CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Matemáticas e Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Generador de mallas de regiones poligonales

Autor: Xiang Chen Chen

Director: Manuel Abellanas Oar

MADRID, JUNIO 2018

Resumen

En este trabajo se estudiará y se implementará un algoritmo para la creación de mallas de triángulos 2-dimensionales.

Las mallas triangulares son muy utilizadas en diversos campos de la ingeniería, principalmente se usan para la aplicación del método de los elementos finitos [1] y, más recientemente, están siendo incorporadas en algoritmos de reconocimiento de imágenes [12].

En ocasiones basta con utilizar métodos de triangulación de polígonos para crear las mallas necesarias, sin embargo, cuando se impone la condición de que no existan ángulos muy pequeños es necesario añadir más vértices a la triangulación.

Nuestro método automático de generación de mallas se basará en la generación de una malla inicial mediante algoritmos para generar Triangulaciones de Delaunay restringidas, la inserción de puntos de Steiner y el movimiento de éstos mediante el método de las fuerzas.

Se implementará dicho algoritmo en Python para su visualización y uso como librería.

Abstract

In this project we will study and implement an algorithm for the creation of two-dimensional triangle meshes.

Triangle meshes are used in many fields, one important application of such meshes is solving complex engineering problems using the finite element method [1]. In recent years, they are also beginning to play a role in some image recognition algorithms [12].

In some cases, it is enough to use polygon triangulation methods to achieve our goal, but when higher quality triangles are needed it is necessary to add more points to the mesh.

Our automatic mesh generation algorithm will start with a Constrained Delaunay triangulation and modify it by adding Steiner points and moving them using the forces method.

The algorithm will be implemented as a Python library where the user will also be able to visualize it.

Índice general

1. INTRODUCCIÓN

Uno de los primeros algoritmos para la generación de mallas triangulares regulares fue el algoritmo de refinamiento que propuso Jim Ruppert en 1995 [2] para reducir el tiempo de cómputo del método de los elementos finitos.

El método de los elementos finitos es un método numérico que aproxima la solución de ecuaciones diferenciales parciales complejas. Su convergencia y estabilidad es afectada por la presencia de triángulos con ángulos muy dispares y, además, su tiempo de cómputo aumenta dependiendo del número de triángulos que forman la malla.

El algoritmo de Ruppert se basa en la inserción de puntos de Steiner en la triangulación, concretamente, inserta iterativamente los circuncentros de los triángulos que poseen algún ángulo menor al indicado y añade puntos sobre aristas consideradas "malas".

A pesar de ser uno de los algoritmos de refinamiento más populares, en algunos casos el resultado está lejos de ser el óptimo. Podemos ver un ejemplo de ello en el artículo de Alper Üngor publicado en 2004, donde propone un nuevo algoritmo basado en "off-centers" y realiza la comparación entre ambos [3]:

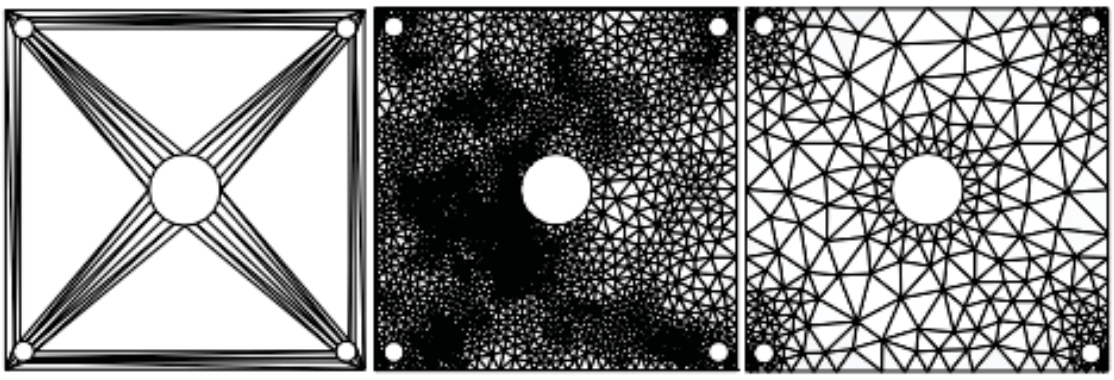


Figura 1.1: La triangulación inicial (a) tiene ángulo mínimo de 1º. El ángulo mínimo tanto en (b) como en (c) es 34º. Sin embargo *triangle*(b) inserta 1984 puntos de Steiner mientras que el método de los "offcenters" inserta solamente 305.

Por ello, se plantea en este trabajo crear un algoritmo similar al de Ruppert, basado en la inserción de baricentros y el movimiento de éstos mediante el método de las fuerzas.

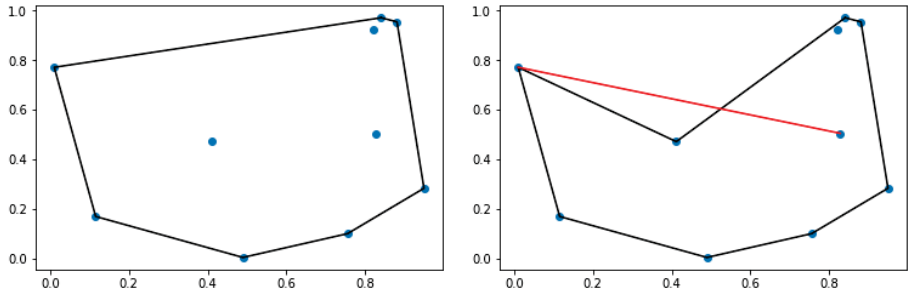
2. TRIANGULACIONES

2.1. Definiciones básicas

Definición. *Un **polígono** es una región acotada del plano cuya frontera es una curva poligonal simple de Jordan.*

Definición. *Un conjunto de puntos en el plano es **convexo** si para cualquier par de puntos del conjunto, el segmento que los une está contenido en él.*

Definición. *El **cierre convexo** de un conjunto de puntos es el menor conjunto convexo que lo contiene.*

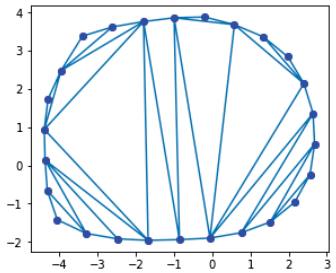


(a) Cierre convexo

(b) No cierre convexo

Definición. *Sea $P := \{p_1, p_2, ..., p_n\}$ un conjunto de puntos en el plano. Llamamos **subdivisión planar maximal** de P a una subdivisión S de su cierre convexo determinada por aristas rectilíneas que conectan pares de puntos de P tal que toda arista que no se encuentra en S interseca a una de las existentes.*

Definición. *Una **triangulación** de P es una subdivisión planar maximal de los vértices P. [7]*

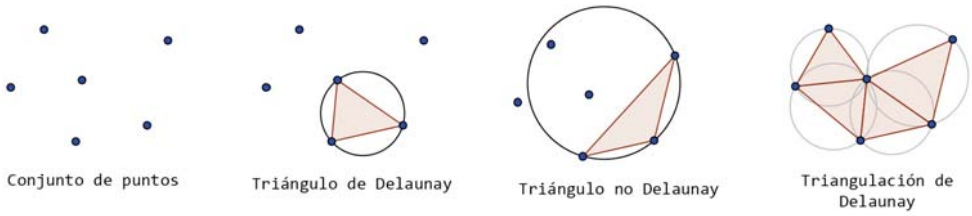


Definición. *Un grafo **plano** o planar es un grafo que puede ser representado en el plano de manera que ninguna arista corte a otra.*

2.2. Triangulación de Delaunay

Definición. *Una triangulación T es una* ***triangulación de Delaunay*** *si ninguna circunferencia circunscrita de un triángulo contiene a un vértice de otro triángulo.*

Tiene la propiedad de maximizar el ángulo mínimo de la triangulación [8], una propiedad imprescindible para generar mallas de buena calidad.



Fuente: mathmunch.org

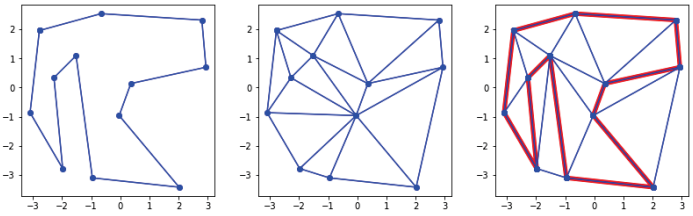
Definición. *Una arista es* ***legal*** *si sus dos triángulos adyacentes cumplen la condición de Delaunay.*

2.3. Triangulación Restringida de Delaunay

Definición. *Dado un grafo plano G, una* ***triangulación restringida*** *de G es una triangulación de sus vértices que incluye las aristas de G como parte de la triangulación.*

Definición. *Una triangulación T es una* ***triangulación restringida de Delaunay*** *(CDT) de G si toda arista de G pertenece a T y para cada arista e restante existe una circunferencia C con las siguientes propiedades: [4]*

1. Los vértices de e se encuentran en el borde de C.
2. Si un vértice v de G se encuentra en el interior de C entonces no es "visible"por uno de los vértices de la arista e. Es decir, al menos uno de los segmentos $\overline{ve_1}$, $\overline{ve_2}$ interseca con una arista de G.



(a) Grafo inicial (b) T. Delaunay (c) CDT

3. DCEL

La principal estructura de datos que se usará para representar los grafos será la **lista de aristas doblemente enlazada**, más conocido como DCEL o **half-edge data structure**.

Esta estructura fue descrita por primera vez por D.E. Müller y F.P. Preparata en 1978 [5], permite representar y manipular grafos planos eficientemente. Se trata de una estructura bastante popular y es usada por librerías importantes como *OpenMesh* [9].

La estructura está formada principalmente por 3 sublistas:

3.1. Vértices

Cada vértice se representa con un nodo que contiene sus coordenadas y una referencia a una semiarista que sale de él.

3.2. Aristas

Cada arista del grafo se representa como dos semiaristas dirigidas con sentido contrario, decimos que una semiarista es gemela de otra si ambas proceden de la misma arista inicial.

Cada nodo arista contiene una referencia a:

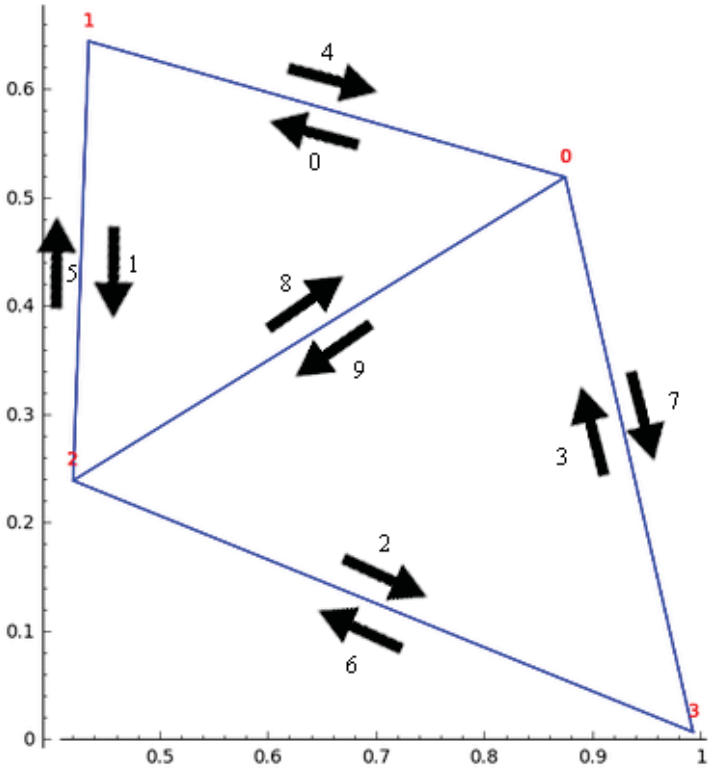
- Su vértice origen
- Su semiarista gemela
- Una semiarista previa
- Una semiarista posterior
- La cara a la que pertecene.

3.3. Caras

Para representar una cara, sólo es necesaria una referencia a una semiarista perteneciente a ella.

3.4. Ejemplo

Veamos un ejemplo de un DCEL junto a su representación gráfica:




```
V =[[0.874, 0.518], 0],
   [[0.434, 0.643], 1],
   [[0.420, 0.238], 2],
   [[0.992, 0.006], 3]]

A = [[0, 4, 8, 1, 1],
     [1, 5, 0, 8, 1],
     [2, 6, 9, 3, 2],
     [3, 7, 2, 9, 2],
     [1, 0, 5, 7, 0],
     [2, 1, 6, 4, 0],
     [3, 2, 7, 5, 0],
     [0, 3, 4, 6, 0],
     [2, 9, 1, 0, 1],
     [0, 8, 3, 2, 2]]

C = [4, 8, 9]
```

- El vértice 0 representado por [[0.874, 0.518], 0] indica que:
  - Se encuentra en las coordenadas x = 0.874, y = 0.518
  - La semiarista con índice 0 sale de ella.
- La semiarista 0 representada por [0, 4, 8, 1, 1] indica que:
  - Sale del vértice 0
  - Su gemela es la semiarista 4
  - Su semiarista anterior es la 8
  - Su semiarista posterior es la 1
  - Se encuentra en la cara 1
- La semiarista 4 pertenece a la cara 0, la semiarista 8 a la cara 1 y la semiarista 9 a la cara 2.

También podemos observar que las caras internas de la triangulación deben tener una orientación positiva mientras que la cara 0 o externa, una orientación negativa.

4. Lista de triángulos

Otra estructura de datos con la que se trabajará será la lista de triángulos.

Está compuesta por 2 sublistas:

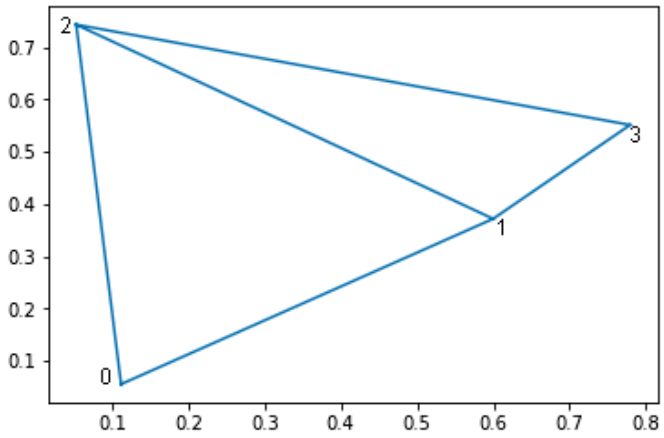
4.1. Vértices

Cada nodo vértice contiene las coordenadas del punto.

4.2. Triángulos

Para cada triángulo se guardan las referencias de los puntos que lo forman y referencias a 3 triángulos adyacentes.

4.3. Ejemplo




```
Puntos = [[0.111  0.055]
          [0.599  0.371]
          [0.052  0.742]
          [0.779  0.550]]

Simplices = [[1  3  2]
             [0  1  2]]

Vecinos = [[-1   1  -1]
           [ 0  -1  -1]]
```

- El símplice [1,3,2] indica que existe un triángulo formado por los vértices 1, 3 y 2.
- La lista [-1,1,-1] indica que para el símplice [1,3,2], la cara contraria al vértice 1 es la -1 (cara externa), la cara contraria al vértice 3 es la cara 1 y la contraria al vértice 2 es de nuevo la -1 (cara externa).

5. Inserción de puntos

Uno de los principales mecanismos que usaremos para mejorar los ángulos de los triángulos de la malla será la inserción puntos.

Dependiendo del tipo de triángulo que debemos mejorar y de si alguna arista suya pertenece a la frontera del polígono, insertaremos un punto en el baricentro del triángulo o partiremos una de sus aristas en dos.

5.1. Triángulo sin aristas restringidas

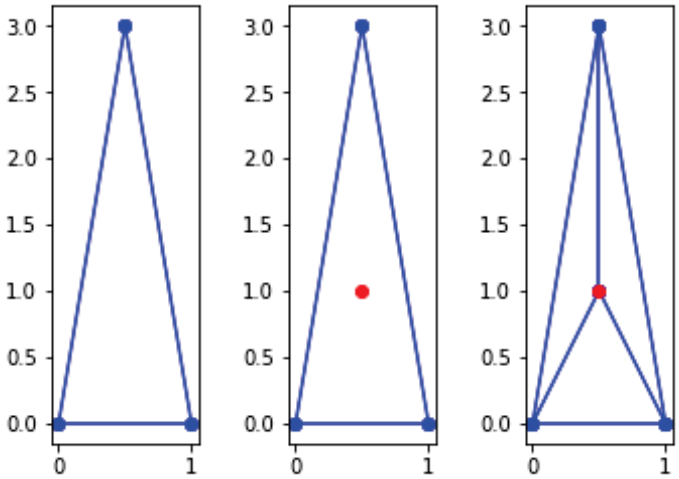
Si se trata de un triángulo sin aristas restringidas, insertaremos su baricentro.

Definición. *Una **mediana** de un triángulo es un segmento que une uno de sus vértices con el punto medio de su arista opuesta.*

Definición. *El **baricentro** o centroide de un triángulo es el punto donde intersecan sus tres medianas.*

La forma de calcular las coordenadas del baricentro G de un triángulo formado por los vértices A,B,C es:

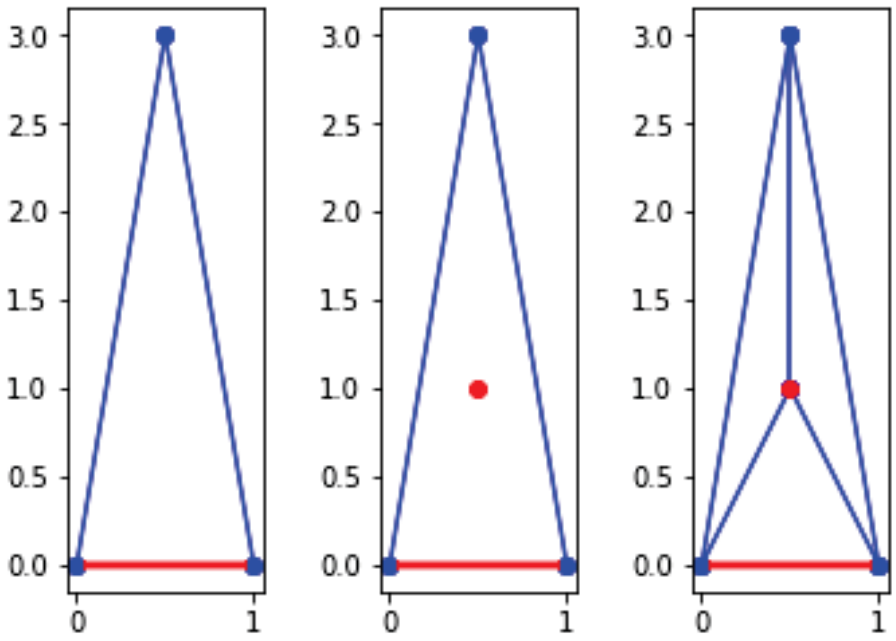
$$G = (\frac{x_A + x_B + x_C}{3}, \frac{y_A + y_B + y_C}{3})$$



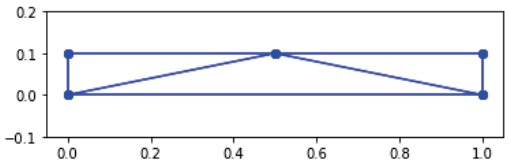
5.2. Triángulo con aristas restringidas

En el caso de que una de las aristas del triángulo sea restringida deberemos analizar dónde se encuentran los ángulos pequeños.

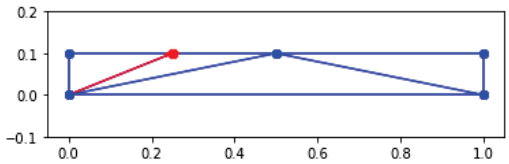
- Si el único ángulo pequeño se encuentra en el vértice opuesto a la arista restringida, añadiremos el baricentro del triángulo como en el caso anterior.



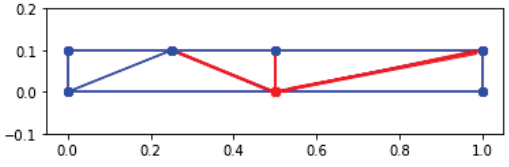
- Si la arista restringida forma parte de un ángulo pequeño, partiremos la arista por la mitad.



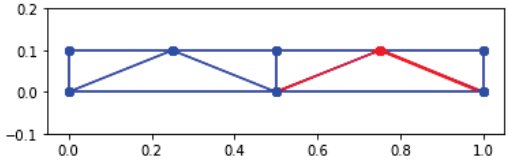
(a) Polígono inicial



(b) Partición de una arista



(c) Partición de una arista



(d) Partición de una arista

6. Método de las fuerzas

Si consideramos la malla generada como si se tratase de una celosía plana o una estructuras de muelles, podríamos calcular la "fuerza" que ejercen las aristas adyacentes a un vértice. En nuestro caso, sólo se moverán los vértices insertados por el algoritmo, los vértices originales no serán modificados.

Dado un vértice v y sea A el conjunto de sus vértices adyacentes, calcularemos el vector suma de la siguiente manera:

$$\sum_{w \in A} v\vec{w}$$

Una vez calculada esta suma, tendremos que identificar de qué tipo de vértice se trata: un vértice interior al polígono o un vértice en la frontera del polígono resultante de la partición de una arista.

Definición. *El **centroide** de un conjunto de finito de k puntos $x_1, x_2, ..., x_k$ es el punto:*

$$C = \frac{x_1 + x_2 + ... + x_k}{k}$$

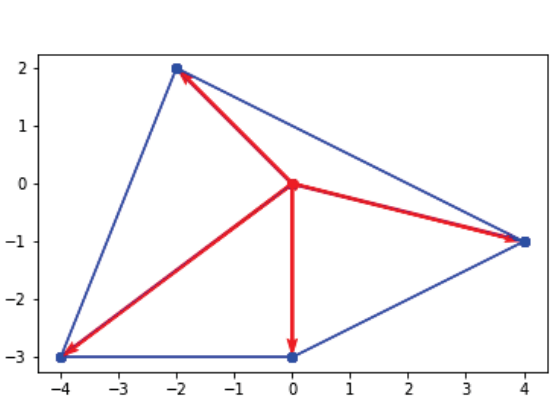
Tiene la propiedad de minimizar la suma de los cuadrados de las distancias entre él y los puntos del conjunto. [14]

6.1. Vértice interior al polígono

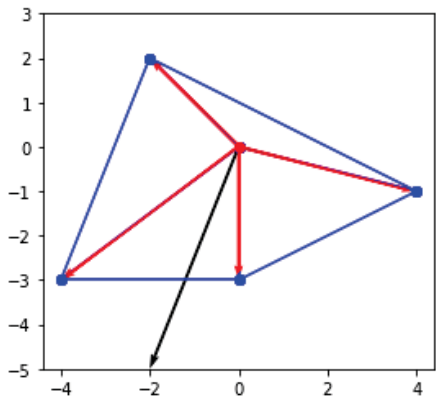
Si movemos el vértice en la dirección del vector suma tras multiplicarlo por un escalar arbitrario ≤ 1, estaremos aproximando el centroide de los vértices adyacentes al punto estudiado y evitando la aparición de triángulos con ángulos muy pequeños.

Veamos un ejemplo:

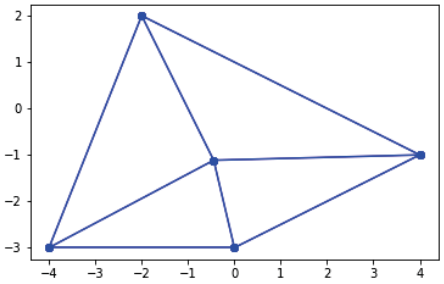
- Punto a estudiar: (0,0)
- Puntos adyacentes: (0,-3),(-2,2),(4,-1),(-4,-3)
- Vector suma: (-2,-5)



(a) Punto (0,0) junto sus vectores



(b) Suma de vectores



(c) Vértice desplazado

Figura 6.1: El ángulo mínimo de la triangulación pasa de (a) 12.5º a 27.9º (c).

En este caso, tras 10 iteraciones, el punto (0,0) ha sido desplazado a la posición (-0.44, -1.11), siendo el centroide del polígono el punto (-0.5, -1.25)

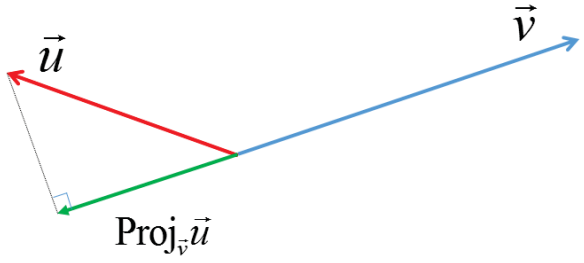
6.2. Vértice resultante de partir una arista

Si el vértice que queremos mover se encuentra en una arista restringida, calcularemos el vector suma de la misma forma que en el caso anterior. Sin embargo, no podemos moverlo directamente ya que se modificaría la arista restringida original.

Por ello, el vector que tomaremos para desplazar el punto será la proyección del vector suma sombre la recta en la que se encuentra la arista.

La **proyección vectorial** de un vector $\vec{v}$ sobre otro vector $\vec{u}$ se calcula de la siguiente forma:

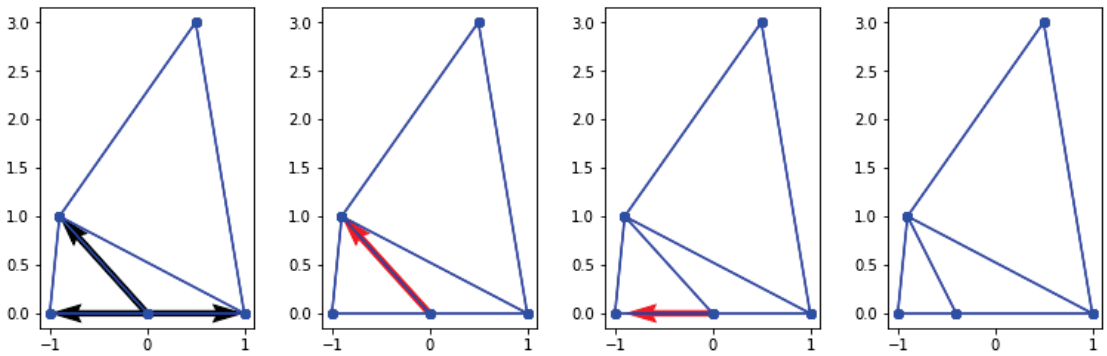
$$proy_{\vec{v}}(\vec{u}) = \frac{\vec{v} \cdot \vec{u}}{\vec{v} \cdot \vec{v}} \vec{v}$$



Fuente: http://jccc-mpg.wikidot.com

En este caso, proyectaremos el vector suma del punto p sobre un vector de la arista restringida, por ejemplo $\vec{pq}$ siendo q un vértice de la arista restringida original.

Ejemplo:



(a) Vectores (b) Vector suma (c) Proyección (d) Desplazamiento

Figura 6.2: El ángulo mínimo pasa de 20.25º a 27.75º.

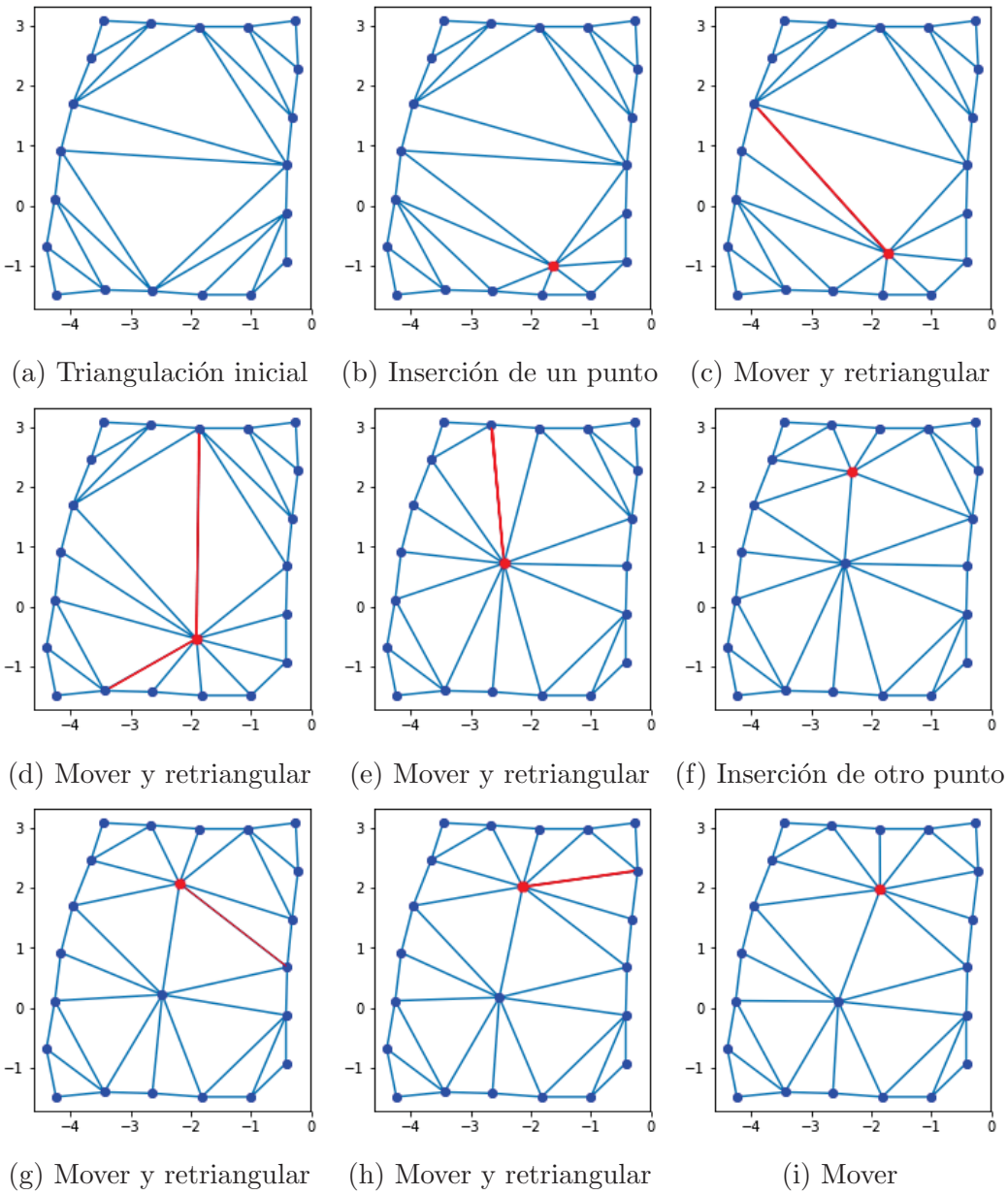
7. Algoritmo

El algoritmo implementado es el siguiente:

Dado un polígono y un ángulo mínimo α:

1. Crear la triangulación de Delaunay restringida del polígono inicial.
2. Mientras el ángulo mínimo de la triangulación o malla sea $< \alpha$:
   a) Buscar la cara o triángulo con el menor ángulo de la malla.
   b) Analizar el tipo del triángulo:
      - Si se trata de un triángulo interior sin aristas restringidas, insertar el baricentro.
      - Si se trata de un triángulo con alguna arista restringida:
        - Partir la arista si la arista restringida forma parte de uno de los ángulos pequeños.
        - En otro caso, añadir el baricentro.
   c) Calcular la triangulación de Delaunay restringida de los puntos nuevos.
   d) Aplicar el método de las fuerzas con todos los puntos y retriangular.
3. Devolver la malla generada.

7.1. Ejemplo



(a) Triangulación inicial (b) Inserción de un punto (c) Mover y retriangular

(d) Mover y retriangular (e) Mover y retriangular (f) Inserción de otro punto

(g) Mover y retriangular (h) Mover y retriangular (i) Mover

Figura 7.1: Ejemplo del algoritmo en acción

8. Implementación en Python

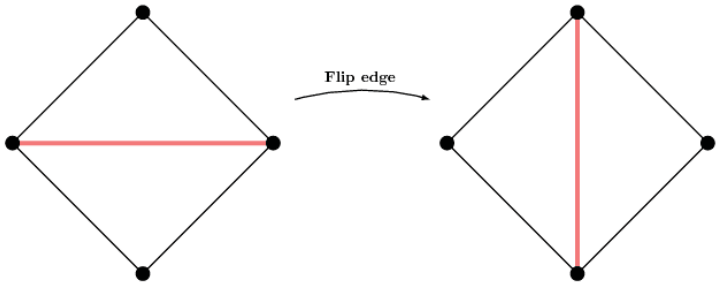
Todo el código se encuentra disponible en la siguiente dirección de Github: `https://github.com/stradivari96/Automatic-mesh-generation`.

8.1. Funciones auxiliares

Algunas de las funciones auxiliares implementadas son las siguientes:

8.1.1. Giro de aristas

Definición. *Sea una arista $e = \overline{p_i p_j}$ de una triangulación T. Si e forma parte de dos triángulos $p_i p_j p_k$ y $p_i p_j p_l$, y, además, estos triángulos forman un cuadrilátero convexo, podemos obtener una nueva triangulación eliminando la arista $\overline{p_i p_j}$ e insertando la arista $\overline{p_k p_l}$, es decir, cambiando una diagonal por la otra. Llamamos a esta operación* ***giro de arista****. [7]*



Fuente: https://stackoverflow.com

8.1.2. Área signada

El área signada de un triángulo formado por los vértices A,B,C se calcula de la siguiente forma:

```
def aux_sarea(A,B,C):
  return ((B[0]-A[0])*(C[1]-A[1])-(C[0]-A[0])*(B[1]-A[1]))/2
```

El signo del resultado indica la orientación de los puntos.

8.2. Triangulación inicial del polígono

El primer paso necesario para generar la malla es obtener una triangulación inicial del polígono.

8.2.1. Triangulación de Delaunay

Dado un conjunto inicial de puntos, haremos uso de la función **Delaunay** del módulo scipy.spatial [10] para crear una triangulación de Delaunay de éstos. Esta función hace uso a su vez de la librería *qhull* [13], una de las librerías más usadas para el cálculo de cierres convexos y triangulaciones.

Tras ello obtenemos una lista de triángulos orientados positivamente que pasaremos a DCEL de la siguiente manera:

1. Copiar la lista de vértices.
2. Crear un diccionario para guardar las referencias de las semiaristas creadas.
3. Para cada triángulo (a,b,c) de la lista de triángulos:
   a) Crear una nueva cara.
   b) Crear las semiaristas $\overline{ab}$, $\overline{bc}$, $\overline{ca}$ y añadirlas al diccionario.
   c) Añadir las semiaristas a los nodos de los vértices correspondientes:
      - Vértice a $\rightarrow$ semiarista $\overline{ab}$.
      - Vértice b $\rightarrow$ semiarista $\overline{bc}$.
      - Vértice c $\rightarrow$ semiarista $\overline{ca}$.
   d) Completar los campos de cada semiarista (posterior,anterior,origen,cara).
   e) Comprobar si la gemela de cada semiarista está en diccionario, en caso afirmativo, añadir las referencias a sus gemelas.
4. Terminar creando las semiaristas de la cara externa con ayuda del cierre convexo de los puntos.

8.2.2. Triangulación restringida

Una vez obtenida la triangulación de Delaunay de los puntos, se impondrán las aristas que forman parte del polígono. Para ello se ha implementado el algoritmo de Sloan [6] de complejidad $O(n^k)$ siendo k una constante en el intervalo [1.12,1.29] que depende del número de restricciones. El algoritmo descrito es el siguiente:

1. (Recorrer la lista de aristas restringidas.) Para cada arista restringida definida por los vértices V_i y V_j, realizar los pasos 2-4.

2. (Buscar aristas que la corten.) Si la arista $V_i - V_j$ está presente en la triangulación, volver al paso 1. En otro caso, marcar todas las aristas que intersecan con $V_i - V_j$.

3. (Girar aristas.) Mientras alguna arista corte a $V_i - V_j$ realizar los siguientes pasos:

   a) Seleccionar una arista $V_k - V_l$ de la lista de aristas que cortan con $V_i - V_j$.

   b) Si los triángulos que comparten la arista $V_k - V_l$ no forman un cuadrilátero convexo, es decir, la arista $V_k - V_l$ no se puede girar, poner de nuevo $V_k - V_l$ en la lista de aristas cortantes y volver al paso anterior. En otro caso, girar la arista y dependiendo de si sigue intersecando con $V_i - V_j$ añadirla a una lista de aristas nuevas, o de nuevo a la lista de aristas que intersecan con $V_i - V_j$.

4. (Restaurar la triangulación de Delaunay.) Repetir los siguientes pasos hasta que no se modifique ninguna arista:

   a) Recorrer la lista de nuevas aristas.

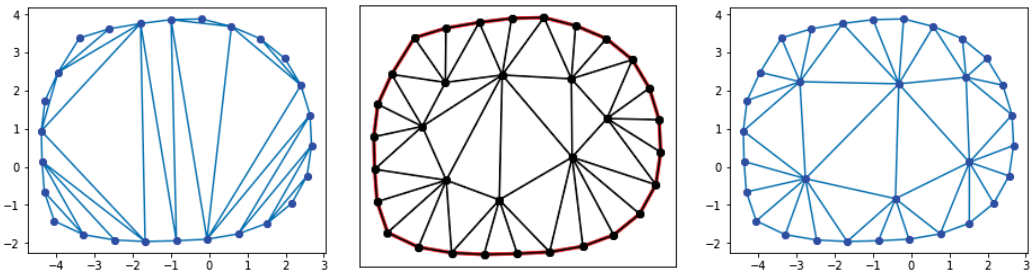
   b) Si la arista $V_k - V_l$ coincide con la arista restringida $V_i - V_j$, volver al paso anterior.

   c) Si la arista $V_k - V_l$ no es legal, girarla y volver a añadirla a la lista de aristas nuevas.

9. Pruebas

En este apartado compararemos nuestro algoritmo con el generador de mallas que proporciona *Triangle* [11], una de las librerías más conocidas y más usadas para la generación de mallas y triangulaciones de Delaunay.

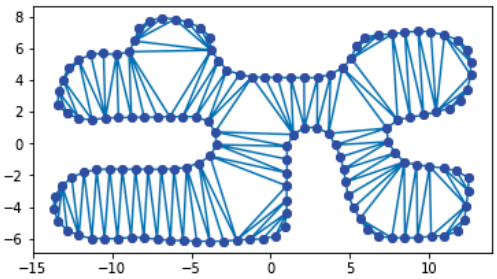
9.1. Polígono con forma convexa



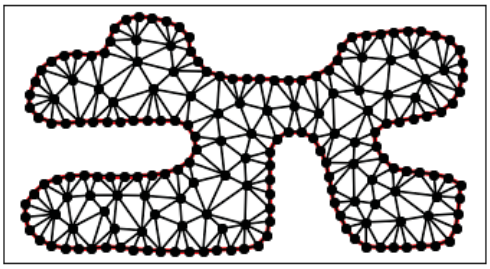
(a) Polígono inicial (b) Triangle (c) Método Fuerzas

- Número de puntos del polígono inicial: 26
- Mínimo ángulo α: 20
- Vértices añadidos (Triangle): 8
- Vértices añadidos (Fuerzas): 6
- Tiempo de ejecución (Fuerzas): 0.19 segundos

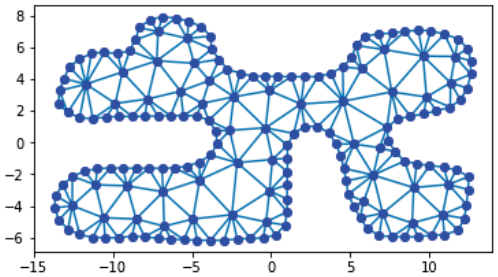
9.2. Polígono no convexo



(a) Polígono inicial



(b) Triangle



(c) Método Fuerzas

- Número de puntos del polígono inicial: 134
- Mínimo ángulo α: 20
- Vértices añadidos (Triangle): 57
- Vértices añadidos (Fuerzas): 47
- Tiempo de ejecución (Fuerzas): 9.31 segundos

10. Conclusiones

Hemos diseñado en este trabajo un algoritmo alternativo para generar mallas de regiones poligonales. Hemos comprobado también que este método, basado en el movimiento de los vértices insertados iterativamente, puede generar triangulaciones con un número menor de puntos que las generadas por la librería *Triangle*, sin embargo, el tiempo de ejecución también es mucho mayor.

En trabajos posteriores se podría implementar el algoritmo en algún lenguaje compilado como C para evitar los costes de tiempo que conlleva usar un lenguaje de programación interpretado como Python. Por otro lado, también se podría realizar un estudio formal para encontrar el factor óptimo por el cual multiplicar el vector suma a la hora de desplazar los puntos, así como el número de iteraciones óptimo entre inserciones de puntos. Otro posible estudio podría ser la generalización del algoritmo a polígonos con agujeros.

Referencias

[1] Sun, L., Yeh, GT., Ma, X. et al. (2017). *Engineering applications of 2D and 3D finite element mesh generation in hydrogeology and water resources.* Computational Geosciences.

[2] Ruppert, J. (1995). *A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation*. Journal of Algorithms.

[3] Üngör, A. (2004) *Off-centers: A new type of Steiner points for computing size-optimal quality-guaranteed Delaunay triangulations.* Springer, Berlin, Heidelberg.

[4] Chew, L. Paul (1987). *Constrained Delaunay Triangulations.* Proceedings of the Third Annual Symposium on Computational Geometry.

[5] Muller, D.E. y Preparata, F.P. (1978) *Finding the intersection of two convex polyhedra.* Theorical Computer Science. North-Holland Publishing Company.

[6] Sloan, S.W. (1993) *A fast algorithm for generating constrained Delanay triangulations.* Computers & Structures Vol.47, No.3.

[7] de Berg, M., Cheong, O., van Kreveld, M. y Overmars M. (2008) *Computational Geometry: Algorithms and Applications*. Berlín, Springer.

[8] R. SIBSON, *Locally equiangular triangulations*, Comput. J., 21 (1978), pp. 243-245.

[9] OpenMes.h *What is OpenMesh?* [Online]. Available: `https://www.openmesh.org/intro/`

[10] Scipy. *scipy.spatial.Delaunay* [Online]. Available: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Delaunay.html`

[11] Triangle. *A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator.* [Online]. Available: `https://www.cs.cmu.edu/~quake/triangle.html`

[12] OpenCV. *Delaunay Triangulation and Voronoi Diagram using OpenCV* `https://www.learnopencv.com/delaunay-triangulation-and-voronoi-diagram-using-opencv-c-python/`

[13] Qhull. *Qhull code for Convex Hull, Delaunay Triangulation, Voronoi Diagram, and Halfspace Intersection about a Point* [Online] Available: http://www.qhull.org/

[14] Hervé Abdi. *Centroid* [Online] Available: https://www.utdallas.edu/~herve/abdi-WireCS-Centroid-2009.pdf

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Thu Jun 14 19:31:01 CEST 2018
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)