FACULTAD DE INFORMÁTICA
UNIVERSIDAD POLITÉCNICA DE MADRID

UNIVERSIDAD POLITÉCNICA DE MADRID

FACULTAD DE INFORMÁTICA

TRABAJO FIN DE CARRERA

"PSEUDO-TRIANGULACIONES DE NUBES DE PUNTOS"

JUNIO 2011

AUTOR: Julio A. Quicios García
TUTOR: Gregorio Hernández Peñalver

INDICE

A mis padres, que han sufrido y celebrado conmigo los fracasos y los triunfos de esta empresa en la que me embarqué hace tanto tiempo…

1. Introducción

1.1 Objetivos

El presente proyecto se centra en el desarrollo de una aplicación gráfica que permita obtener y representar visualmente pseudo-triangulaciones de nubes de puntos y polígonos simples, incluyendo la posibilidad de realizar modificaciones en la configuración de la nube y ver el impacto de estos cambios en el resultado obtenido, mediante recalculado en tiempo real.

A tal fin, la aplicación pone a disposición del usuario una variedad de métodos de cálculo, cada uno de los cuales tiene sus aplicaciones concretas y produce unas salidas características. La multiplicidad de resultados para una misma distribución de puntos, según el algoritmo escogido, y la cantidad de acciones disponibles para modificarla e incluso para generar distribuciones aleatorias, concede a la aplicación un carácter de herramienta para la experimentación, más que para la simple exposición de conceptos.

El principal objetivo del trabajo es, por tanto, dotar al usuario de una facilidad que le permita realizar pruebas, pruebas y más pruebas que le ayuden en el estudio de las propiedades y el comportamiento de las pseudo-triangulaciones (a partir de ahora PTs), esas interesantes estructuras geométricas que en breve nos aprestamos a explicar.

Un objetivo secundario sería el de servir como plataforma a un proyecto más ambicioso dotado de mayor funcionalidad, que tuviera mayor capacidad de representación, para nubes de miles de puntos, con posibilidades gráficas como detección de nodos en mapas, zoom, etc.

Siempre pensando en futuras ampliaciones, la aplicación se ha desarrollado modularmente, lo que permite añadir métodos adicionales sin excesiva complicación. Basta con implementar los algoritmos en un módulo separado, que podrá beneficiarse de las facilidades ya programadas para la representación en pantalla, obtención de parámetros característicos y otras tareas comunes. La estructura de la interfaz gráfica de usuario también está preparada para esta eventualidad, permitiendo la adición fácil de nuevos botones de acción y opciones de menú que se integrarán a las barras de herramientas de la aplicación.

1.2 Alcance de la aplicación

Describimos en esta sección las capacidades de la herramienta y los límites a los que se circunscribe: qué es lo que se puede hacer con la herramienta y lo que no.

- *Límites de representación:*

La aplicación está preparada para funcionar con cualquier resolución de pantalla, ajustándose el aspecto de la interfaz de usuario y presentándose maximizada al arrancar. No obstante, no se recomienda usar resoluciones de 800 x 600 e inferiores, pues aparte de perderse algunos elementos de las barras de herramientas, el área de dibujo es pequeña y el resultado será bastante pobre.
El tamaño máximo de la nube está limitado a 200 puntos. Para valores superiores el tiempo de ejecución de algunos algoritmos empieza a ser lo bastante alto como para perder suavidad en el redibujado cuando se modifica la nube en tiempo real.

- *Límites de operación:*

Las operaciones que se pueden realizar sobre el lienzo de dibujo son las siguientes:

- Definición manual de una nube de puntos insertándolos uno a uno
- Generación aleatoria de una nube de puntos que siga una distribución uniforme o bien normal.
- Modificación de la nube, moviendo los puntos individualmente, lo que conlleva actualización automática de la PT dibujada.
- Eliminación individual de puntos, lo que conlleva actualización automática de la PT dibujada.
- Definición de un polígono simple insertando los puntos a uno o uniendo manualmente los de una nube ya presente.
- Cargar una imagen de fondo para usar como referencia en la confección de la nube.
- Guardar en fichero la imagen de pantalla actual.
- Dibujar la PT por el método escogido.
- Configuración del color y grosor de los puntos y aristas de la PT.

Las operaciones que se pueden realizar sobre la distribución de puntos son:

- Cargar la distribución desde un fichero de texto
- Guardar la distribución en fichero de texto
- Calcular la PT por el método escogido
- Obtener los parámetros característicos de la PT

- Métodos de cálculo

Los algoritmos implementados en la última versión de la aplicación son los siguientes:

- Barrido (permite elegir la recta de barrido)
- Capas convexas
- Remoción de aristas (triangulación inicial de Delaunay)
- Incremental (caras de grado acotado: permite elegir la triangulación inicial)
- Caminos mínimos (permite elegir la triangulación inicial)

Los parámetros de la PT obtenibles en la última versión de la aplicación son los siguientes:

- Peso de la PT
- Grado máximo de los vértices
- Grado máximo de las aristas
- Número de puntos de la PT
- Número de caras de la PT
- Número de triángulos de la PT

2. Estudio teórico

2.1 ¿Qué es una Pseudo-triangulación?

En la geometría plana Euclidiana, una triangulación es una partición de una región del plano en triángulos. Así que, sin resultar demasiado osados, podemos pensar que una *pseudo-triangulación* será una partición de una región del plano en pseudo-triángulos. Pero… ¿y qué es un pseudo-triángulo?

Un *pseudo-triángulo* se definió originalmente como una región del plano simplemente conexa, limitada por 3 curvas convexas suaves que son tangentes en sus puntos de unión. Sin embargo en sucesivos trabajos se ha ampliado esa definición para incluir polígonos, que al estar compuestos por segmentos rectos permiten ángulos mayores que cero en los vértices. En el presente trabajo nos centraremos exclusivamente en los pseudo-triángulos poligonales.

Así pues, para el contexto de este trabajo, podemos definir un pseudo-triángulo como un polígono simple de n lados, con 3 y solo 3 vértices convexos (vértices cuyo ángulo asociado abarca menos de π radianes), también llamados *esquinas*.

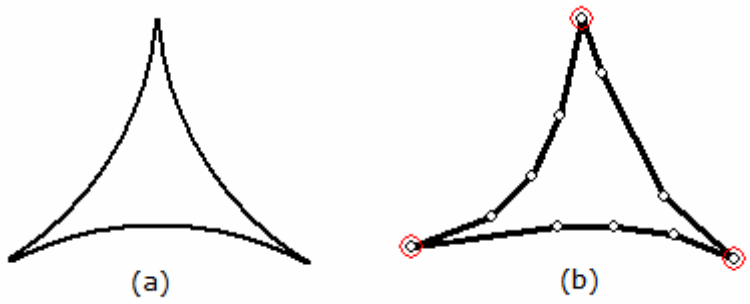


Figura 2.1: (a) pseudo-triángulo, (b) pseudo-triángulo poligonal

Como puede verse a través de estas definiciones, un triángulo no sería más que un caso particular de pseudo-triángulo, con exactamente 3 lados. Y una triangulación sería un caso particular de pseudo-triangulación. Es por esto que las pseudo-triangulaciones generalizan y heredan ciertas propiedades de las triangulaciones corrientes.

Según sea la región sobre la que se realiza la pseudo-triangulación, tendremos una de tres variantes :

- Pseudo-triangulación de un polígono simple *R*
  Es una subdivisión del interior de *R* en pseudo-triángulos, usando solamente los vértices del polígono.

- Pseudo-triangulación de un polígono *R* y un conjunto finito de puntos *P*, interno al polígono y que incluye los vértices de *R*.
  Particiona el interior del polígono usando todos los puntos de *P*.

- Pseudo-triangulación de un conjunto finitos de puntos *P*.
  Similar al anterior, donde el polígono R es el cierre convexo de *P*.

En cuanto a nuestro caso de estudio, nos centraremos en la pseudo-triangulación de un conjunto finito de puntos (al que llamaremos nube) y la del polígono resultante de la unión aleatoria de los puntos de la nube.

2.2 ¿Para qué sirve una Pseudo-triangulación?

El estudio de las pseudo-triangulaciones es una ciencia de muy reciente creación, que se remonta a mediados de la década de los 90, cuando fueron acuñadas por Pocchiola y Verter en 1993 durante el transcurso de unos estudios sobre visibilidad en obstáculos convexos. No obstante, en este tiempo se ha encontrado cierto número de aplicaciones para estas interesantes estructuras matemáticas, además de impulsar varias líneas de investigación. Entre estas últimas podemos destacar:

- Teoría de la Rigidez, con el descubrimiento de que las PTs son estructuras rígidas.
- Propiedades combinatorias de las pseudo-triangulaciones: estudio de su enumeración, de sus parámetros para diferentes conjuntos de puntos y su relación con los conceptos análogos para triangulaciones.
- Conexión entre grafos planos y pseudo-triangulaciones.

En cuanto a las aplicaciones prácticas, podemos encontrarlas en:

- Problemas relacionados con visibilidad, como iluminación de galerías de arte, planificación de la vigilancia en recintos, etc
- Planificación de movimientos en mecanismos robotizados (brazo robótico)
- Detección de colisiones de obstáculos poligonales

Antes de pasar a la presentación de la aplicación y los algoritmos implementados, conviene establecer una terminología básica en torno al asunto:

2.3 Algunas definiciones

- **Grafo**: es un par ordenado G = (V, E) donde:
  - V es un conjunto de vértices o nodos
  - E es un conjunto de arcos o aristas, que relacionan estos nodos

  Se llama *órden de G* a su número de vértices.

- **Grafo plano**: es aquél grafo que puede ser dibujado en el plano cartesiano sin cruce de aristas (las aristas solo intersecan en sus puntos extremos)

- **Polígono:** Figura geométrica formada por una sucesión de segmentos rectos unidos por los extremos que forman un ciclo. Un polígono es un grafo.

- **Polígono simple:** es un polígono cuyos lados no se intersecan. Divide al plano que lo contiene en dos regiones: la interior al polígono y la exterior a él. Un polígono simple es un grafo plano.

- **Vértice:** cada uno de los puntos de unión entre dos segmentos de un polígono. Un vértice de una región poligonal puede ser *convexo*, *llano* o *reflex* dependiendo de si el ángulo abarcado por sus dos aristas incidentes es menor, igual o mayor de π radianes (180º).

  Los vértices convexos de una cara se llaman *esquinas*

  Un vértice interno de una pseudo-triangulación se llama *puntiagudo* o *reflex* cuando tiene algún ángulo incidente > π radianes.

- **Pseudo-k-gono:** Un polígono simple con exactamente k esquinas.

- **Pseudo-Triángulo**: Caso especial de pseudo-k-gono con k = 3.

- **Pseudo-Triangulación**: Grafo plano sobre un conjunto de vértices *P* que subdivide el *convexo(P)* en pseudo-triángulos.

- **Fórmula de Euler**: La fórmula de Euler enuncia que si un grafo plano conexo es dibujado sobre el plano sin intersección de aristas, se cumple:

$$v - a + c = 2$$

  siendo v el número de vértices, a el número de aristas y c el número de caras (regiones delimitadas por aristas, incluyendo la región externa e ilimitada)

2.4 Pseudo-triangulaciones mínimas y minimales

Pseudo-Triangulación *Mínima (puntiaguda)*

Definición:

Aquella en la que en cada vértice tiene una región incidente (bien una cara interna o la cara externa) cuyo ángulo es mayor de π radianes, y tal que no se puede insertar ninguna arista entre dos vértices sin quebrantar esta propiedad.

Alternativamente, es aquella que tiene el menor número de aristas (o equivalentemente, de caras) posible, dado el conjunto S de puntos.

Alternativamente, es aquella que tiene todos sus vértices puntiagudos.

Propiedades:

- nº aristas = 2n – 3 n = número de puntos
- nº caras = n – 2
- Todo conjunto de puntos S admite una pseudo-triangulación puntiaguda
- Si el máximo grado de los vértices de una pt es 5, entonces es puntiaguda
- Si el máximo grado de las caras de una pt es 4, entonces es puntiaguda

Para una pseudo-triangulación con n vértices y t pseudo-triángulos se obtiene por la fórmula de Euler antes vista que el número de aristas es $a = (t + 1) + n - 2$. Recuérdese que la cara externa entra en la cuenta, de ahí el 1 que acompaña a t.
El grado total de los vértices (suma del total de aristas que inciden en ellos) puede entonces ser acotado por arriba considerando las caras: hay $3t$ esquinas (puesto que cada cara tiene tres y solo tres) y a lo sumo, n no-esquinas, puesto que las no-esquinas tienen un ángulo mayor que π. (En el peor caso, si el conjunto de puntos S define un polígono, todos los vértices serán no esquinas, al incidir en ellos la región externa. También serán todos esquinas, al incidir en ellos la respectiva cara interna). Así, puesto que cada arista incide en dos vértices, el grado total será:

$$2a = 2t + 2n - 2 \leq 3t + n$$
$$n - 2 \leq t$$

Para que se cumpla la igualdad, cada vértice debe tener exactamente un ángulo $>\pi$

Pseudo-Triangulación *Minimal*

Definición:

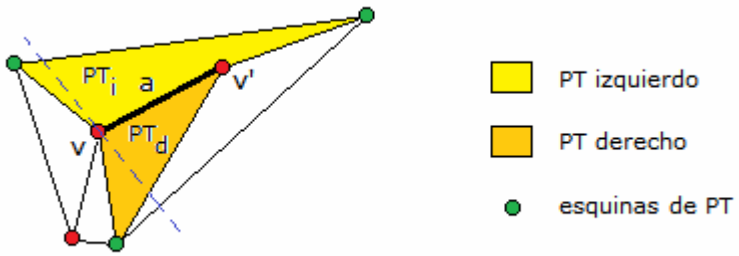
Aquella de que la que no se puede eliminar ninguna arista, porque ningún subgrafo de ella es una pseudo-triangulación que cubra la misma región convexa del plano.

Alternativamente, es aquella en la que la unión de dos caras cualesquiera separadas por una arista no forma un pseudo-triángulo.

Propiedades:

- nº caras ≥ 2n – 3
- si nº caras = 2n – 3, entonces también es puntiaguda
- Si el máximo grado de los vértices de una pt minimal es 4, entonces también es puntiaguda

Supongamos que tenemos una pseudo-triangulación minimal con máximo grado de los vértices igual a 4. Si la pseudo-triangulación no es mínima entonces, según vimos anteriormente, debe haber un vértice *v* sin una región incidente cuyo ángulo sea mayor que π. Puesto que *v* tiene a lo sumo grado 4, podemos encontrar una línea que pase a través de *v* y que separe limpiamente una de sus aristas incidentes del resto. Si suprimiéramos esta arista *a* de la pseudo-triangulación uniríamos los dos pseudo-triángulos que la comparten (PT_{izq} y PT_{der}) en un pseudo-triángulo *PT*. Lo cual demostraría que la pseudo-triangulación no era minimal.



Podemos ver que *PT* es un pseudo-triángulo: en primer lugar, el vértice v es una esquina en PT_{izq} y en PT_{der} por construcción. La remoción de la arista *a* creará un ángulo mayor que π en *v* dentro de *PT*. Por otra parte, el otro extremo de la arista (*v'*) es una esquina en alguno de los dos PT_{izq} y PT_{der}. Su remoción solo creará una esquina *v'* en *PT* si ya lo era en ambos. Esto implica que el número de esquinas en *PT* es exactamente 3, por lo que *PT* es un pseudo-triángulo.

2.5 Conceptos previos

Antes de proceder con la presentación de los diferentes algoritmos implementados en la aplicación es necesario definir unos cuantos conceptos que serán recurrentes a lo largo de las explicaciones y con los que conviene familiarizarnos cuanto antes. Daremos una descripción sucinta de cada uno de ellos:

- Cierre convexo: el cierre convexo de una nube de puntos es el polígono resultante de unir los puntos más externos de la nube de forma que ningún punto quede fuera del polígono y todos sus ángulos internos sean menores o iguales a 180º.
- Capa convexa: Cada uno de los cierres convexos resultantes de excluir de la nube de puntos el cierre convexo anterior. Una nube de puntos puede reducirse por tanto a n capas convexas conteniendo cada una de ellas al menos tres puntos y la más interna al menos uno. La suma de los puntos de las capas es equivalente a la nube inicial.

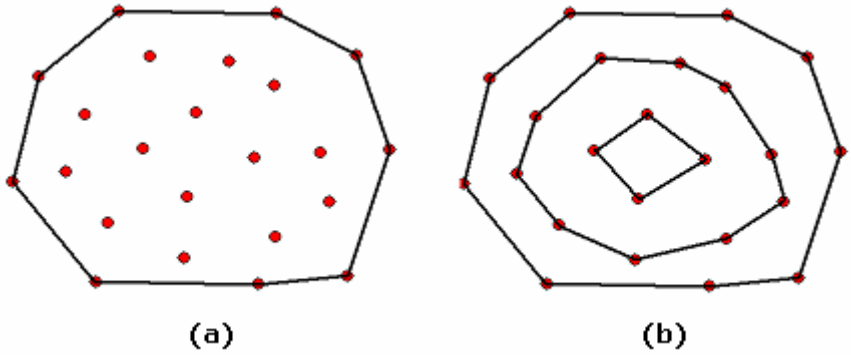


Figura 2.5.1: (a) Cierre convexo. (b) Capas convexas

- Sentido de giro: Dados 3 puntos *p*,*q*,*r* decimos que el sentido de giro del ángulo *pqr* es horario si para ir de *p* a *r* pasando por *q* se describe un giro en el sentido de las agujas del reloj. En caso contrario decimos que el giro es antihorario. Si los 3 puntos están alineados decimos que el sentido de giro es ninguno.

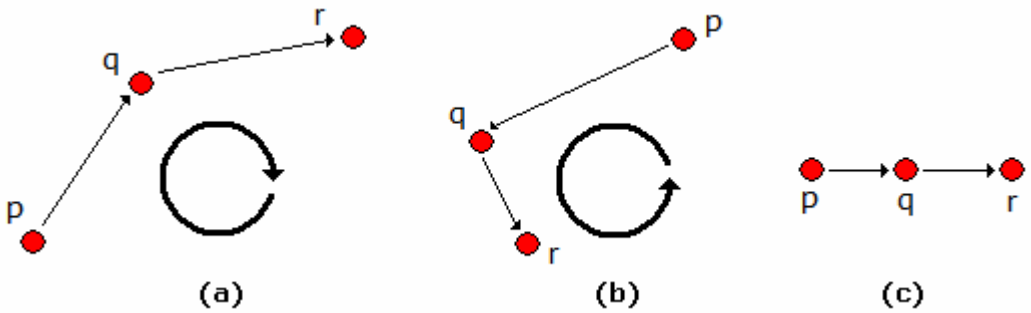


Figura 2.5.2: (a) Sentido horario. (b) Sentido antihorario. (c) Sentido ninguno

- Visibilidad entre vértices de un polígono: Dados dos vértices *u* y *v* pertenecientes a polígono simple *P* sobre el plano, diremos que *u* y *v* son visibles entre sí siempre y cuando sea posible trazar una línea interna al polígono entre *u* y *v* (al ser interna se deduce naturalmente que no puede atravesar ninguna de las aristas del polígono).

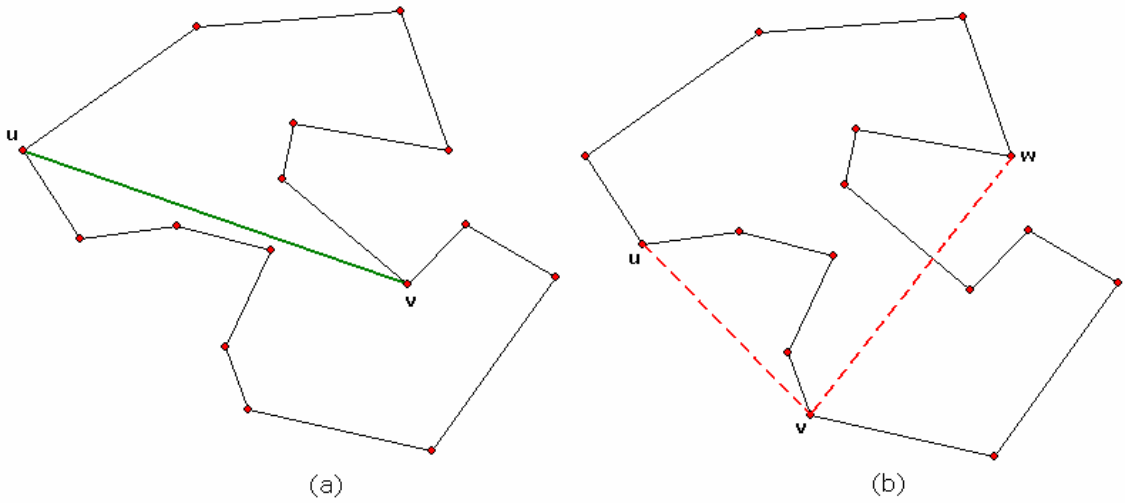


Figura 2.5.3: (a) Los vértices *u* y *v* son visibles entre sí. (b) Ni los vértices *u* y *v* son visibles entre sí (hay línea directa pero por fuera del polígono) ni *v* y *w* (la línea que los une atraviesa aristas del polígono)

- Camino mínimo: Llamamos camino mínimo entre dos nodos de un grafo a la sucesión de arcos del grafo cuya suma de longitudes minimiza su distancia. Análogamente podemos aplicar este concepto a los vértices de un polígono.

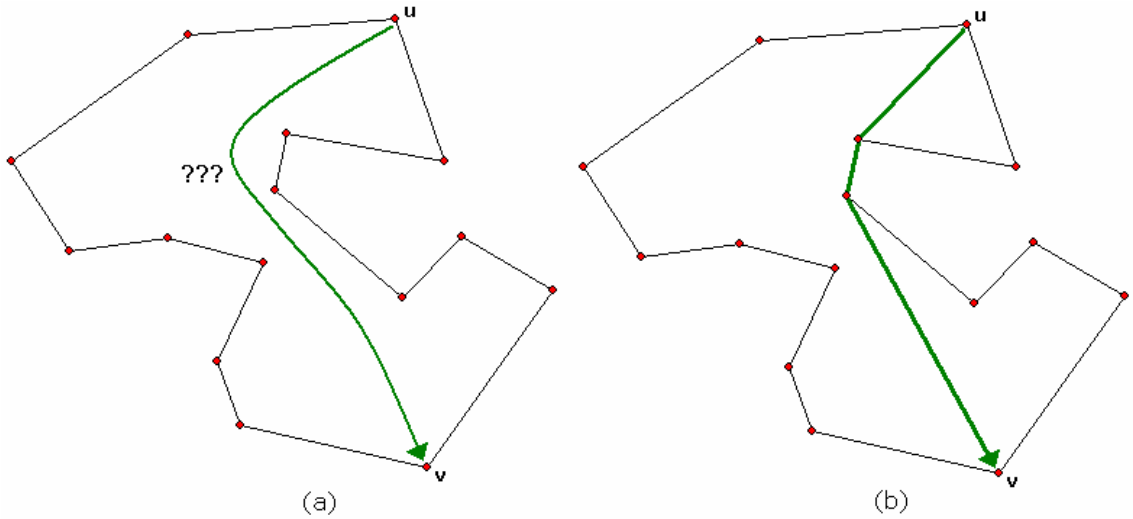


Figura 2.5.4: (a) ¿Cuál es el camino más corto entre los vértices u y v?
(b) Sucesión de aristas de suma mínima de longitudes

- Recta soporte: En el contexto de nuestro estudio, las rectas soporte están ligadas al concepto de visibilidad. Supongamos un objeto plano convexo *C* y un punto *p* externo al mismo.
  Las rectas soporte desde el punto *p* hacia el convexo *C* son aquellas que partiendo de *p* son tangentes a *C*. Es fácil comprobar que la porción de convexo encerrado entre los puntos de tangencia es la parte del convexo visible desde el punto *p*.

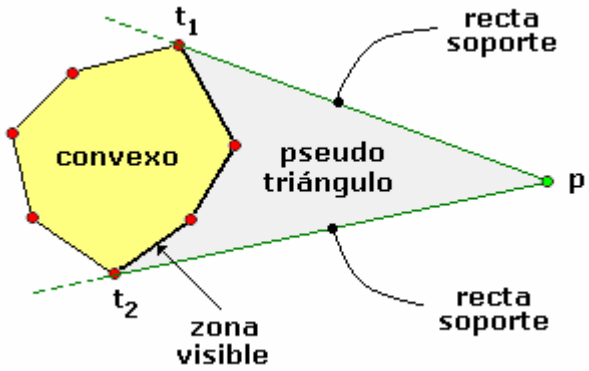


Figura 2.5.5: *p* es el punto externo. t_1, t_2 son los puntos de tangencia

Cada punto *p* lanza dos rectas soporte hacia un cierre convexo, de suerte que el espacio delimitado por las rectas y el convexo definen un pseudo-triángulo.

- Cara: llamamos cara a cada uno de los polígonos internos resultantes de una pseudo-triangulación.
- Vértice réflex: se dice que en un vértice es réflex o puntiagudo cuando alguno de los ángulos formados por dos aristas incidentes en él es mayor de 180º. En la figura de abajo, el vértice marcado como réflex tiene un ángulo obtuso en la cara coloreada de amarillo. El vértice marcado como no-réflex tiene ángulos agudos en todas las caras de las que forma parte, excepto en la cara externa que, naturalmente, no se cuenta.

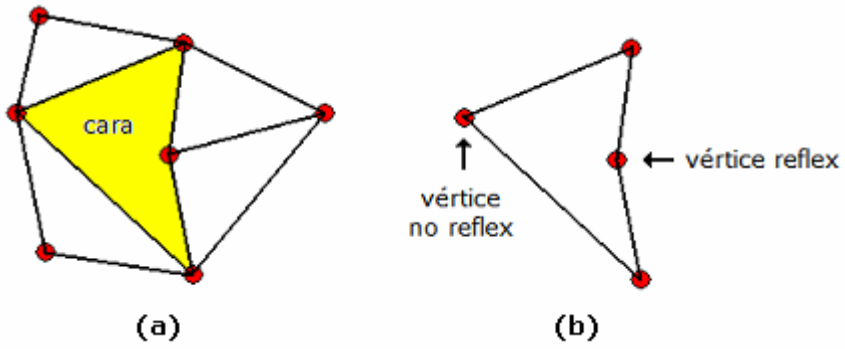


Figura 2.5.6: (a) Una cara de la pseudo-triangulación. (b) vértices réflex y no réflex

- Flip: En términos generales, un flip es una transformación de una pseudo-triangulación *T* en otra mediante la inserción y/o el borrado de una arista. Tenemos tres tipos de flips:
  - Por borrado: la remoción de una arista interior $e \in T$, si el resultado es otra pseudo-triangulación
  - Por inserción: la inserción de una arista interior $e \notin T$, si el resultado es otra pseudo-triangulación.
  - Diagonal: si *e* es una arista interior cuya remoción no produce un pseudo-triángulo, entonces existe una única arista *e'* distinta de *e* que puede añadirse para obtener una nueva pseudo-triangulación $T - e + e'$

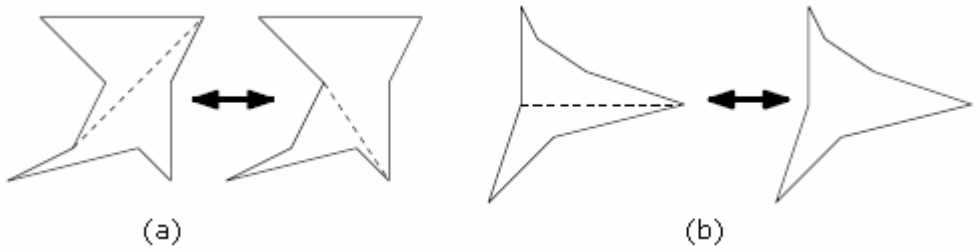


Figura 2.5.7: (a) Flip diagonal. (b) Flips de inserción/borrado
Estos pseudo-triángulos pueden formar parte de una PT mayor.

En el caso concreto de nuestra aplicación, la mayoría de las operaciones de transformación o flips van a ser diagonales. Y los pseudo-triángulos sobre los que se van a realizar serán triángulos unidos por su base. Veámoslo con un ejemplo: Supongamos el cuadrilátero formado por dos triángulos que comparten un lado. Este lado es una arista que une los vértices v_1 y v_2 del cuadrilátero. Un flip consiste en sustituir esta arista por la que une los vértices v_3 y v_4, generando dos triángulos distintos (puesto que en un cuadrilátero solo puede haber dos aristas, la otra será la resultante de unir el otro par de vértices)

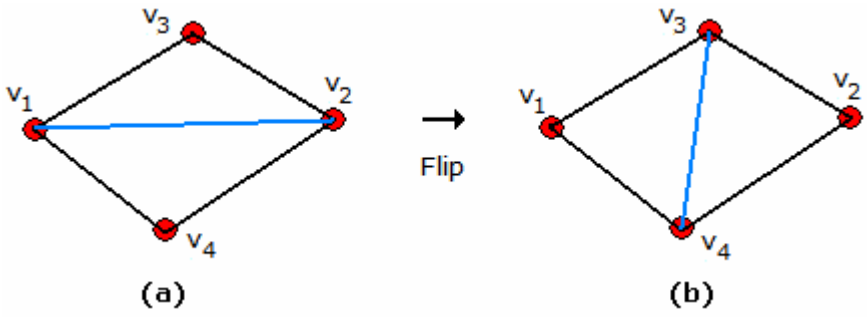


Figura 2.5.8: (a) pseudo-triangulación original. (b) pseudo-triangulación transformada

3. Algoritmos de pseudo-triangulación

En esta sección vamos a hacer un recorrido por los métodos de pseudo-triangulación implementados por la aplicación. El esquema de presentación es el siguiente:

- Explicación teórica de los fundamentos
- Cuadro con los pasos del algoritmo
- Ejemplo práctico con la aplicación
- Complejidad
- Estructuras necesarias

3.1 Barrido

3.1.1 Concepto

El método de barrido es quizás la forma más rápida y fácil de realizar la pseudo-triangulación de una nube de puntos. Requiere pocos cálculos y las estructuras de datos necesarias para almacenar la información son bastante simples, lo que resulta en una implementación muy sencilla.

El algoritmo implica la preordenación de los puntos según la dirección de una "recta de barrido" escogida previamente por el usuario. Sobre esta recta se levanta una perpendicular que va "barriendo" los puntos y nos indica cuál es el próximo a incluir en la PT realizada hasta la fecha. Como se ve, sigue un esquema incremental en el que vamos expandiendo la PT apoyándonos sobre lo que ya llevamos computado.

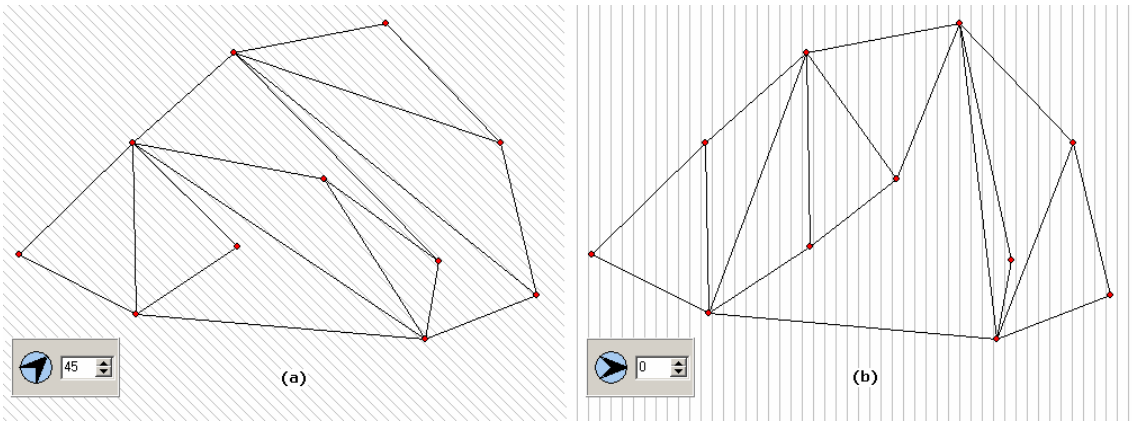


Figura 3.1.1: (a) Recta de barrido a 45º. (b) recta de barrido a 0º
Obsérvese la diferente ordenación de los puntos según son barridos

Como se puede apreciar en la figura 3.1.1, el resultado final cambia con el ángulo de la recta de barrido escogida, dando lugar a fuertes variaciones en los parámetros de la pseudo-triangulación. Por ejemplo, en la PT de la izquierda el máximo grado de las caras es 4, mientras que en la PT de la derecha hay una cara de grado 5. Asimismo, en la PT de la izquierda hay 6 triángulos mientras que la de la derecha cuenta con 7. Parece también que la de la izquierda tiene mayor peso (suma de las longitudes de sus aristas).

3.1.2 Algoritmo

Veamos pues como formular el algoritmo a partir de la idea expuesta. Los pasos a realizar serían los siguientes:

(Suponemos pre-ordenación de los puntos en la dirección de la recta de barrido. Hay que establecer unos pesos P_x, P_y asociados a las coordenadas x e y, para deshacer empates en caso de que dos puntos sean barridos al mismo tiempo. Llamaremos *nube* al conjunto de los puntos sin ningún orden especial, y *vector ordenado* o *vector de puntos* a la estructura que contiene la nube tras ser sometida al proceso de ordenación)

1) Establecer el triángulo inicial
   Esta primera operación consiste en tomar los tres primeros puntos del vector y comprobar que no están alineados. En caso de que así fuera, se introduce una perturbación en uno ellos para deshacer la alineación. Una vez seguros de que no la hay, se procede a unir los 3 puntos en un triángulo. Este será el cierre convexo inicial, el cual se irá expandiendo por la inclusión de sucesivos puntos.
2) Iteración principal: hallar las dos rectas soporte para cada punto
   A partir del cuarto punto, se ejecuta la iteración principal que implica el cálculo de las dos rectas soporte (una en sentido horario y otra en sentido antihorario) que unen el punto actual con el cierre convexo calculado hasta la fecha. El cierre convexo se expande con el nuevo punto y las rectas así calculadas.
   Los puntos y las aristas del anterior cierre que son 'cubiertas' por las rectas soporte se eliminan del nuevo cierre.

Como puede verse, el procedimiento es sencillísimo, una operación inicial y un bucle simple sobre el vector de puntos. La mayor dificultad radica en el cálculo de las rectas soporte. Veamos primero una serie de figuras ilustrativas del proceso de

pseudo-triangulación y seguidamente explicamos el mecanismo utilizado para el cálculo de rectas soporte.

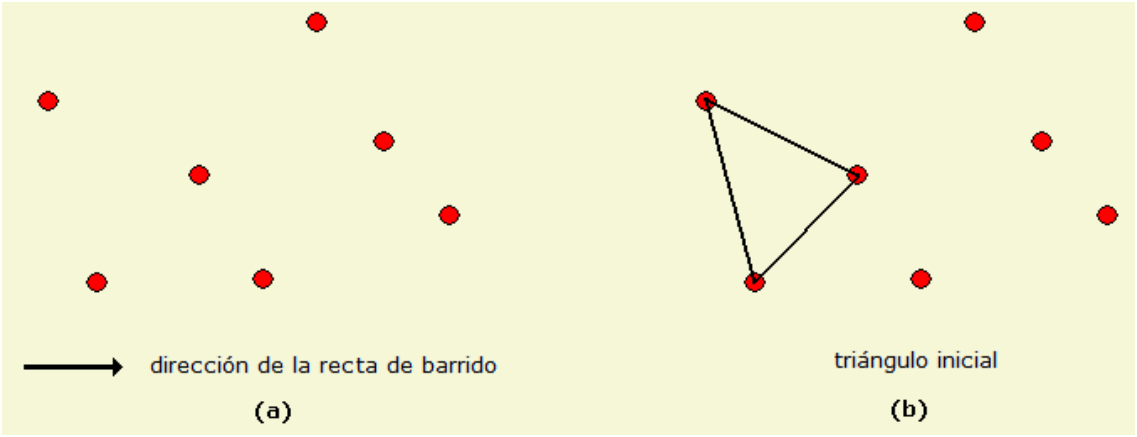


Figura 3.1.2: (a) elección de la recta de barrido. (b) formación del triángulo inicial

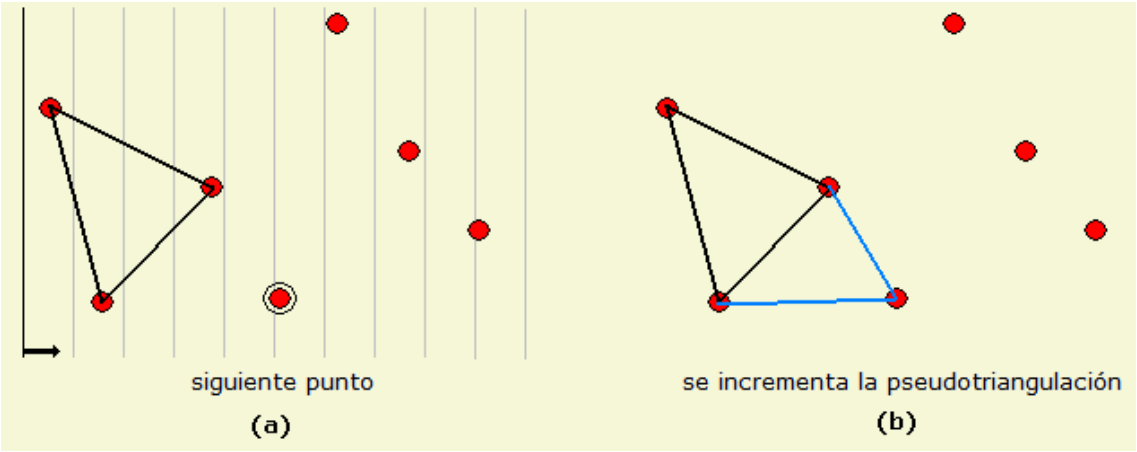


Figura 3.1.3: (a) barrido de un nuevo punto. (b) expansión del convexo

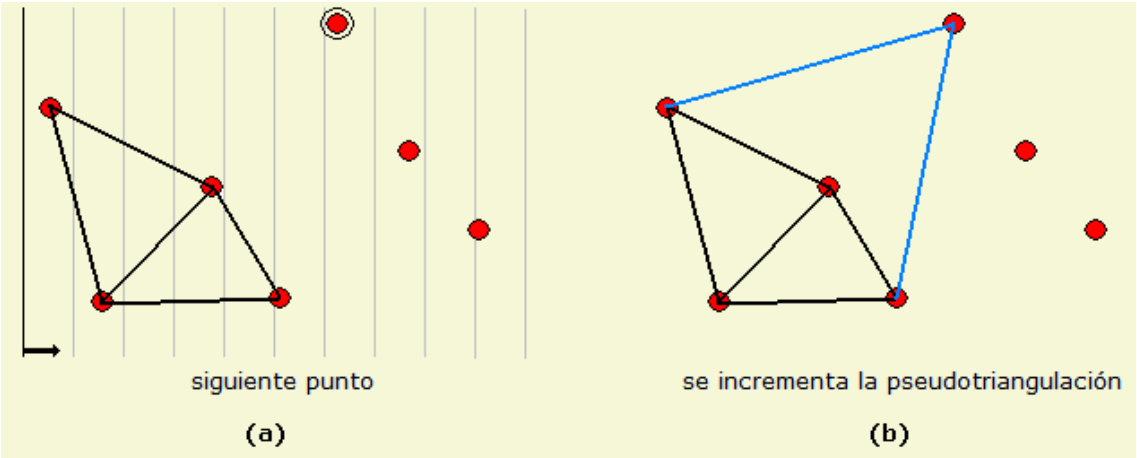


Figura 3.1.4: (a) barrido de un nuevo punto. (b) expansión del convexo

3.1.3 Cálculo de las rectas soporte

Para el cálculo de una recta soporte partimos de 3 puntos *p*,*q*,*r* :

- *p* es el punto i-ésimo del vector de puntos (el punto actual)
- *q* es el punto anterior a *p* en el vector ordenado (el último punto que se añadió a la pseudo-triangulación)
- *r* es el punto siguiente a *q* en el cierre convexo (en sentido horario/antihorario según la recta soporte que estemos calculando)

La recta definida por los puntos *p* y *r* será recta soporte en *sentido horario* si el sentido de giro *pqr* es horario. La recta definida por los puntos *p* y *r* será recta soporte en *sentido antihorario* si el sentido de giro *pqr* es antihorario.
Mientras no se cumpla la condición para ser recta soporte, se actualizan los puntos *q* y *r* con los siguientes puntos del cierre en el sentido correspondiente. Puesto que las rectas soporte van ligadas al concepto de visibilidad en los extremos, tiene que haber dos y solo dos rectas soporte, y necesariamente han de ser distintas.

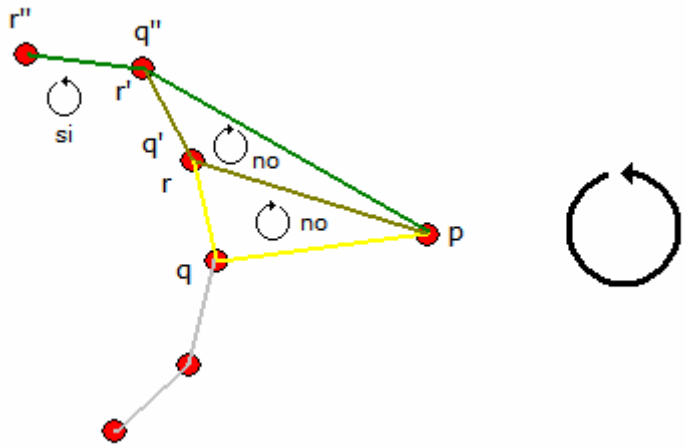


Figura 3.1.5: Búsqueda de la recta soporte en sentido antihorario

Fijándonos en la figura superior, *p* sería el punto actual de la iteración, *q* sería el último punto iterado y *r* el siguiente punto en sentido *antihorario* en el cierre convexo actual. Ahora tendríamos que ir fijándonos en el sentido de giro *pqr*.

- *pqr* tiene sentido de giro horario, luego no nos sirve.
  Hacemos $q \leftarrow q'$, $r \leftarrow r'$
- *pq'r'* tiene sentido de giro horario, tampoco nos vale
  Hacemos $q \leftarrow q''$, $r \leftarrow r''$
- *pq''r''* tiene sentido de giro antihorario, luego *pq''* será recta soporte.

3.1.4 Ejemplo con la aplicación

Vamos a ver un caso sencillo, con una nube de 20 puntos para la que vamos a establecer una línea de barrido a 60º.

Podemos colocar los puntos manualmente o dejar que sea la propia aplicación quien nos genere una distribución aleatoria. Por ser más cómodo vamos a utilizar este segundo método. Desde la barra de menú, seleccionamos distribución uniforme. En la barra de botones pulsamos el botón de distribución aleatoria. Hacemos n = 20 y pulsamos OK.

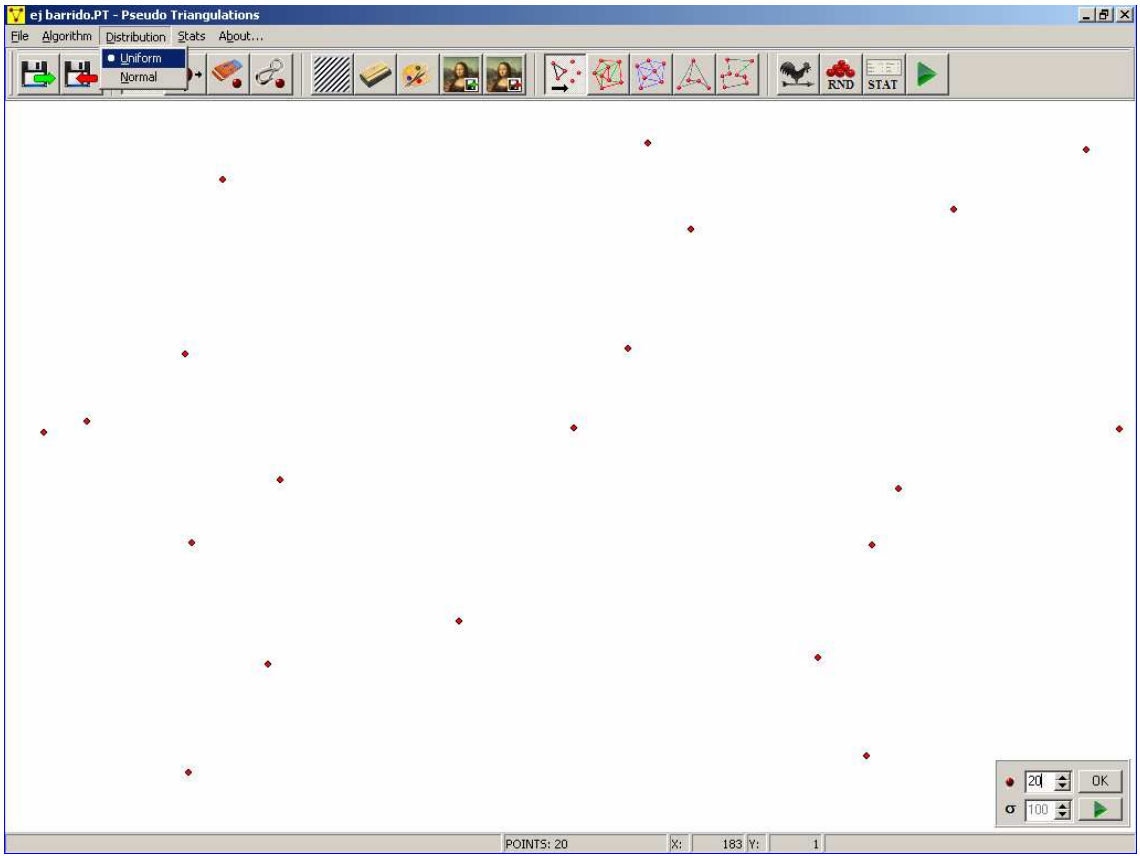


Figura 3.1.6: Generación de una distribución aleatoria normal de 20 puntos

Ahora establecemos la línea de barrido. Pulsando en el botón con el icono de la veleta aparece el panel de elección de ángulo. Introducimos el valor 60.
Con el método de barrido seleccionado (el primero de los botones de algoritmos ha de estar pulsado), pulsamos el botón de ejecución (el último de los botones de la barra). La PT es calculada automáticamente y se nos presenta el dibujo en pantalla:

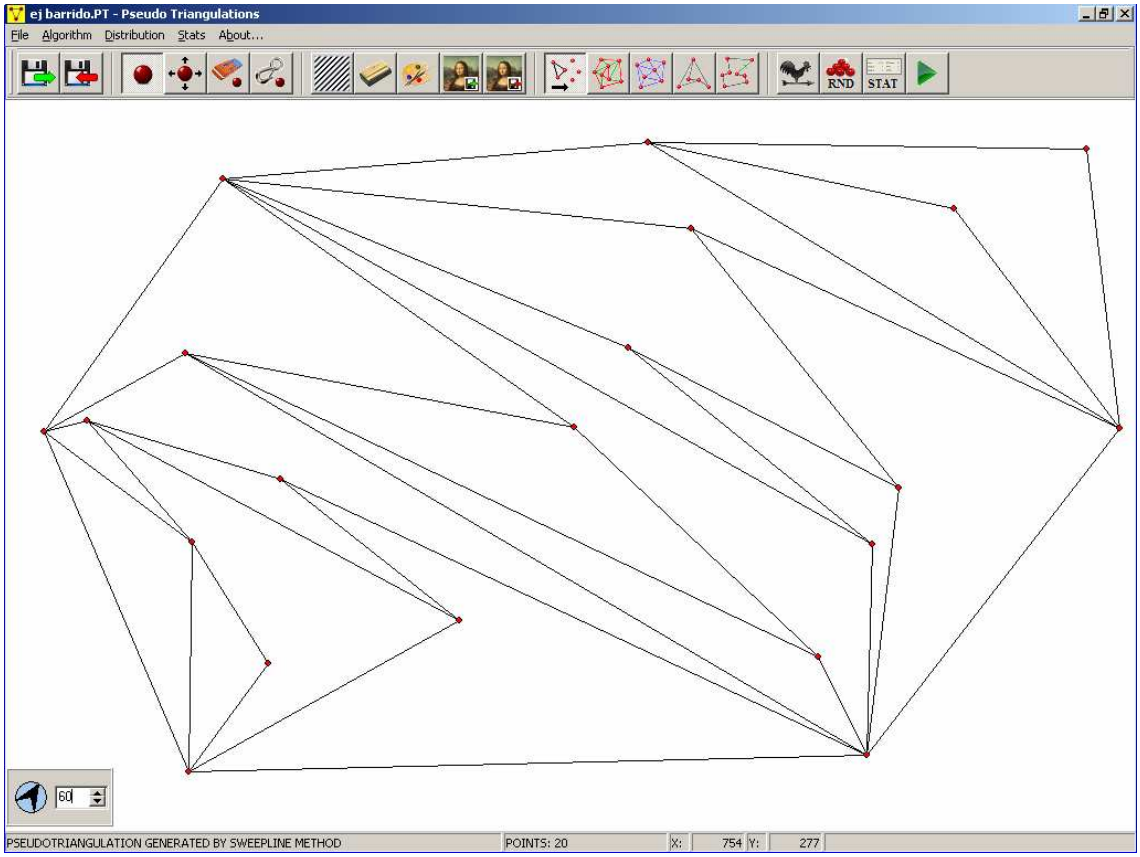


Figura 3.1.7: Pseudo-triangulación resultante con línea de barrido a 60º

El resultado es el mostrado en la figura 3.1.7.
Este método tiene la característica de producir PTs poco vistosas. A menos que los puntos estén bien elegidos las líneas tienden a ser muy largas y las caras muy irregulares.
La PT obtenida siempre es *mínima*, independientemente del ángulo de la recta de barrido.

3.1.5 Complejidad

La complejidad es O(n log(n)) ya que hay una iteración principal para recorrer los n puntos y dentro de ésta, las rectas soporte se calculan en tiempo logarítmico. La operación para calcular el triángulo inicial se realiza en tiempo constante.
Esto convierte al algoritmo de barrido en uno de los métodos más rápidos para obtener una pseudo-triangulación.

Como ya hemos dicho, suponemos que el vector de puntos está preordenado según la dirección marcada por la línea de barrido. No obstante, si sobre una PT terminada, insertáramos un nuevo punto, habría que reordenar el vector, por lo

que el tiempo de ejecución total ya vendría a depender del algoritmo utilizado para la ordenación. Si usamos un algoritmo de tiempo cuadrático, tipo burbuja, su complejidad se trasladaría al algoritmo de barrido, empeorando su eficiencia.

3.1.6 Estructuras necesarias

Para almacenar la estructura interna de la PT

Si solo nos interesa dibujar la PT, bastaría con un vector de duplas como el siguiente:

```
TDupla =  record
            p1, p2: integer;
          end;
TVector = array [0..n-1] of  TDupla;
```

donde:

n es el número de puntos de la PT

p1, p2 son dos punteros desde el punto actual a los extremos de las rectas soporte (que al ser también puntos, son índices del vector).

Así pues, si la posición 7 del vector contiene los punteros p1 = 3, p2 = 5 esto indicaría al procedimiento de dibujado que debe trazar una recta desde el punto 7 al punto 3 y otra desde el punto 7 al punto 5.

No obstante, como en la práctica también nos interesa conocer el valor de una serie de parámetros de la PT, es aconsejable (como se ha hecho en el presente trabajo) utilizar una estructura de mayor versatilidad como el DCEL.

El **DCEL** (Doubly-Connected Edge List) es una estructura de datos centrada en el concepto de arista, aunque en un modo no del todo intuitivo. Cada arista se compone en realidad de dos semi-aristas gemelas, apuntando en direcciones opuestas. Cada arista almacena una referencia a su origen pero no a su destino. El destino de una arista puede saberse consultando el origen de su gemela. Esta estrategia organizativa sugiere que cada semi-arista tiene una orientación, que es siempre contraria a la de su gemela.

La estructura se compone de tres tipos de objetos: vértices, aristas y caras. Cada uno de estos objetos contiene básicamente punteros o referencias a los otros dos tipos. Lo que se consigue con esto es relacionar los elementos de la figura de tal

forma que podemos navegar a través de su topología, recorrerla de extremo a extremo sabiendo siempre dónde nos encontramos.

La implementación que de esta estructura se ha hecho en el presente trabajo es similar a la descrita por la siguiente declaración parcial:

```
TDCEL = class
   private
        v: array of integer;
        a: array of record orig,ant,pos,gem,cara: integer; end;
        c: array of integer;
        numv, numa, numc: integer;
        …
```

- *v* es un vector de vértices. Cada celda *v[i]* contiene un puntero al vector de aristas indicando que el vértice i es origen de la arista *v[i]*

- *c* es el vector de caras. Cada celda *c[i]* contiene un puntero al vector de aristas indicando que la cara i comienza o está representada por la arista *c[i]*

- *a* es el vector de aristas. Cada celda *a[i]* contiene varios campos:

  orig es un puntero al vector de puntos. Indica el vértice origen de la arista
  ant es un puntero al propio vector de aristas. Indica la arista anterior a *a[i]*
  pos es un puntero al propio vector de aristas. Indica la arista posterior a *a[i]*
  gem es un puntero al propio vector de aristas. Indica la arista gemela a *a[i]*
  cara es un puntero al vector de caras. Indica la cara de la que forma parte *a[i]*

- Finalmente, *numv*, *numa* y *numc* son el número de vértices, aristas y caras que contiene la figura representada por la estructura.

Para almacenar el cierre convexo

Para poder realizar los pasos del algoritmo, se necesita una estructura que guarde el orden de los puntos del cierre convexo. Puesto que la actualización del cierre implica numerosas inserciones por la mitad, se desaconseja el uso de vectores. Pese a la inmediatez de la indexación, cada actualización del cierre implicaría acrecentar la dimensión del vector y desplazar buena parte de las celdas del mismo. Por tanto, se ha optado por un mecanismo de memoria dinámica plasmado en una clase *ListaCierre* que se define como un *deque* (lista circular doblemente enlazada). Cada celda contiene la identidad del punto (su índice en el vector de puntos) y dos punteros, uno al punto anterior y otro al posterior.

3.2 Capas convexas

3.2.1 Concepto

Este método consiste en obtener la descomposición en *n* capas convexas de la nube de puntos original, para posteriormente trazar aristas desde los vértices de una capa externa a los de la capa inmediatamente más interna, formando así pseudo-triángulos entre las capas. Realmente es una variación del método de barrido que en lugar de aplicarse a lo largo de una línea se aplicaría en espiral desde el centro de la nube.

Por tanto, al igual que el barrido tradicional, se hace uso del mismo concepto de recta soporte. Cada punto de una capa *i* tendrá una visibilidad sobre el convexo de la capa *i+1*. El trazado de estas líneas de visibilidad es lo que nos da la descomposición en pseudo-triángulos.

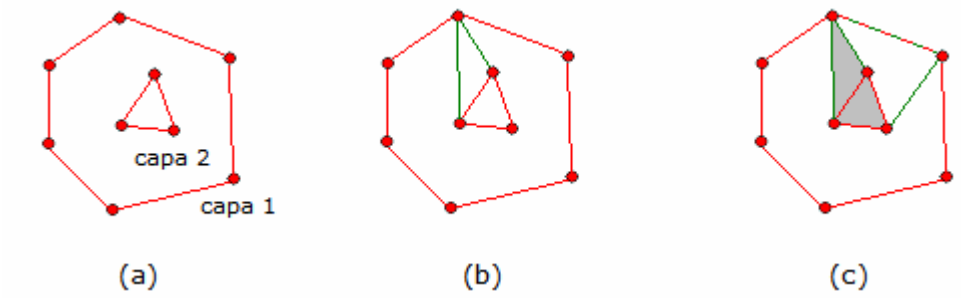


Figura 3.2.1: proceso de obtención de la PT para un ejemplo de 2 capas (a) cálculo de las capas convexas. (b) desde el primer punto se trazan rectas soporte a la capa interna. (c) desde los siguientes se trazan rectas a la capa engordada

Aquí hay que distinguir entre *capa interna* y *capa interna engordada*.
Para el primer punto de la capa externa (b) se trazan rectas hacia la capa interna.
Para los siguientes puntos (c) se trazan rectas a la capa engordada, es decir, el convexo resultante de añadir a la capa interna los pseudo-triángulos ya generados para los puntos anteriores en la capa externa (zona gris).

Las capas externas estarán formadas por un mínimo de 3 puntos.
La capa más interna puede estar formada por:

- 1 punto: se deja como está. El proceso de pseudo-triangular la capa inmediatamente superior consistirá en unir los dos primeros puntos con éste y el resto tirará rectas al convexo.

- 2 puntos: se unen en una arista. El primer punto de la capa inmediatamente superior se unirá a estos dos, formando un triángulo (el convexo inicial). El resto de puntos tirará rectas al convexo.
- 3 puntos: se unen en un triángulo. Los puntos de la capa inmediatamente superior tirarán rectas al convexo.
- 4 puntos: se triangula la capa aleatoriamente (la aplicación usa una triangulación en abanico). Los puntos de la capa inmediatamente superior tirarán rectas al convexo.

3.2.2 Algoritmo

El algoritmo de PT por capas convexas queda fijado en los siguientes pasos:
(Suponemos *n* capas, siendo capa 1 la más externa y capa *n* la más interna)

```
Calcular las capas convexas (mediante Graham Scan, Jarvis March, …)
Triangular la capa n, si fuera aplicable (si consta de 4+ puntos)
capa ← n-1
repetir
        punto ← PrimerPunto(capa)
        repetir
                trazar rectas soporte desde punto a la capa engordada
                punto ← SiguientePunto(punto, capa)
        hasta punto = PrimerPunto(capa)
        capa ← capa -1
hasta capa = 0
```

Como en el caso del método de barrido la operación más laboriosa no forma parte del proceso de pseudo-triangulación en sí, sino de la preordenación inicial de la nube. Calcular las capas convexas puede verse como un equivalente a ordenar los puntos para poder iterar sobre ellos en pasos posteriores.

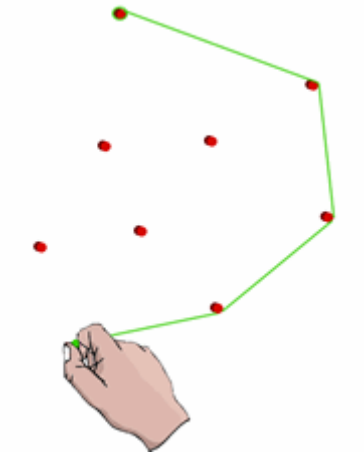
Veamos detenidamente como se calculan las capas y se numeran los puntos dentro de ellas.

3.2.3 Cálculo de las capas convexas

Para obtener las *n* capas de una nube de puntos se sigue un sencillo proceso iterativo consistente en:

- Hallar el cierre convexo de la nube (esta será la capa 1)
- Marcar de alguna manera los puntos del cierre obtenido para que no vuelvan a computarse
- Volver a hallar el cierre de la nube. Esta será la capa 2.
- Marcar los puntos del nuevo cierre
- Repetir la operación hasta agotar todos los puntos.

Ahora bien, para calcular cada una de las capas independientes necesitaremos un algoritmo elaborado que nos resuelva el problema en un tiempo razonable. Para esta tarea ya existen unos cuantos bien documentados, entre ellos el de *Graham* o la *Marcha de Jarvis*. En nuestra aplicación, y por una mera cuestión de sencillez en la implementación decidimos usar el segundo.



Este método se basa en una técnica conocida como 'envoltorio de regalo' y se pude visualizar de la siguiente manera:

Supongamos un tablero con diversos clavos o agujas clavadas en su superficie. Atamos una cuerda al clavo situado más hacia un extremo (por ejemplo el extremo superior del tablero) y la vamos enrollando alrededor del conjunto de clavos. Obtenemos una configuración como la de la figura de la izquierda, similar a la de una cinta de envolver regalos.

Fig 3.2.2: Gift Wrapping

La manera en que funciona esta Marcha de Jarvis es la siguiente:

Convirtamos los clavos en puntos. Debemos encontrar el punto extremo del que hablábamos en el párrafo anterior. A tal fin sería conveniente ordenar los puntos por *coordenada y*, por ejemplo. De menor a mayor ordenada. Así, el primer punto del vector (el de mínima ordenada o el de la parte superior de la pantalla) será la referencia que estamos buscando. Este punto, por definición, estará situado en el cierre convexo. Además es una buena elección porque los ángulos relativos a otros puntos de la nube varían entre 90º y 270º, lo que es perfecto para nuestros propósitos, como enseguida veremos. Además de este punto también debemos localizar el situado en el extremo opuesto del eje (el punto de máxima ordenada o extremo inferior de la pantalla).

Entonces, comenzando con el punto mínimo, tanteamos todos los puntos de la nube, computamos su ángulo relativo y nos quedamos con aquél que dé el valor más bajo. Necesitamos comprobar todos los puntos (excepto el actual) ya que su distribución es arbitraria. El punto encontrado en esta iteración será el siguiente punto en el cierre convexo en sentido horario. Convertimos este nuevo punto en el punto actual e iteramos de nuevo recalculando los ángulos relativos de cada punto a esta nueva referencia. Como se puede ver, los cálculos angulares anteriores no nos sirven al cambiar el punto, por lo que no sirve de nada almacenarlos y deben ser desechados.

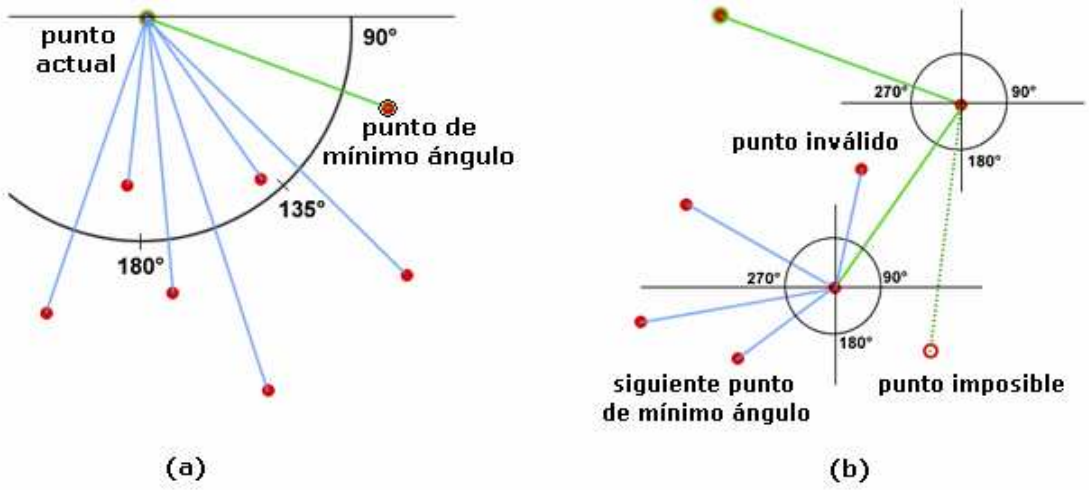


Figura 3.2.3: (a) Cálculo del punto de mínimo ángulo.
(b) Cálculo del i-ésimo punto del cierre convexo

Este proceso se repite, saltando de punto en punto, hasta llegar al punto de máxima ordenada. En este momento, tenemos el lado derecho del cierre completado. El mismo proceso comienza de nuevo desde el punto de mínima ordenada, pero esta vez, en lugar de tantear buscando el punto de menor ángulo relativo, buscamos el de mayor ángulo. Ahora estamos computando la parte izquierda del cierre, y al igual que antes, nos detendremos al llegar de nuevo al punto de máxima ordenada. Una vez acabada esta operación, tendremos el cierre convexo completo.

Ejecutaremos este algoritmo tantas veces como capas convexas puedan extraerse de la nube. La estructura que contenga la nube marcará los puntos ya usados como pertenecientes a la capa en que se eligieron, de forma que no se computen en la próxima iteración. Un valor de capa igual a 0, por ejemplo, podría indicar que el punto aún no se ha utilizado.

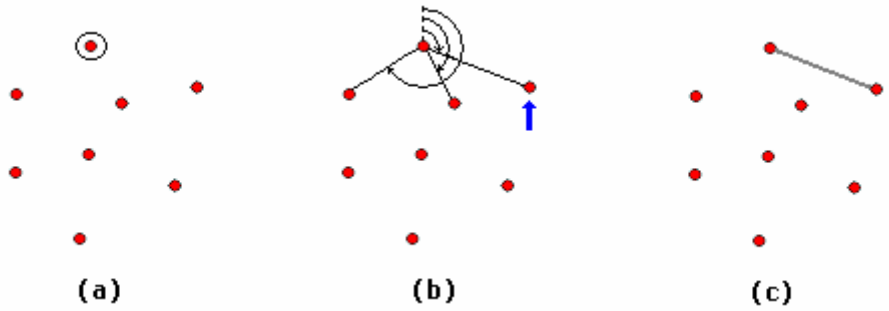


Figura 3.2.3: (a) Encontrar el punto de máxima ordenada. (b) Encontrar el punto con mínimo ángulo. (c) Incluirlo en la capa actual e iterar de nuevo

3.2.5 Ejemplo con la aplicación

Veamos un ejemplo con una distribución aleatoria normal de 20 puntos. Puesto que las capas pueden aproximarse mucho y confundirse si los puntos están cercanos, elegimos una desviación típica alta, de forma que resulte una distribución abierta. También podríamos haber seleccionado una distribución uniforme, ya que la dispersión de los puntos suele ser alta.

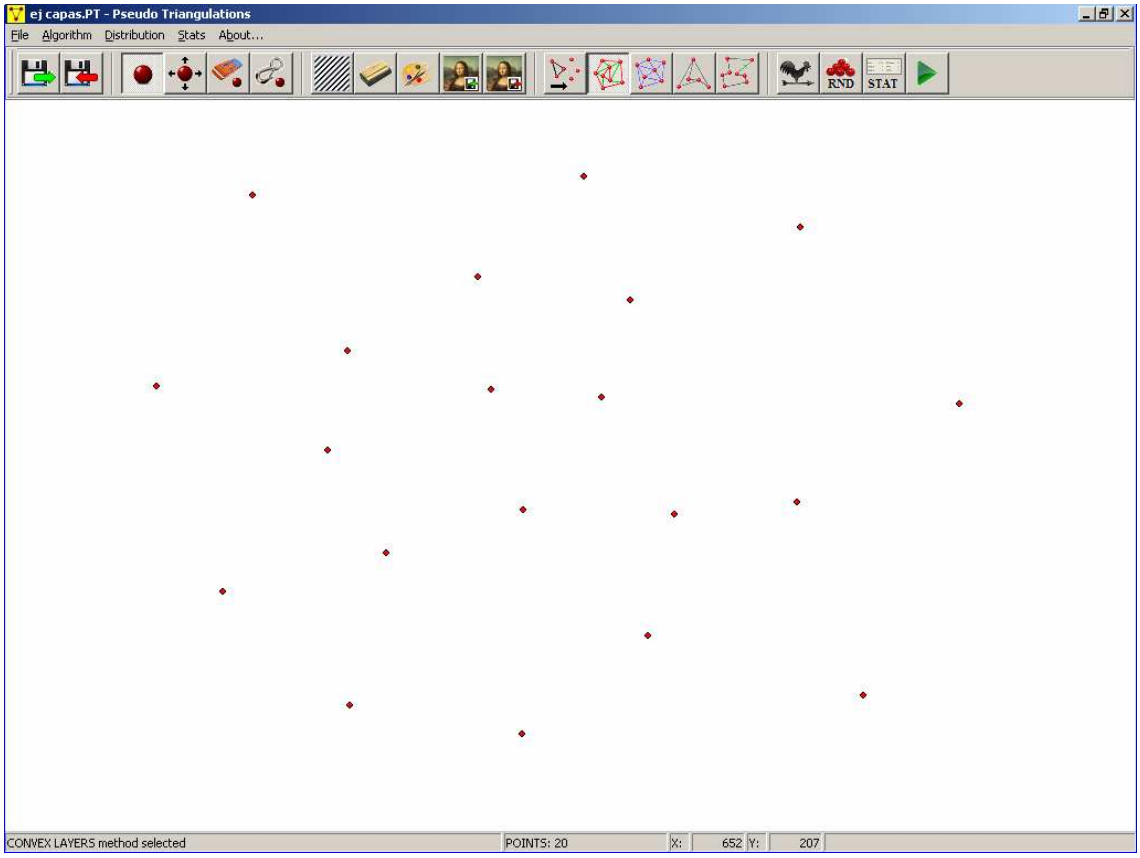


Figura 3.2.4: distribución de 20 puntos para cálculo de PT por capas

Seleccionamos el método de capas convexas pulsando el botón correspondiente (segundo de los botones de algoritmos) y a continuación pulsamos el botón de ejecución. El resultado es el siguiente:

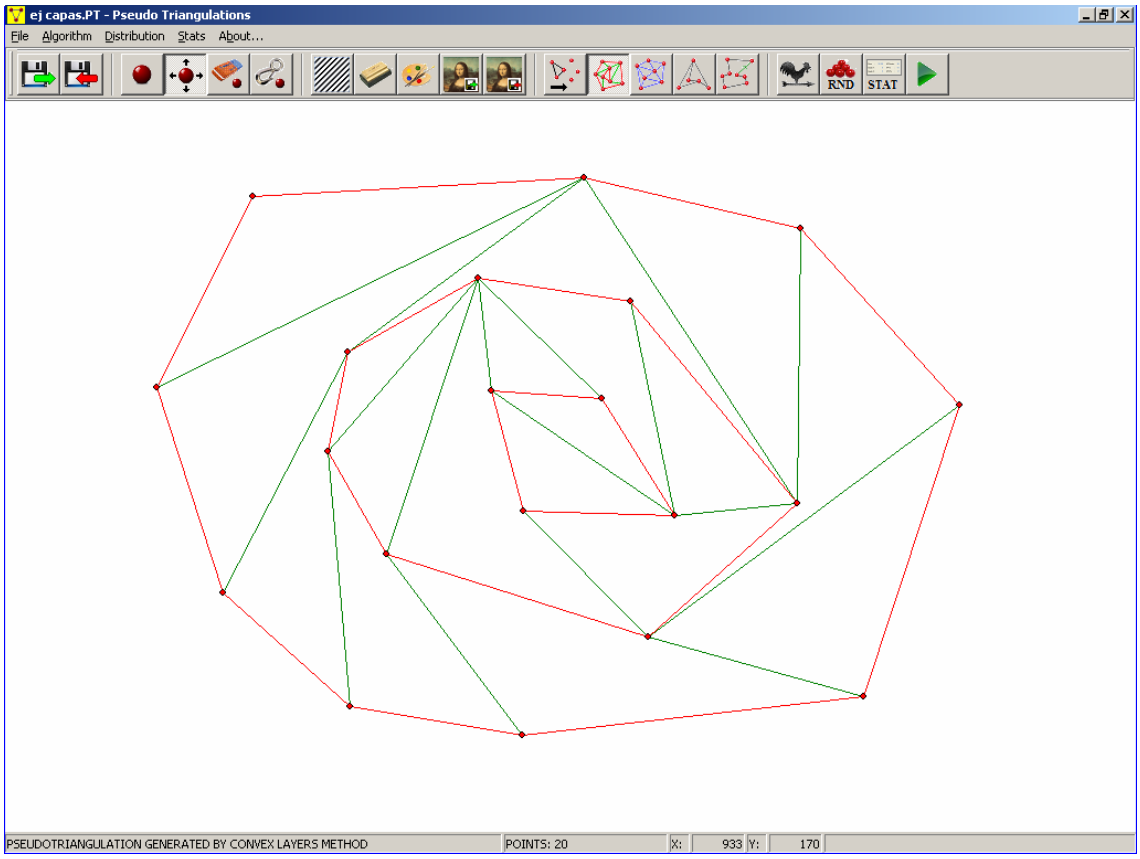


Figura 3.2.5: Pseudo-triangulación resultante de 3 capas

En rojo aparecen las capas convexas. En verde la intercapa. Obsérvese como cada arista perteneciente a ésta viene asociada a una línea de visibilidad desde el punto del que parte hacia el convexo en construcción.

3.2.5 Complejidad

Este algoritmo es bastante más trabajoso de implementar que el de barrido convencional, tanto en la cantidad de código necesario (hay muchos casos especiales que tratar) como en su complejidad y la de la estructura necesaria para guardar la información de la PT. Analicemos los tiempos de cada operación:

- La marcha de Jarvis da una respuesta en tiempo O(nh) siendo n el número total de puntos y h el número de puntos situados en el cierre convexo. En el peor de los casos (n=h) la complejidad es $O(n^2)$. En una distribución

aleatoria con un n de cierta entidad, normalmente el cierre convexo se formará con unos pocos puntos del total de la nube. Pero recuérdese que aquí estamos hallando no el cierre sino todas las capas, por lo que todo punto formará parte de algún cierre. Por tanto, la cota superior de complejidad será cuadrática.

- El cálculo de la intercapa implica el cálculo de las rectas soporte para cada punto, que como vimos en el método de barrido, se conseguía en tiempo logarítmico. Por tanto, complejidad O(log n).

Dado que la iteración para calcular la intercapa se encuentra fuera del bucle de Jarvis (son operaciones secuenciales, no anidadas) la complejidad total del algoritmo es la de la operación más costosa. Por tanto, $O(n^2)$. Algo más ineficiente que el barrido, pero aún así, una de las formas más rápidas de obtener una PT.

3.2.6 Estructuras necesarias

Para almacenar la estructura interna de la PT

Al igual que en el caso del barrido, podríamos haber usado una clase DCEL para este propósito. Sin embargo, las características especiales del algoritmo de capas nos forzaban a extender la definición de la clase para tratar con caras con 'agujeros'. Recuérdese que el primer paso consiste en calcular las capas convexas, por lo que tendríamos una situación con varias caras sin puntos de contacto entre ellas. A fin de mantener la implementación lo más sencilla posible se optó por la utilización de una estructura más ligera. No obstante, dado que el resto de los algoritmos implementados han acabado usando la clase DCEL, sería recomendable reconvertir en un futuro el método de capas para que funcionara con esta clase. Es interesante mantener, en la medida de lo posible, la homogeneidad en los métodos de almacenamiento y dibujado.

La estructura usada en la versión actual de la aplicación se asemeja a la descrita en los siguientes tipos:

```
TCelda =  record
                capa: integer;
                p1, p2: integer;
                p3, p4: integer;
          end;
TVector = array [0..n-1] of  TCelda;
```

El vector tiene tantos elementos como puntos haya en la nube. Cada celda del vector contiene un registro con 5 elementos, que explicamos a continuación:

- *capa* indica el número de capa en la que se encuadra el punto. Este campo puede tomar los valores [1..*n*] siendo *n* el número de capas de la nube. Un valor cero indica que el punto aún no ha sido asignado a ninguna capa. Una vez finalizado el proceso de cálculo de las capas, ningún elemento del vector puede tener este valor a 0.
- *p1* y *p2* son punteros a los puntos anterior y siguiente, respectivamente, en la capa a la que pertenece el punto. Proporcionan la información para dibujar las capas convexas.
- *p3* y *p4* son punteros a los extremos de las rectas soporte que se proyectan desde el punto. Proporcionan información para dibujar la intercapa. Normalmente los puntos señalados por estos punteros estarán en la capa inmediatamente inferior, pero en algunos casos tanto *p3* como *p4* pueden estar en la misma capa que el punto.

Para almacenar el cierre convexo

Al igual que en el algoritmo de barrido, también se necesita una estructura para mantener el cierre convexo, pues aunque la estructura definida en el apartado anterior nos permite guardar las distintas capas y el orden en que vienen sus puntos, aún necesitamos una estructura auxiliar para ir guardando la *capa engordada* o convexo entre capas que se va construyendo. Para este fin, reutilizamos la clase *ListaCierre* usada en el método de barrido, que se definía como una lista circular doblemente enlazada.

3.3 Triangular y suprimir

3.3.1 Concepto

El modo de operación de este método consiste en realizar primeramente una triangulación *T* de la nube de puntos *S*. A continuación, en una primera pasada, se eliminan las aristas que cumplan ciertas condiciones. En una segunda pasada, se afina más intentando suprimir triángulos enteros. El resultado final será una pseudo-triangulación *minimal* contenida en *T*, que designaremos por P^T.

Para obtener la triangulación inicial podemos hacer uso de cualquiera de los algoritmos conocidos y disponibles a tal fin. Es importante señalar que la PT final heredará de alguna manera 'la forma' de la triangulación, por lo que parece conveniente elegir un algoritmo que produzca triangulaciones 'vistosas', por así decirlo. Con este objetivo en mente, hemos elegido la triangulación de Delaunay porque tiene la característica de maximizar los ángulos interiores de los triángulos, resultando formas muy regulares, si se nos permite la expresión, en oposición a triángulos con ángulos muy agudos y lados muy juntos que a menudo resultan difíciles de identificar en el dibujo, especialmente en PTs con gran número de caras.

La triangulación de Delaunay obliga a todos sus triángulos a cumplir la llamada *condición de Delaunay*, que especifica que para cada triángulo, la circunferencia circunscrita asociada no puede contener otros vértices aparte de los 3 que la definen.

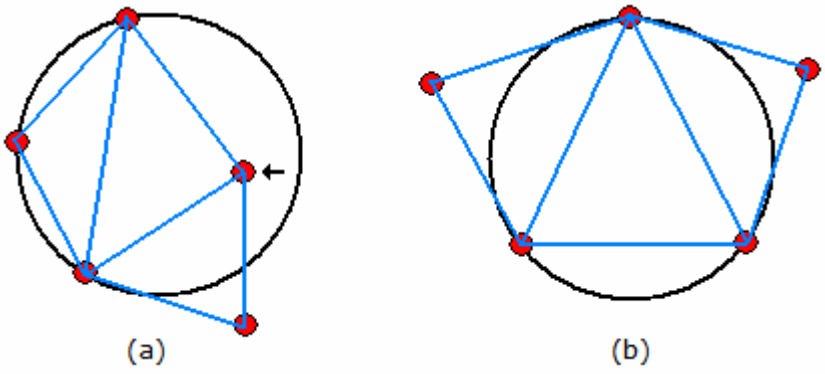


Figura 3.3.1: (a) no cumple condición de Delaunay. (b) sí la cumple

Hay varias maneras de obtener la triangulación de Delaunay. Una de las más sencillas, y por tanto la que hemos usado, consiste en partir de una triangulación cualquiera y hacer flips de aquellas aristas que provoquen que algún triángulo no cumpla la condición (aristas ilegales), repitiendo la operación hasta que todos los triángulos la cumplan. Aprovechando que tenemos ya implementado el método de

pseudo-triangulación por barrido, podemos realizar unos cambios mínimos en el algoritmo para que también sea capaz de generar triangulaciones por barrido. Basta con que, para cada nuevo punto añadido, trace además de las rectas soporte también las rectas que no cumplieron con la condición para serlo.

Ahora bien, ¿cómo sabemos cuando una arista o un triángulo se pueden suprimir? El siguiente lema nos dice cuando es factible realizar la eliminación:

Lema:

(a) *Sea P una pseudo-triangulación y $e \in P$ una arista. Entonces P – e es una pseudo-triangulación si y solo si la eliminación de e crea un nuevo vértice réflex, es decir, si un extremo de e no es réflex en P y sí es réflex en P – e.*

(b) *Sea P una pseudo-triangulación y $\{e_1, e_2, e_3\} \in P$ una cara triangular de P. Entonces P – $\{e_1, e_2, e_3\}$ es una pseudo-triangulación si y sólo si la eliminación del triángulo convierte a los tres vértices en réflex, o más precisamente, si los 3 vértices de $\{e_1, e_2, e_3\}$ no son réflex en P y sí lo son en P – $\{e_1, e_2, e_3\}$.*

Recordemos que un vértice es réflex o puntiagudo si uno de los ángulos formados por las aristas incidentes en él es mayor que 180º.
Las condiciones descritas en el lema anterior son sencillas de comprobar, computacionalmente hablando. Sean *a* y *b* dos puntos de la pseudo-triangulación *P*, y sea *e* = *ab* la arista que los une. Sean α_1 y α_2 los ángulos incidentes en *e* desde el punto *a, y sean* β_1 y β_2 los ángulos incidentes en *e* desde el punto b. Entonces *P – e* es una pseudo-triangulación si y solo si $\alpha_1 < \pi$, $\alpha_2 < \pi$ y $\alpha_1 + \alpha_2 > \pi$, o bien $\beta_1 < \pi$, $\beta_2 < \pi$ y $\beta_1 + \beta_2 > \pi$. Esta condición puede ser extensible al apartado (b) del lema, para la eliminación de un triángulo. Habría que comprobarla para cada una de las 3 aristas constituyentes.

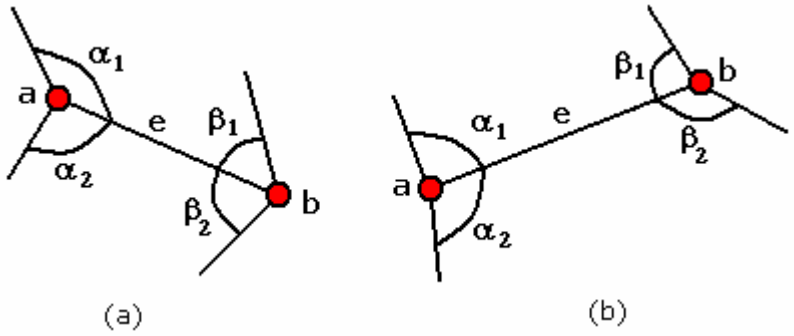


Figura 3.3.2: (a) La arista *e* se puede eliminar. El punto *a* pasa de no-reflex a reflex mientras que el punto *b* permanece reflex
(b) La arista *e* no se puede eliminar. Ambos vértices se convierten en reflex.

Es importante observar que para poder eliminar la arista, sólo uno de sus extremos debe cambiar de carácter, es decir sólo uno de los vértices debe convertirse en réflex, mientras que el otro debe conservarse como es.

3.3.2 Algoritmo

Veamos entonces la formulación del algoritmo. Necesitaremos de una estructura auxiliar para guardar las aristas susceptibles de eliminación. Una simple pila nos servirá para este propósito. Iniciamos la pila a vacía y recibimos el conjunto de puntos S. El algoritmo consta de los siguientes pasos:

1) Realizar la triangulación T de la nube de puntos S.
2) Comprobar una a una todas las aristas, ver si satisfacen la condición (a) del lema anterior. Si una arista *e* la satisface se introduce en la pila. Este bucle se realiza en tiempo lineal. Hacemos P = T.
3) Iteración en aristas: se extrae la cabeza de la pila y se elimina de P la arista indicada. La eliminación de una arista puede afectar las condiciones de sus vecinas, de suerte que es necesario comprobar (a) para las dos aristas vecinas en cada extremo de la arista eliminada. Si alguna de ellas la cumpliera, se introduce en la pila. Estas aristas pueden comprobarse en tiempo constante. Repetir hasta vaciar la pila.
4) Iteración en caras: comprobar uno a uno todos los triángulos de P. Si alguno cumple la condición (b) del lema, se eliminan las tres aristas que lo componen. La eliminación de un triángulo no puede crear una nueva arista o triángulo eliminable. Por tanto, esta comprobación puede hacerse en tiempo lineal.

Al finalizar habremos obtenido una pseudo-triangulación sin aristas o triángulos eliminables, siendo por tanto *minimal*, aunque por lo general, no será mínima.

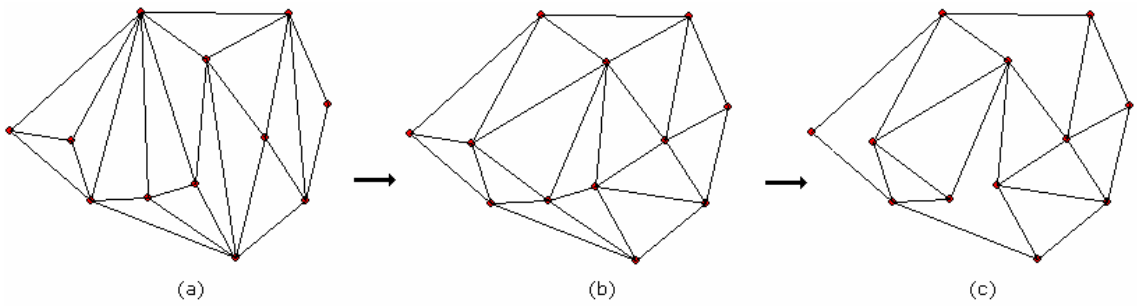


Figura 3.3.3: (a) Triangulación por barrido. (b) Transformación a Delaunay mediante flips. (c) Eliminación de aristas

3.3.3 Ejemplo con la aplicación

La aplicación permite ver cada una de las etapas de construcción de la PT mediante las opciones de la barra de menú. Vamos a ver un ejemplo con una distribución aleatoria.

El primer paso es generar la nube de puntos S. Para que el ejemplo no resulte trivial hagamos n = 50 puntos. Seleccionamos la opción de generar una distribución aleatoria de carácter normal, con una desviación típica lo bastante alta como para no concentrar demasiado los puntos en torno al centro.

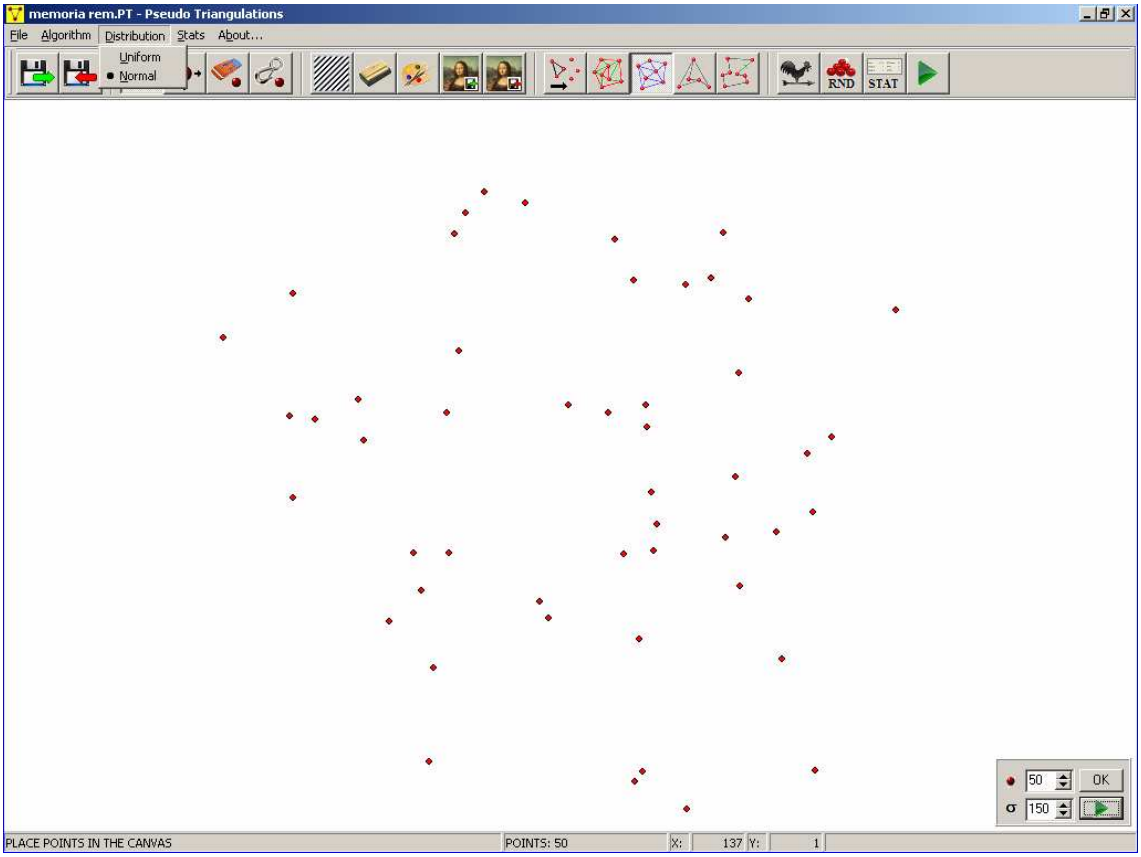


Figura 3.3.4: Generación aleatoria de una nube de 50 puntos

Ahora procederemos a realizar la triangulación de la nube. Como ha sido explicado antes, internamente la aplicación ejecuta un algoritmo de triangulación por barrido que parte desde el punto de menor coordenada (por defecto, el punto más a la izquierda en el eje de las *x*, pero depende de la configuración de la línea de barrido), y se va moviendo hacia el punto de mayor coordenada. Con los tres primeros puntos encontrados forma el triángulo inicial y cada punto nuevo es unido al convexo anterior añadiendo uno o varios triángulos. El nuevo punto se unirá a todos aquellos puntos sobre los que tenga visibilidad. Si el punto *i* tiene visibilidad sobre *k* puntos del convexo actual, se formarán *k*-1 nuevos triángulos.

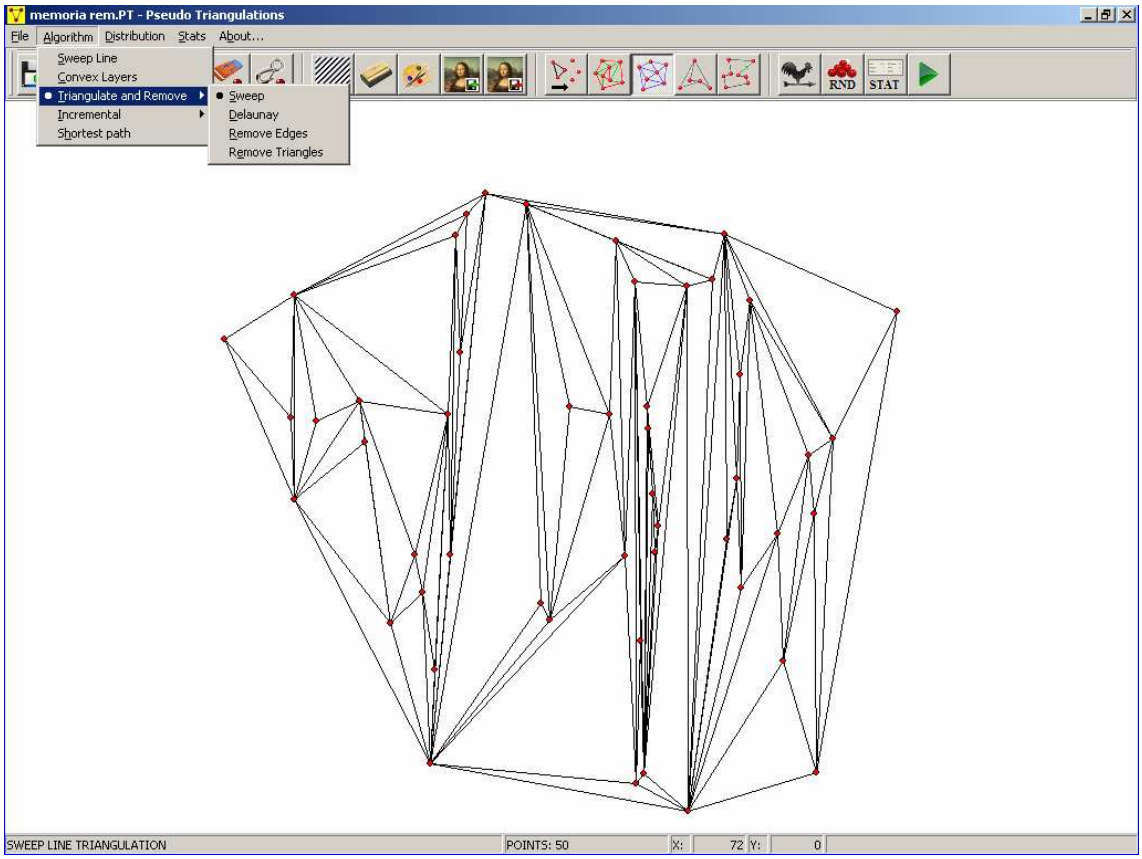


Figura 3.3.5: Triangulación de S mediante técnica de barrido

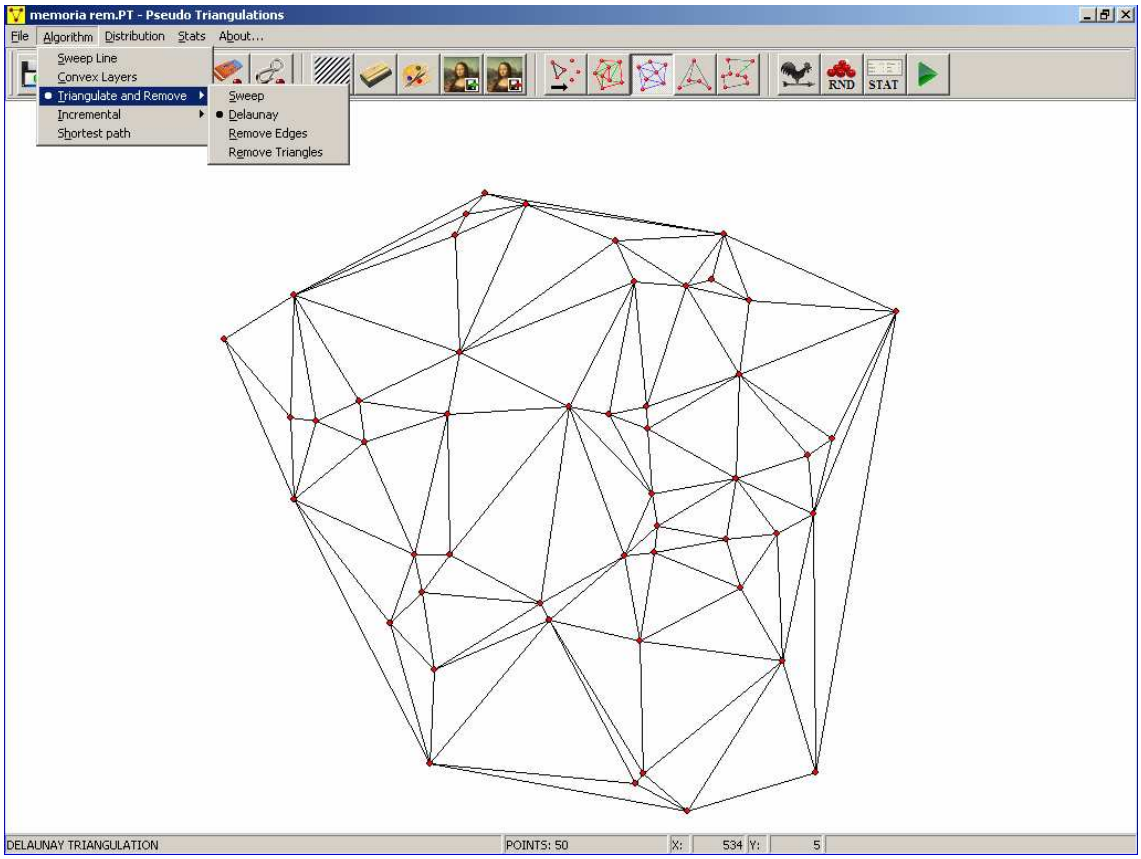


Figura 3.3.6: Transformación en triangulación de Delaunay

Como se puede observar en la figura 3.3.5, la triangulación por barrido produce unos resultados muy poco elegantes. La mayoría de los triángulos son excesivamente alargados, con ángulos internos muy pequeños, lo que hace prácticamente indistinguibles unos triángulos de otros. Además las líneas están demasiado juntas y no se sabe muy bien desde qué punto parten y en cuál acaban.

Para solucionar todos estos inconvenientes transformamos la triangulación inicial en una triangulación de Delaunay, mediante escaneo de aristas y realizando 'flips' en aquellas que no cumplan la condición de localidad de Delaunay. El resultado puede verse en la figura 3.3.6. ¡Esto sin duda es mucho mejor!

Veamos ahora el resultado ejecutando hasta el tercer paso.

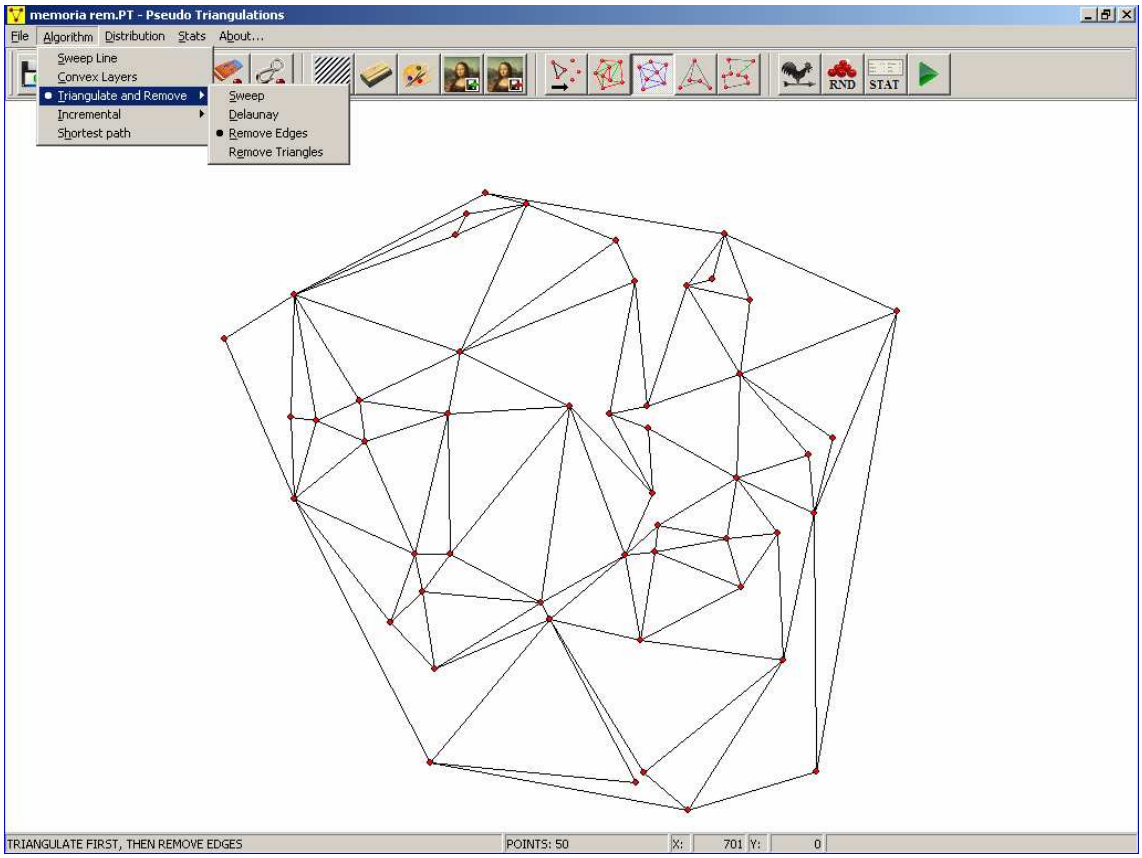


Figura 3.3.7: Eliminación de aristas

Podemos ver que algunas de las aristas han desaparecido, causando que varios triángulos se unieran en polígonos con mayor número de caras que, no obstante, solo tienen 3 vértices convexos, por lo que continúan siendo pseudo-triángulos.
Para terminar nos quedaría ver si es posible reducir aún más el número de aristas suprimiendo caras enteras. Aunque en la práctica esto no suele ocurrir para un número bajo de puntos, hemos tenido suerte con esta distribución.

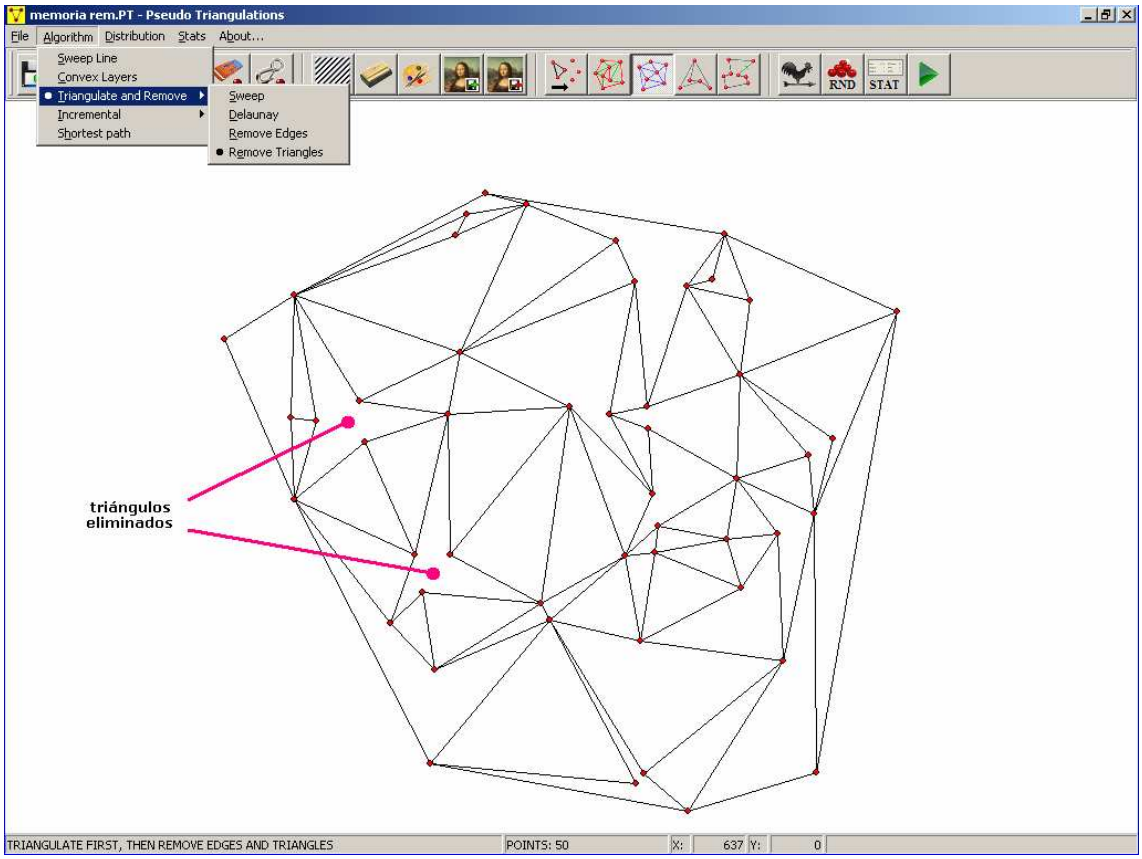


Figura 3.3.8: Eliminación de triángulos

Como puede apreciarse, hacia la parte central izquierda, había un par de estructuras compuestas por tres triángulos unidos por sus bases a un cuarto triángulo central. Este último se ha eliminado resultando unos pseudo-triángulos de 6 caras similares a estrellas de 3 puntas.

Por las características del algoritmo, hemos llegado a una PT que no puede contener otra PT de menor peso o de menor número de aristas.

3.3.4 Complejidad

Para hallar la complejidad del algoritmo deberemos examinar la de cada una de las operaciones individuales que se realizan:

- La triangulación por barrido se realiza en O(n log(n)), ya que hay una iteración principal para recorrer los n puntos y dentro de esta, las rectas hacia el convexo se calculan en O(log(n)).
- La transformación en triangulación de Delaunay se efectúa en tiempo $O(n^2)$ en el pero caso: si solo hay una arista ilegal en la triangulación, tendremos que revisar todas las aristas para encontrarla. Entonces puede ocurrir que al

legalizarla se genere una nueva arista ilegal única. Como esto se puede producir tantas veces como aristas interiores tenga la triangulación llegamos a complejidad cuadrática en el peor caso (por cada arista, hemos de comprobar todas)

- El escaneo de aristas eliminables se realiza en tiempo lineal O(n), como ya vimos.
- El escaneo de triángulos eliminable también se realiza en O(n).

Dado que todas estas operaciones se realizan secuencialmente, la complejidad total nos la da la operación con la mayor complejidad, que es el cálculo de la triangulación de Delaunay.

El algoritmo de triangulación y eliminación de aristas tiene, por tanto, complejidad cuadrática $O(n^2)$.

3.3.5 Estructuras necesarias

Este algoritmo es con diferencia, el más costoso de implementar de los tres estudiados hasta la fecha, viéndonos en la necesidad de definir numerosas funciones auxiliares y un par de clases abstractas en las que apoyarse:

- Para almacenar las aristas susceptibles de eliminación se ha usado una clase Pila, con sus dos operaciones básicas *Push* y *Pop*.

- Para guardar la estructura interna de la PT se optó por una clase DCEL, ya que son muchas las consultas necesarias sobre aristas y vértices para los cálculos del algoritmo.

3.4 Incremental

Un método de pseudo-triangulación incremental es aquél en el que primeramente se realiza el cierre convexo de la nube de puntos y posteriormente, se va definiendo la PT mediante el tratamiento de los puntos interiores que se van incorporando sucesivamente. Aunque en esta definición entrarían bastantes métodos, lo usaremos aquí para referirnos específicamente a dos de ellos, que tienen la característica común de acotar algún parámetro de la PT. En concreto, uno de ellos logra acotar el grado máximo de las caras y el otro el grado máximo de los vértices. El objetivo de estas acotaciones es asegurar que las PTs obtenidas sean *mínimas*. Pasamos a explicarlos.

3.4.1 Caras de grado acotado

Este método propone una implementación que reduce el máximo grado de las caras (número de aristas que las conforman) a 4. Es decir, todas las caras de la PT serán o bien triángulos o bien cuadriláteros.

3.4.1.1 Concepto

Partimos de una nube de puntos con su cierre convexo calculado. La siguiente tarea que hemos de realizar es triangular el cierre de forma que el interior quede dividido en un conjunto de caras de grado 3. En principio, a efectos de la explicación, la forma de la triangulación nos es indiferente. Por hacerlo fácil, podríamos aplicar una triangulación en abanico, uniendo cada punto del cierre con el de ordenada superior. Aunque nos vale una triangulación aleatoria cualquiera.

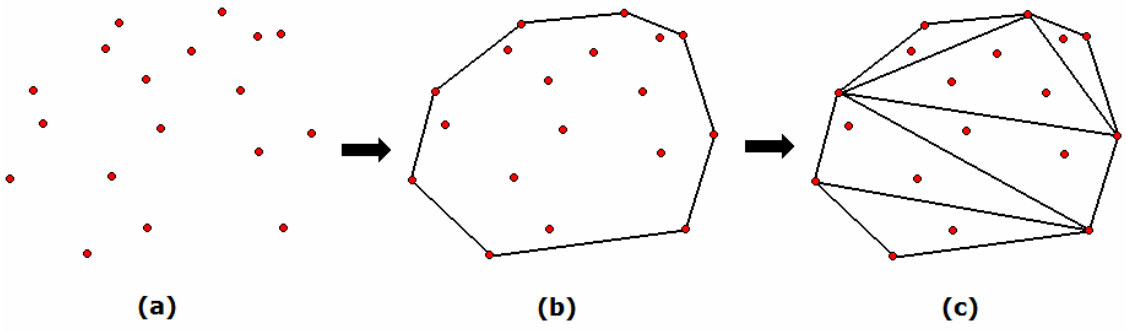


Figura 3.4.1: (a) Nube de puntos original. (b) Cálculo del cierre convexo. (c) Triangulación del convexo

Una vez dividido el interior del cierre en triángulos, iteramos sobre los puntos internos. Vemos que cada uno de estos cae dentro de algún triángulo. La cuestión

es identificar dentro de qué triángulo cae un punto dado. Una vez detectado este triángulo, podemos subdividirlo en otro triángulo y un cuadrilátero, tal como se aprecia en la figura 3.4.2a, uniendo el nuevo punto con dos puntos cualesquiera del triángulo original (normalmente los más cercanos).
Si un nuevo punto cayera dentro de un cuadrilátero resultante de una división previa, habría que subdividir este en dos cuadriláteros (teniendo en cuenta que ambos han de ser pseudo-triángulos). Hay dos formas de realizar esta subdivisión, dependiendo de si el punto cae por encima o por debajo de la proyección de las aristas incidentes en el vértice réflex. En el primer caso se une el punto con los vértices adyacentes al réflex (fig b). En el segundo, se une el punto al vértice réflex y su opuesto (fig. c).

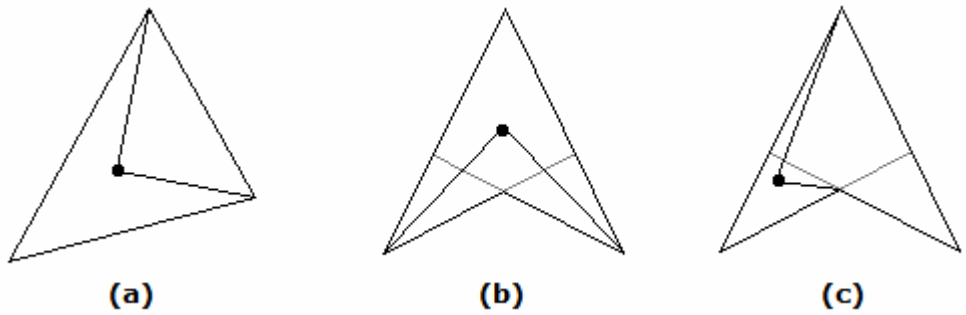


Figura 3.4.2: (a) División de un triángulo. (b) División de un cuadrilátero, caso 1. (c) División de un cuadrilátero, caso 2.

Este procedimiento iterativo de subdivisión nos irá transformando la triangulación inicial en una pseudo-triangulación en la que, por construcción, todas las caras tendrán grado 4 a lo sumo.

Es interesante hacer notar que la triangulación de partida influirá en la forma del resultado final. Una triangulación por abanico tenderá a concentrar gran cantidad de aristas incidentes en el punto radial, produciendo una PT bastante confusa. Por esta razón, y para dotar al usuario del mayor grado de libertad posible, la aplicación permite elegir la triangulación inicial sobre la que ejecutar el algoritmo.

3.4.1.2 Algoritmo

Como se puede apreciar en la explicación conceptual del apartado anterior, la técnica de construcción de la PT es extremadamente sencilla. Partiendo de la nube original hacemos un par de operaciones previas y acto seguido iteramos en el vector de puntos. Veamos como queda resumido el algoritmo en una secuencia de pasos:

```
1) Obtener el cierre convexo de la nube de puntos
   cierre ← CierreConvexo(nube)
2) Triangular aleatoriamente el cierre convexo
   pt ← Triangular(cierre)
3) Iteración en el vector de puntos
   - si el punto pertenece al cierre, ignorarlo
   - si el punto es interno, dividir la cara en la que cae
            cara ← EnQueCara(pt, punto)
            Dividir(pt, cara, punto)
```

Aunque conceptualmente el algoritmo es sencillo, algunas de las operaciones que realiza no lo son tanto. La obtención del cierre convexo ya se ha discutido en métodos anteriores y no reviste especial dificultad. La triangulación aleatoria del convexo es igualmente sencilla si bien puede complicarse al pretender ser capaces de alcanzar cualquier posible combinación. Es la localización del punto en una división del espacio la que nos plantea los mayores problemas. Pasamos a examinar las dos últimas operaciones citadas.

3.4.1.3 Triangulación del convexo

Triangular un polígono convexo es relativamente sencillo. Se puede usar un algoritmo como el corte de orejas o el voraz para llegar rápidamente a una solución. Sin embargo, ser capaces de generar todas y cada una de las triangulaciones posibles de un convexo dado para poder elegir una de entre ellas es una tarea difícil. Sobre todo porque el número posible de combinaciones crece desmesuradamente conforme aumenta el número de puntos del convexo.

La enumeración de triangulaciones viene asociada al orden lexicográfico de los pares de puntos que definen las aristas de cada triangulación particular. Implementar un mecanismo para esto se hace bastante costoso y excede los propósitos de la aplicación, sobre todo teniendo en cuenta que esta permite trabajar con una cantidad apreciable de puntos. No tiene mucho sentido pretender recorrer todo el espacio posible de triangulaciones para elegir una cuando este se compone de trillones de ellas o incluso más. Es por eso que la aplicación introduce un mecanismo para poder generar cualquier triangulación pero sin ningún control sobre la propia generación. Es decir, aunque en teoría podemos alcanzar el espacio completo no podemos transitar ordenadamente de una a otra, ni elegir una a voluntad introduciendo su orden lexicográfico.

El método que hemos usado para la generación aleatoria es muy simple y se resume en lo siguiente: partimos de un polígono regular con el mismo número de vértices que el cierre convexo. Realizamos la triangulación por abanico uniendo cada punto del polígono con el punto superior. Y a partir de aquí aplicamos un número aleatorio de flips de aristas en aristas aleatorias. Este número, a pesar de ser aleatorio, es proporcional al número de vértices, de forma que a mayor número de aristas en la triangulación, mayor número de flips se realizarán.
La operación de generación aleatoria se puede repetir tantas veces como queramos pudiendo alcanzar, en teoría, todo el espacio de soluciones.

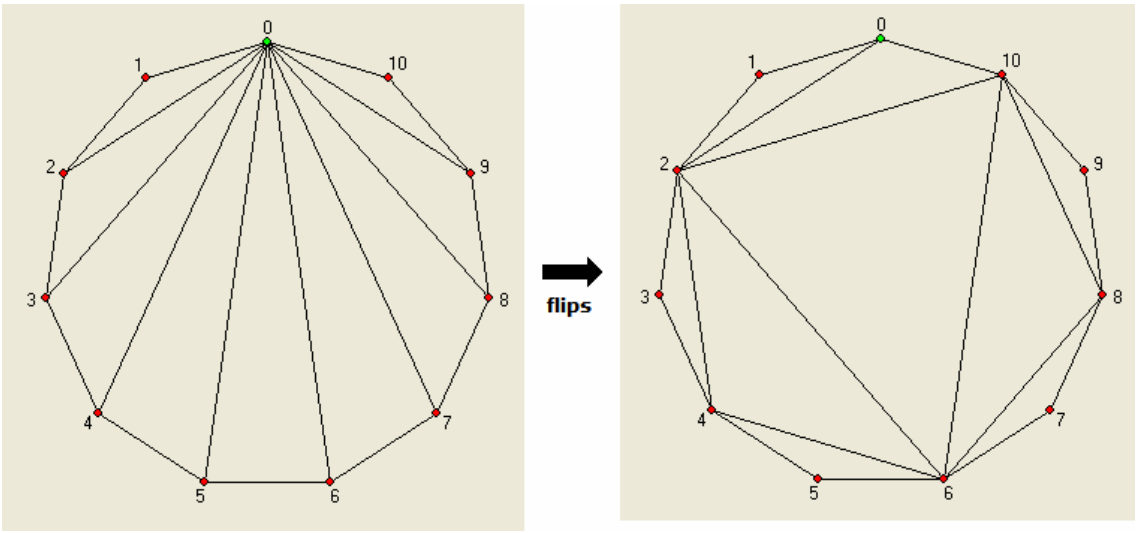


Figura 3.4.3: Triangulación aleatoria mediante flips partiendo del abanico

Para el ajuste fino, por así decirlo, también se ha programado una operación que efectúa un solo flip en una arista concreta, siguiendo un orden predeterminado. Esta segunda operación no nos permite alcanzar todo el espacio de soluciones a base de aplicarla repetidamente, pero sí nos sirve para movernos en las inmediaciones de una triangulación concreta, cuyas características nos interesen, para realizar pequeños cambios.
La triangulación efectuada sobre el polígono se traslada al convexo de la nube de puntos sin mayor problema, puesto que el cierre tiene sus puntos ordenados en sentido antihorario.

3.4.1.4 Localización del punto en la triangulación

Hay varios algoritmos establecidos para la localización de un punto en una división del plano. Por ejemplo, el de Kirkpatrick, que involucra una jerarquía de triangulaciones mantenida por un árbol. Aunque la complejidad ofrecida es buena,

el esfuerzo de implementación quizá resulta excesivo para lo que se pretende con la aplicación, por lo que en su desarrollo se ha optado por utilizar un método diferente. Se trata de una variante del *ray-tracing* o proyección de rayos, que funciona muy bien con una estructura como el DCEL y nos ahorra el tener que definir clases adicionales. Resumidamente, consiste en lo siguiente:

Tenemos un punto *p* del cual queremos averiguar su localización en alguna de las caras de la pseudo-triangulación. Igualmente, esta se encuentra contenida en un DCEL con algunas caras ya formadas.
Escogemos un punto cualquiera de alguna de las caras, al cual llamaremos *ancla*.
Entonces trazamos un rayo desde el ancla hasta el punto *p* que atravesará sucesivas caras y aristas en la trayectoria. Para empezar, sabemos en qué cara se encuentra el ancla, y qué aristas la componen. Por lo tanto podemos calcular qué arista atraviesa el rayo. Al saber la arista, sabemos su gemela y por tanto la cara adyacente a la que acabamos de atravesar. En esta nueva cara, comprobamos todas sus aristas para ver cuál de ellas interseca con el rayo y es la más próxima al punto final. Obteniendo su gemela pasamos a una nueva cara, y así vamos aproximándonos al punto final *p* atravesando sucesivas caras intermedias. Detenemos el proceso cuando detectemos que el punto p está dentro de la cara actual (necesitamos definir una función que nos lo indique).

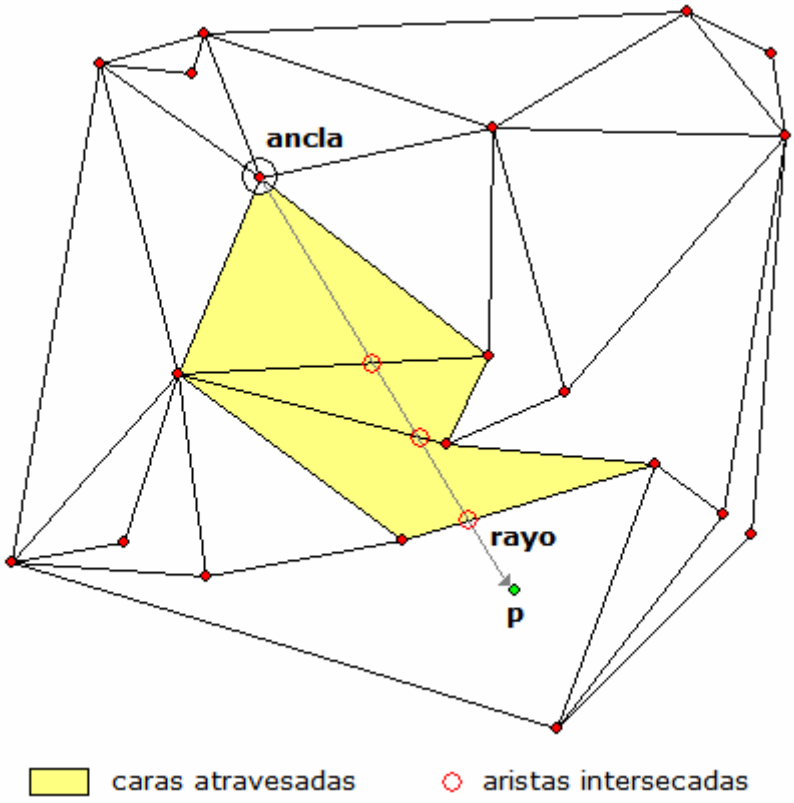


Figura 3.4.4: Localización de un punto p en la pseudo-triangulación

3.4.1.5 Ejemplo con la aplicación

Las PT producidas por este método tienden a generar muchas líneas de unión con los vértices del cierre convexo, dando lugar a representaciones visuales bastante abigarradas. Esto es especialmente notable cuando el número de puntos de la nube es grande en comparación con el de los puntos situados en el cierre. Por tanto, en vez de probar con una distribución aleatoria (que suele dar lugar a estas situaciones) vamos a configurar manualmente la nube de forma que tenga bastantes puntos situados en la periferia.

De esta manera, con el botón de *insertar puntos* seleccionado, pinchamos a nuestro antojo en la pizarra (superficie de dibujo), hasta conseguir una distribución de este tenor:

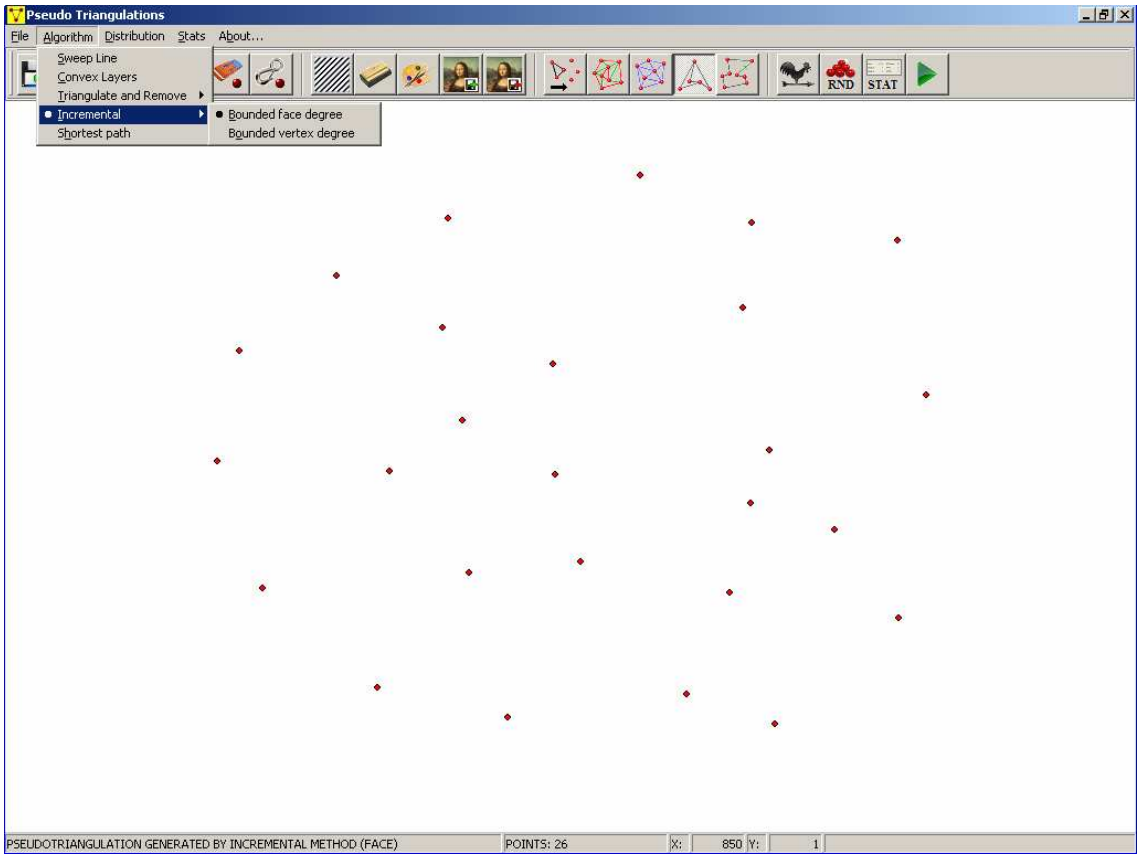


Figura 3.4.5: Nube de puntos para cálculo de PT incremental por caras

Seleccionamos la opción caras de grado acotado desde la barra de menú (opción Algorithm -> Incremental -> Bounded face degree). Con el botón de método incremental pulsado (cuarto de los botones de algoritmos) hacemos click en el

botón de ejecución. La aplicación computará el cierre convexo y nos pedirá que definamos su triangulación inicial antes de pasar a calcular la PT.

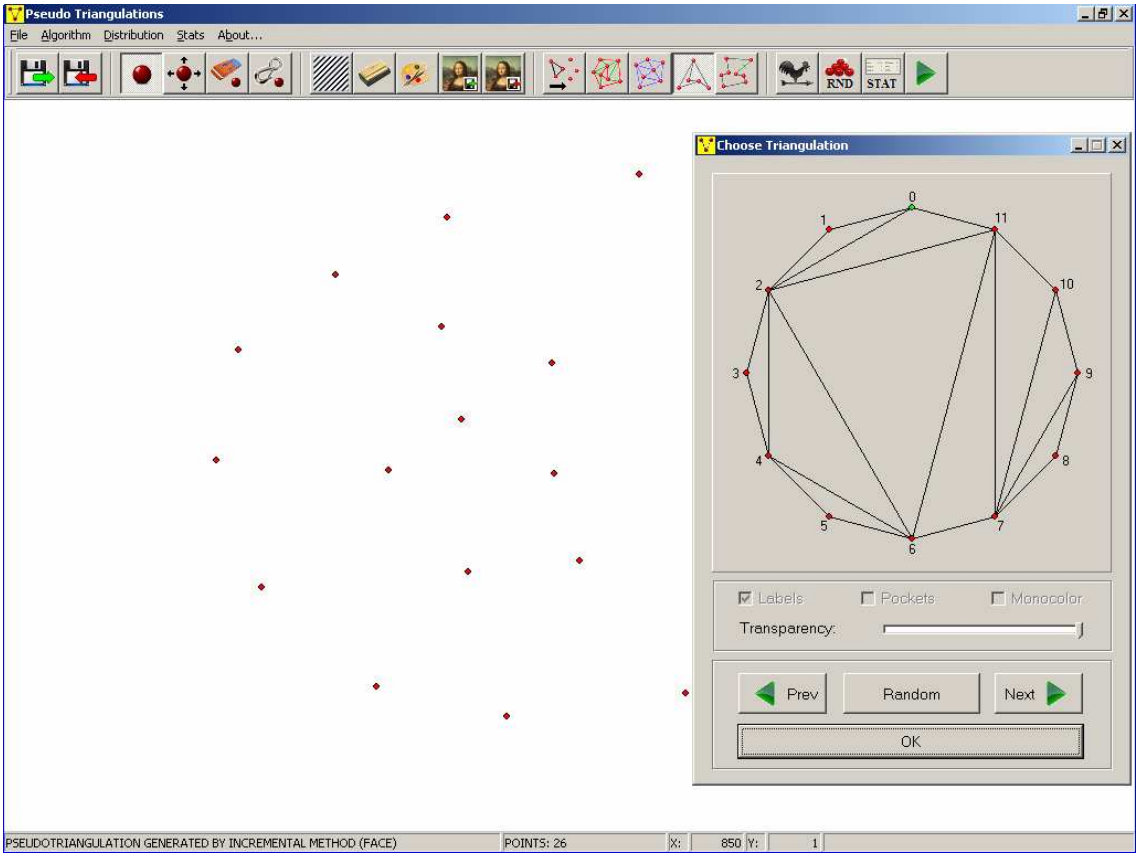


Figura 3.4.6: Selección de la triangulación inicial del cierre convexo

En esta ventana interpuesta podemos usar el botón *Random* cuantas veces consideremos necesario hasta dar con una triangulación a nuestro gusto, o bien usar los botones *Prev* y *Next* para hacer pequeños cambios en la triangulación actual. Una vez escogida, pulsamos el botón OK y el programa dibujará la PT resultante (fig. 3.4.7)

Obsérvese como los puntos del cierre concentran gran cantidad de aristas (su grado es muy alto). Es una característica de este método de pseudo-triangulación. Logramos acotar el grado de las caras en detrimento del grado de los vértices. El aspecto de las PT es también característico, pudiendo distinguirse sin dificultad del de las PT generadas por otros métodos.

Una vez dibujada la PT podemos volver a probar la misma nube de puntos con una triangulación diferente. Basta con volver a pulsar el botón de ejecución y la pantalla de selección volverá a aparecer, con la triangulación actual por defecto.

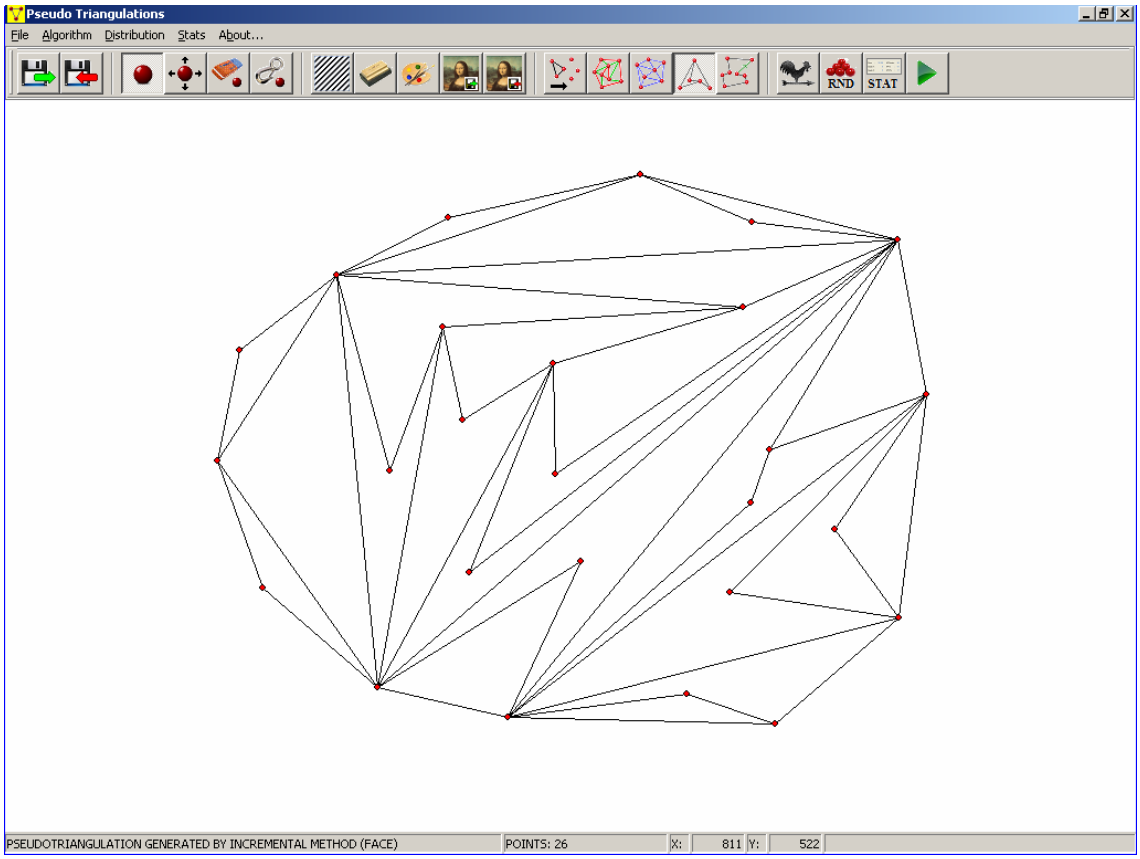


Figura 3.4.7: Resultado de la PT por el método de caras de grado acotado

3.4.1.6 Complejidad

La PT puede construirse en O(n logn).

No obstante, como paso previo hay que calcular el cierre convexo, que puede elevar la complejidad total del método dependiendo del algoritmo utilizado. Usando Jarvis, el peor tiempo posible sería cuadrático, pero en la práctica el cierre convexo nunca va a contener todos los puntos de la nube, por lo que podemos concluir que la complejidad total se aproxima a O(n logn)

3.4.1.7 Estructuras necesarias

Para guardar la estructura de la pseudo-triangulación

Como ya explicamos en apartados anteriores, el algoritmo de localización de un punto en la división del espacio requiere de una estructura a la que se le puedan realizar preguntas tales como cuáles son las aristas de una cara, cuál es la gemela de una arista dada, etc. La estructura DCEL nos viene como anillo al dedo para este

particular, y puesto que ya la tenemos implementada para ser usada con otros métodos, parece la opción lógica.

Para guardar la estructura de la triangulación

Puesto que la PT se construirá a partir de la triangulación inicial, tiene todo el sentido del mundo usar la misma estructura para ambas cosas. Y puesto que usamos un DCEL para contener la PT, usamos otro para contener la triangulación. El DCEL de la ventana de selección se traslada al de la ventana principal y a partir de este operamos. La clase DCEL contiene métodos para insertar nuevas aristas, dividir caras y todas las operaciones necesarias en la implementación de este algoritmo.

3.4.2 Vértices de grado acotado

Este método consigue reducir el máximo grado de los vértices (número de aristas que confluyen en un vértice) a 5, mediante un procedimiento recursivo que va operando sobre conjuntos de puntos cada vez más pequeños hasta llegar a un caso base. Veamos el concepto.

3.4.2.1 Concepto

Denotamos por *P* al polígono resultante de calcular el cierre convexo de la nube de puntos. Y sea *B(P)* el conjunto de puntos pertenecientes al borde del polígono. En cada paso del algoritmo aplicamos una de dos operaciones posibles sobre *P* para obtener polígonos de menor tamaño. Estas operaciones son:

- Partición
  Se usa un pseudo-triángulo para partir *P* en dos polígonos convexos P_1 y P_2, que comparten exactamente un punto *p* en *B(P)* y pueden ser separados por una línea vertical que pase por *p* (figura 3.4.8b).

- Poda
  Con esta operación se recorta un punto de *B(P)*, formándose un pseudo-triángulo y un polígono convexo *P'* más pequeño (figura 3.4.8c)

¿Cómo sabemos cuándo aplicar una operación u otra y sobre qué punto realizarla? Consideremos el problema: puesto que ningún vértice en la pseudo-triangulación

final puede tener un grado mayor que 5, las aristas incidentes en los puntos de *B(P)* nos limitan la elección de aristas para el interior de *P*. Por tanto debemos encontrar un mecanismo para llevar la cuenta del número de aristas incidentes en cada punto y que nos permita discernir sobre qué puntos realizar las operaciones.

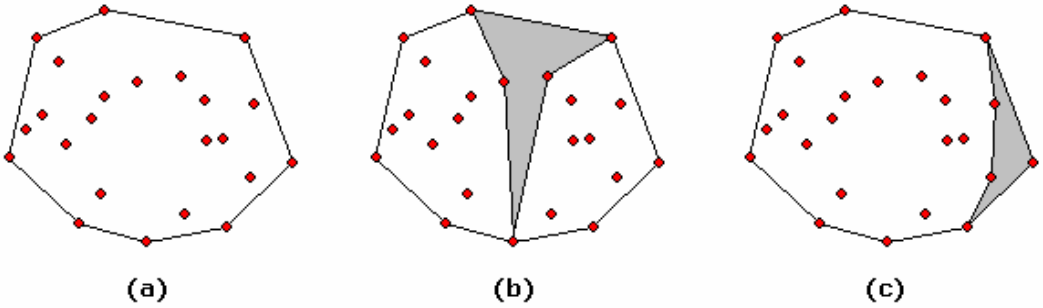


Figura 3.4.8: (a) Cierre convexo de la nube. (b) Partición. (c) Poda

Introducimos pues una magnitud llamada *carga* de un punto *p*, que denotaremos por c*(p)*, y que equivale al grado de *p* menos 2 (es decir, al número de aristas que inciden sobre el punto le restamos 2). Así, un punto de grado 4 tendrá carga 2 y un punto de carga 2 tendrá carga 0. La carga de un punto no es una cantidad estática sino que va cambiando según la pseudo-triangulación va construyéndose. Por tanto, cuando nos referimos a la carga de un punto, nos referimos siempre a su carga en la PT actual.
Para completar la definición, llamaremos *carga del polígono P* a la suma de las cargas de los puntos del borde (o cierre convexo).

$$c(x) = \text{grado}(x) - 2$$

$$c(P) = \sum_{x \in B(P)} c(x)$$

Veamos ahora como esta magnitud nos va guiando en el proceso de elección.
Es preciso, a fin de poder ejecutar los pasos recursivos que nos permiten ir reduciendo el tamaño del problema, mantener en todo momento un par de *invariantes*, condiciones que han de cumplirse a la fuerza y que no podemos transgredir. Se enuncian así:

- **Invariante 1**: Para todo punto *x* del borde de *P* se cumple que $c(x) \leq 3$. Más aún, a lo sumo hay un punto *p* del borde con $c(p) = 3$.

- **Invariante 2**: si la carga de todo punto $p \in B(P) \leq 2$, entonces $c(P) \leq 5$. En otro caso, $c(P) \leq 6$.

Es fácil ver que estos invariantes se cumplen en la situación inicial, cuando *P* es el cierre convexo de la nube. Entonces todos los puntos del cierre tienen carga 0.
Vamos a ver primero cómo se realizan las operaciones anteriormente citadas y cuando seleccionar cada una. En el proceso de explicarlas veremos que su aplicación no altera los invariantes.

Criterio de selección:

- Si la carga de todos los puntos de B(P) es 0 ó 1, se elige PARTICIÓN
- Si hay algún punto de B(P) con carga 2 ó 3, se elige PODA

PARTICION:

Asumamos que tenemos un polígono *P* que satisface ambos invariantes y cuyos puntos del borde tienen como mucho carga 1. Esto implica que *P* contiene como máximo 5 puntos de carga 1 en el borde (por el invariante 2).

Si *P* tiene 5 vértices de carga 1 en el borde, entonces seleccionamos el punto *mediana* (es decir, aquel cuya coordenada x sea la mediana de las 5). Si tuviera menos de 5, elegimos cualquier punto *p* que no sea extremo del borde y cuidando de que a lo máximo nos queden 2 vértices de carga 1 a cada extremo de una línea vertical trazada por *p*. A efectos de simplicidad trataremos *p* como un punto de carga 1.

Ahora dividimos *P* en dos polígonos convexos P_1 y P_2 formados a partir de los puntos que quedan a izquierda y a derecha de la línea trazada sobre el punto *p*. Ambos polígonos contienen a *P*, por lo que su carga aumenta de 1 a 3.

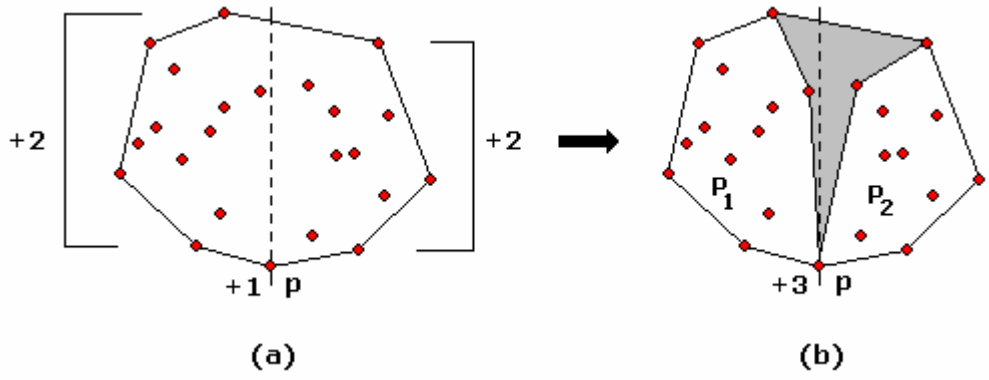


Figura 3.4.9: (a) Selección del punto p. La carga del punto es 1. A cada lado de la vertical la carga máxima solo puede ser 2.
(b) Partición de P en dos polígonos menores. La carga de p pasa a ser 3.

El polígono P_1 contiene exactamente un punto de carga 3 (el punto p). Esto satisface el invariante 1 e implica que para cumplir el invariante 4, $c(P_1)$ debe ser menor o igual a 6. Esto es así, puesto que además de p, $P1$ contiene como mucho 2 puntos más de carga 1 en su borde y el cierre convexo solo contribuye con una carga adicional de 1. Todo esto se aplicaría lo mismo al caso de P_2.

PODA:

Asumamos que tenemos un polígono P que satisface ambos invariantes y que tiene al menos un punto de carga 2 ó 3. Sea p_{max} el punto del borde con la carga más alta. Denotamos P' al polígono resultante de podar p_{max} (haciendo uso de los puntos interiores para calcular el nuevo convexo). Nótese que el pseudo-triángulo resultante de la poda no tiene puntos en su interior.

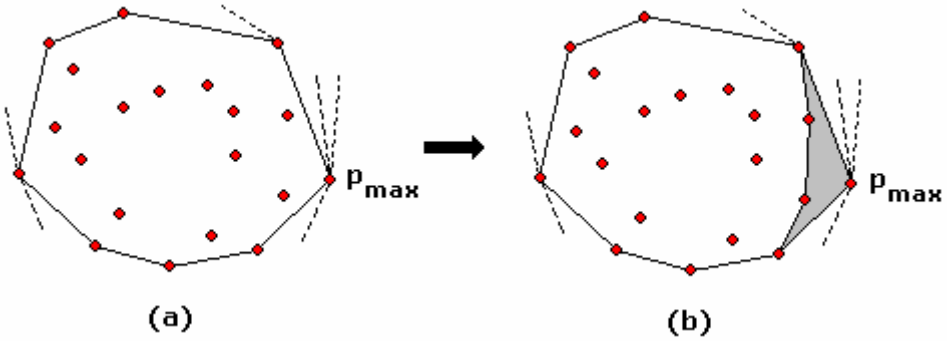


Figura 3.4.10: (a) Selección del punto de máxima carga. (b) Poda.

La carga de P' es $c(P') = c(P) - c(p_{max}) + 2 \leq c(P)$, puesto que las nuevas aristas del convexo suman 2 a la carga de P', pero p_{max} es un punto de carga al menos 2. Específicamente:

Si $c(p_{max}) = 3$ entonces $c(P') = c(P) - 3 + 2 \leq 6 - 3 + 2 \leq 5$

Si $c(p_{max}) = 2$ entonces $c(P') = c(P) - 2 + 2 \leq 5 - 2 + 2 \leq 5$

Es decir, en cualquier caso $c(P') \leq 5$, lo que satisface el invariante 2.

Si $c(p_{max}) = 3$ entonces había como mucho un punto de carga 2 en $B(P)$ y todos los demás puntos debían tener carga 1 o menos. Esto implica que solo uno de los vecinos de p_{max} en el borde tiene carga 2 y por tanto las aristas del nuevo cierre crean a lo máximo un punto de carga 3.

Si $c(p_{max}) = 2$ entonces p_{max} no tiene por qué ser único, pero el invariante 2 implica que en este caso hay como mucho dos puntos de carga 2. La poda elimina uno de ellos e incrementa la carga de los vecinos, dejando a lo sumo un punto de carga 3.

Como hemos podido ver, tanto en la situación de partida como en la aplicación de las operaciones, se mantienen los invariantes, por lo que la carga de cada punto será siempre ≤ 3 y su grado ≤ 5.

3.4.2.2 Algoritmo

Hasta aquí la explicación del método. La implementación está sujeta a variaciones, sobre todo en la forma de guardar los sucesivos convexos y de controlar la recursividad. Puesto que los lenguajes imperativos no están bien dotados para el uso de funciones recursivas, hemos convertido el proceso a un esquema iterativo.

Básicamente definimos las funciones de partición y poda con una serie de parámetros que nos indican entre qué límites horizontales va a ejecutarse la operación, en qué cara (convexo) y sobre qué punto podamos/partimos. Otro parámetro necesario es un vector que contiene la carga actual de cada punto.
Para sortear la recursividad y trasladarla a un esquema iterativo usamos una pila en la que iremos introduciendo las sucesivas caras o convexos (con puntos en su interior) que aún quedan por procesar. Cuando la pila quede vacía, habremos terminado. Podemos describir el procedimiento con el siguiente pseudocódigo:

```
1. Obtener cierre convexo
   cierre ← cierreConvexo(nube)
2. Introducir en la pila la primera cara (la cara inicial o polígono)
   cara ← 0
   Push(pila, cara)
3. Iteración en la pila. Mientras pila no vacía:
   Pop(pila, cara)
   si cara es un triangulo sin puntos interiores, saltar a la siguiente iteración
   pmc ← punto de máxima carga en la cara
   maxcarga ← carga(pmc)
   - si maxcarga = 0 | 1 entonces
         Poda(pmc, cara, cierre …)
         Push(pila, cara)
   - si maxcarga = 2 | 3 entonces
         pmc ← verticeMediana(cara)
         Partición(pmc, cara, cierre …)
         Push(pila, cara+1)
         Push(pila, cara+2)
```

3.4.2.3 Ejemplo con la aplicación

Vamos a probar con una distribución aleatoria normal de 20 puntos. Usamos un valor de desviación típica elevado para dispersar los puntos y que se vea mejor la estructura de la PT. Con el fin de que los pseudo-triángulos se vean claramente podemos modificar la nube generada con el botón de mover puntos.

Desde la barra de menú seleccionamos el algoritmo incremental y en el submenú que se abre elegimos la variante vértice de grado acotado (Bounded vertex degree).

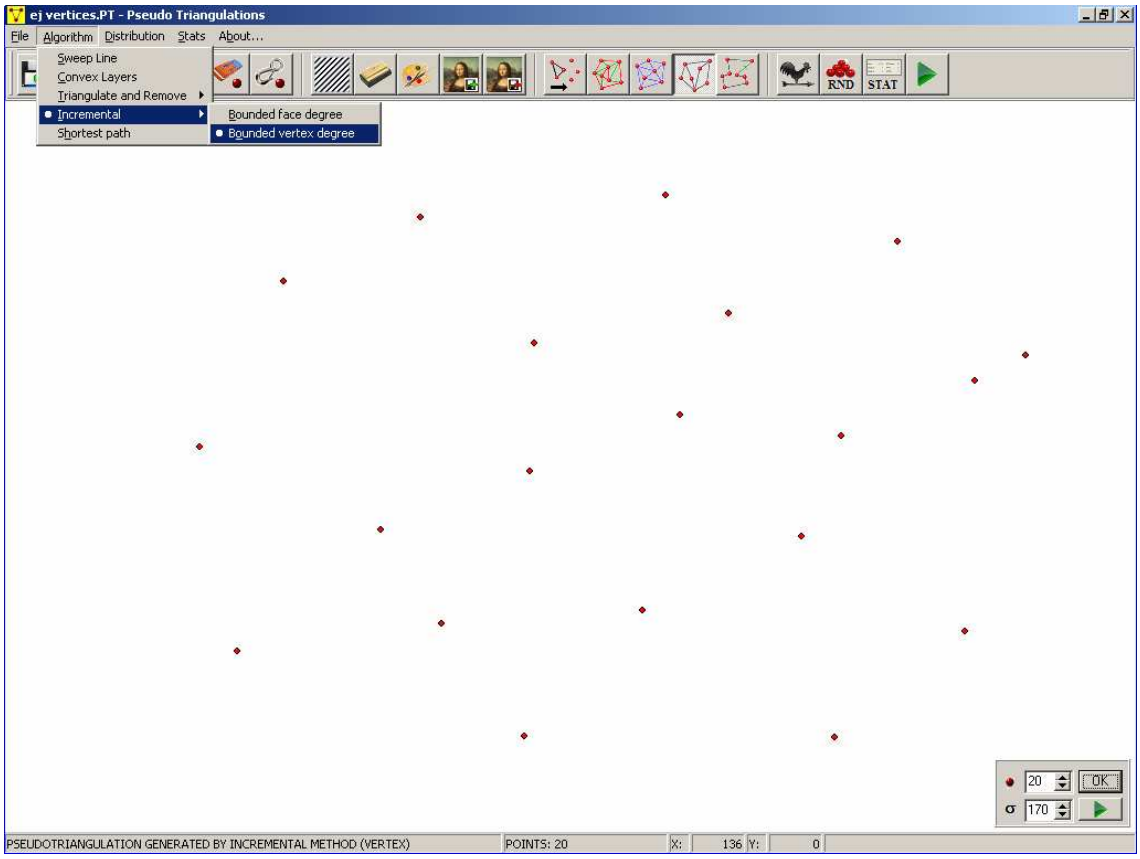


Figura 3.4.11: Generación de una nube para el cálculo de la PT por el método incremental en su variante de vértices de grado acotado.

Este método, tal como está implementado, no otorga al usuario ningún grado de libertad una vez que se ha definido la nube. El único factor que puede variar el resultado de la PT es la elección del punto sobre el que trazar la vertical en las operaciones de partición, pero de esto se hace cargo el propio algoritmo. Siempre intenta coger el punto mediana, y si el número de puntos elegibles fuera par, escoge aleatoriamente uno de los dos puntos mediana.

Con la nube definida y el botón de método incremental seleccionado, pulsamos el botón de ejecución. Y he aquí el resultado:

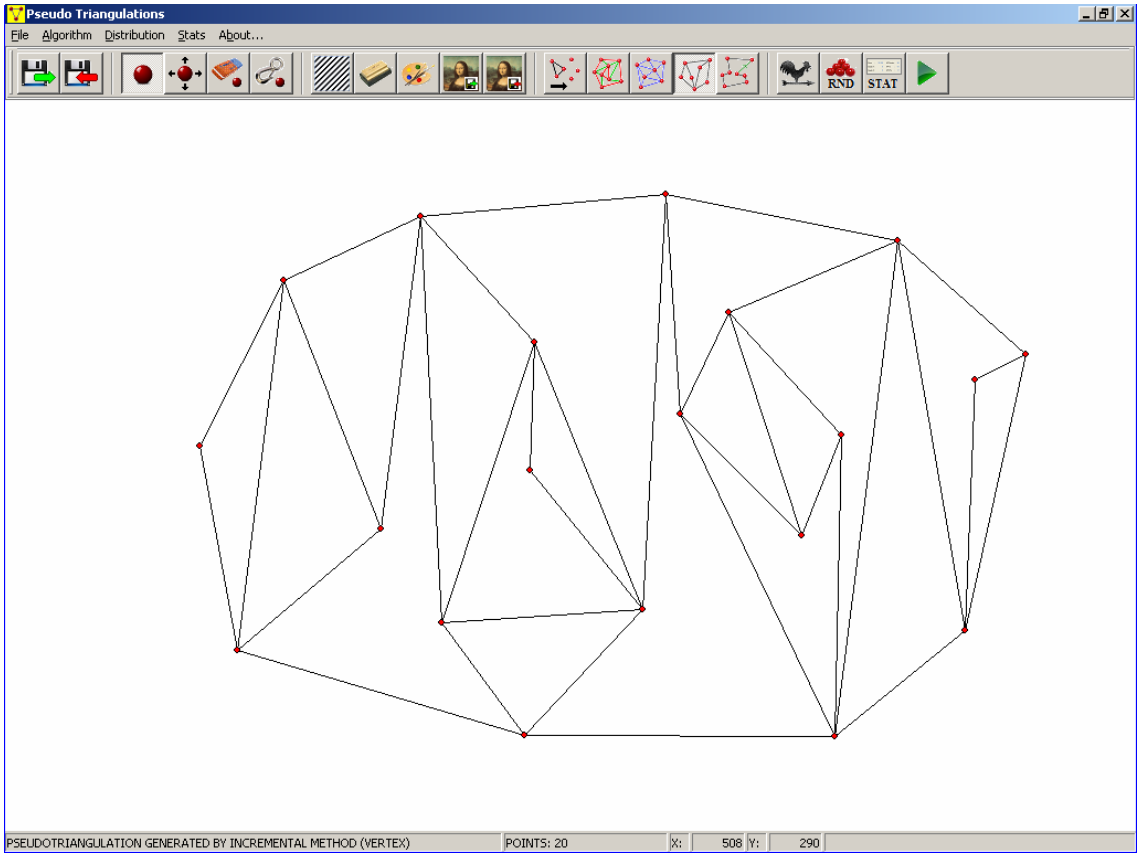


Figura 3.4.12: PT generada por el método incremental en su variante de vértices

3.4.2.4 Complejidad

A la hora de implementar el procedimiento descrito, nos sería útil mantener los sucesivos cierres convexos bajo las operaciones de borrado de un punto y partición por una línea vertical. A estos efectos, puede usarse una estructura como un árbol de convexos (hull tree) que permite realizar ambas operaciones en tiempo logarítmico O(logn) por cada una. Al final puede demostrarse que dado cualquier conjunto *S* de puntos en el plano, una PT mínima de *S* con máximo grado de sus vértices igual a 5 es computable en tiempo O(n logn).

Sin embargo, la mencionada estructura resulta algo compleja de implementar y manejar, por lo que en nuestra aplicación hemos recurrido a un mecanismo más sencillo si bien algo más ineficiente, como es el de la mencionada pila de convexos o caras. Además, puesto que ha de computarse el cierre convexo de la nube, dependemos de nuevo del algoritmo usado para ello de cara al cálculo de la complejidad final.

3.4.2.5 Estructuras necesarias

Para guardar la estructura de la pseudo-triangulación

Una vez más hacemos uso de la clase DCEL. El número de operaciones de consulta e inserción de aristas y caras es tan elevado y hay que hacerlas con tanta frecuencia que se hace del todo necesario contar con una estructura de gran potencia.

Para guardar los polígonos intermedios resultantes de la poda o la partición

Realmente no hemos utilizado más que un objeto de la clase DCEL para guardar toda la información relativa tanto a la PT como a los polígonos. En cada operación de partición o poda el DCEL se acota con un límite izquierdo y un límite derecho de forma que solo se computan los vértices contenidos entre esos límites (los puntos internos al convexo). Esto, unido al conocimiento de la forma del convexo que nos da la correspondiente cara del DCEL nos permite operar justo en la zona requerida: los puntos de la frontera del polígono y de su área interna.

También, y puesto que cada uno de estos polígonos es asimilable a una cara de la estructura DCEL, hemos usado una clase Pila (ya usada en métodos anteriores) para almacenar los índices de las caras que se van generando y que quedan por procesar.

Así, si en el tratamiento de una cara *i* ha de realizarse una operación de partición, el pseudo-triángulo divisor pasará a ser la nueva cara *i* (este no requiere mayor proceso), mientras que los polígonos resultantes a izquierda y a derecha pasan a ser las caras *i+1* e *i+2* y se meten en la pila puesto que pueden contener puntos internos que obliguen a seguir procesando.

Si la operación es de poda, el pseudo-triángulo podado pasa a ser la cara *i+1* (no requiere proceso) mientras que la cara recortada sigue siendo la *i*, que se introduce de nuevo en la pila para su reproceso.

3.5 Caminos mínimos

A diferencia de los métodos vistos hasta ahora, el método de caminos mínimos no se aplica sobre una nube de puntos sino sobre un polígono. Mas exactamente, la nube de puntos inicial se poligoniza formando una figura cerrada cóncava. Los demás algoritmos implicaban el cálculo de un convexo, bien fuera el cierre convexo de la nube y el posterior tratamiento iterativo de los puntos internos, o bien un convexo inicial y su posterior expansión por tratamiento igualmente iterativo del resto de puntos. Aquí todos los puntos van a estar en la frontera de la pseudo-triangulación, no quedando ninguno en el espacio delimitado por el polígono.

3.5.1 Concepto

Puesto que la nube de puntos de poligoniza dando lugar a una figura cerrada, cóncava, sin cruce de aristas ni puntos internos, la única opción que nos queda para pseudo-triangular consiste en unir los puntos del polígono mediante aristas internas ¿Cómo hacer esto y cuántas posibles combinaciones se pueden dar? Para responder a estos interrogantes usaremos una técnica consistente en la construcción desde la triangulación de un convexo. Consideremos el siguiente escenario:

Dado un polígono *P* con *n* vértices, se considera un polígono convexo *P** con el mismo número de vértices y una triangulación cualquiera *T* de *P**. Cada arista *uv* de *T* se convierte en *P* en el camino mínimo (geodésico) entre los vértices *u* y *v* dentro de *P*. Así se consigue una pseudo-triangulación de *P*.
De esta forma, cada triangulación de *P** origina una pseudo-triangulación de *P*, por lo que disponemos de un mecanismo que nos permite obtener una infinidad de PTs sobre el polígono *P*, en concreto tantas como triangulaciones posibles tenga *P**.

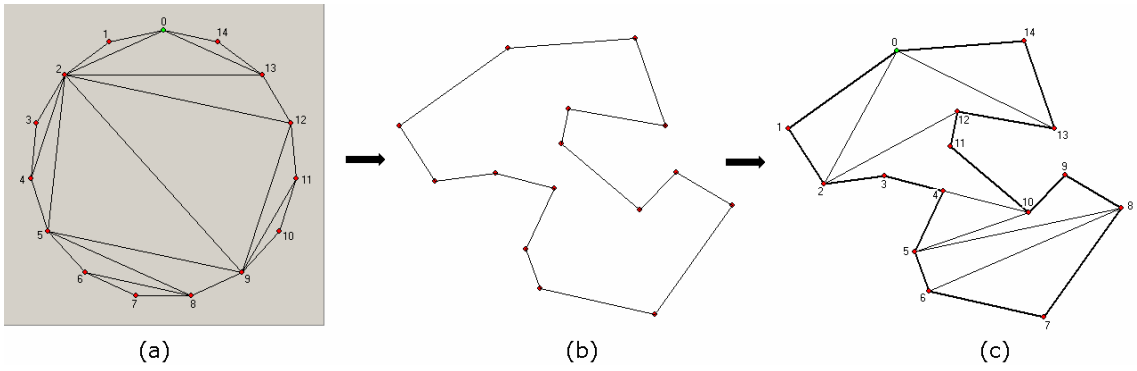


Figura 3.5.1: (a) Triangulación del convexo. (b) Aplicación sobre la nube poligonizada. (c) Pseudo-triangulación obtenida al convertir las aristas del convexo en caminos mínimos en el polígono.

3.5.2 Cálculo de los caminos mínimos

Para calcular los caminos mínimos usaremos una variante del algoritmo de Dijkstra, modificada para adaptarla a nuestras necesidades. La idea tras el mencionado algoritmo consiste en la exploración sistemática de todos los caminos que parten de un vértice A y llegan al resto.

Primeramente se recorren los nodos a los que se puede llegar directamente, etiquetándolos con la distancia desde el nodo origen. Los nodos no alcanzables se etiquetan con valor infinito. Hecho esto se cierra el nodo A y se escoge el nodo más cercano (B) y se procede de la misma manera. Si desde B se llega a un nodo C, la etiqueta de este se actualiza con el mínimo valor entre la etiqueta actual y la suma de la etiqueta de B más la distancia de B a C.

Cuando se recorren todos los nodos accesibles directamente desde un nodo *i*, se cierra y nos colocamos en el nodo abierto de etiqueta menor, repitiéndose el proceso anterior hasta haber explorado todos los caminos.

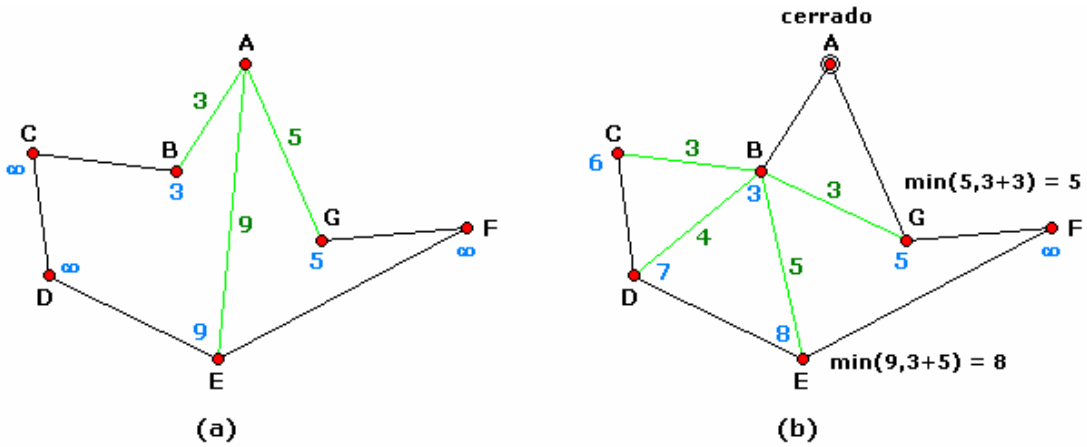


Figura 3.5.2: Algoritmo de Dijkstra. (a) Exploración del nodo origen. (b) Cerrado el nodo origen se explora el siguiente nodo con menor etiqueta

Tras haber cerrado todos los nodos, tenemos una configuración en la que cada nodo tiene la distancia mínima desde el origen. Esto por si solo no nos indica el camino mínimo. Para hallarlo, es necesario retroceder desde el nodo destino saltando a nodos anteriores que difieran exactamente en la longitud del arco conector, hasta llegar al nodo origen.

La implementación del algoritmo usada en nuestra aplicación es un poco diferente, pues se va construyendo el camino a medida que actualizamos la información de los nodos. De forma que al terminar con el proceso, ya tenemos el camino definido.

Para ayudarnos en la tarea, necesitamos una serie de estructuras que guardarán información sobre los nodos y sus relaciones. Veámoslas:

3.5.3 Matriz de visibilidad

Para poder efectuar el etiquetado necesitamos saber, por ejemplo, qué nodos son accesibles directamente desde un nodo determinado. En un grafo esta información nos la proporcionan los arcos. Dos nodos serán adyacentes si existe un arco que los une, de forma que se puede viajar de uno a otro directamente. Como en nuestro caso no trabajamos sobre un grafo sino sobre un polígono llamaremos *vértices* a los nodos y *visibilidad* a la adyacencia. Por tanto, necesitamos calcular previamente la *matriz de visibilidad* que nos indique, por cada vértice, qué otros vértices son *visibles* desde él.

La composición de esta matriz será muy sencilla: si un vértice *j* es visible desde un vértice *i*, entonces el elemento *mv[i,j]* = 1, en caso contrario, *mv[i,j]* = 0. Puesto que la visibilidad es una propiedad simétrica, también lo será la matriz.

Esta matriz nos servirá para el cálculo de los caminos mínimos desde cualquier vértice, ya que la visibilidad entre vértices es una relación fija que no cambia con el vértice escogido, permaneciendo inmutable mientras no cambie la forma del polígono. Por tanto, se calcula una sola vez al inicio del algoritmo.

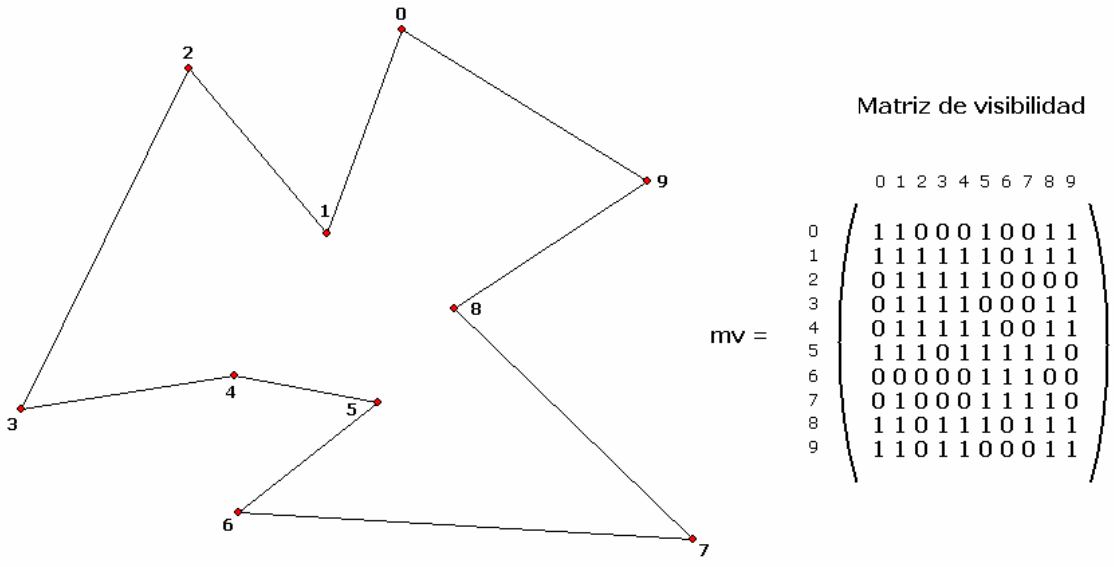


Figura 3.5.2: Matriz de visibilidad de los vértices del polígono

Como podemos observar en la matriz, el vértice 7 no es visible desde el vértice 3, puesto que *mv[3,7]* = 0, pero el vértice 9 sí que lo es, puesto que *mv[3,9]* = 1.

3.5.4 Matriz de distancias

Una segunda información que necesitamos es la distancia desde cada vértice a todos los demás, a fin de poder comparar las etiquetas. Podemos colocar esta información en otra matriz llamada *matriz de distancias*. Esta segunda matriz puede parecer superflua, pero en nuestro algoritmo, tal como lo hemos implementado, servirá a dos propósitos: en la situación inicial nos indicará la distancia desde un vértice *i* hasta un vértice *j* (o bien infinito si no son visibles entre si). Pero en las sucesivas iteraciones del algoritmo, irá almacenando las distancias mínimas calculadas desde un punto cualquiera al resto. Por ejemplo, fijándonos en la figura 3.5.3: en la situación inicial, el vértice 7 no es visible desde el vértice 0, con lo que la matriz albergaría un valor infinito en la posición *md[0,7]*. Pero al explorar el vértice 1, como sí tiene visibilidad sobre el vértice 7, se podrá actualizar el valor de *md[0,7]* como *md[0,1] + md[1,7]*.

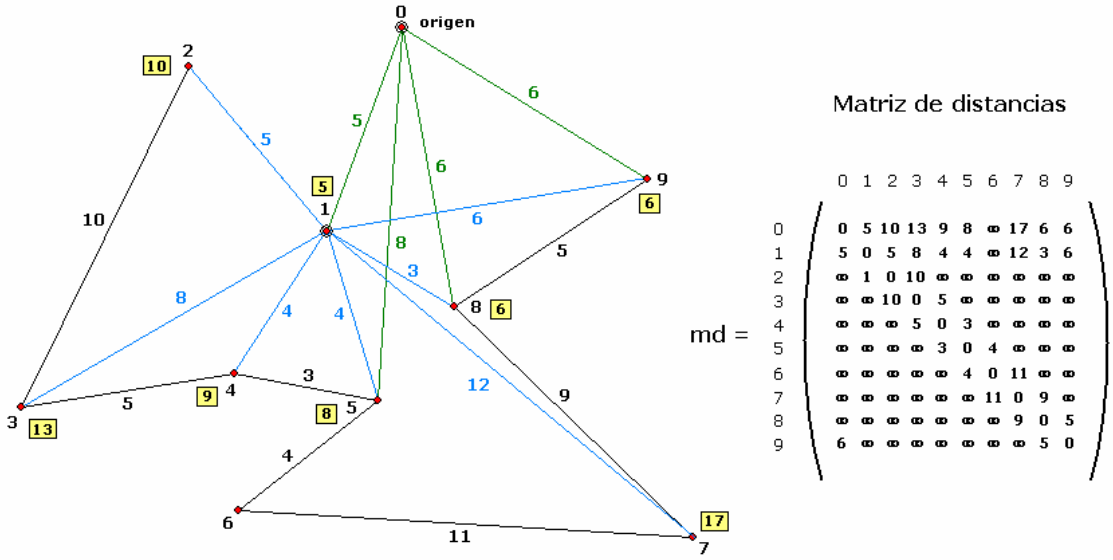


Figura 3.5.3: Matriz de distancias tras la exploración de los vértices 0 y 1

Al igual que en el caso de la matriz de visibilidad, la matriz de distancias se crea al principio del algoritmo, con una información inicial. Pero a diferencia de esta, su contenido se irá modificando con el paso de las iteraciones.

Nos queda definir un tercer elemento que a la postre será el que realmente nos indique el camino mínimo desde un vértice cualquiera al resto de vértices. Así como con la ayuda de matriz de visibilidad creamos la matriz de distancias, con la ayuda de la matriz de distancias crearemos esta última matriz que llamaremos *matriz de trazabilidad*. Esta matriz se irá actualizando junto con la matriz de distancias, cada vez que se encuentre una distancia menor desde el nodo origen a uno destino.

3.5.5 Matriz de trazabilidad

Esta tercera matriz guarda la información de los caminos mínimos en la siguiente forma: si queremos conocer el camino mínimo del vértice *i* al vértice *j* consultamos el elemento *mt[i,j]*. El valor contenido en la celda es el del próximo vértice que debemos recorrer en el camino (llamémosle *k*). Pasamos entonces a consultar el elemento *mt[k,j]* cuyo valor nos enrutará por el siguiente vértice. El camino finalizará cuando *mt[k,j] = k*, lo que nos estará indicando que *j* es visible desde *k*.

Esta matriz se irá formando fila a fila, con cada iteración del bucle principal de vértices. Cuando se llame a la función para calcular los caminos mínimos del vértice *i*, se inicializará la fila *i* de la matriz con los siguientes valores: si los vértices *i*, *j* son visibles entre sí, entonces *mt[i,j] = i*. En caso contrario, *mt[i,j]* = -1.

Durante la ejecución del algoritmo, cada vez que una distancia entre puntos se actualice con un valor menor, la celda correspondiente de la matriz de trazabilidad también se actualizará. Supongamos que se ha encontrado un camino más rápido para ir de *i* a *j* pasando por el vértice *k*. Entonces, si *k* es visible desde *i*, haremos *mt[i,j] = k*. En caso contrario, *mt[i,j] = mt[i,k].*

Es decir:

Si *i* ve a *k*, enrutamos por *k*

Si *i* no ve a *k*, enrutamos por el vértice que enruta desde *i* para llegar a *k*

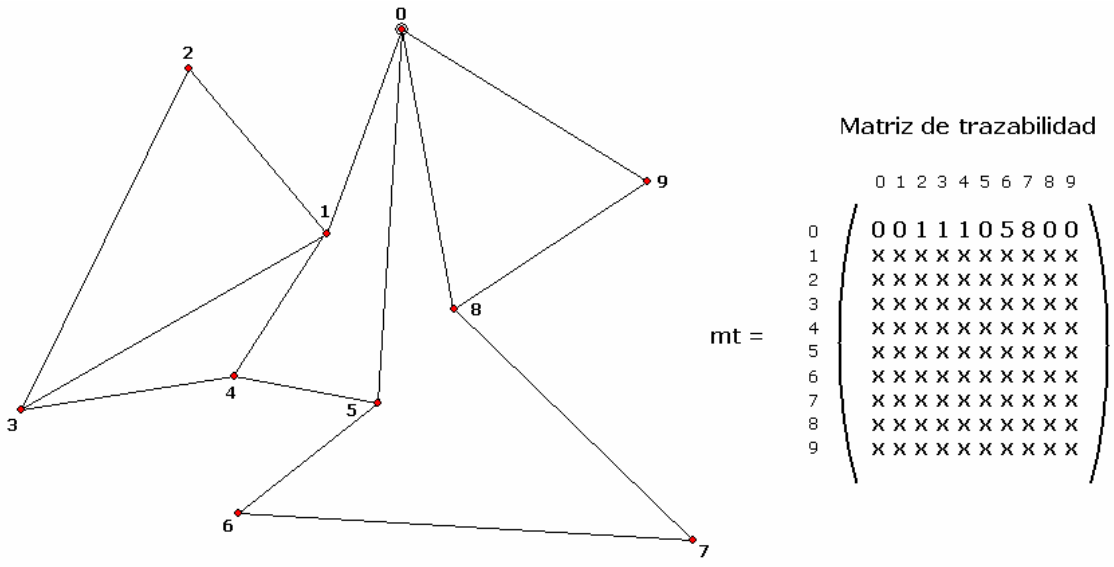


Figura 3.5.4: Matriz de trazabilidad para el vértice 0. Los vértices alcanzables directamente tienen un 0 en la posición mt[0,i]. Los vértices no visibles desde 0 tienen el valor del siguiente vértice en el camino en la posición mt[0,i]. Pej.: mt[0,3] = 1, que es el siguiente punto a 0 en el camino 0->3

Según se deduce de la explicación dada hasta la fecha, en general no se podrá calcular el camino mínimo desde un vértice *i* a un vértice *j* hasta no tener la matriz de trazabilidad completa, pues los elementos de cada fila solo nos enrutan a través del primer vértice en el camino.

La ventaja es que esta matriz solo se calcula una vez (aunque cada fila se computa en una llamada diferente a la función de buscar caminos mínimos). Y una vez calculada, se pasa al procedimiento de transformación que opera sobre la estructura poligonal original dividiéndola mediante la inserción de las aristas correspondientes. A su vez, la estructura modificada se pasa al procedimiento de dibujado que lee cada arista y la dibuja en pantalla. Además, una vez terminado el proceso, tenemos la información de TODOS los caminos mínimos en la memoria, lo que nos puede resultar de utilidad para algunas cosas.

La desventaja es que se requiere mucho almacenamiento en memoria, al contrario que un algoritmo que desalojara la información de los caminos desde cada vértice una vez calculado. Pero dado que la cantidad de vértices con que vamos a trabajar es en cualquier caso pequeña, eso no supondrá ningún problema serio.

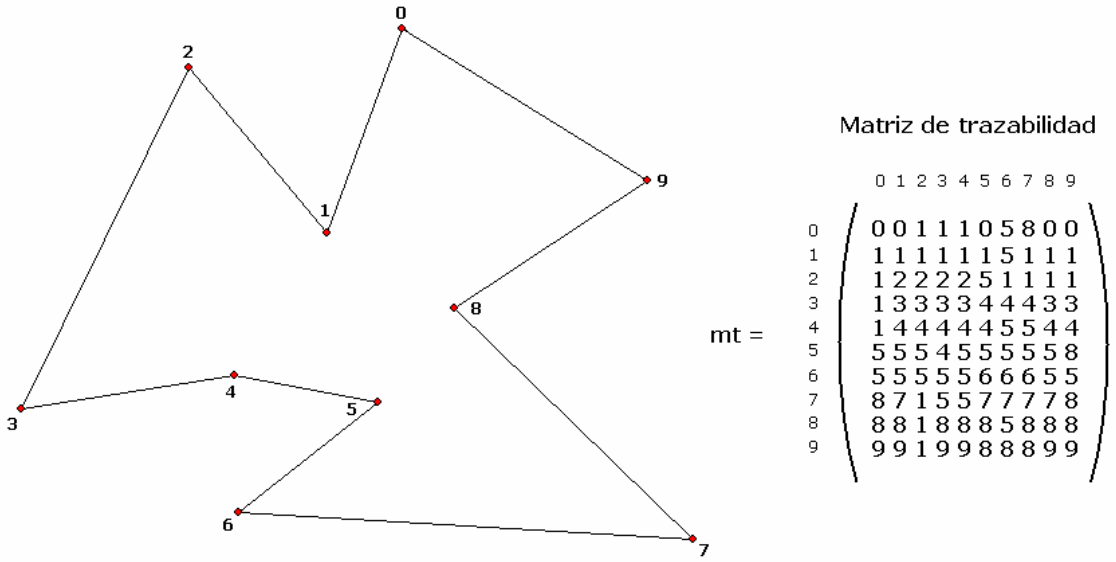


Figura 3.5.5: Matriz de trazabilidad completa
Ej.: Camíno mínimo (7,3) → mt[7,3] = 5; mt[5,3] = 4; mt[4,3] = 4
Hemos llegado al final. Camino mínimo (7,3) = 7-5-4-3

Obsérvese en la matriz de la figura cómo cuando un vértice *j* es visible desde *i*, *mt[i,j]* = *i*. Es decir, para ir de *i* a *j* se enruta por si mísmo, no por el punto final. Esto es fundamental para detectar la condición de parada. Igualmente *mt[i,i]* = *i*, como se puede comprobar examinando la diagonal de la matriz.

3.5.6 Ejemplo con la aplicación

Vamos a ver el funcionamiento del algoritmo con un polígono de varias decenas de puntos que tenga varias concavidades, de forma que los caminos mínimos no sean triviales.

Nos colocamos en el modo de definición de polígono (pulsando el botón de engarzar o bien el del algoritmo de caminos mínimos) y vamos insertando puntos en la pizarra. Cada vez que insertamos uno se une mediante una arista con el anterior. Es importante señalar que el polígono debe ser simple, cerrado y sin cruce de aristas (la aplicación detectará inserciones ilegales y las rechazará). Es decir, no conviene insertar los puntos al azar o pronto nos quedaremos sin opciones. Para formar una figura, es aconsejable insertar el primer punto y seguir definiendo el contorno manteniendo un sentido de giro. Es irrelevante el sentido elegido, aunque una vez cerrado el polígono, la aplicación renumerará los puntos para que sigan un orden antihorario.

Para nuestro ejemplo, hemos elegido una forma similar a una figurilla humana.

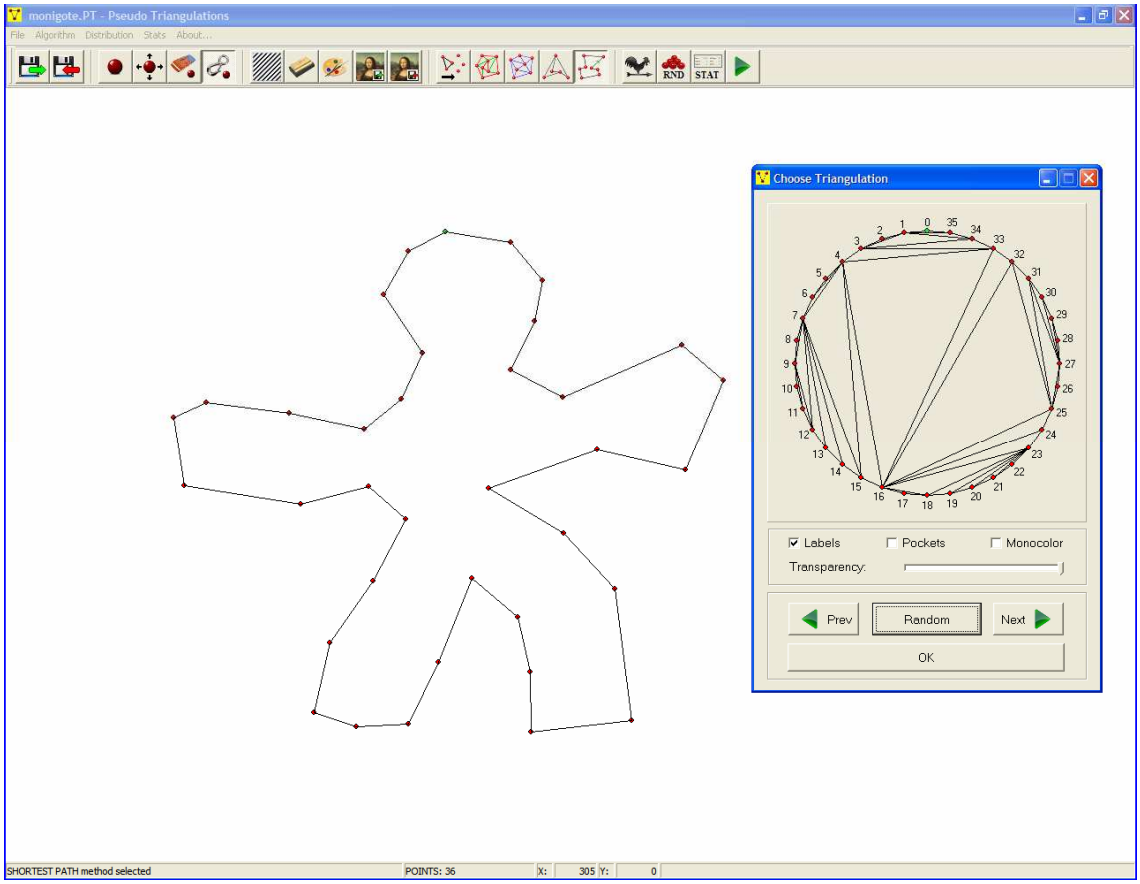


Figura 3.5.4: Definición del polígono y elección de la triangulación del convexo

Una vez definido el polígono, pulsamos el botón de ejecución. La aplicación nos mostrará entonces la pantalla con el convexo y nos permitirá que modifiquemos la triangulación a nuestro antojo. Esta triangulación será la que se traslade al polígono en forma de caminos mínimos. Con el botón *Random* podemos ir generando triangulaciones aleatorias, y con los botones *Prev* y *Next* hacer pequeños ajustes arista a arista. Cuando encontremos una a nuestro gusto, pulsamos el botón OK para generar la PT. Con las casillas de verificación podemos indicar el modo en que queremos visualizar el resultado. Vamos a ir viendo todas las opciones posibles:

Con la casilla *etiquetas (labels)* marcada, la representación es la siguiente: los puntos del polígono se etiquetan con números, el contorno del polígono se dibuja con grosor extra para ayudar a diferenciarlo (sobre todo en formas complejas) y el interior se pseudo-triangula con aristas de grosor normal (el grosor y el color de las líneas se puede configurar desde la barra de botones, ver manual de usuario).

Con la triangulación del convexo al lado podemos ir comprobando arista por arista y ver que la traslación se ha hecho correctamente.

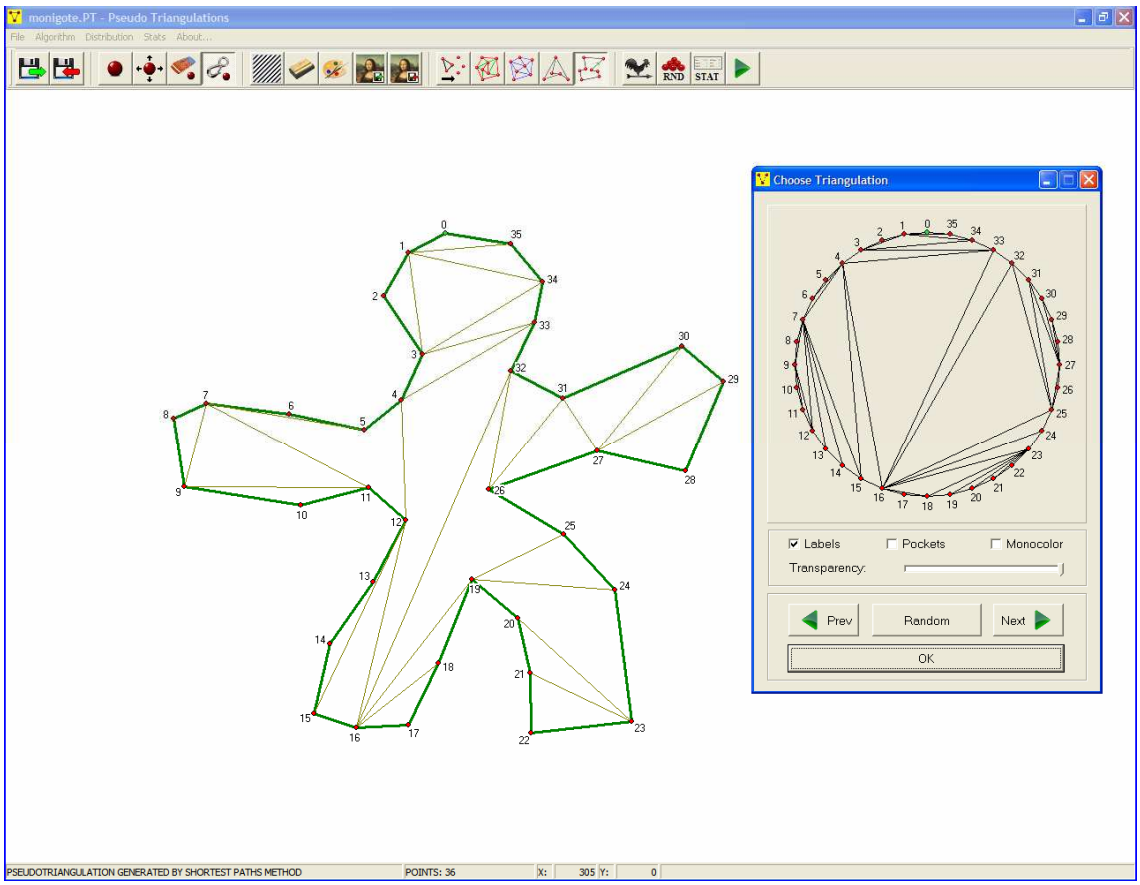


Figura 3.5.5: Pseudo-triangulación del polígono y etiquetado de puntos

La siguiente opción contempla el rellenado de las concavidades del polígono, también llamadas *bosillos*. Si marcamos la casilla *pockets* se calculará una PT aleatoria para cada uno de estos bolsillos. Las aristas se dibujan en un color distinto para diferenciarlas claramente del resto de la PT. Conviene desmarcar la casilla *labels* para hacer desaparecer los números. Una vez que ya hemos comprobado que los caminos se han calculado correctamente no son necesarios y nos quitan visibilidad.

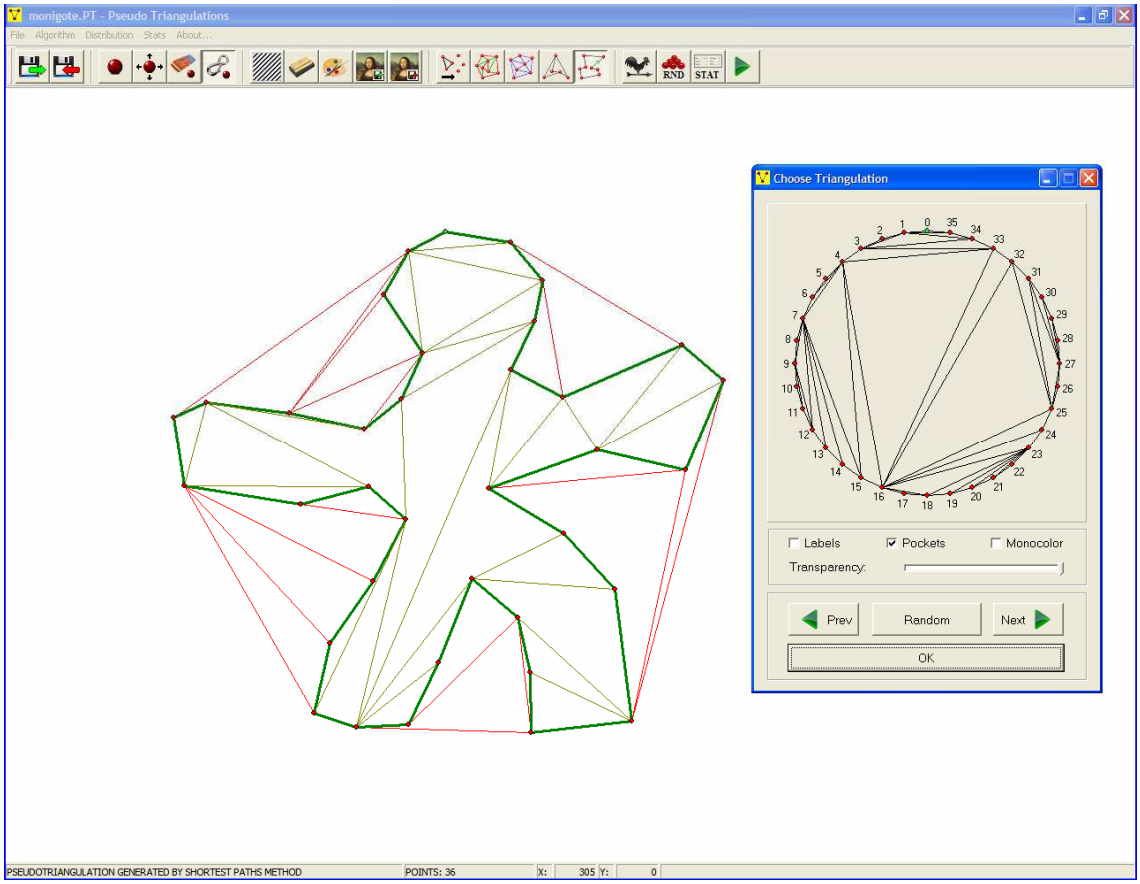


Figura 3.5.6: Pseudo-triangulación de los bolsillos

Como se puede apreciar en la figura, la forma original queda encerrada en una envoltura convexa que le da un aspecto similar al de las PTs generadas por el resto de métodos. Sin embargo, tal como está, la PT es un batiburrillo de líneas de diferentes colores y grosores. Quizá deseamos, una vez que hemos visto el proceso completo de generación del resultado, tener una vista "plana" de la PT. Es decir, tan solo el resultado final, sin pistas que nos indiquen de donde ha salido cada pseudo-triángulo. Esto lo conseguimos marcando la tercera y última casilla, etiquetada *monocolor*. Con esta opción el procedimiento de dibujado iguala el color y el grosor de todas las líneas, de forma que ya no queda rastro del polígono original.

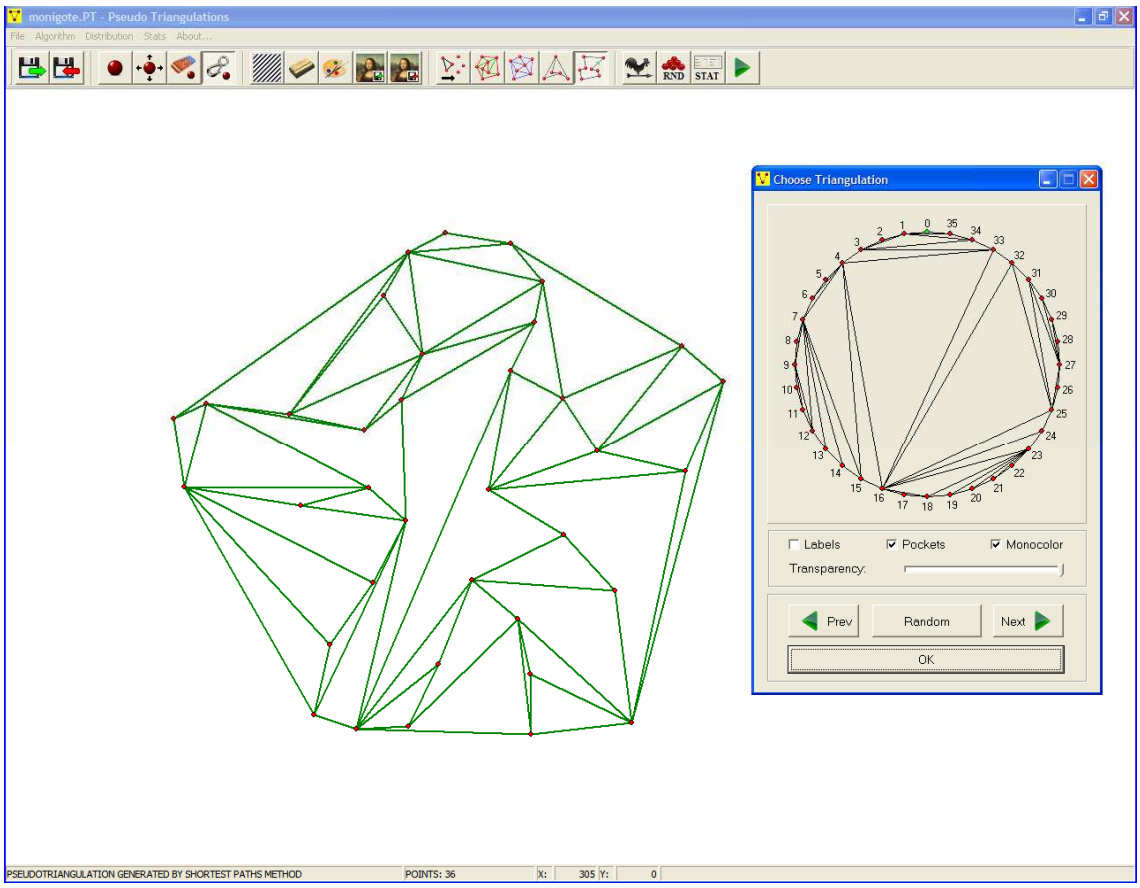


Figura 3.5.7: Redibujado de la PT en color y grosor únicos

Si la ventana de elección de triangulación nos molestara o impidiera la visión podemos hacer varias cosas: una es cerrarla. En cualquier momento se puede volver a invocar pulsando el botón de ejecución. La ventana reaparecerá con la triangulación actual y nos permitirá cambiarla para probar con la nueva. Otra opción es mover la ventana a un lugar donde no nos moleste. Incluso podemos hacerla medio desaparecer arrastrándola al borde inferior o lateral siempre que dejemos visible un trozo de la barra superior para poder arrastrarla de nuevo. La tercera opción consiste en ajustar el valor de transparencia de la ventana al mínimo, de forma que se volverá semitransparente y nos permitirá ver a través de ella.

Por último, cabe decir que este método de pseudo-triangulación no produce formas características, y es imposible a primera vista saber con qué método se ha calculado. No obstante, y puesto que ya conocemos los perfiles del resto de PTs explicadas, podemos deducirlo por eliminación.

3.5.7 Algoritmo

El algoritmo completo utilizado para la PT por caminos mínimos es un poco complicado. Intentaremos sintetizarlo en el siguiente pseudo-código. Omitimos la explicación del cálculo de las matrices de visibilidad y distancias, suponiendo que hay dos funciones que nos las devuelven y nos centramos más bien en el cálculo de los caminos en sí:

```
1. mv ← CalcularMatrizVisibilidad(poligono)
2. md ← CalcularMatrizDistancias(mv, poligono)
3. for i = 0 to numVertices – 1
       IniciaFila(mt, i)
       cerrados ← [i]
       repetir
           vme ← VerticeMenorEtiqueta(md, cerrados)
           for j = 0 to numVertices -1
               d ← min ( md[i,j]  ,  md[i,vme] + md[vme,j] )
               si d < md[i,j] entonces
                   md[i,j] ← d
                   si mt[i,vme] = i entonces
                       mt[i,j] ← vme
                   en caso contrario
                       mt[i,j] ← mt[i,vme]
           cerrados ← cerrados + [vme]
       hasta cerrados = [0..numVertices -1]
4. Por cada arista (i,j) en la triangulación,
   trazar el camino mínimo i → j en el polígono
```

Es decir, para cada vértice *i*:

- Se establece un bucle en el que se busca el nodo abierto con la etiqueta de menor valor (o lo que es lo mismo, el vértice más cercano a *i* aún no explorado).
- Entonces para cada vértice *j* se compara la distancia actual de *i* a *j* con la distancia de *i* a *j* pasando por el *vme* (vértice más cercano).
- Si la distancia pasando por el *vme* es menor que la actual, se actualiza con ese valor y a su vez se modifica la matriz de trazabilidad:
- Si *i* ve a *vme*, se enruta por *vme*
  En caso contrario, se enruta por el vértice que enruta para ir de *i* a *vme*.

3.5.8 Complejidad

Este es claramente el método de mayor complejidad de todos los contemplados.
La triangulación del convexo es un problema ya visto anteriormente y puedo realizarse manteniendo la complejidad baja, siendo $O(n^2)$ en el peor de los casos.
El verdadero problema es el cálculo de los caminos mínimos: hay que calcular n árboles de caminos mínimos (uno por cada vértice), lo que representa un total de n^2 caminos. Cada camino cuesta O(log n) en el peor caso.
Por lo tanto, la complejidad se va a $O(n^2 \log n)$. Bastante más que un algoritmo sencillo de barrido, por ejemplo.

El cálculo de alguno de los elementos usados en nuestra implementación particular también resulta muy elevado: con algoritmos refinados, la matriz de visibilidad puede calcularse en tiempos bastante bajos, pero el método sencillo usado por nuestra aplicación (comprobar si la recta (*i*,*j*) que une dos vértices interseca con alguna de las aristas del polígono) lo eleva hasta $O(n^2 \log n)$. Por fortuna, esta operación es secuencial con el cálculo de los caminos mínimos, por lo que no incrementa la complejidad total.

3.5.9 Estructuras necesarias

Para guardar la estructura de la triangulación y la pseudo-triangulación:

De nuevo hacemos uso de la clase DCEL y sus facilidades para inserción de aristas, división de caras y consultas de todo tipo. Dado el intenso trabajo de transformación del polígono original es impensable utilizar estructuras más ligeras como en el caso del barrido por capas.
En este caso haremos uso de dos objetos DCEL, uno para el convexo que se triangulará aleatoriamente y otro para el polígono que se modificará con los caminos mínimos.

Para guardar las matrices de visibilidad, distancias y trazabilidad:

Las tres son matrices $n \times n$, siendo n el número de vértices del polígono original. La diferencia estriba en el tipo del elemento que alojan. La matriz de visibilidad contiene 0 ó 1, elementos que se pueden asimilar a un tipo binario. La matriz de distancias contiene distancias entre vértices, que en general serán números reales.
La matriz de trazabilidad contiene números de vértice (índices), por lo que se pueden asimilar a números enteros.

Dado que *n* puede ser cualquier número y se desconoce en tiempo de compilación, la definición de las matrices debe quedar como vectores dinámicos o vectores abiertos.

Por tanto, la declaración de los tipos sería similar a la siguiente:

TMatrizVisibilidad = array of array of bolean;

TMatrizDistancias = array of array of real;

TMatrizTrazabilidad = array of array of integer;

4. Estructura del programa

La aplicación se ha realizado en Object Pascal, usando el entorno de desarrollo de Borland Delphi 6.0. Está compuesta por 15 ficheros de extensión *.pas* o unidades de carga entre las que se reparte todo el código, un fichero *.dpr* con la información del proyecto y tres ficheros de extensión *.dfm* con la interfaz gráfica de usuario.

La compilación genera un único fichero ejecutable llamado *pseudotri.exe* que corre en cualquier máquina con sistema operativo Windows.

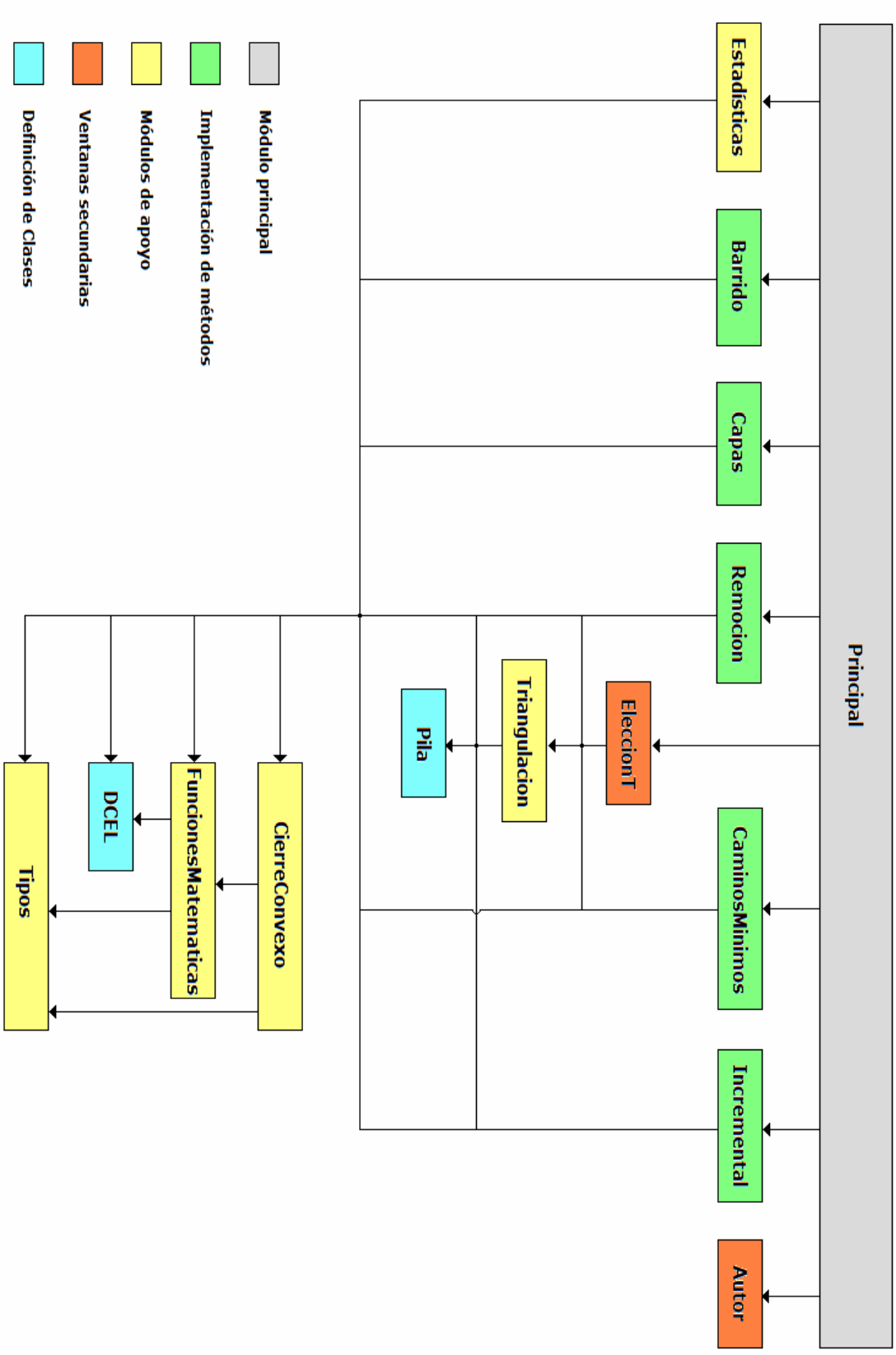
Delphi es un lenguaje RAD de desarrollo rápido especialmente dotado para construir y manejar interfaces gráficos con facilidad. Aunque incorpora el paradigma de programación orientada a objetos, también permite la programación orientada a procedimientos. En este sentido, es más flexible que otros lenguajes OO que nos obligan a tratar exclusivamente con clases.

En el presente trabajo, se ha seguido un esquema mixto. Por un lado se han definido librerías de funciones y procedimientos que operan sobre tipos básicos y algunos tipos definidos por el usuario. Estas funciones no pertenecen a ninguna clase en concreto y pueden usarse desde cualquier módulo que importe la librería. Por otra parte, algunas estructuras necesarias para los algoritmos sí han sido definidas como clases, con sus propiedades y sus métodos públicos. Pueden ser usadas desde cualquier módulo que las importe y declare un objeto de esa clase.

En base a esto, pueden clasificarse los módulos utilizados en distintos tipos, identificados en el esquema de la página siguiente con diferentes colores:

- El módulo principal
- Los módulos con la implementación de los algoritmos de cada método
- Módulos de apoyo con funciones matemáticas y de cálculo de estructuras usados por el módulo principal y los módulos de métodos
- Módulos con la definición de clases
- Módulos con el interfaz gráfico de ventanas secundarias

La relación de dependencia de los módulos se indica mediante flechas. Si una flecha parte desde un módulo hacia otro quiere decir que el primero necesita importar al segundo. Cuando varias líneas confluyen en un nodo quiere decir que todos los módulos de los que parten esas líneas dependen del módulo apuntado por la flecha que sale del nodo. La jerarquía de dependencias se presenta así ordenada de arriba abajo, con los módulos principales en la parte superior, y según descendemos nos encontramos los módulos a los que importan.


Módulo principal
Implementación de métodos
Módulos de apoyo
Ventanas secundarias
Definición de Clases
Principal
Estadísticas
Barrido
Capas
Remocion
EleccionT
Triangulacion
Pila
CaminosMinimos
Incremental
Autor
CierreConvexo
FuncionesMatematicas
DCEL
Tipos

La siguiente relación especifica el contenido de cada módulo, que también se indica en la cabecera del fichero:

Principal.pas	Contiene todo la lógica de control de la pantalla principal, los eventos de ratón y los procedimientos de trazado gráfico.
Barrido.pas	Contiene las funciones necesarias para el cálculo de la PT por barrido.
Capas.pas	Contiene la estructura usada para el almacenamiento del la PT por capas convexas, así como las funciones necesarias para su cálculo.
Remoción.pas	Contiene las funciones necesarias para el cálculo de la PT por remoción.
Incremental.pas	Contiene las funciones necesarias para el cálculo de la PT por el método incremental en sus variantes de caras y vértices de grado acotado, así como funciones para localización de puntos en divisiones del plano.
CaminosMinimos.pas	Contiene las estructuras y las funciones necesarias para el cálculo de los caminos mínimos y los bolsillos de una forma poligonal.
EleccionT.pas	Contiene la lógica de control de la ventana de selección de triangulación, así como las funciones necesarias para generar triangulaciones aleatorias de un polígono regular.
Triangulacion.pas	Contiene funciones para calcular triangulaciones por el método de barrido, abanico y Delaunay.
CierreConvexo.pas	Contiene la definición del tipo abstracto CierreConvexo, usado auxiliarmente en el cálculo de la PT por alguno de los métodos.

DCEL.pas	Contiene la clase abstracta DCEL que permite modelar una PT y guardar toda la información relativa a sus vértices, aristas y caras. Usada en la mayoría de métodos de PT.
Pila.pas	Contiene la clase abstracta Pila, que facilita la creación de objetos de este tipo así como las usuales operaciones de Push y Pop.
FuncionesMatematicas.pas	Contiene diversas funciones de índole algebraica como cálculo de determinantes, sentido de giro, etc.
Estadisticas.pas	Contiene las funciones necesarias para el cálculo de las características de la PT (peso, grado, etc)
Tipos.pas	Contiene varios tipos de usuario comunes usados por el resto de los módulos.
Autor.pas	Contiene la definición de la ventana con el nombre del autor y la versión de la aplicación.
Principal.dfm	Contiene la definición de los elementos gráficos de la ventana principal (interfaz de usuario).
EleccionT.dfm	Contiene la definición de los elementos gráficos de la ventana de Selección de Triangulación.
Autor.dfm	Contiene la definición de los elementos gráficos de la ventana del autor.
PseudoTri.dpr	Contiene el fichero de proyecto con la inicialización de la aplicación, la creación de las ventanas de usuario y la orden de ejecución.

4.1. Funciones implementadas por cada módulo

PRINCIPAL.PAS

Manejadores de eventos

…

Procedimientos gráficos

- PizarraNueva
- DibujaPunto
- DibujaPolígono
- DibujaRejilla
- DibujaPT
- DibujaDCEL
- CopiaPizarra
- PegaPizarra
- PintaFlecha
- LimpiaPanelStat
- Refresca

Operaciones sobre estructuras

- ReordenarPuntos
- EliminaPunto
- NormalizaNube

Numeros Aleatorios

- DistribuciónUniforme
- DistribuciónNormal

Algoritmos

- PTBarrido
- PTPorCapas
- PTRemocion
- PTIncremental
- PTCaminos

Estadísticas

- CalculaEstadisticas

BARRIDO.PAS

Funciones de cálculo

CalculaBarrido

CAPAS.PAS

Funciones de cálculo

CalculaCapas

CalculaInterCapas

Funciones de apoyo

Indice

NumCapas

PuntosCapa

REMOCION.PAS

Funciones de cálculo

RemoverAristas

RemoverTriangulos

Funciones de apoyo

EsReflex

EsRemovible

EsUnTriangulo

YaInsertada

INCREMENTAL.PAS

Funciones de cálculo (caras)

PTriangulaTriangulo

PTriangulaCuadrilatero

Funciones de apoyo (caras)

DetectaReflexCuadrilatero

HayAlineacion

TrasladaDCEL

Funciones de cálculo (vértices)

ProcesaConvexo
PTriangulaConvexo
ParticionCierre
PodaCierre

Funciones de apoyo (vértices)

ConvexoVacio
LimitesDelConvexo
VerticeMediana

Funciones comunes

EnQueCara
EstaDentro
EsPuntoDelCierre
PerturbaPunto

CAMINOSMINIMOS.PAS

Funciones de cálculo

CalculaMatrizVisibilidad
CalculaMatrizDistancias
CalculaMatrizTrazabilidad
TriangulacionACaminosMinimos
TrataBolsillos
PTBolsillo
EncajaBolsillo

Funciones de apoyo

DetectaCara

ELECCIONT.PAS

Manejadores de eventos

…

Procedimientos gráficos

LimpiaImagen

DibujaDCEL

DibujaPoligonoRegular

Funciones de cálculo

ObtienePuntos

RandomFlip

TRIANGULACION.PAS

Funciones de cálculo

TriangulacionPorBarrido

TriangulacionEnAbanico

TriangulacionDeDelaunay

Funciones de apoyo

EsDelaunayLocal

PerturbaPunto

CIERRECONVEXO.PAS

Funciones de cálculo

CierreConvexoJarvis

CierreSecuencial

ConvierteJarvisASecuencial

Operaciones

InsertaInicio

InsertaFinal

InsertaDespues

InsertaEntre

BorraInicio

BorraEnmedio

EliminaLista

Siguiente

Anterior

DCEL.PAS

Getters y Setters

...

Métodos generales

Create
InicializaDcel
CopiaDcel

Métodos de vértices

InsertaVertice
OrigenDeArista
FinalDeArista

Métodos de aristas

InsertaArista
EliminaArista
Anterior
Posterior
Gemela
AristaDeVertice
AristaDeCara
AristaDeVerticeEnCara
CambiaGemela
CambiaCara
EsDiagonal
Flip

Métodos de caras

InsertaCara
EliminaCara
EsCaraExterna
CaraDeArista
PonAristaGuia
DivideCara

PILA.PAS

Métodos

- Create
- Push
- Pop
- Vacia

FUNCIONESMATEMATICAS.PAS

Funciones de cálculo (segementos)

- Intersecan
- PosicionRelativa
- EstanAlMismoLado
- LongitudSegmento
- PuntoDeCorte
- EsRectaSoporte

Funciones de cálculo (determinantes y productos)

- Determinante
- SignoDet3
- SignoDet4
- ProductoVectorialPlano

Funciones de cálculo (puntos)

- PuntoMinimaOrdenada
- PuntoMaximaOrdenada

Funciones de cálculo (ángulos)

- PuntoMinimoAngulo
- PuntoMaximoAngulo
- CalculaAngulo
- PuntoMayorAngulo

Funciones de cálculo (giros)

- SentidoDeGiro

Funciones de cálculo (areas)

- AreaPoligono

ESTADISTICAS.PAS

Funciones de cálculo

Peso
GradoVertices
GradoCaras
EstadCaras

Podemos representar las dependencias entre los módulos con un diagrama de clases. Con el prefijo '-' aparecerán los atributos privados usados por las funciones del módulo y con el prefijo '+' los métodos públicos visibles y utilizables desde módulos externos.

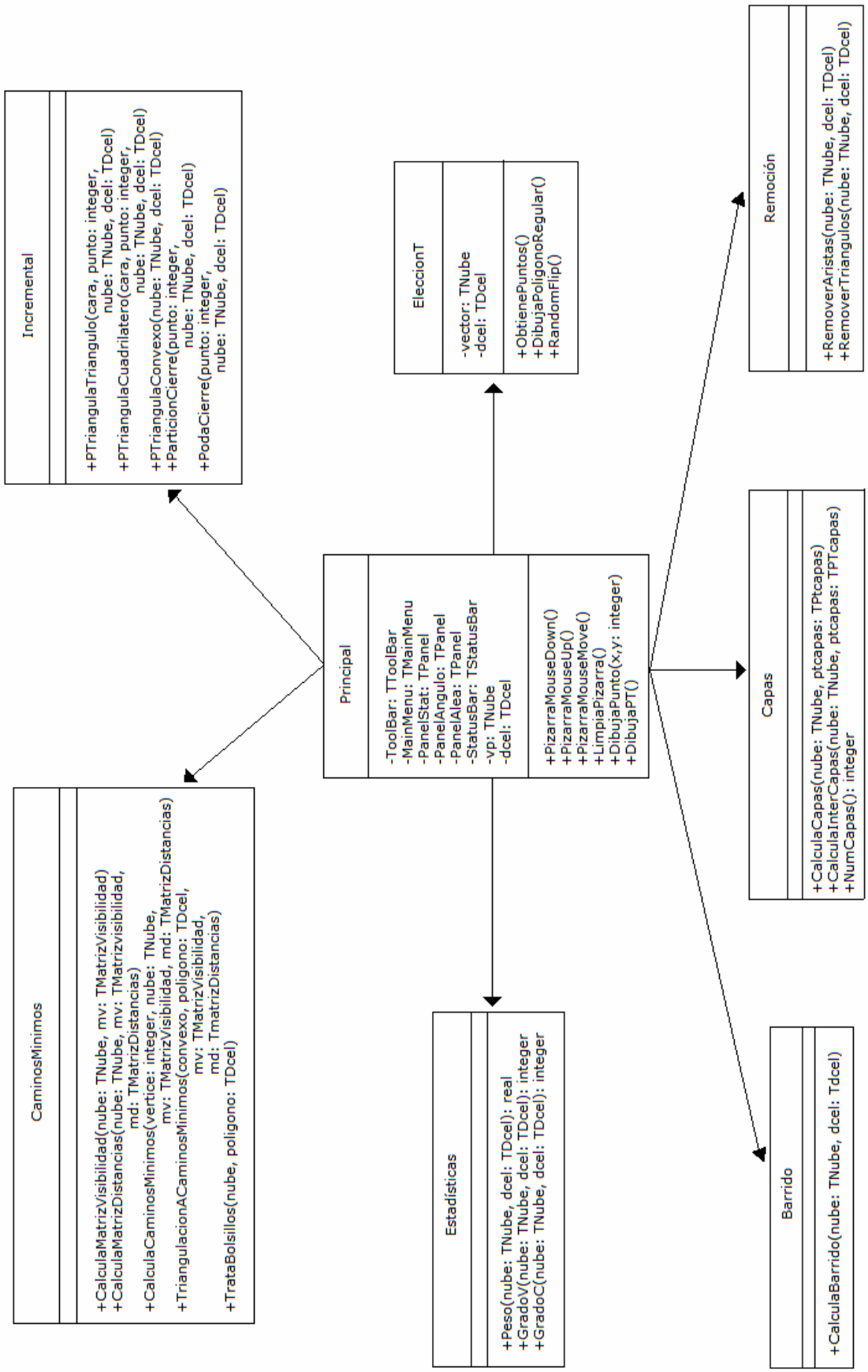
Hay que tener en cuenta que, a diferencia de Java, Pascal no obliga a una función Main() en el módulo principal. Este, junto con los módulos de ventanas secundarias, son todos clases del tipo formulario, por lo que cuentan entre sus miembros públicos con los elementos visuales y los manejadores de eventos. Por ello, en la caja asociada al módulo principal aparecen funciones distintas a Main.

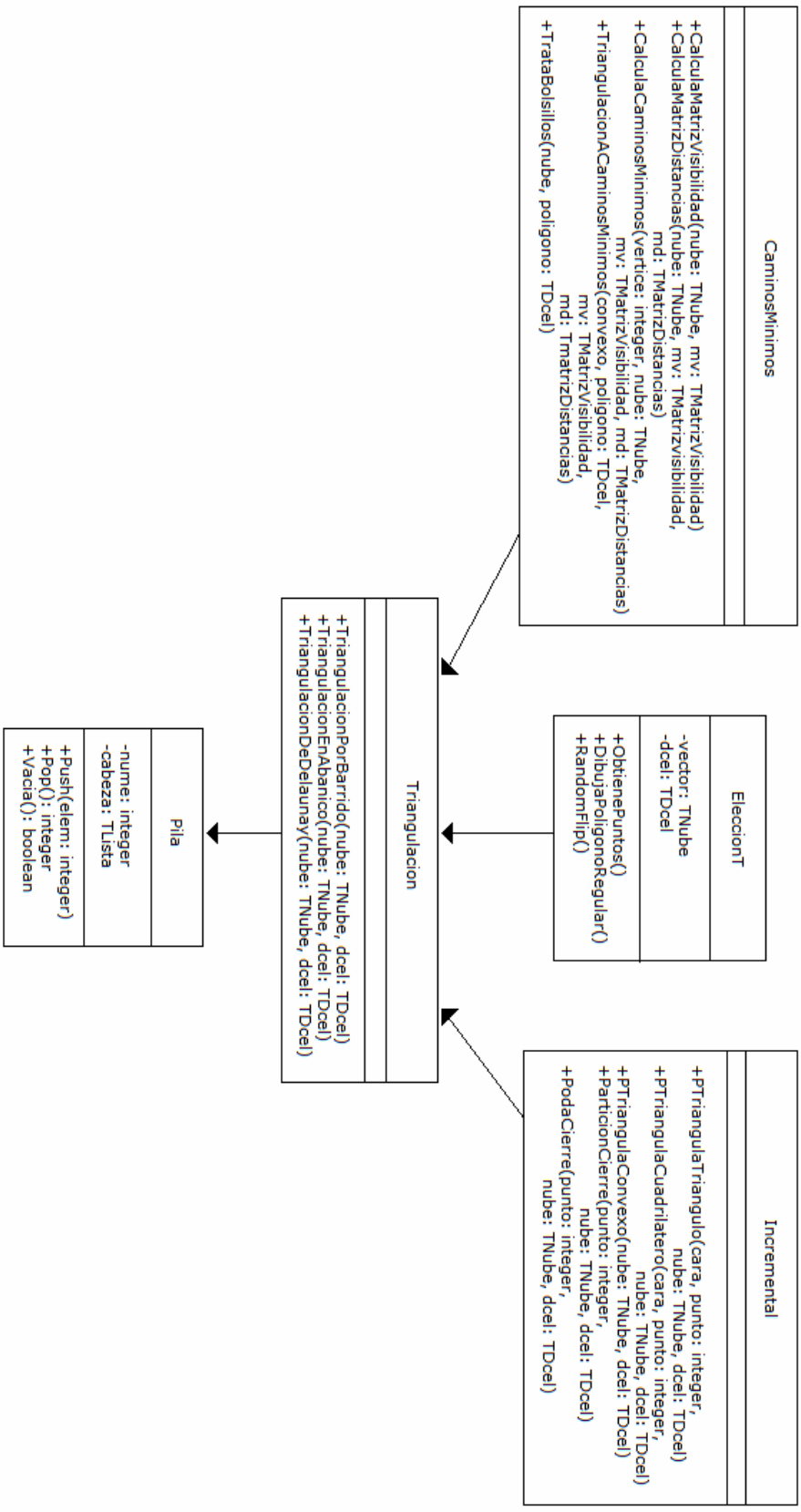
En las siguientes páginas se expone el diagrama por partes:

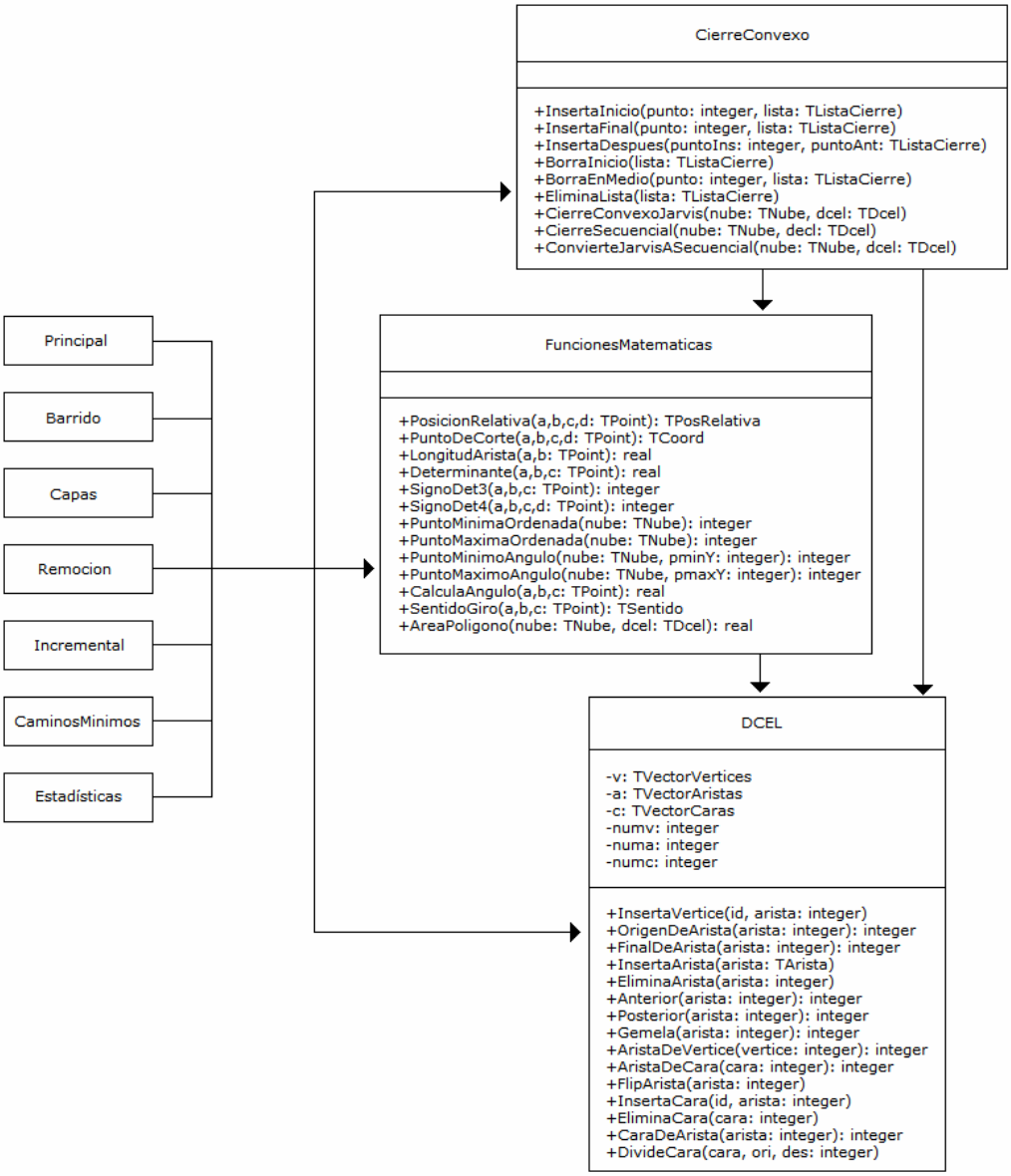
La primera contiene las relaciones entre la clase principal, los módulos con la implementación de los algoritmos de pseudo-triangulación y la ventana de selección de triangulación inicial.

La segunda contiene las dependencias entre la ventana de selección de triangulación, el módulo de apoyo para triangulación, la clase Pila y los módulos de métodos que hacen uso de las funciones de triangulación.

La tercera muestra las relaciones entre los módulos de métodos, las clases DCEL y CierreConvexo que usan la mayoría de métodos y los módulos de apoyo con funciones matemáticas.


Incremental
+PTriangulaTriangulo(cara, punto: integer, nube: TNube, dcel: TDcel)
+PTriangulaCuadrilatero(cara, punto: integer, nube: TNube, dcel: TDcel)
+PTriangulaConvexo(nube: TNube, dcel: TDcel)
+ParticionCierre(punto: integer, nube: TNube, dcel: TDcel)
+PodaCierre(punto: integer, nube: TNube, dcel: TDcel)
EleccionT
-vector: TNube
-dcel: TDcel
+ObtienePuntos()
+DibujaPoligonoRegular()
+RandomFlip()
Remoción
+RemoverAristas(nube: TNube, dcel: TDcel)
+RemoverTriangulos(nube: TNube, dcel: TDcel)
CaminosMinimos
+CalculaMatrizVisibilidad(nube: TNube, mv: TMatrizVisibilidad)
+CalculaMatrizDistancias(nube: TNube, mv: TMatrizvisibilidad, md: TMatrizDistancias)
+CalculaCaminosMinimos(vertice: integer, nube: TNube, mv: TMatrizVisibilidad, md: TMatrizDistancias)
+TriangulacionACaminosMinimos(convexo, poligono: TDcel, mv: TMatrizVisibilidad, md: TmatrizDistancias)
+TrataBolsillos(nube, poligono: TDcel)
Principal
-ToolBar: TToolBar
-MainMenu: TMainMenu
-PanelStat: TPanel
-PanelAngulo: TPanel
-PanelAlea: TPanel
-StatusBar: TStatusBar
-vp: TNube
-dcel: TDcel
+PizarraMouseDown()
+PizarraMouseUp()
+PizarraMouseMove()
+LimpiaPizarra()
+DibujaPunto(x,y: integer)
+DibujaPT()
Capas
+CalculaCapas(nube: TNube, ptcapas: TPtcapas)
+CalculaInterCapas(nube: TNube, ptcapas: TPTcapas)
+NumCapas(): integer
Estadísticas
+Peso(nube: TNube, dcel: TDcel): real
+GradoV(nube: TNube, dcel: TDcel): integer
+GradoC(nube: TNube, dcel: TDcel): integer
Barrido
+CalculaBarrido(nube: TNube, dcel: Tdcel)


CaminosMinimos
+CalculaMatrizVisibilidad(nube: TNube, mv: TMatrizVisibilidad)
+CalculaMatrizDistancias(nube: TNube, mv: TMatrizvisibilidad,
md: TMatrizDistancias)
+CalculaCaminosMinimos(vertice: integer, nube: TNube,
mv: TMatrizVisibilidad, md: TMatrizDistancias)
+TriangulacionACaminosMinimos(convexo, poligono: TDcel,
mv: TMatrizVisibilidad,
md: TmatrizDistancias)
+TrataBolsillos(nube, poligono: TDcel)
Triangulacion
+TriangulacionPorBarrido(nube: TNube, dcel: TDcel)
+TriangulacionEnAbanico(nube: TNube, dcel: TDcel)
+TriangulacionDeDelaunay(nube: TNube, dcel: TDcel)
EleccionT
-vector: TNube
-dcel: TDcel
+ObtienePuntos()
+DibujaPoligonoRegular()
+RandomFlip()
Pila
-nume: integer
-cabeza: TLista
+Push(elem: integer)
+Pop(): integer
+Vacia(): boolean
Incremental
+PTriangulaTriangulo(cara, punto: integer,
nube: TNube, dcel: TDcel)
+PTriangulaCuadrilatero(cara, punto: integer,
nube: TNube, dcel: TDcel)
+PTriangulaConvexo(nube: TNube, dcel: TDcel)
+ParticionCierre(punto: integer,
nube: TNube, dcel: TDcel)
+PodaCierre(punto: integer,
nube: TNube, dcel: TDcel)


CierreConvexo
+InsertaInicio(punto: integer, lista: TListaCierre)
+InsertaFinal(punto: integer, lista: TListaCierre)
+InsertaDespues(puntoIns: integer, puntoAnt: TListaCierre)
+BorraInicio(lista: TListaCierre)
+BorraEnMedio(punto: integer, lista: TListaCierre)
+EliminaLista(lista: TListaCierre)
+CierreConvexoJarvis(nube: TNube, dcel: TDcel)
+CierreSecuencial(nube: TNube, decl: TDcel)
+ConvierteJarvisASecuencial(nube: TNube, dcel: TDcel)
Principal
Barrido
Capas
Remocion
Incremental
CaminosMinimos
Estadísticas
FuncionesMatematicas
+PosicionRelativa(a,b,c,d: TPoint): TPosRelativa
+PuntoDeCorte(a,b,c,d: TPoint): TCoord
+LongitudArista(a,b: TPoint): real
+Determinante(a,b,c: TPoint): real
+SignoDet3(a,b,c: TPoint): integer
+SignoDet4(a,b,c,d: TPoint): integer
+PuntoMinimaOrdenada(nube: TNube): integer
+PuntoMaximaOrdenada(nube: TNube): integer
+PuntoMinimoAngulo(nube: TNube, pminY: integer): integer
+PuntoMaximoAngulo(nube: TNube, pmaxY: integer): integer
+CalculaAngulo(a,b,c: TPoint): real
+SentidoGiro(a,b,c: TPoint): TSentido
+AreaPoligono(nube: TNube, dcel: TDcel): real
DCEL
-v: TVectorVertices
-a: TVectorAristas
-c: TVectorCaras
-numv: integer
-numa: integer
-numc: integer
+InsertaVertice(id, arista: integer)
+OrigenDeArista(arista: integer): integer
+FinalDeArista(arista: integer): integer
+InsertaArista(arista: TArista)
+EliminaArista(arista: integer)
+Anterior(arista: integer): integer
+Posterior(arista: integer): integer
+Gemela(arista: integer): integer
+AristaDeVertice(vertice: integer): integer
+AristaDeCara(cara: integer): integer
+FlipArista(arista: integer)
+InsertaCara(id, arista: integer)
+EliminaCara(cara: integer)
+CaraDeArista(arista: integer): integer
+DivideCara(cara, ori, des: integer)

5. Estudio de una nube de puntos

Después del repaso que hemos dado a los diferentes métodos de pseudo-triangulación implementados, nos surge una pregunta natural: ¿qué método es el mejor de entre todos los vistos?

No es una pregunta con respuesta sencilla, entre otras cosas, porque cada tipo de PT responde a unas necesidades y tiene sus usos particulares, por lo que no puede haber un método mejor absolutamente. Algunos son más rápidos en la respuesta, otros acotan ciertas características de la PT, otros nos dan mayor libertad de elección…

Todo lo que podemos hacer es estudiar los parámetros característicos de las PT generadas por cada método y compararlas entre sí, para ver dónde están los puntos fuertes y débiles de cada uno. Y esto es lo que nos proponemos hacer a continuación. Para ello escogeremos una distribución de n puntos, siendo n un número de cierta entidad pero no demasiado alto: digamos n = 30. Este número dará lugar a PTs de complejidad suficiente para que las variaciones en los valores de los parámetros arrojados por cada método sean apreciables y representativas.

La misma distribución se someterá a cada método por separado y discutiremos los valores de los parámetros comparándolos con los obtenidos por el resto de métodos. Los parámetros que examinaremos serán los siguientes:

Peso: la suma de las longitudes de todas las aristas de la PT. Por lo general resultará conveniente mantenerlo lo más bajo posible.

Grado de los vértices: el máximo número de aristas que inciden en algún vértice de la PT. También resulta interesante mantener bajo este valor puesto que a valores menores que 5 siempre resulta posible encontrar una PT mínima.

Grado de las caras: el máximo número de aristas que componen alguna cara de la PT. Al igual que en el caso de los vértices, puede resultar conveniente mantener bajo este valor. Aunque por lo general no será posible mantener bajos ambos a la vez.

Número de caras: el número de caras que componen la PT, y qué porcentaje de estas son triángulos simples. Cuantos menos triángulos contenga una pseudo-triangulación mejor, puesto menos aristas habrá, y mayor probabilidad de ser mínima.

Elección de la distribución de puntos:

Hemos confeccionado la nube de puntos manualmente, intentando evitar las frecuentes alineaciones que se producen si dejamos que sea un proceso aleatorio el que la genere. Insertando los puntos con cuidado de no ponerlos muy juntos y moviéndolos cuando dan lugar a caras muy irregulares de aristas indistinguibles, llegamos, por un proceso de ensayo y error, a una distribución como la siguiente, que produce pseudo-triangulaciones aceptables para la totalidad de los métodos.

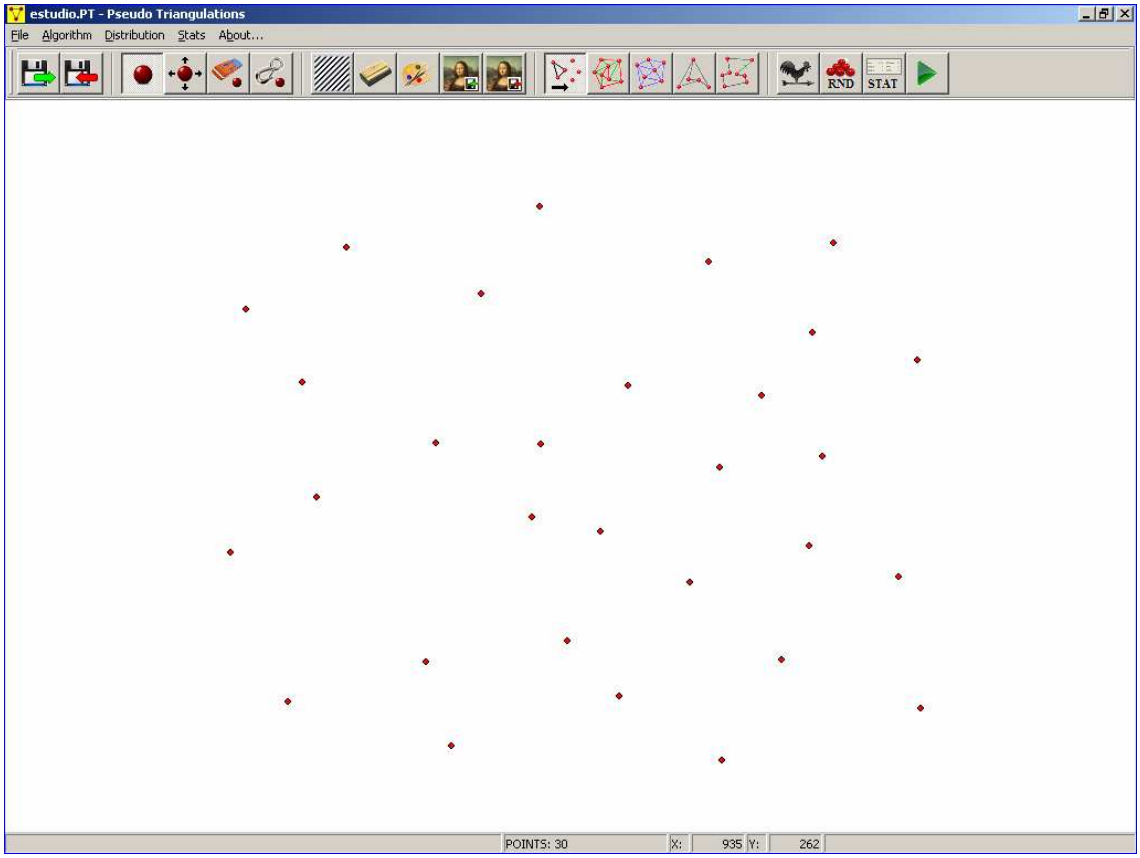


Figura 5.1: Distribución de trabajo

Es apreciable el intento por equiespaciar los puntos, con cierta distancia mínima de separación de forma que no de lugar a caras muy pequeñas. Igualmente evitamos poner puntos cerca de los bordes para que el contorno de la PT pueda verse sin problemas, manteniendo la nube centrada en la pantalla.

Hemos intentado también que el cierre convexo de la nube tenga una forma regular y equilibrada, sin mayor carga de puntos hacia un lado. Así mismo, la proporción de puntos pertenecientes al convexo con respecto al total es alta para evitar gran acumulación de aristas en algoritmos como el de caras de grado acotado.

Veamos pues, el resultado producido por cada método concreto:

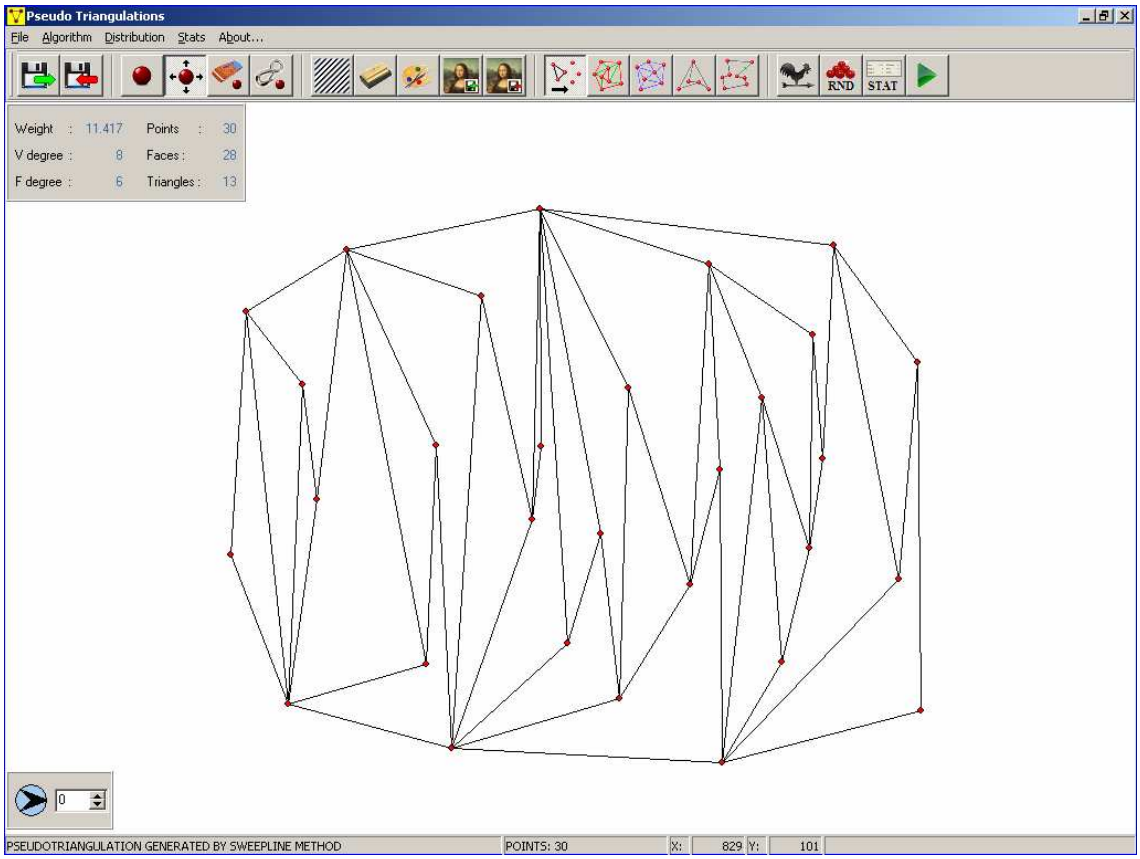


Figura 5.2: Resultado por el método de barrido

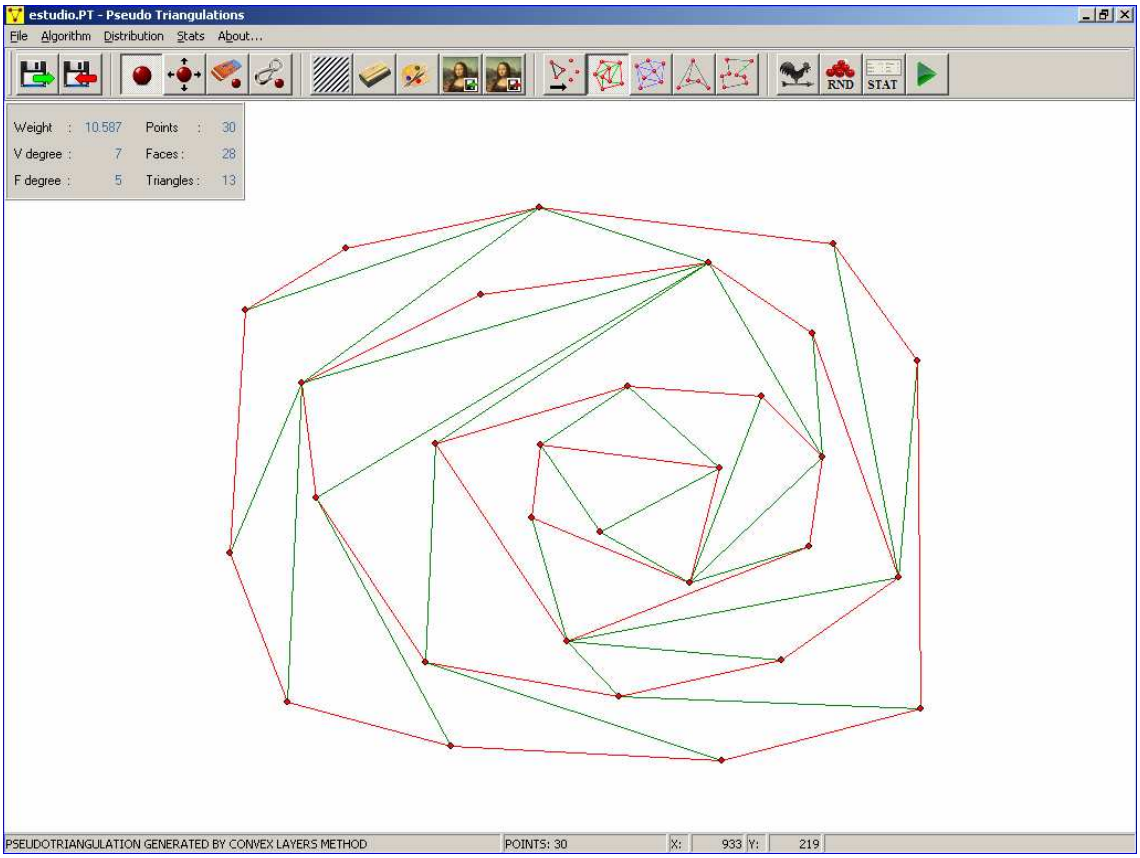


Figura 5.2: Resultado por el método de capas convexas

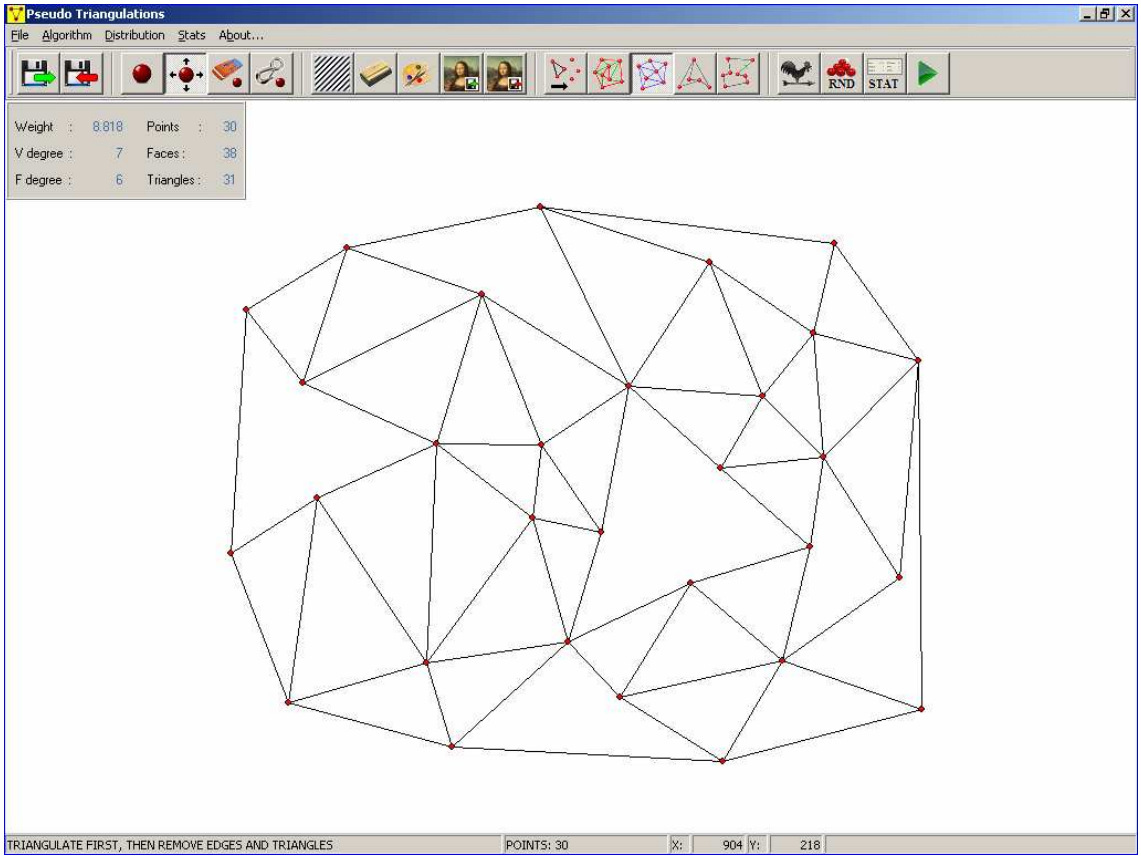


Figura 5.3: Resultado por el método de triangular y suprimir

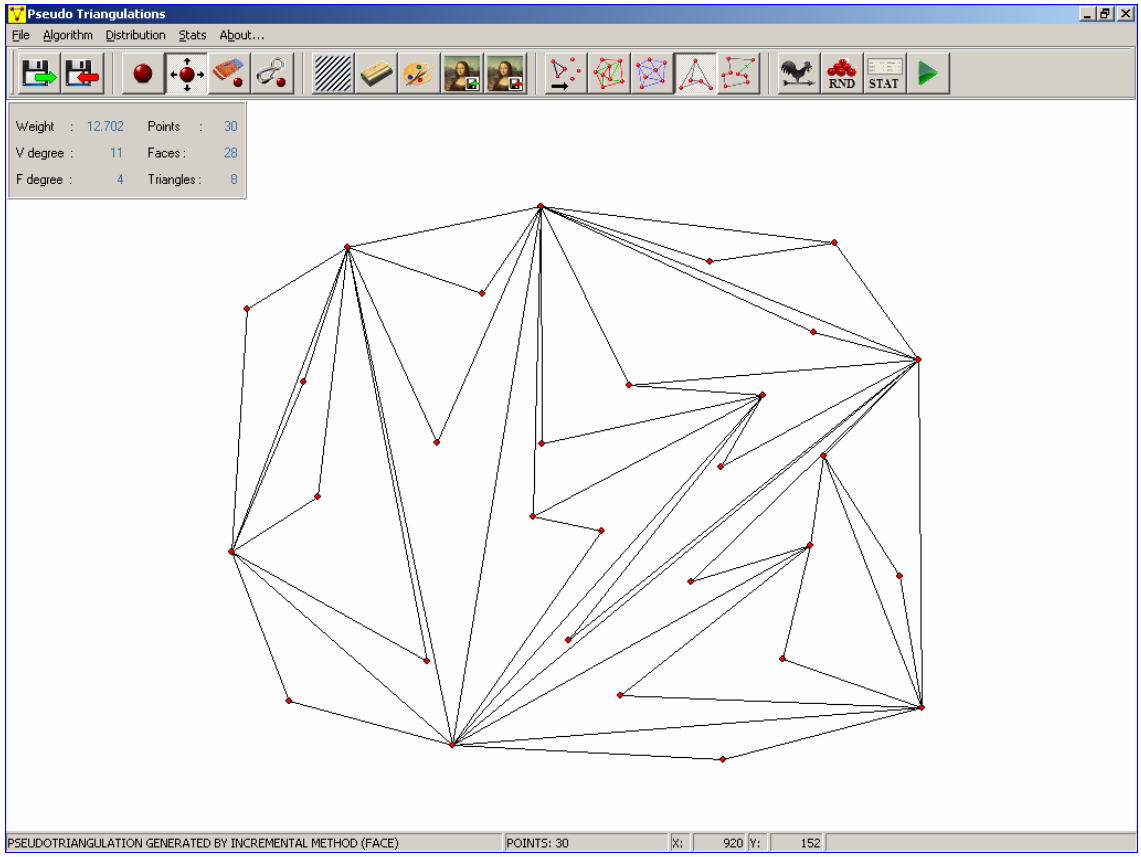


Figura 5.4: Resultado por el método incremental (caras de grado acotado)

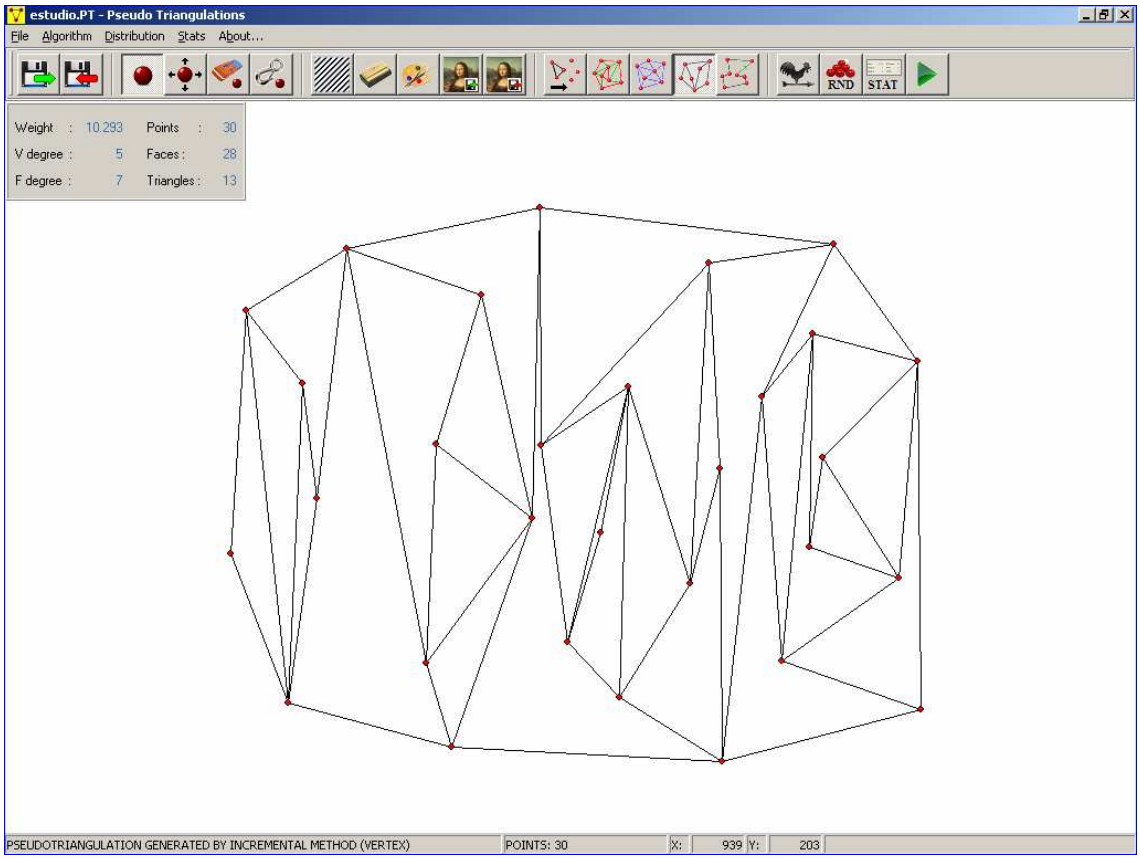


Figura 5.5: Resultado por el método incremental (vértices de grado acotado)

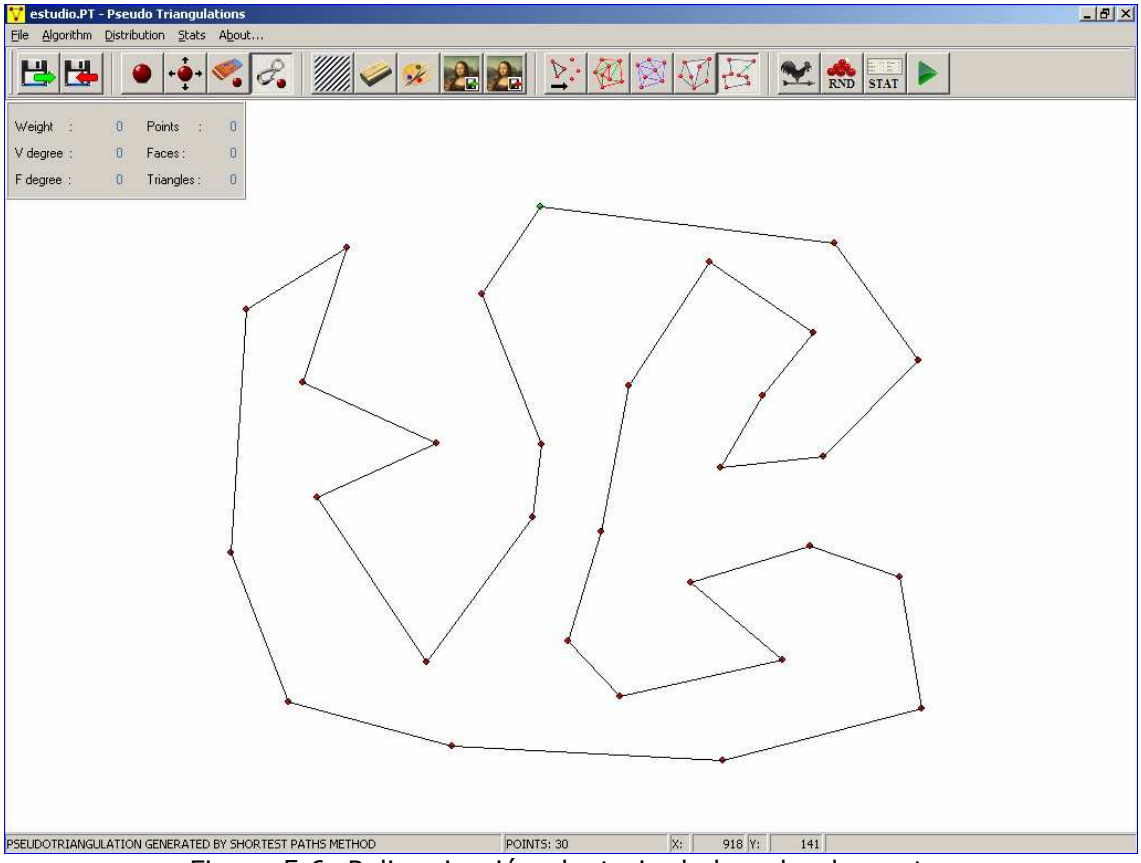


Figura 5.6: Poligonización aleatoria de la nube de puntos

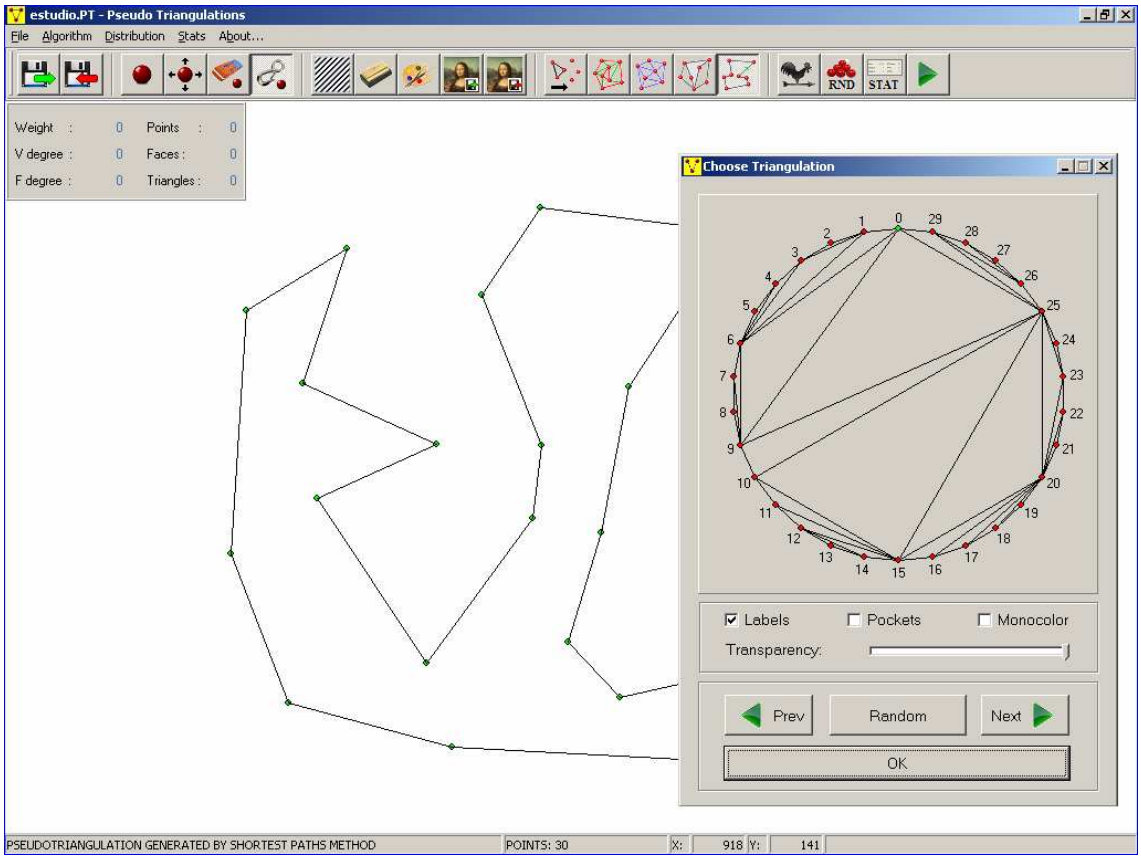


Figura 5.7: Triangulación aleatoria del convexo asociado

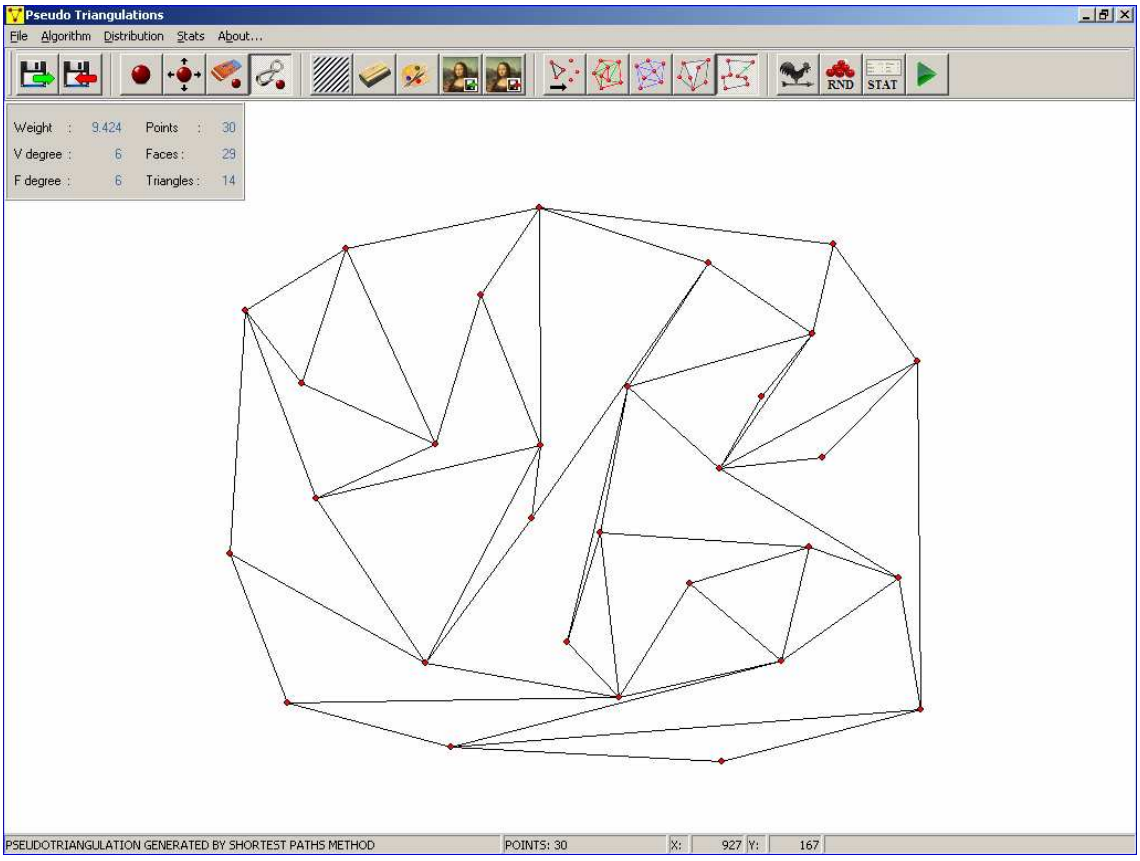


Figura 5.8: Resultado por el método de caminos mínimos

	Peso	Max Grado V	Max Grado C	Número de Caras	Número de Triángulos
Barrido	11.417	8	6	28	13
Capas	10.587	7	5	28	13
Remoción	8.818	7	6	38	31
Incremental (caras)	12.702	11	4	28	8
Incremental (vértices)	10.293	5	7	28	13
Caminos mínimos	9.424	6	6	29	14

Tabla 5.1: valores para una distribución de 30 puntos

Colocamos los resultados en una tabla para poder compararlos mejor. Y ahora, estudiamos cada parámetro concreto:

5.1. Peso

Según se puede observar, y así ocurre en la mayoría de las distribuciones, el método incremental en su variante de caras de grado acotado arroja el mayor valor para este parámetro, seguido muy de cerca por el método de barrido. A continuación vienen los métodos de capas e incremental en su variante de vértices, con un peso notablemente inferior, aunque esto se hace más evidente en PTs de mayor número de puntos. El algoritmo de caminos mínimos tiene un comportamiento variable, pues hay muchas combinaciones posibles entre elección de polígonos, caminos y relleno de bolsillos. No obstante, el peso tiende a estar por debajo de todos los anteriores. Finalmente, el método que invariablemente arroja un peso menor es el de remoción. Vamos a intentar razonar el porqué de estos resultados.

Método incremental (caras)

Recuérdese que en este método se comenzaba por triangular el cierre convexo, por lo que aquí ya tenemos un primer conjunto de aristas que atraviesan la distribución de extremo a extremo. Pero es que además, el resto de los puntos se añaden a la

PT uniéndose a dos vértices del pseudo-triángulo en el que se alojan, que en la mayoría de los casos están situados en el cierre convexo. Por tanto la mayoría de los puntos, tantos los periféricos como los situados en el centro de la nube, acaban tirando una recta a los puntos externos situados en el cierre, lo que incrementa la longitud media de las aristas, con la consiguiente elevación del peso.

Método de Barrido

Algo similar ocurre en el caso del barrido. Al agregar un nuevo punto a la PT se proyectan las dos rectas soporte hacia el convexo actual. La mayoría de las veces esto implica recorrer distancias muy largas hasta puntos muy lejanos. A menos que los puntos se distribuyan en una banda muy estrecha en la dirección de la recta de barrido, el convexo irá aumentando su diámetro en la perpendicular a la recta de barrido, por lo que se sitúe donde se sitúe el siguiente punto, o bien lanza una recta soporte corta hacia el extremo cercano y una recta soporte larga hacia el extremo lejano, o bien lanza dos rectas soporte de longitud media. El resultado vuelve a ser una longitud media de arista bastante elevada.

Método de Capas

Dado que este método es una variación del método de barrido cabría esperar que los pesos de sus PTs fueran similares a los de este. Sin embargo, si nos detenemos a examinar paso a paso la construcción de la PT nos daremos cuenta enseguida que para la mayoría de los puntos solo se traza una línea soporte, pues la otra se suele solapar con la arista que lo une al punto anterior en la propia capa. Puesto que el punto anterior ya ha sido tratado antes, y este se encuentra una capa por encima de aquellos a los que se une, siempre formará parte del convexo actual. Como el punto actual debe unirse al convexo actual tendrá que unirse al punto anterior, pero de hecho ya se encuentra unido por una arista de capa. Por tanto el peso de las PTs calculadas por capas suele ser inferior en todo caso a las PTs por barrido.

Método incremental (vértices)

El modo de operación de este método consiste en ir reduciendo cada vez más el convexo de puntos sobre el que se opera, por lo que la longitud de las aristas irá disminuyendo conforme avanzan las iteraciones.

Método de remoción

Es bien sabido que la triangulación de Delaunay, en la que se basa este método, tiende a producir los triángulos más regulares posibles, maximizando sus mínimos

ángulos internos. Esto redunda en la longitud de las aristas, que tienden a ser lo más pequeñas posible.

La diferencia en los pesos es un factor que depende de la distribución concreta, claro está. Pero en la generalidad de los casos tiende a ser como se ha explicado: encabeza el grupo (incremental caras, barrido) seguido de (incremental vértices, capas) seguido de caminos mínimos seguido de remoción. Las diferencias entre el primer grupo y el último son siempre notables.

5.2. Grado de los vértices

En el caso de este parámetro, los extremos de la jerarquía están muy claros: el método incremental en su variante de caras siempre da el peor resultado con mucha diferencia. Por el contrario, el método incremental en su variante de vértices siempre da el mejor resultado (no en balde su meta es mantener el grado de los vértices lo más bajo posible). Por el medio la cosa no está tan clara. Capas y remoción suelen dar valores muy por debajo de incremental caras, pero entre ellos no suele haber mucha diferencia. Barrido con frecuencia arroja valores altos, aproximándose en ocasiones a incremental caras. Aunque puede dar valores similares a capas y remoción, lo normal es que esté siempre por encima de ellos. Y finalmente, caminos mínimos tiene un comportamiento variable. Al igual que en el caso del peso, hay muchos factores aleatorios que pueden influir sobre el valor final. Depende sobre todo de la pseudo-triangulación de los bolsillos. Pero en general, el máximo grado se moverá en valores medios-altos. Examinemos ahora las posibles causas de cada comportamiento:

Método incremental (caras)

Como ya se comentó anteriormente, la mayoría de los puntos de la nube se unirá a alguno de los puntos situados en el cierre convexo. Esto provocará que los puntos del cierre acumulen gran cantidad de aristas incidentes.

Barrido

Muchas veces un punto de la nube alcanza gran altura sobre el resto, convirtiéndose así en un obstáculo para la visibilidad del resto del convexo. Entonces una gran parte de los puntos que quedan por debajo se unirán a él como extremo visible del convexo, acumulando gran cantidad de aristas incidentes.

Capas

En este caso, los nuevos puntos que se van agregando a la PT van rodeando el convexo en formación, por lo que ningún punto se convierte en obstáculo a la visibilidad como en el caso del barrido, o al menos no por mucho tiempo.

Remoción

En las PTs generadas por este método la mayoría de los vértices tienden a unirse con todos aquellos vértices que les rodean, dando lugar a configuraciones conocidas como ruedas de carro. Al distribuir de esta manera las aristas ningún punto llega a concentrar demasiadas, a menos que el centro de la rueda proyecte muchos radios.

Caminos mínimos

En este caso no hay una pauta establecida. Un punto puede convertirse en estación de tránsito para una gran cantidad de caminos mínimos. Aunque no sea el caso general, basta con que haya un punto de estas características para aumentar el valor del parámetro. También, dependiendo de la pseudo-triangulación de los bolsillos, un punto puede concentrar gran cantidad de aristas, por ejemplo, si la PT del bolsillo se ha hecho en abanico, partiendo todas las aristas de un punto radial.

5.3. Grado de las caras

Este parámetro es de alguna manera complementario al parámetro anterior. Aquí el mejor resultado siempre lo proporciona el método incremental caras, puesto que su meta es precisamente acotar su grado máximo. Y el peor suele darlo el método incremental vértices. El resto se confunde en el medio sin que pueda señalarse uno por encima o por debajo del resto.

En el caso del incremental caras está claro que da el valor más bajo por la propia construcción de la PT. En el caso del incremental vértices, los pseudo-triángulos de las operaciones de partición se definen encontrando dos cadenas cóncavas desde el punto mediana hasta el extremo opuesto del convexo. Estas dos cadenas pueden involucrar a gran cantidad de puntos intermedios, por lo que el grado de estos pseudo-triángulos puede llegar a ser elevado.

En cuanto al resto de métodos no tienen una característica innata que les lleve a formar caras de alto o de bajo grado. Suelen mantenerse en valores intermedios entre estos dos extremos.

5.4. Número de caras

Un rápido vistazo a esta columna de la tabla nos revela un dato sorprendente: tanto barrido, como capas como incremental en sus dos variantes generan una PT con el mismo número de caras. En realidad esto no es nada sorprendente, ya que los 4 algoritmos citados tienen la característica común de producir PTs *mínimas* o *puntiagudas*, como se ha visto en sus correspondientes apartados. En concreto, a la PT por barrido se le llama *PT mínima canónica*. Recuérdese que una PT mínima tiene el mínimo número de caras posibles. Por eso los 4 métodos coinciden en el valor de este parámetro, no solo para esta distribución en concreto, sino para cualquier distribución.

Remoción no genera PTs puntiagudas, sino minimales (esto es, de las que no puede eliminarse ninguna arista porque la unión de dos caras cualesquiera no forma un pseudo-triángulo). A menos que el máximo grado de los vértices fuera 4, en cuyo caso la PT sería a la vez puntiaguda y minimal. Pero no es el caso en la distribución bajo estudio. Por esa razón da un número de caras mucho mayor mediante este método.

Caminos mínimos tampoco produce PTs puntiagudas, y como siempre, el valor del parámetro puede variar ampliamente, aunque suele ser alto.

5.5. Número de triángulos

Donde sí pueden variar los 4 métodos anteriormente citados es en el número de caras de entre el total que son triángulos simples, puesto que estamos constreñidos solo por el número de caras totales y no por la forma de estas. Por tanto, cada método puede arrojar un valor diferente para este parámetro aunque curiosamente, para nuestra distribución de prueba, tres de estos métodos han dado el mismo número.

En general, incremental caras siempre dará el menor valor, por el siguiente motivo: el algoritmo implica que se parte de una división del convexo en triángulos. Ahora bien, cada vez que se añade un nuevo punto, el triángulo original es dividido en otro triángulo y un cuadrilátero. Los cuadriláteros a su vez se descomponen en dos cuadriláteros. No es difícil comprobar que la PT resultante contendrá por tanto tantos triángulos como la triangulación original del convexo. Es decir, n-2 siendo n el número de puntos en el cierre convexo. A menos que el cierre contenga la

mayoría de los puntos de la nube, el número de triángulos será en general bastante bajo.

Remoción suele dar el valor más alto por motivos ya explicados anteriormente. Este método tiende a unir los puntos con sus más cercanos dando lugar a ruedas de carro, que son esencialmente polígonos convexos triangulados uniendo sus vértices con un punto central. Pero esto solo es así en la medida en que hemos escogido la triangulación de Delaunay para la implementación del algoritmo. Distintos tipos de triangulación podrían dar lugar a resultados distintos.

En cuanto al barrido y capas, dependerá de lo pegados que estén los sucesivos puntos a sus convexos anteriores y el ángulo que tengan sobre ellos. Un punto muy cercano a la parte media de una arista tiene muchas probabilidades de ver solo esa arista del convexo, por lo que su inclusión generaría un triángulo. Puntos más alejados del convexo anterior o situados cerca de una esquina tenderán a generar caras de mayor grado.

Para terminar vamos a poner otra tabla con los valores medios de cada parámetro obtenidos tras efectuar pruebas con 30 distribuciones aleatorias normales de 50 puntos, con desviación típica 180 píxeles.

N = 50	Peso	Max Grado V	Max Grado C	Número de Caras	Número de Triángulos	% tri
Barrido	23.915	11,9	6,4	48	17,73	37%
Capas	17.659	7,1	6,3	48	18	37%
Remoción	13.939	7,9	6,2	61,9	42,7	69%
Incremental (caras)	24.551	18,2	4	48	6,1	13%
Incremental (vértices)	18.509	5	7,5	48	21,1	44%
Caminos mínimos	17.081	9,3	6,7	61,3	42	69%

Tabla 5.2: valores promedio para 30 distribuciones de 50 puntos

Como se puede constatar, los valores de esta segunda tabla siguen las pautas inferidas a partir de la primera, por lo que nuestros razonamientos han sido esencialmente correctos.

Quedaría por probar como se comporta cada algoritmo ante distribuciones uniformes de puntos, o como afecta la mayor o menor dispersión de los puntos en las distribuciones normales (como afecta la desviación típica).

6. Manual de usuario

La aplicación se ha diseñado como herramienta para generación y visualización de pseudo-triangulaciones, lo que conlleva usar la mayor resolución de pantalla posible y dejar un amplio espacio para que el usuario introduzca las distribuciones de puntos a su antojo. Esto implica que la estructura de la interfaz se asemeje a un programa de edición de gráficos típico: una zona de dibujo ocupando la práctica totalidad de la pantalla, con algunos elementos de control en la parte superior e inferior. Veamos el aspecto general de la aplicación:

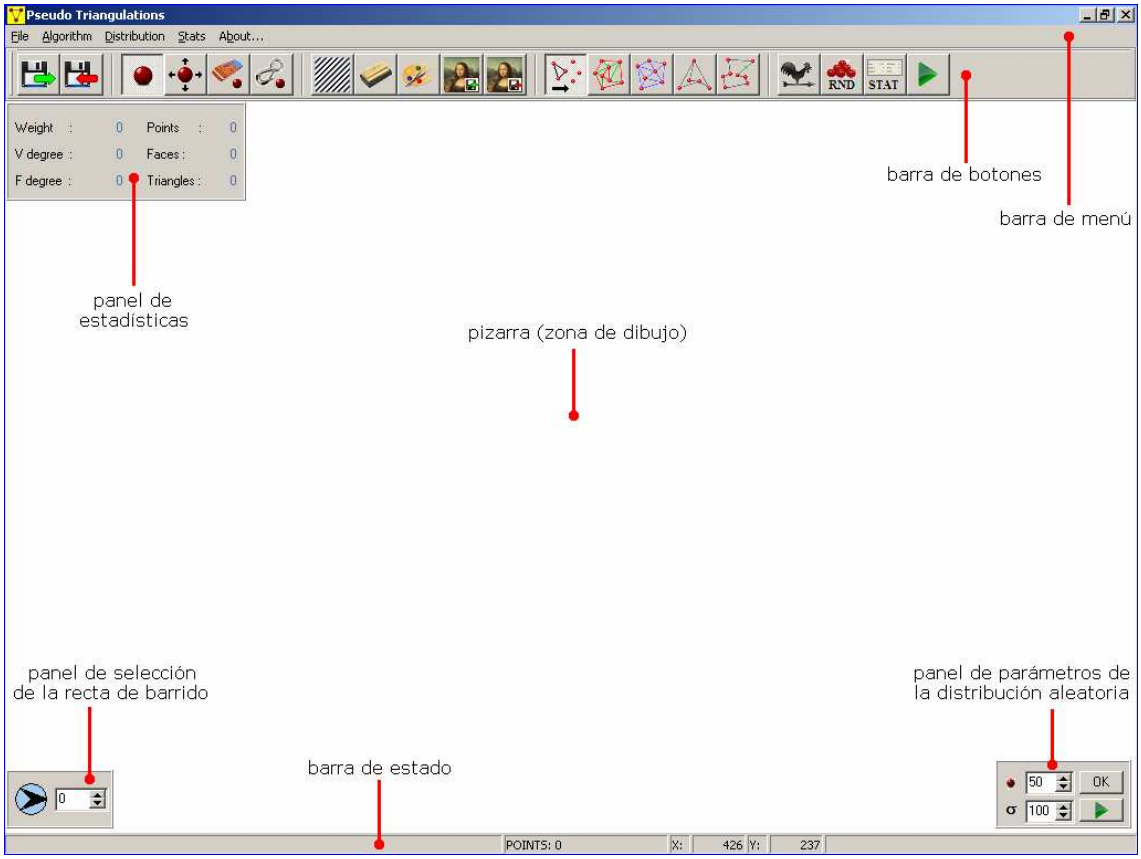


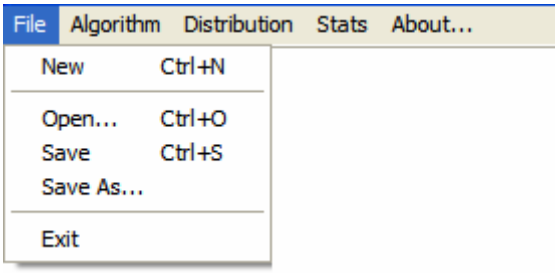
Figura 6.1: elementos de la pantalla principal

Como se puede observar en la figura, la interfaz consiste en una superficie de dibujo que llamaremos pizarra en la que se pueden insertar, mover o borrar los puntos de la nube; una barra de botones de acción, una barra de menú superior, una barra de estado inferior, y unos paneles auxiliares que emergerán cuando se pulsen ciertos botones y que pueden tornarse visibles o invisibles a voluntad del usuario, de forma que no afecte a la visibilidad de la pizarra.

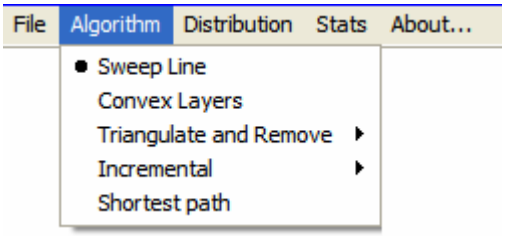
Pasamos a examinar cada elemento de los enumerados anteriormente.

6.1 Funciones de la barra de menú

File: este submenú es el típico de las aplicaciones Windows estándar, con las mismas teclas de acceso rápido y las mismas operaciones básicas:
New borra la pizarra y deja todo listo para empezar a definir otra distribución.
Open carga una PT desde un fichero en disco.
Save guarda la distribución actual a un fichero en disco. Si es la primera vez que la guardamos nos pedirá el nombre del archivo. En veces sucesivas sobreescribirá el contenido de este.
Save As guarda la distribución actual al fichero que le indiquemos.
Exit cierra la aplicación.

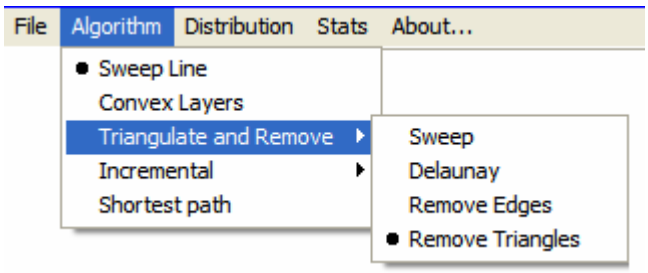


Algorithm: en este submenú es posible elegir el algoritmo de pseudo-triangulación deseado. Equivale a pulsar el botón correspondiente en la barra de botones. La barra de menú y la de herramientas están enlazadas, de manera que al pulsar el botón de un método cualquiera la selección se ve reflejada en la barra de menú y viceversa, al seleccionar el algoritmo desde el menú, se oprime el botón correspondiente.



Sin embargo, este elemento no es totalmente redundante, puesto que también permite especificar algunas opciones adicionales que no están disponibles desde la barra de botones. Por ejemplo, en el algoritmo de remoción, podemos indicar hasta que fase queremos ejecutarlo: triangulación inicial, transformación en Delaunay, remoción de aristas y remoción de triángulos. Esto nos permite ver en mayor detalle el proceso de cálculo de la PT.

Es interesante para observar como se transforma la triangulación inicial por barrido en triangulación de Delaunay, el número de aristas que se eliminan y las condiciones que llevan a poder suprimir triángulos. Es decir, permite hacernos una mejor idea de la eficiencia del algoritmo.

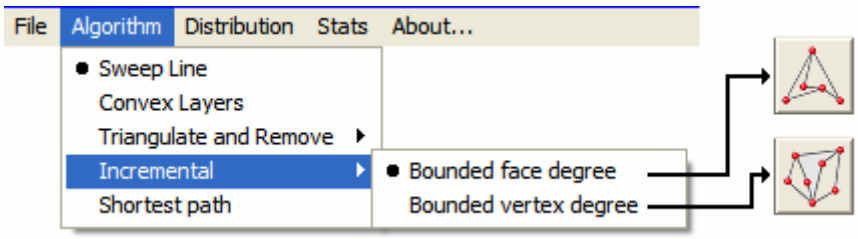


Desde este submenú también se nos permite especificar, para el algoritmo incremental, sobre qué elemento de la PT vamos a realizar la acotación, si sobre vértices o sobre caras.

Una PT con vértices de grado acotado, tendrá sus vértices con un máximo de 5 aristas incidentes.

Una PT con caras de grado acotado tendrá todas sus caras con un máximo de 4 aristas.

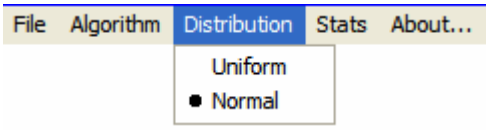
Al seleccionar cualquiera de estas dos opciones el botón asociado cambia de aspecto, según la acotación escogida. Al pulsar el botón de ejecución, se calculará la PT según la opción correspondiente.



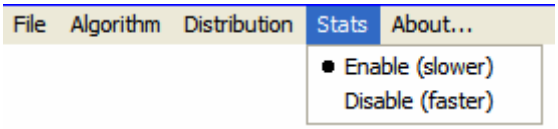
Distribution: en este submenú es posible seleccionar el tipo de distribución aleatoria con el que trabajará la aplicación.

Una distribución *uniforme* esparcirá los puntos por toda la pizarra con igual probabilidad.

Una distribución *normal* tenderá a agrupar los puntos en el centro de la pizarra, pues la probabilidad aumenta hacia la parte media de los ejes de coordenadas.



Stats: Aquí se puede habilitar o deshabilitar el cálculo de las estadísticas de la PT.
Enable (slower): con esta opción marcada se calculan las estadísticas, tanto si está visible la pantalla de estadísticas como si no.
Disable (faster): con esta opción marcada no se calcula ninguna estadística. Tampoco se puede mostrar la pantalla de estadísticas.
Hay que tener en cuenta que para obtener las características de la PT es necesario realizar iteraciones adicionales que añaden ciclos de computación, con lo que la velocidad de redibujado de la PT puede verse afectada. Esto solo es realmente molesto cuando se está modificando la PT en tiempo real (moviendo alguno de sus puntos). En cualquier caso, la velocidad no se ve afectada en exceso a menos que trabajemos con un número alto de puntos.



6.2 Funciones de la barra de botones

Los distintos botones de la barra se encuentran agrupados por funcionalidades. Vamos a ir viendo cada grupo en orden comenzando desde la parte izquierda y hasta el final de la barra. En cada apartado se recuadran en rojo los botones pertenecientes a la categoría.

Cargar y Guardar:
Con el botón de *Guardar* (diskette con flecha roja de entrada) se puede almacenar la nube de puntos en un fichero (solo la nube, la PT se puede calcular a partir de ella).
Con el botón de *Cargar* (diskette con flecha verde de salida) se puede recuperar la nube desde un fichero guardado o generado a mano.
Los ficheros tienen extensión .PT y su estructura es muy sencilla: una línea conteniendo la coordenada x del punto y otra línea conteniendo la coordenada y. Repetido n veces para el caso de n puntos.

Insertar, mover, borrar y unir puntos:

Estos 4 botones son depresibles y solo uno puede estar pulsado a la vez. Determinan el modo de operación al pinchar sobre la superficie de dibujo.

- Con el botón de *Insertar* (el primero) cada click sobre la pizarra inserta un punto en esas coordenadas.
- Con el botón de *Mover* (el segundo), si hacemos click sobre un punto, y arrastramos manteniendo el botón izquierdo del ratón pulsado, el punto se irá moviendo a las nuevas coordenadas. Si la PT está dibujada, ésta se actualizará en tiempo real.
- Con el botón de *Borrar* (el tercero), si hacemos click sobre un punto conseguiremos eliminarlo. Si la PT está dibujada, se actualizará en tiempo real.
- El botón de *Engarzar* (el cuarto) tiene dos modos de uso:
  1) Si lo pulsamos con la pizarra vacía cada nuevo punto que insertemos se unirá al inmediatamente anterior, definiendo un polígono. Para cerrarlo, basta con pulsar el botón derecho del ratón en cualquier parte de la pantalla. Pulsando Ctrl+Z podemos borrar el último punto insertado.
  2) Si lo pulsamos con una distribución de puntos en pantalla, se borrará cualquier PT presente, dejando tan solo los puntos. Al situar el puntero del ratón sobre un punto veremos que cambia de color, lo cual nos indica que el punto está seleccionado. Haciendo click sobre él se unirá al último que seleccionamos en la misma forma. Esto nos permite definir un polígono sobre la nube a nuestro antojo. Para cerrarlo se debe hacer click sobre el primer punto que seleccionamos. No se permite cerrar si no se han usado todos los puntos. Pulsando Ctrl+Z podemos borrar la última unión efectuada

  El modo de operación de este botón solo es compatible con PTs generadas mediante el algoritmo de caminos mínimos. Por tanto, al pulsarlo se seleccionará automáticamente este método de entre todos los disponibles.

Rejilla:

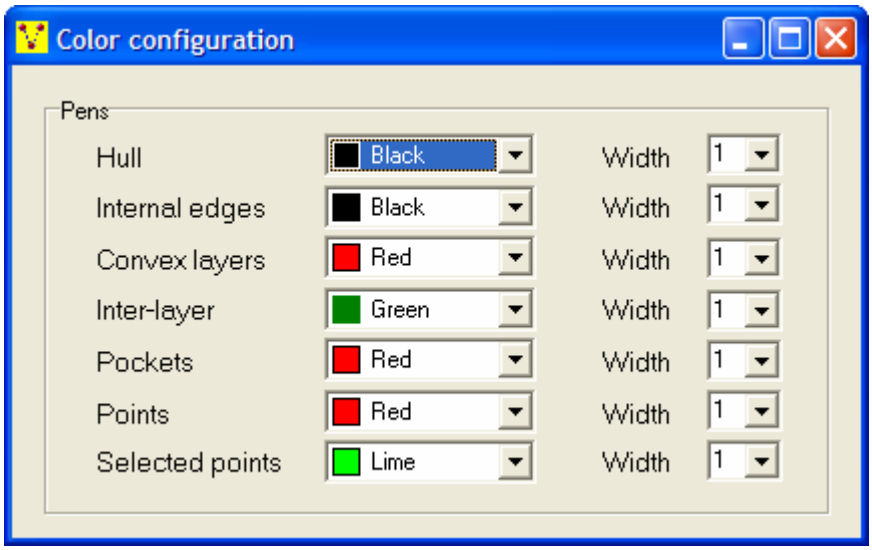
Al pulsar este botón se imprime un patrón de líneas paralelas perpendicular al vector director de la línea de barrido. Es muy útil para saber qué punto viene a continuación en el proceso de barrido. Pulsando de nuevo el botón, la rejilla desaparece. Funciona aún cuando el método seleccionado no sea el barrido.

Limpiar:

Borra el todo contenido de la pantalla, tanto los puntos, como la PT, como la imagen de fondo si la hubiera.

Configuración del color:

Con este botón se accede a una ventana donde se presentan cada uno de los elementos de dibujo susceptibles de configuración, permitiendo cambiar tanto el color con el que se pintan como el grosor del trazo utilizado.



Color y grosor de las plumas de dibujo

Hull es tanto el cierre convexo de la PT como el borde del polígono.
Internal edges son el resto de aristas interiores
Convex layers son las diferentes capas convexas en el algoritmo de capas
Inter-layer se refiere a las aristas inter-capa en el mismo algoritmo
Pockets son los 'bolsillos' en la PT por caminos mínimos (las aristas que rellenan las concavidades de la PT, formando un convexo)
Points son los puntos que componen la distribución
Selected points son aquellos puntos que ya han sido seleccionados en el proceso de definición de un polígono

Cargar y guardar imagen de fondo:

Con el botón de *Cargar* (Mona Lisa con flecha verde) se puede cargar una imagen BMP desde fichero y colocarla de fondo en la superficie de dibujo. El modo de copia es transparente, de forma que si hay una PT dibujada, esta permanecerá sobre el fondo.

Con el botón *Guardar* (Mona lisa con flecha roja) se puede salvar a fichero BMP o JPG la imagen de la pizarra.

Escoger método de pseudo-triangulación:

Estos 5 botones son depresibles y solo uno puede estar pulsado a la vez.

Al apretar uno de ellos se selecciona un método de pseudo-triangulación diferente, de manera que al pulsar la tecla de ejecución con una distribución de puntos en pantalla, se calculará la PT por ese método.

Los métodos asociados a cada botón son, de izquierda a derecha: barrido, capas convexas, remoción, incremental y caminos mínimos.

Si se pulsa el botón del método de caminos mínimos, automáticamente se selecciona el botón de engarzar puntos (modo definición de polígono), ya que este método requiere un polígono en lugar de una nube de puntos para su ejecución. No obstante, el usuario aún puede seleccionar cualquiera de los otros modos (inserción, movimiento y borrado de puntos) y operar a su antojo. Una vez terminadas las modificaciones, podemos proceder a unir los puntos pulsando de nuevo el botón de engarzar.

De igual manera, pulsando el botón de engarzar se selecciona automáticamente el método de caminos mínimos.

Escoger recta de barrido:

Con este botón se hace visible el panel que permite escoger el ángulo de la recta de barrido. Pulsándolo de nuevo, el panel desaparece.

Generar distribución aleatoria:

Pulsando este botón se hace visible el panel que permite escoger el número de puntos de la distribución aleatoria y, en caso de haber especificado distribución *Normal* desde la barra de menú, la desviación típica cuantificada en pixels. A menor valor de σ mayor agrupación de los puntos en torno al centro de la pizarra. A mayor valor, mayor dispersión.

Pulsando de nuevo el botón, el panel desaparece.

Mostrar estadísticas de la PT:

Con este botón se hace visible el panel que muestra las características de la PT. Si la PT está dibujada y se modifica, los valores del panel se irán actualizando en tiempo real.

Pulsando de nuevo el botón, el panel desaparece. Si deshabilitamos las estadísticas desde la barra de menú, el panel no se mostrará y nos avisará mediante mensaje.

Deshabilitar las estadísticas mejorará algo el rendimiento de la aplicación, al suprimir cierta cantidad de cálculos. No obstante, esto apenas se notará a menos que trabajemos con gran número de puntos o en una máquina lenta.

Ejecución (cálculo de la PT):

Al pulsar este botón la aplicación procede al cálculo y dibujado de la PT según el método seleccionado. Este mismo icono, en miniatura, también se encuentra situado en el panel de parámetros de la distribución para poder probar sucesivas distribuciones con mayor comodidad, evitando engorrosos desplazamientos de ratón arriba y abajo.

Si queremos observar el resultado de los distintos métodos sobre una misma distribución de puntos, no tenemos más que elegir secuencialmente cada método y pulsar el botón de ejecución.

6.3 Barra de estado

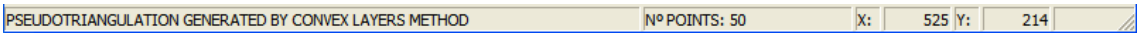
La barra de estado contiene los siguientes elementos:

Celda 1: operación realizada / ayuda

Celda 2: nº de puntos presentes en pantalla

Celda 4: coordenada x del puntero del ratón

Celda 6: coordenada y del puntero del ratón



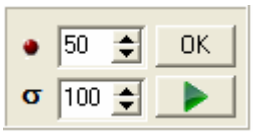
6.4 Paneles auxiliares

Panel de selección de recta de barrido:

Muestra la dirección de la recta de barrido en la forma de una pequeña brújula. Se puede cambiar el ángulo bien introduciendo un número en el campo de edición, bien pulsando en las dos flechas a la derecha (incrementan/decrementan el ángulo en 1 unidad).

Si la PT está dibujada, se actualizará en tiempo real, al igual que la rejilla, si estuviera visible.

Panel de parámetros de la distribución aleatoria:



Muestra los dos valores parametrizables de la distribución:

El *número de puntos*

La *desviación típica*

Si hemos seleccionado distribución uniforme desde la barra de menú, solo nos permitirá cambiar el primero. Si hemos seleccionado distribución normal podremos cambiar los dos.

Cada vez que pulsamos el botón OK se generará una nueva distribución, borrando la anterior junto con la PT si estuviera dibujada.

Al pulsar el botón de ejecución (la pequeña cuña verde) se calculará la nueva PT según el método seleccionado actualmente.

Panel de estadísticas:

Muestra los valores característicos de la PT calculada:

Peso (Weight) : la longitud total en pixels de las aristas de la PT.

Grado de los vértices (V degree) : el máximo número de aristas concurrentes en un vértice.

Grado de las caras (F degree) : el máximo número de aristas de una cara

Nº de puntos (Points) : el número de puntos de la PT

Nº de caras (Faces) : el número de caras totales de la PT

Nº de triángulos (Triangles) : el número de caras que son triángulos

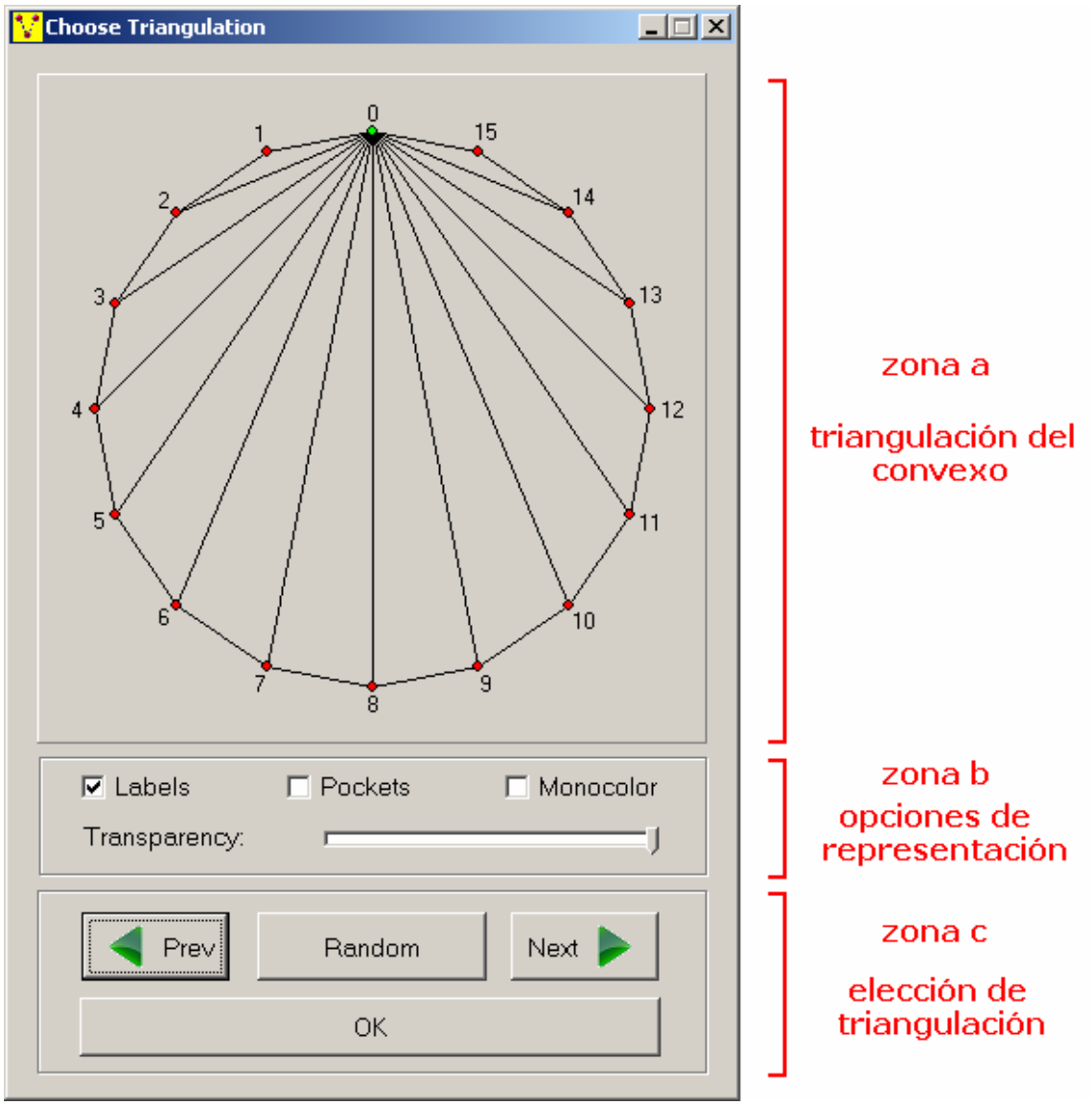
6.5 Ventanas auxiliares

Ventana de selección de triangulación

Algunos métodos de cálculo de la PT requieren la entrada de información adicional por parte del usuario, pues en algunos pasos del algoritmo se introduce cierta arbitrariedad y en general no producen una solución única. La aplicación resuelve esta contingencia presentando una pantalla interpuesta que permita al usuario establecer estas condiciones adicionales, con la posibilidad de cambiarlas cuantas veces se estime necesario antes de dar paso a la ejecución del método. Incluso una vez calculada y dibujada la PT, nos permite invocar de nuevo la ventana para probar con otro conjunto de condiciones.

Es el caso de los métodos incremental (en la vertiente de caras de grado acotado) y caminos mínimos. Ambos métodos invocan la citada ventana de selección; no obstante, algunas de las funciones accesibles solo están habilitadas para el método de caminos mínimos, pues carecen de sentido en el método incremental.

Echemos un vistazo a los elementos constitutivos de la ventana:



Zona a

En ella aparece un polígono regular de n lados con los vértices numerados en sentido antihorario, comenzando por el 0 en la parte superior. La triangulación inicial que se presenta es por abanico, con el vértice 0 siendo el punto radial. Cada vez que se elija una triangulación diferente, el dibujo cambiará para reflejarla.

Zona b

Aquí contamos con 3 casillas de verificación marcables y desmarcables a voluntad más una barra de deslizamiento ajustable en toda su longitud.

Las casillas solo están habilitadas en el método de caminos mínimos, mientras que la barra puede usarse en cualquier circunstancia. En el método incremental las casillas permanecen en color gris y no responden a las pulsaciones del ratón.

La casilla *Labels* indica al procedimiento de dibujado de la PT que etiquete cada vértice con un número, empezando en 0 por el primer vértice insertado y continuando en sentido antihorario. La etiqueta se colocará en el borde exterior del polígono dibujado.

La casilla *Pockets* indica al procedimiento de cálculo de la PT que debe ejecutar el código para calcular los bolsillos del polígono, de forma que el procedimiento de dibujado pintará la PT completa.

La casilla *Monocolor* indica al procedimiento de dibujado que ignore la configuración de color y grosor de líneas y dibuje todas las aristas de la PT en el mismo color y grosor, para darle un aspecto homogéneo. Esta casilla va ligada a la casilla *Pockets* de tal forma que si elegimos monocolor, siempre se dibujarán los bolsillos. Pero no al revés, si elegimos bolsillos podemos o no elegir monocolor.

La barra deslizante sirve para ajustar el valor de la transparencia de la ventana. Llevándola hacia el extremo derecho la ventana aparecerá completamente sólida. Según la llevamos hacia el extremo izquierdo irá transparentándose más, dejando ver la parte de la pizarra que hay bajo ella.

Zona c

Esta zona contiene 4 botones para controlar la generación de triangulaciones.

El botón *Random* genera una triangulación aleatoria a partir de la triangulación actual, mediante la aplicación de un número aleatorio de flips sobre aristas aleatorias.

El botón *Prev* realiza un flip sobre la triangulación actual en una arista determinada. Cada vez que se pulsa este botón, la arista afectada cambia, en un ciclo que va desde las aristas de numeración más alta hacia las más bajas.

El botón *Next* realiza un flip sobre la triangulación actual en una arista determinada. Cada vez que se pulsa este botón, la arista afectada cambia, en un ciclo que va desde las aristas de numeración más baja hacia las más altas.

Por último, el botón *OK* selecciona la triangulación actual y devuelve el control al programa para que calcule la PT y la dibuje. No obstante, la ventana permanece activa para que si no nos gusta el resultado podamos elegir otra triangulación. La ventana solo se cerrará si explícitamente la cerramos con el icono de la parte superior derecha.

Ventana de autor:

Proporciona información sobre el autor y la versión de la aplicación.

7. Conclusión

El motivo de la elección de este trabajo por parte del autor se debe a la fascinación que de siempre le ha suscitado la Geometría, especialmente el campo de la geometría plana euclidiana. Con el advenimiento de los ordenadores y el desarrollo de la Geometría computacional las posibilidades de exploración del campo se han visto incrementadas exponencialmente, permitiendo la creación de herramientas visuales que simplifican y ayudan enormemente en la labor de experimentación del matemático actual. Así pues, el desarrollo del proyecto ha resultado ser una agradable (si bien estresante por momentos) experiencia en la que se ha aunado la afición por las matemáticas teóricas con la pasión por la programación de ordenadores, cuyo resultado es esta herramienta que viene a sumarse a todas aquellas que anteriores aficionados a la Geometría realizaron con el noble propósito de ayudar en su trabajo a los estudiosos de esta ciencia. El autor espera que su pequeña aportación sirva al mismo fin en la medida en que puede hacerlo.

Por supuesto que la realización del presente trabajo ha supuesto la superación de numerosos obstáculos:

De una parte, la escasa bibliografía en la que documentarse, debido al ya mencionado carácter reciente de la disciplina bajo estudio (pseudo-triangulaciones). Justo es decir, sin embargo, que la labor del tutor ha sido encomiable al seleccionar y proporcionar los textos necesarios para la comprensión de los conceptos y el funcionamiento de los distintos algoritmos a implementar. De la misma manera sus explicaciones han resuelto cualquier duda que el autor pudiera tener en el plano puramente teórico, dejando a su propia inventiva la resolución de los problemas de programación que surgieran en el desarrollo.

Por otra parte, también resultan contadas las fuentes en Internet donde encontrar ejemplos con código relativo a métodos de pseudo-triangulación o contenido similar que pudiera ayudar en la programación del código propio. Además, las pocas que existen o bien contienen textos y explicaciones farragosos o bien usan estructuras de datos que no se ajustan bien a las propias necesidades. La consecuencia directa de esta escasa disponibilidad de material de referencia es que la mayoría del código final de la aplicación es original del autor, como por otra parte se supone que debe ser en un proyecto de fin de carrera.

Al hilo de esto, resulta interesante destacar que el espíritu que ha guiado en todo momento la realización del proyecto ha sido el de resolver todos los problemas de implementación por los propios medios. Partiendo, no diremos de cero, pero si de un punto lo suficientemente bajo como para necesitar definir los tipos complejos, las estructuras de datos geométricas, y otra serie de elementos que quizá ya existan en las colecciones o en las librerías de clases de alguno de los modernos lenguajes de programación, pero que el autor ha preferido modelar a su gusto en lugar de intentar adaptar a las propias necesidades objetos ya existentes. En primer lugar, porque de esta manera es como realmente se aprende, intentando hacer frente a las dificultades por uno mismo. En segundo lugar, porque un proyecto de fin de carrera es el escenario ideal para poner en práctica todos los conocimientos que se han ido acumulando a lo largo del ciclo de estudios. En este sentido, nos parece una traición al espíritu del proyecto el realizar una mera labor de gestión de material realizado por terceras personas.

Para terminar, nos gustaría señalar que aún hay mucho espacio en este campo para la investigación y la experimentación. Este mismo trabajo puede servir de base a proyectos más ambiciosos para futuros estudiantes, que podrían ampliarlo con sus propios conocimientos para producir herramientas más poderosas que permitan, por qué no pensarlo así, llegar a interesantes demostraciones que aumenten el conocimiento matemático acumulado o que puedan servir para el mejoramiento de los algoritmos o procesos utilizados en la técnica, la industria o cualesquiera otras actividades humanas.

8. Bibliografía

[1] G. Carmichael, "A survey of Delaunay Triangulations. Algorithms and Applications", 2008.

[2] E.O. Gagliardi, "Geometría Computacional y Metaheurísticas: Aproximaciones sobre Pseudotriangulaciones". Plan de tesis doctoral. Universidad Nacional de San Luis, Argentina. 2009.

[3] C. Harrison, "An investigation of Graham's Scan and Jarvis' March". http://www.chrisharrison.net/projects/convexHull/index.html

[4] R. Holmes, "The DCEL data structure for 3D Graphics". http://www.holmes3d.net/graphics/dcel/

[5] L. Kettner, D.G. Kirkpatrick, A. Mantler, J. Snoeyink, B. Speckmann, F. Takeuchi, "Tight degree bounds for pseudo-triangulations of points", 2003, pp.3-12.

[6] D. Lischinski, "Incremental Delaunay Triangulation". Graphic Gems IV. 1994.

[7] M. Pocchiola, G. Vegter, "Pseudo-triangulations: theory and applications", 1996, pp.291-300.

[8] M. Pocchiola, G. Vegter, "Topologically Sweeping Visibility Complexes via Pseudotriangulations", Discrete & Computational Geometry 1996, pp.419-453.

[9] D. Randall, G. Rote, F. Santos, J. Snoeyink, "Counting triangulations and pseudo-triangulations of wheels", CCCG, 2001, pp.149-152.

[10] G. Rote, F. Santos, I. Streinu, "Pseudo-Triangulations – a Survey. Surveys on Discrete and Computacional Geometry: Twenty years later", 2006.

[11] G. Rote, A. Schulz, "A pointed Delaunay pseudo-triangulation of a simple polygon", EuroCG, 2005, pp.77-80.

[12] J.R. Shewchuk, "Triangulation Algorithms and Data Structures". http://www.cs.cmu.edu/~quake/tripaper/triangle2.html