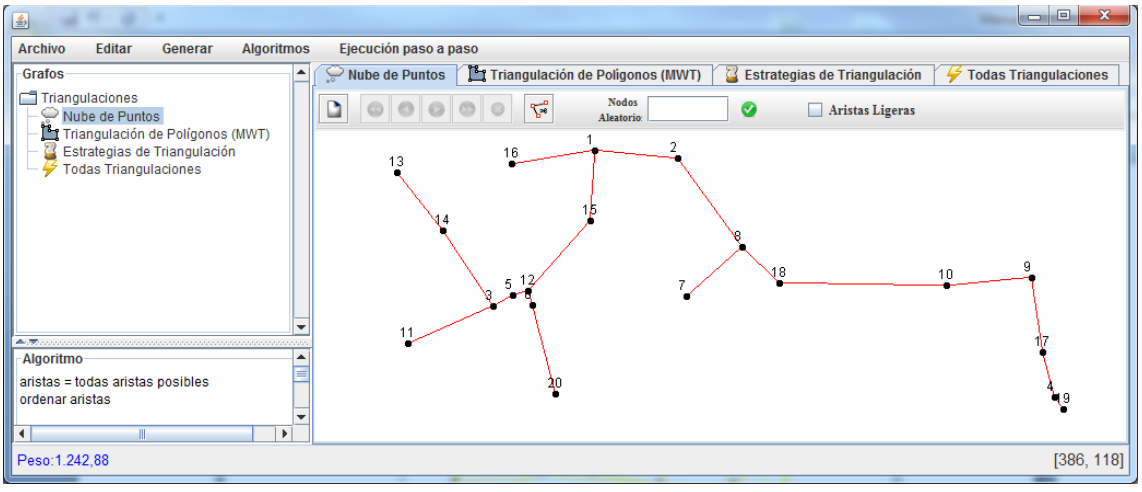
2. Árbol de perspectivas
3. Descripción de los algoritmos
4. Barra de estado
5. Menú (diferente para cada perspectiva)

Todas las perspectivas nos permiten añadir nodos manualmente, moverlos o borrarlos. La perspectiva de polígonos, nos permite visualizar el polígono que se está creando, pero no permite la generación de nodos aleatorios, como las demás perspectivas.

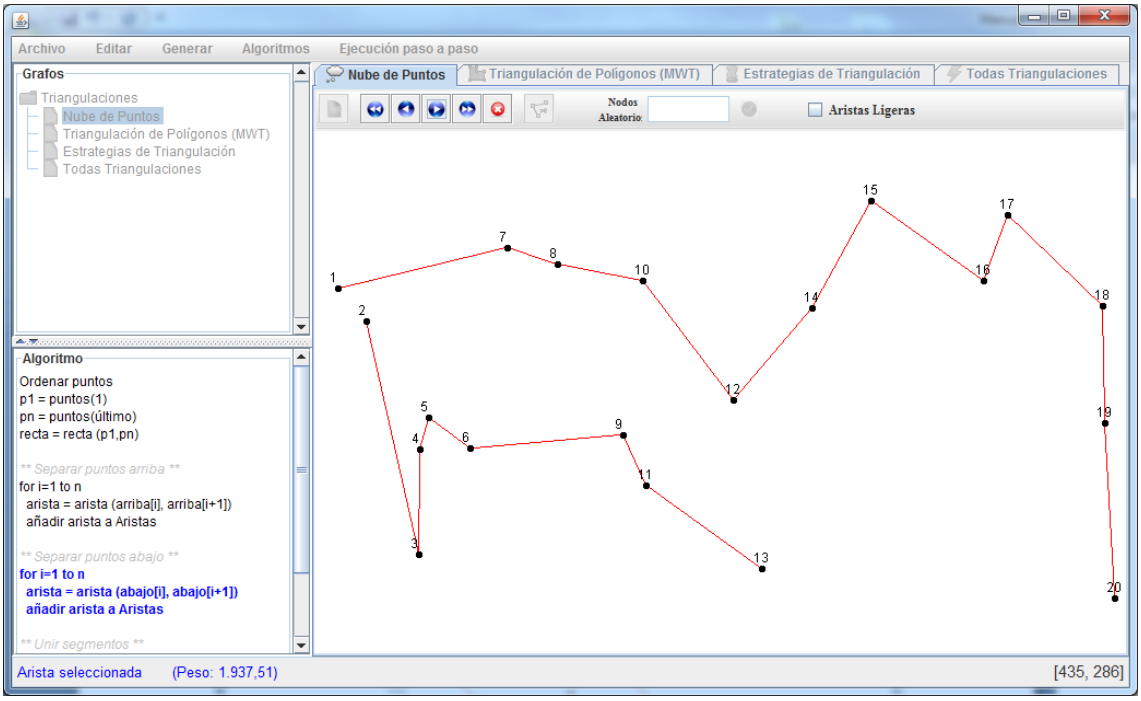
Podemos ver las aristas ligeras (en amarillo) de una nube de puntos o de una triangulación, marcando el checkbox "Aristas ligeras" en la barra de herramienta:




Podemos ejecutar los algoritmos de forma directa, de tal forma que solo veremos los resultados, y no los pasos que ha dado para alcanzar el resultado:

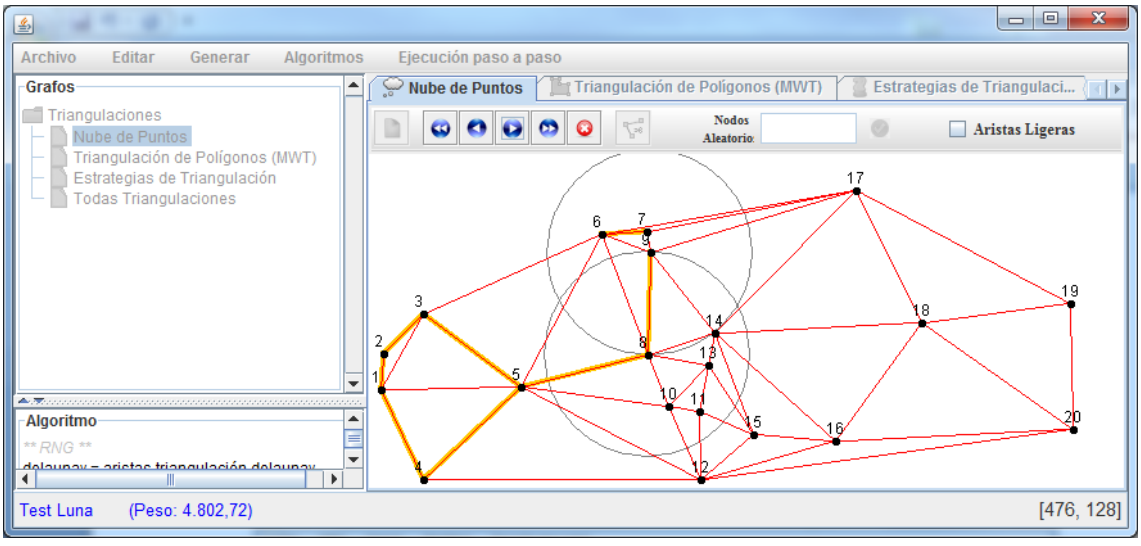



Al ejecutar cualquier algoritmo o estrategia, vemos el pseudocódigo del algoritmo en la parte inferior izquierda de la pantalla. Si se trata de una ejecución paso a paso, vemos que comandos se están ejecutando (pintados en azul en el pseudocódigo):




Además, en la pantalla anterior, al ejecutar un algoritmo paso a paso, vemos que se bloquean todos los elementos y botones de la pantalla que no intervienen en la ejecución. Una vez que se cancele la ejecución paso a paso, estos elementos vuelven a activarse.

Cuando ejecutamos un algoritmo paso a paso, vemos las acciones que se ejecutan reflejadas en el canvas. Por ejemplo, al ejecutar el algoritmo que calcula el grafo RNG:




Utilizando las flechas (elementos no bloqueados durante la ejecución paso a paso), podemos avanzar o retroceder para poder visualizar los pasos de los algoritmos.

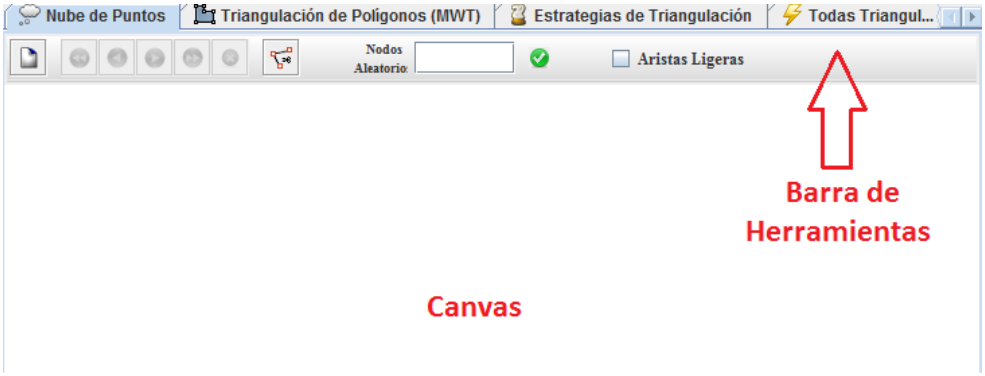
Además, en la barra de de estado (barra inferior), vemos una breve descripción de cada paso, además del peso de la grafo o triangulación enseñado en pantalla.

Podemos copiar los nodos de una perspectiva a otra utilizando el menú “editar”. Esta funcionalidad puede ser útil a la hora de comparar estrategias con algoritmos.

Podemos también generar nodos aleatorios al pulsar en el menú  y luego en “generar nodos aleatorios”. También podemos hacerlo a través de la barra de herramientas en cada perspectiva.

## 6.3 PANEL DE PERSPECTIVAS

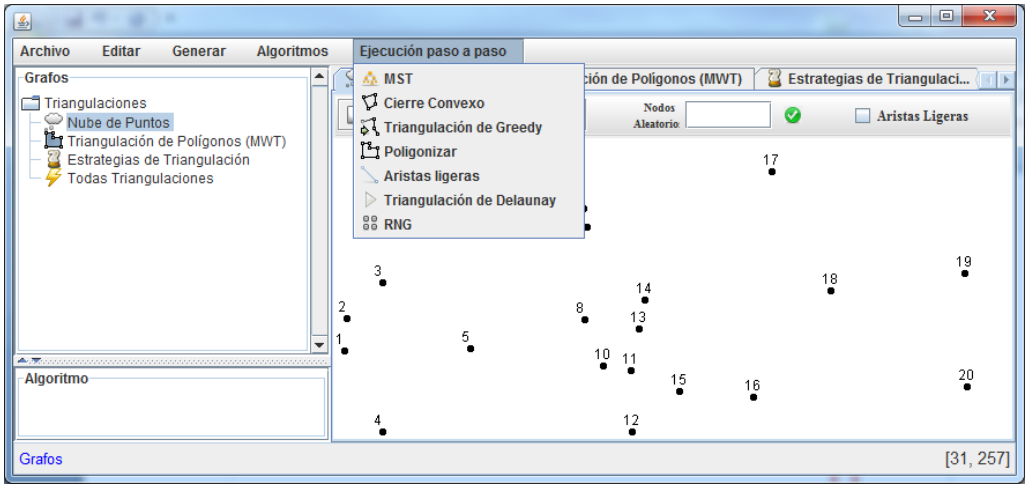
Los elementos de cada perspectiva son la barra de herramientas, el canvas (paleta donde se muestran los elementos de las triangulaciones) y el menú.




La barra de herramientas y el menú son diferentes en cada perspectiva. La barra de herramienta nos permite utilizar las funcionalidades generales de cada perspectiva, mientras que el menú nos permite ejecutar los algoritmos específicos de las perspectivas.

### 6.3.1 NUBE DE PUNTOS

En la primera perspectiva podemos ejecutar algoritmos generales de las nubes de puntos. Es decir, los algoritmos que se ha implementado como apoyo para programar las estrategias.



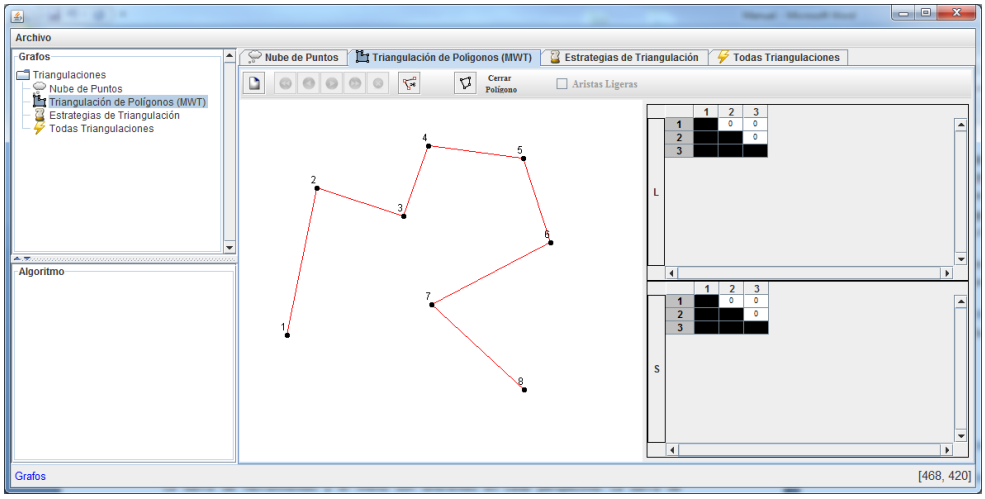

Como podemos ver en la imagen anterior, los algoritmos generales que podemos ejecutar son:

- Árbol generador mínimo (MST)
- Cierre convexo
- Triangulación de Greedy
- Poligonización monótona
- Aristas ligeras
- Triangulación de Delaunay
- Grafo RNG

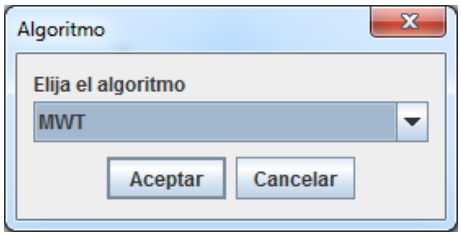
Los algoritmos se ejecutan si hay más de 3 nodos dibujados. Todos estos algoritmos se pueden ejecutar de forma directa o paso a paso.

### 6.3.2 TRIANGULACIÓN DE POLÍGONOS

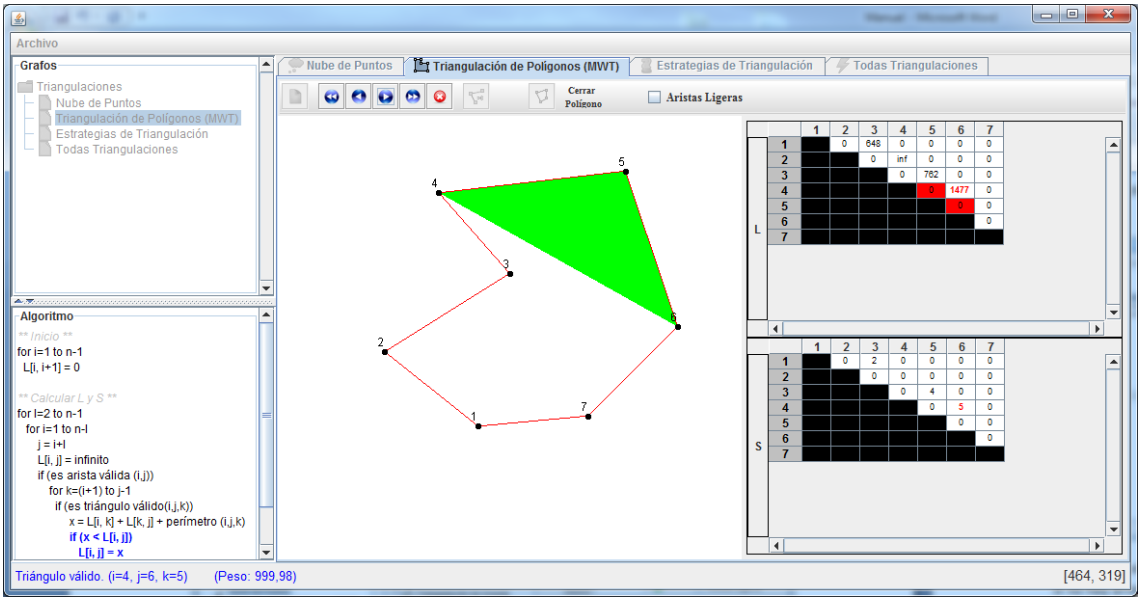
Esta perspectiva permite ejecutar y visualizar los pasos de los algoritmos que se ejecutan sobre polígonos.



Ejecutamos el algoritmo pulsando en el botón “cerrar polígono”, en la barra de herramientas. Si no hay aristas que se corten, podemos elegir entre 3 algoritmos distintos: MWT, MLT o EAR:



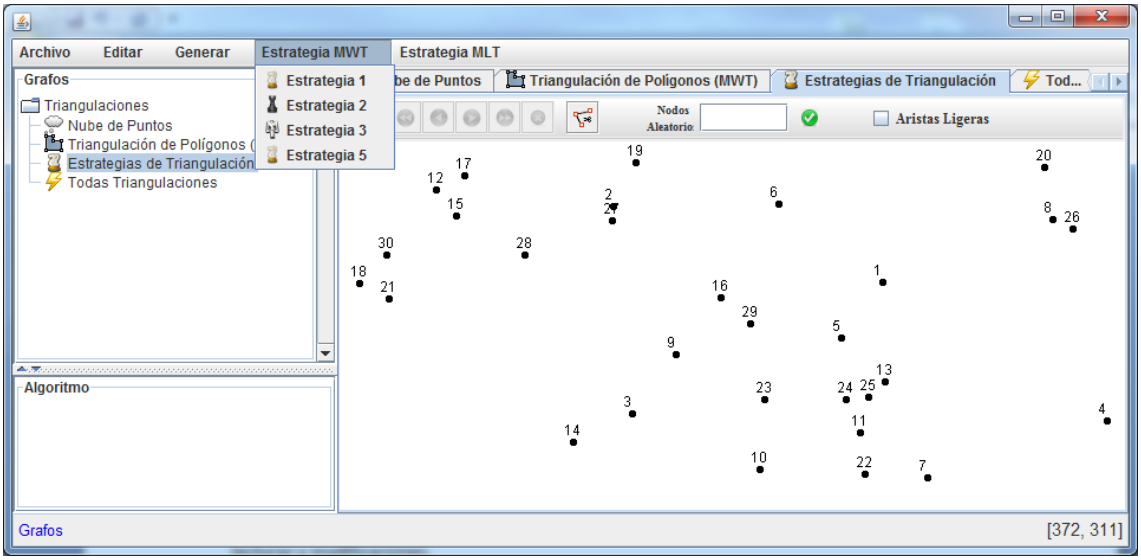

Se trata de una ejecución paso a paso. Por tanto, podemos ver las diferentes acciones durante la ejecución:




Como podemos ver, hay dos elementos más en el panel. Se trata de las matrices L y S. Los valores se actualizan cuando se ejecuta el algoritmo MWT o el MLT, indicando en rojo las lecturas y modificaciones.

### 6.3.3 ESTRATEGIAS DE TRIANGULACIÓN

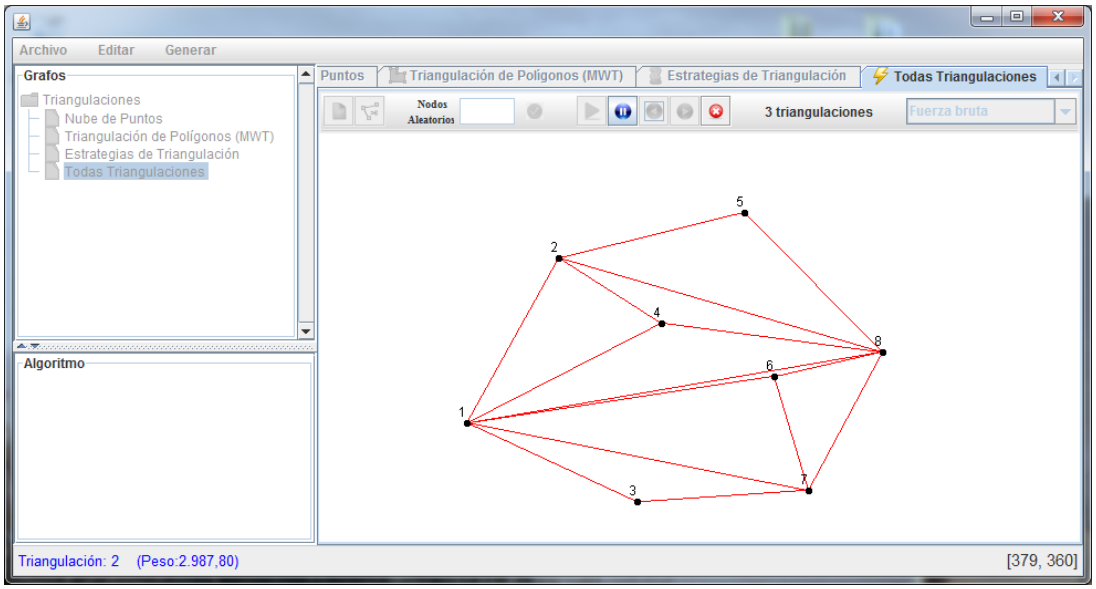
Es muy parecida a la primera perspectiva (nube de puntos). La única diferencia es el menú, donde nos permite seleccionar las estrategias que queremos ejecutar: MWT o MLT.




### 6.3.4 TODAS TRIANGULACIONES

Por último, tenemos la perspectiva que nos permite visualizar todas las triangulaciones. Los algoritmos implementados son de fuerza bruta, así que podemos tener limitación en el número de puntos dependiendo de la memoria RAM disponible.

Se ha implementado utilizando un thread para ejecutar los algoritmos, así que se puede pausar la ejecución para visualizar las triangulaciones encontradas hasta el momento, y luego reanudar la ejecución.



# 7 RESULTADOS Y CONCLUSIONES

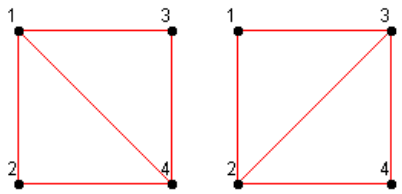
En este apartado, nos centraremos en presentar los resultados obtenidos en la aplicación, comparando los resultados entre las distintas técnicas implementadas para estrategias de triangulación de nubes de puntos y de polígonos.

## 7.1 RESULTADOS OBTENIDOS

### 7.1.1 RESULTADOS: TRIANGULACIÓN DE POLÍGONOS

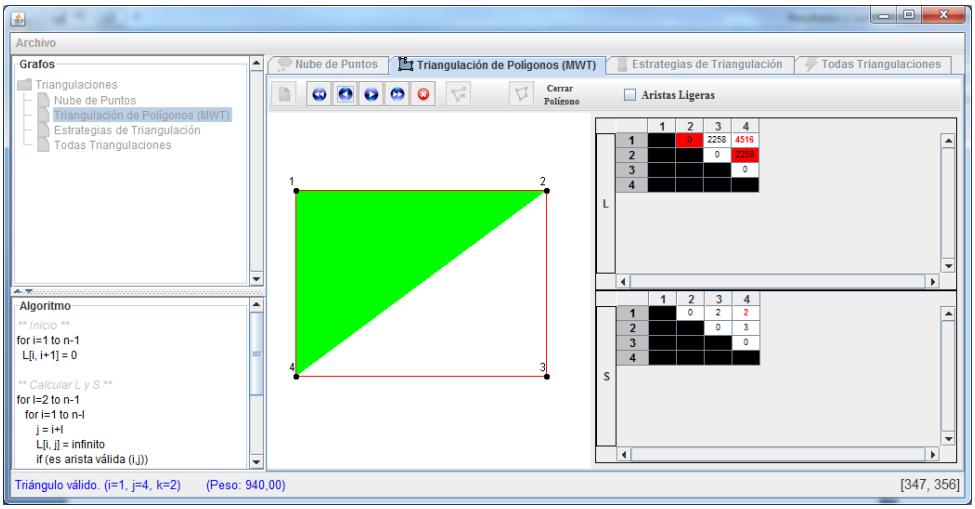
Analizaremos los 3 algoritmos que se ha implementado para calcular la triangulación de un polígono: MWT, MLT y EAR.

Como ya sabemos, la triangulación MWT nos devuelve la triangulación de peso mínimo, mientras que la triangulación MLT, nos devuelve la triangulación cuya arista de mayor longitud sea la mínima posible.

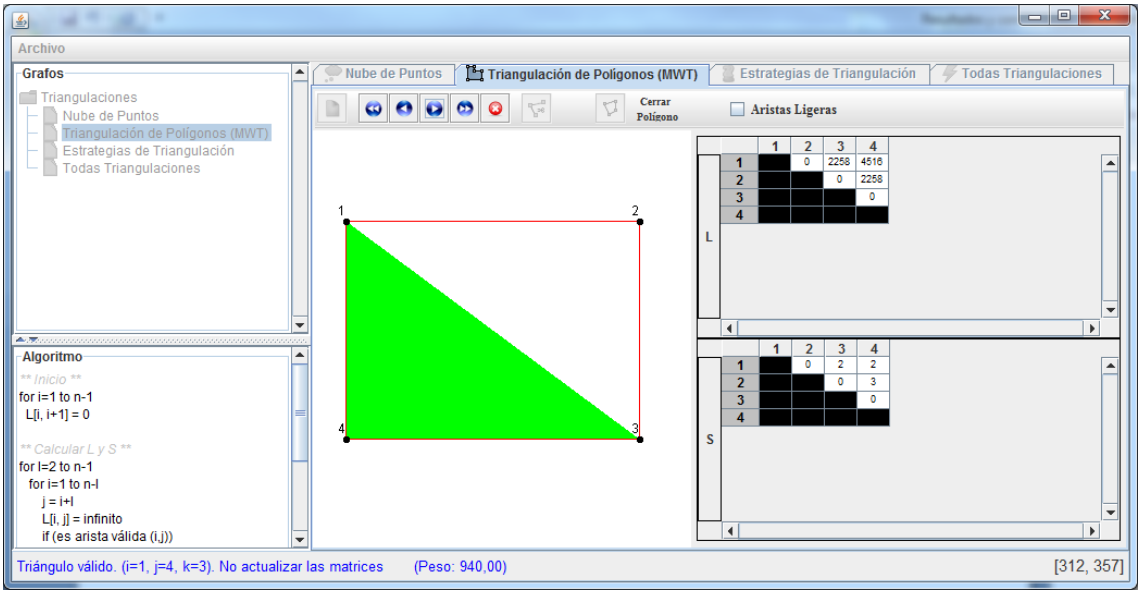



Ambas triangulaciones no son únicas. Por ejemplo, un cuadrado regular tendrá dos triangulaciones, y ambas son MWT y MLT: ((100, 100), (100, 200), (200, 100) , (200, 200))

En estos casos, sabiendo que el algoritmo ejecuta un bucle triplo para buscar las aristas que cumplen un determinado criterio, hemos decidido almacenar en las matrices L y S la primera arista que encuentra en caso de igualdad. Esto es, si durante la ejecución del algoritmo, encuentra una arista con el mismo peso o longitud que la anterior, no actualizará las matrices.



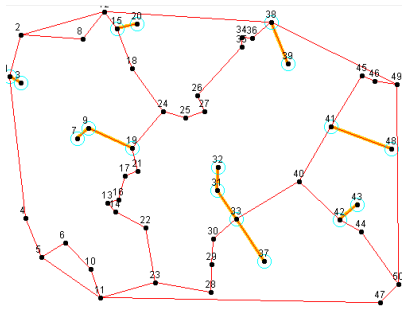

En la imagen anterior, vemos como se actualiza las posiciones (1, 4) de las matrices para el polígono ((30, 100), (300, 100), (300, 300), (30, 300)).




Y en la siguiente ejecución del algoritmo, vemos como las matrices no se han actualizado, aunque el peso obtenido sea el mismo.

Como ya hemos adelantado en apartados anteriores, tenemos dos variantes para los algoritmos de MWT y MLT, cuando los polígonos tienen ramas. Obtenemos estos polígonos cuando ejecutamos las estrategias de triangulación de nubes de puntos. Y en estos casos, como también ya lo hemos comentado, tenemos puntos repetidos en el polígono, y para todos los puntos repetidos, siempre actualizamos las mismas filas y columnas de las matrices L y S.

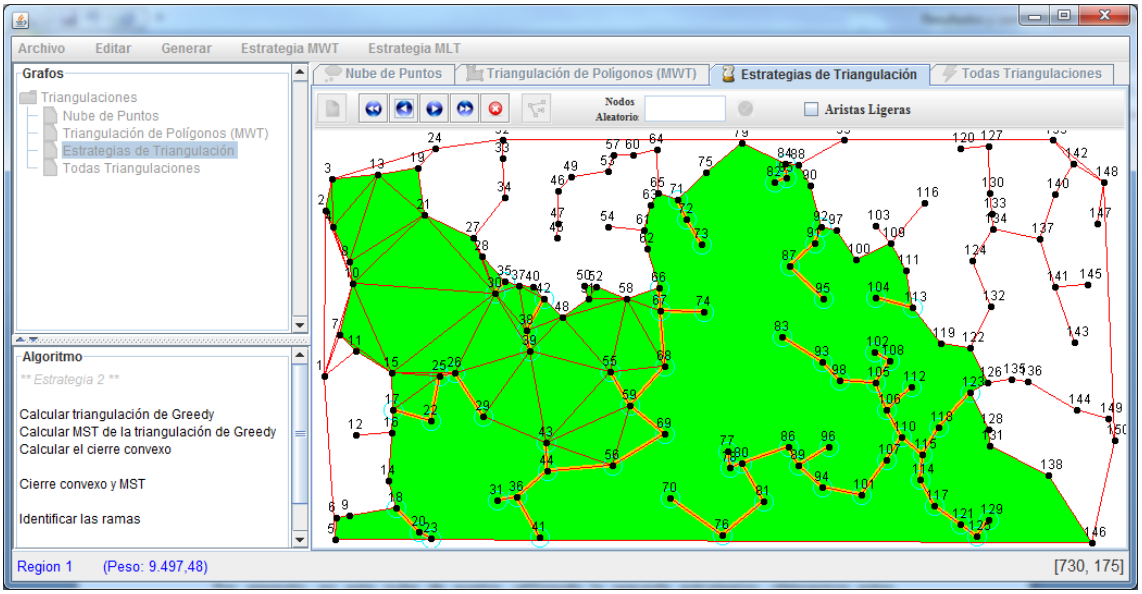
Por ejemplo, en esta nube de puntos, utilizando la segunda estrategias, obtenemos estas regiones con ramas (en naranja):



Podemos ver ha encontrado 8 ramas en 7 regiones diferentes.

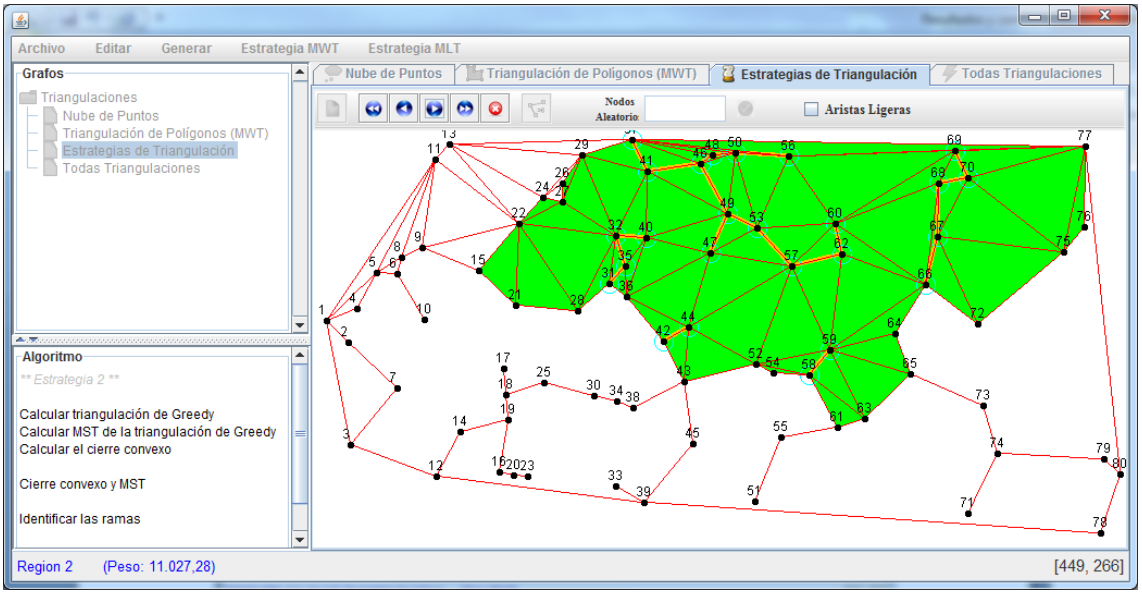
En las regiones (polígonos), con ramas, tenemos puntos que aparecen varias veces en la definición del polígono. Y para todos estos puntos, le asignamos solamente una columna en las matrices L y S

Esta implementación, cuando se ejecuta con éxito, nos devuelve el resultado exacto al que esperamos. Sin embargo, el algoritmo no funciona perfectamente en algunos casos, principalmente cuando las ramas encontradas son muy complejas:




Podemos ver que el algoritmo no ha podido completarse con éxito. Podemos ver que las ramas son muy complejas, casi dividiendo la región en polígonos menores.

Sin embargo, en algunos casos parecidos, el algoritmo funciona perfectamente:



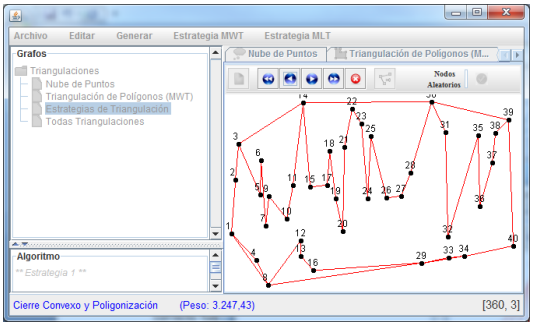

Hemos identificado que en la mayoría de las veces, es decir, un 60% de los casos, el algoritmo funciona perfectamente independientemente de la complejidad de las ramas. Y en los casos que las ramas son muy simples, es decir, que no haya demasiadas bifurcaciones dentro de las ramas y que no sean muy extensas, el algoritmo funciona el 90% de las veces.

La triangulación EAR no presenta ninguna propiedad de minimización de pesos, y tampoco devuelve un único resultado. Simplemente la implementamos para poder comprobar el correcto funcionamiento de algunas de las funcionalidades de la interfaz gráfica, debido a su sencillo algoritmo.

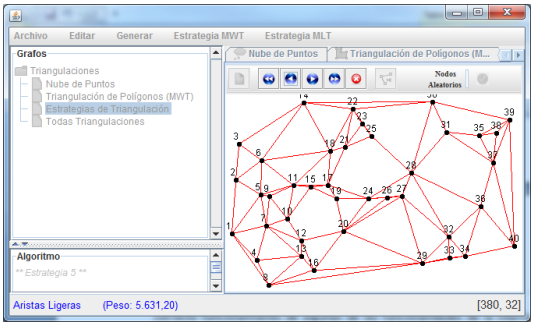
### 7.1.2 RESULTADOS: ESTRATEGIAS Y PROGRAMACIÓN DINÁMICA

De las 5 estrategias implementadas, las únicas que presentan la propiedad de completitud son las estrategias 1 y 5. La primera estrategia obtiene las regiones a través del cierre convexo y de la poligonización monótona, y la quinta estrategia, obtiene las regiones a partir de las aristas ligeras.

Ambas estrategias nos devuelve regiones sin ramas, y por tanto, el algoritmo de triangulación de polígonos nos devuelve el resultado exacto para cada una de las regiones obtenidas.

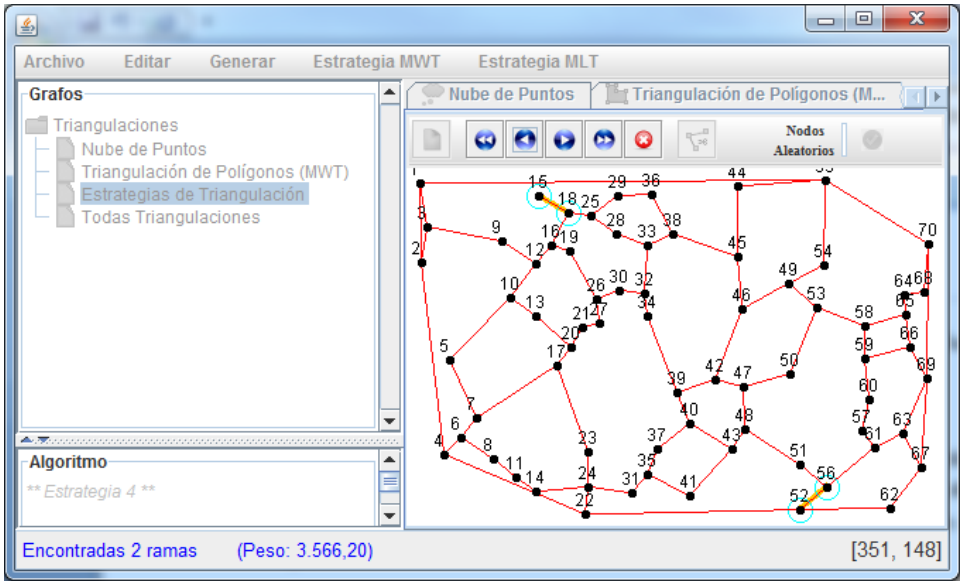



**Estrategia 1**



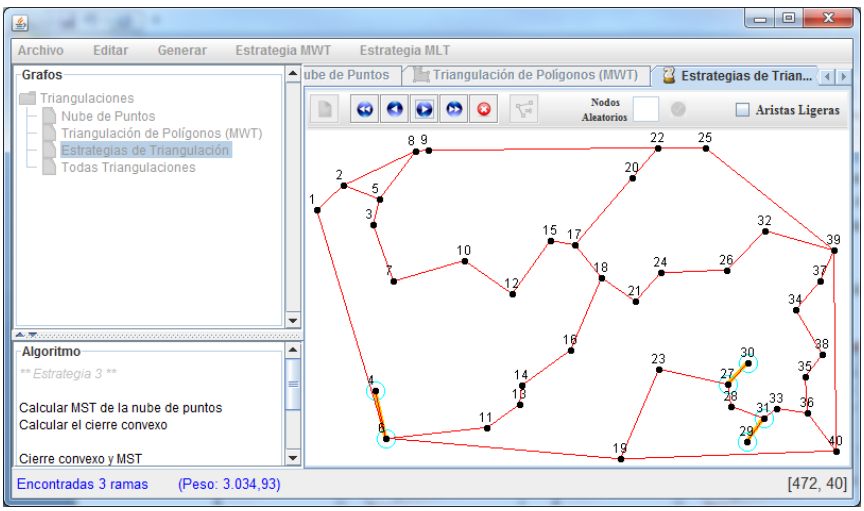

**Estrategia 5**

En la **estrategia 4**, donde obtenemos las regiones a partir del cierre convexo y del grafo RNG, hemos llegado a la conclusión de que el algoritmo funciona casi todas las veces, debido a que las ramas que se generan en las regiones son muy simples, y muchas veces únicas. Por tanto, la variante del algoritmo MLT para triangulación con ramas, suele funcionar sin ningún problema.

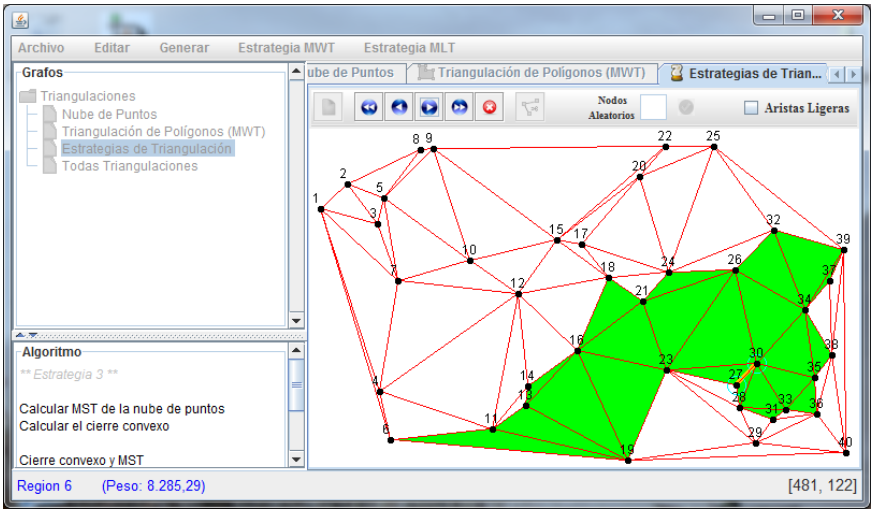



Las **estrategias 2 y 3** generan regiones con ramas. Si hay muchos puntos en la nube de puntos, y además, muy pegados entre sí, las ramas creadas suelen ser muy complejas, y por tanto, la probabilidad de que el algoritmo funcione correctamente es relativamente baja, ya que la probabilidad de que funcione depende del producto de las probabilidades de las correctas ejecuciones de todas las regiones con ramas.

Sin embargo, cuando las regiones no son muy complejas (normalmente cuando hay pocos puntos en la nube), el algoritmo funciona perfectamente para ambas estrategias:




Podemos ver que ha encontrado pocas regiones, y que además, las regiones son muy simples. Por tanto, el algoritmo funcionará perfectamente en estos casos:




Por último, vale la pena comentar que el programa no es robusto en lo que se refiere a utilización de la memoria, ya que no ha sido objetivo del proyecto contemplar casos donde la nube de puntos o el polígono fuesen demasiado extensos. Eso se debe a que, por tratar de una aplicación gráfica, cada paso del algoritmo utiliza una parte de la memoria para guardar la información que se debe enseñar en pantalla. S hay muchos nodos, el número de pasos a ejecutar aumenta, y por tanto, también aumenta el consumo de la memoria.

Además, como ya hemos comentado en el alcance del proyecto, debido a esta problemática, mejorar la complejidad de los algoritmos en conjunción con la interfaz gráfica no ha entrado en los objetivos del proyecto.

## 7.2 CONCLUSIONES

Debido a que el cálculo de la MWT es un problema NP-Duro (Loera, 2009), hemos implementado 4 estrategias diferentes para calcular la triangulación de peso mínimo de una nube de puntos y 1 estrategia para calcular la triangulación MLT.

Las estrategias se basan en dividir la nube de puntos en polígonos menores, donde se aplica de forma individual el algoritmo de MWT de polígonos, que nos devuelve el resultado exacto para estas pequeñas regiones.

Por tanto, el objetivo de la programación dinámica es de obtener las mejores regiones para poder aplicar el algoritmo MWT. Cada una de las 5 estrategias implementadas ejecuta una secuencia de código diferente para obtener las regiones.

Cada estrategia nos devuelve un resultado diferente. Y comparar estos resultados y el tiempo de ejecución es uno de nuestros objetivos:

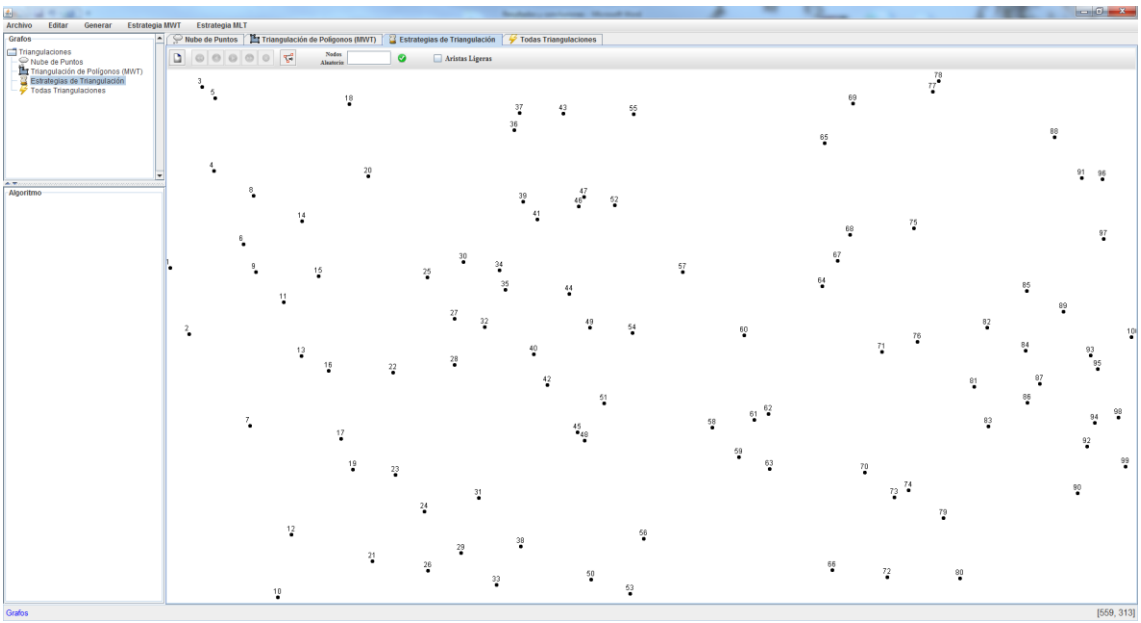
### 7.2.1 COMPARAR RESULTADOS

Comparando los resultados de las 5 estrategias para la misma nube de puntos, hemos obtenido la siguiente información:

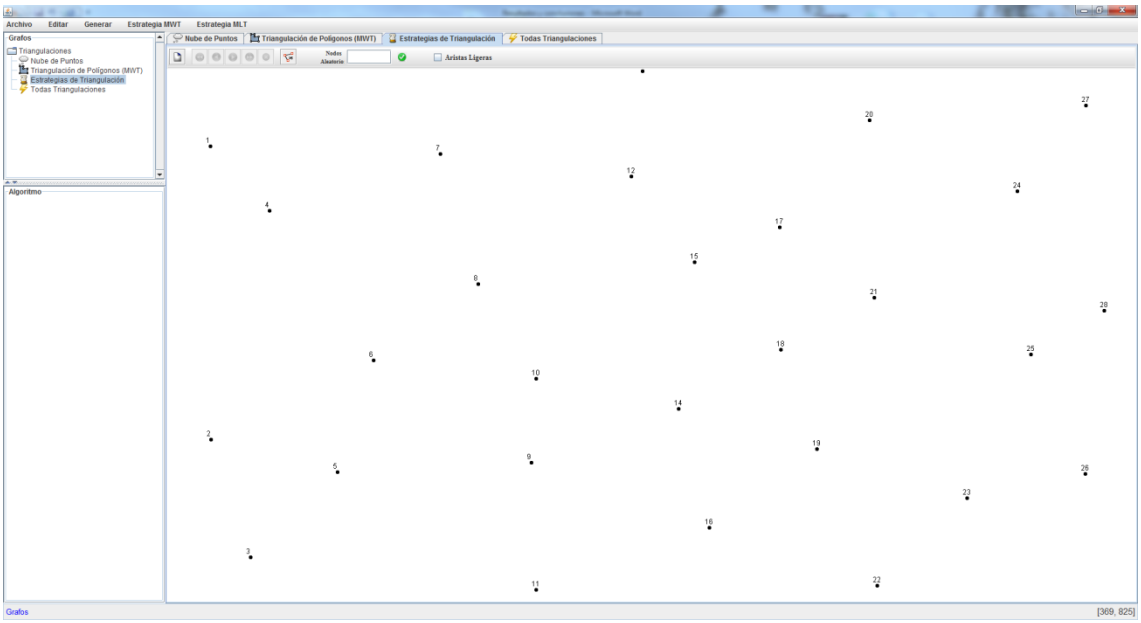
- ***La estrategia 1 siempre nos devolverá el peor resultado.***
  Además, se trata de la triangulación menos homogénea de las 5 (por la poligonización monótona de la nube de puntos). Como consecuencia, el resultado suele ser bastante superior a los demás.
- ***Las estrategias 2 y 3 siempre nos dan el mismo resultado.***
  La diferencia en los pasos de ejecución de esta estrategia es que una calcula el árbol generador mínimo de la nube de puntos, y el otro de la triangulación de Greedy. Y en todas pruebas que hemos realizado, no hemos encontrado ningún caso donde el árbol MST de la nube de puntos fuera distinto al árbol MST de la triangulación de Greedy. Es decir, en todos los casos prácticos que hemos analizados, el árbol MST de la nube de puntos estaba contenido en la triangulación de Greedy de la misma nube de puntos.
- ***La estrategia 4 devuelve el mejor resultado.***
  Para todos los casos que hemos analizados, la estrategia 4 siempre ha devuelto el mejor resultado o igual al mejor resultado de las demás triangulaciones.
- ***Cuanto más puntos haya en la nube, mayor la diferencia de pesos entre las estrategias***
  Hemos observado que al aumentar el número de puntos, aumentamos la complejidad de las operaciones, y por tanto, vemos grandes diferencias entre los resultados obtenido de las diferentes ejecuciones.

A continuación, vamos a ejecutar las 5 estrategias para las siguientes nubes de puntos y comparar los resultados:

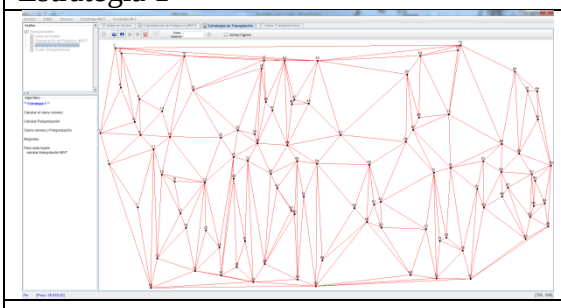
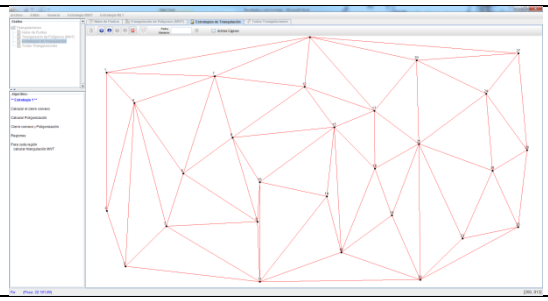
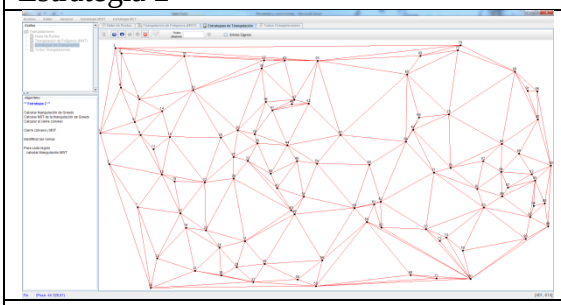
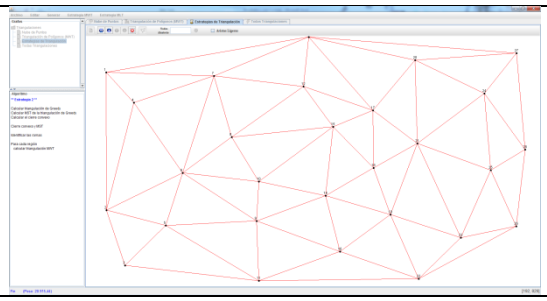
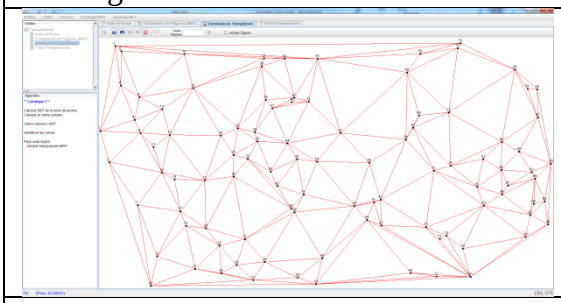
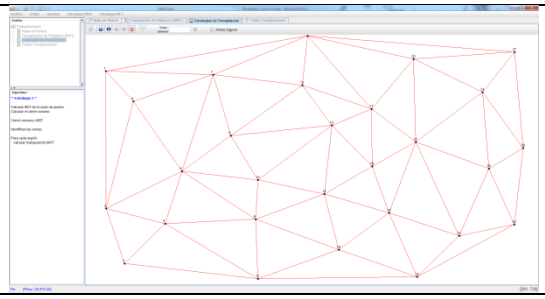
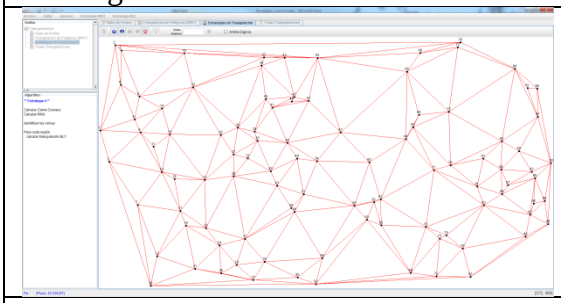
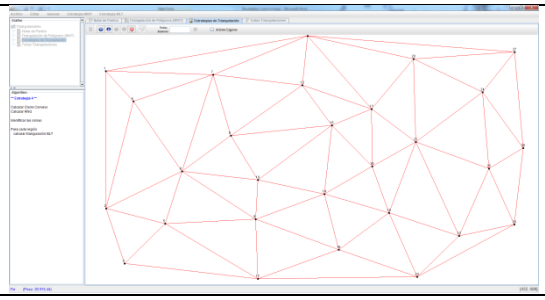
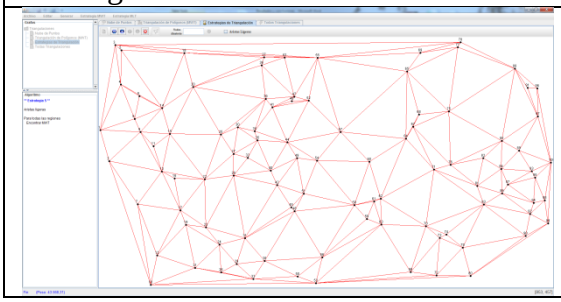
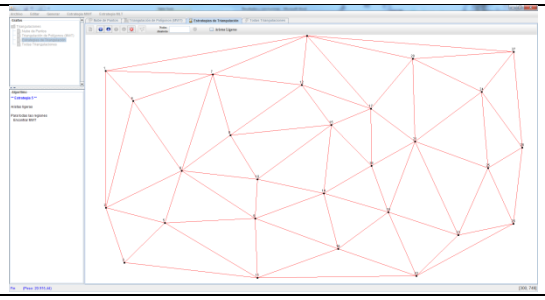
**Nube de puntos 1**




**Nube de puntos 2**




Ahora, ejecutaremos las 5 estrategias para las dos nubes de puntos:

| Nube de puntos 1 | Nube de puntos 2 |
|---|---|
| Estrategia 1 | |
|  |  |
| Estrategia 2 | |
|  |  |
| Estrategia 3 | |
|  |  |
| Estrategia 4 | |
|  |  |
| Estrategia 5 | |
|  |  |

Los pesos de las triangulaciones obtenidas son los siguientes:

| | Nube de puntos 1 | Nube de puntos 2 |
|---|---|---|
| **Estrategia 1** | 50.659,42 | 22.181,89 |
| **Estrategia 2** | 44.328,01 | 20.915,44 |
| **Estrategia 3** | 44.328,01 | 20.915,44 |
| **Estrategia 4** | 43.524,87 | 20.915,44 |
| **Estrategia 5** | 43.568,31 | 20.915,44 |

Para la primera nube de puntos, vemos que se cumplen los 4 puntos comentados anteriormente:

- La estrategia 1 devuelve el peor resultado
- La estrategia 4 devuelve el mejor resultado
- Los resultados de las estrategias 2 y 3 son iguales
- Los resultados de las distintas estrategias (excepto de la 2 y 3), son diferente entre sí

Para la segunda nube, vemos que la primera estrategia es la única que ha presentado un resultado diferente (peor que las demás).

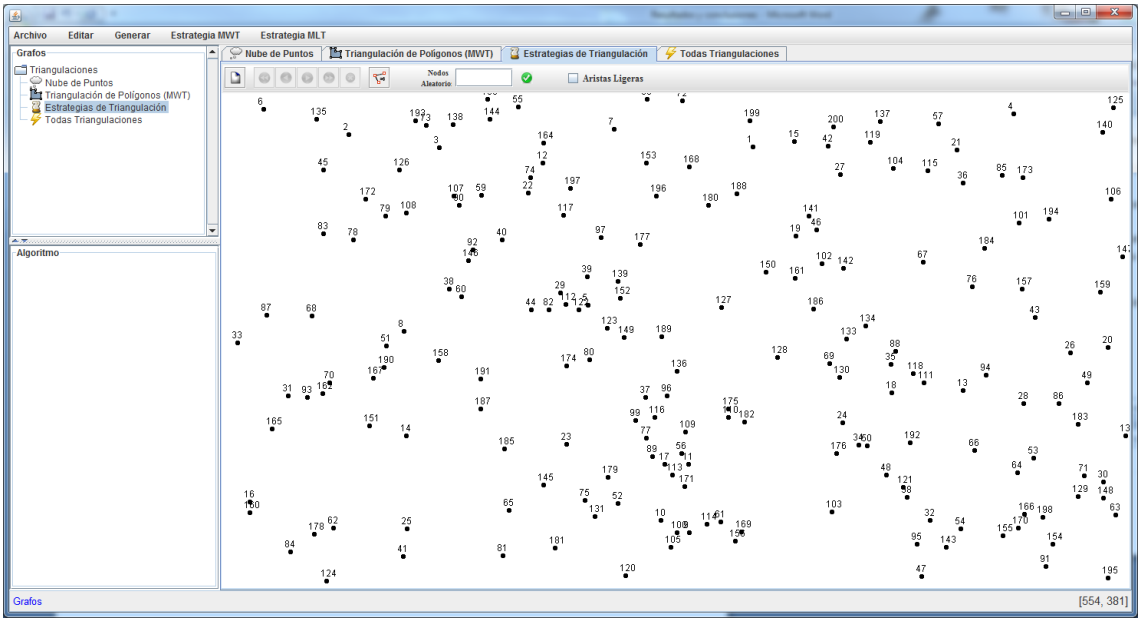
Y al comparar las dos nubes, vemos que cuantos más nodos haya en la nube, mayor es la diferencia de resultados entre las distintas estrategias.

### 7.2.2 TIEMPO DE EJECUCIÓN

Al comparar el tiempo de ejecución de las diferentes estrategias, obtenemos:

- Resultados muy parecidos entre la estrategia 2 y 3 (con pequeñas variaciones).
- La estrategia 1 es la más rápida.
- La estrategia 5 es la que más tarda, debido a la complejidad del algoritmo que calcula las aristas ligeras.
- La estrategia 4 suele tardar menos que las estrategias 2 y 3, y un poco más que la primera estrategia. En algunos casos, la estrategia 4 ha tardado un poco más que la 4, pero en general suele tardar menos.

A continuación, presentaremos los tiempos de ejecución para una nube de 200 puntos generada de forma aleatoria:




Los tiempos obtenidos son los siguientes:

| | Nube de 200 puntos |
|---|---|
| **Estrategia 1** | 2 segundo |
| **Estrategia 2** | 8 segundos |
| **Estrategia 3** | 9 segundos |
| **Estrategia 4** | 2 segundos |
| **Estrategia 5** | 15 minutos |

Es enorme la diferencia entre la quinta estrategia y las demás, mientras que las estrategias 1 y 4 son 4 veces más rápidas que las estrategias 2 y 3.

## 7.3 CONCLUSIONES GENERALES

El desarrollo de este proyecto me ha permitido introducirme en el mundo de la algebra computacional, hasta entonces desconocido para mí.

Las ayudas que he recibido de Gregorio Hernández, tanto de la parte teórica como la de recursos, ha sido fundamental para poder entender la teoría y los problemas de triangulación, como para implementar los algoritmos y las estrategias de programación dinámica.

Además, al desarrollar el proyecto en el lenguaje JAVA, me he familiarizado con las librerías de interfaz gráfica (Java SWING) y con la puesta en práctica de los patrones de diseños estudiados en ingeniería del software.

Para futuras líneas de investigación y continuación del proyecto, se podría implementar otras estrategias distintas, o incluso corregir la funcionalidad del cálculo de la triangulación MWT de polígonos, para que presente un mejor resultado en los casos donde haya alguna rama.

La finalización de este proyecto ha sido una tarea ardua y dura, que me ha permitido comprender la necesidad de superación y persistencia para cumplir los objetivos, además de valorar el trabajo que realizan otras personas, que me han permitido partir de una base sólida para implementar el proyecto.

# 8 BIBLIOGRAFÍA

Abellanas, M. (vol 11 (2008)). *Conectando puntos: Poligonizaciones y otros problemas relacionados.* La Gaceta de la RSME.

Claus, D. (Febrero de 2004). Nearest Neighbour: Condensing and Editing.

Kunzweb, S. (2004). *MWT*. Obtenido de http://stefan.kunzweb.net/en/study/java/mwt/

Loera, J. A. (2009). *Triangulations: Structures and Algorithms.*

Mark de Berg, O. C. (2008). *Computational Geometry: Algorithms and Applications.* Springer-Verlag.

Marques, M. d. (Mayo de 2005). Geometría Computacional Aplicada.

Moreno, L. M. (2007). *DMA*. Obtenido de http://www.dma.fi.upm.es/docencia/trabajosfindecarrera/programas/geometriacomputacional/TrianArteaga/TRIANGULACIONES.html

Peñalver, G. H. (s.f.). Árboles generadores mínimos.

Peñalver, G. H. (2006). *De triángulos a triangulaciones: una nueva mirada geométrica.*

Tan, H. E. (1993). *A quadratic time algorithm for the minmax length triangulation.*

Torres, C. L. (2009). *Trabajo final de análisis y diseño de algoritmos II.*