



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

GENERACIÓN DE MALLAS POLIGONALES A PARTIR DE CAVIDADES

PROPUESTA DE TEMA DE MEMORIA PARA OPTAR AL TÍTULO DE  
INGENIERO CIVIL EN COMPUTACIÓN

NICOLÁS ESCOBAR ZARZAR

MODALIDAD:  
Memoria

PROFESORA GUÍA:  
Nancy Hitschfeld K.

PROFESOR CO-GUÍA:  
Sergio Salinas

SANTIAGO DE CHILE  
2025

# 1. Introducción

Las mallas poligonales son estructuras fundamentales en el modelado geométrico y la simulación computacional. Se definen como colecciones de vértices, aristas y caras que, en conjunto, representan la superficie de un objeto, generalmente mediante una subdivisión en triángulos. Esta representación es ampliamente utilizada en áreas como el diseño asistido por computador (CAD), gráficos por computadora, ingeniería estructural, biomecánica y videojuegos. La eficiencia y calidad de las mallas generadas influye directamente en la precisión de las simulaciones y en el rendimiento computacional de los sistemas que las utilizan.

Uno de los problemas recurrentes en este ámbito es la generación automática de mallas poligonales a partir de un conjunto discreto de puntos en el plano, también conocido como problema de triangulación. Si bien existen algoritmos eficientes como Triangle [1] o Detri2 [2] para formar una triangulación de Delaunay [3], la construcción de formas poligonales a partir de dichas triangulaciones sigue siendo un área de investigación activa. En particular, existe un interés creciente por encontrar métodos que generen mallas con mayor fidelidad geométrica, adaptabilidad local, y que mantengan cualidades deseables como ángulos adecuados y buena distribución de los elementos.

Una triangulación de Delaunay se define como tal si cumple con la propiedad de que para todos los triángulos de la triangulación, se cumple que el circuncírculo del triángulo solo contiene a los vértices de su triángulo respectivo y no los vértices de cualquier otro (ver Figura 1). Estas son de particular importancia porque maximizan el tamaño del ángulo más pequeño de todos los triángulos que la componen, esto es relevante porque previene problemas de precisión que ocurren al tener ángulos muy agudos los cuales propician “casos degenerados” donde los puntos de un triángulo son interpretados como colineales lo cual puede provocar cálculos erróneos e incluso que los programas que trabajen con las mallas resultantes que no sean lo suficientemente robustos fallen por completo en el peor de los casos.

Este trabajo de memoria se enfoca en el desarrollo y análisis de una estrategia alternativa de generación de mallas en dos dimensiones, basada en el concepto de **cavidad** o *concavity* como se describe en el artículo Triangle [1]. La idea central es que, a partir de una triangulación de Delaunay (como la de la Figura 1), se seleccionan ciertos triángulos utilizando un criterio definido por el usuario. Luego, se calculan los circuncentros de estos triángulos, y se identifican los conjuntos de triángulos de la malla cuyo circuncírculo contiene alguno de estos puntos por separado. La unión de las aristas del borde de estos triángulos vecinos para un punto  $p$  forma una **cavidad**, la cual define un nuevo polígono. Este procedimiento permite generar regiones poligonales que pueden servir como base para construir un nuevo tipo de mallas poligonales.

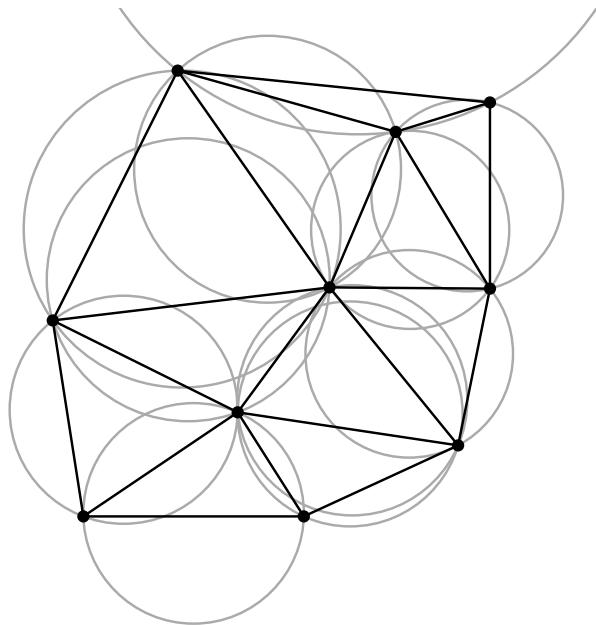


Figura 1: Triangulación de Delaunay con circuncírculos compuesta por 10 vértices.

A modo ilustrativo, en la Figura 2 se presenta una selección de triángulos a partir de la triangulación de la Figura 1.

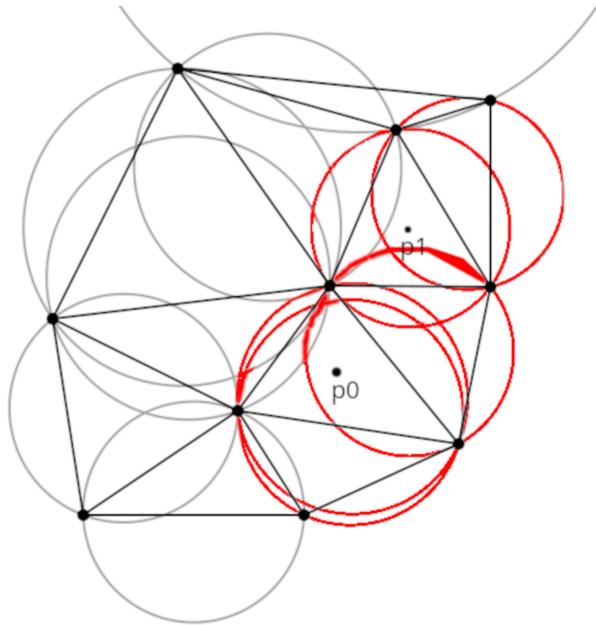


Figura 2: Selección de triángulos para formar un polígono a partir de una cavidad.

En este ejemplo, se seleccionaron dos triángulos  $t_0$  y  $t_1$ , cuyos respectivos circuncentros están representados por los puntos  $p_0$  y  $p_1$ . Estos puntos están contenidos en los circuncírculos de varios triángulos vecinos, incluyendo aquellos que los generaron. La unión de las aristas de borde de los triángulos que contienen a  $p_0$  y la unión de las aristas de borde

de los triángulos que contienen a  $p_1$  forman cada uno una cavidad, las cuales se pueden ver en la Figura 3.

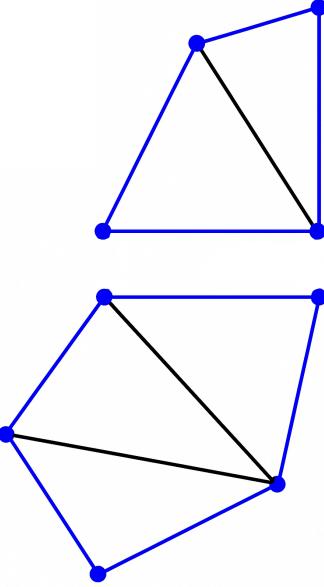


Figura 3: Polígonos resultantes de las cavidades marcados en azul

Este enfoque mediante cavidades no solo busca explorar una nueva forma de construir mallas, sino también comparar su rendimiento y características con métodos existentes. En particular, se contrastará esta técnica con el generador de mallas ***Polylla*** [4], un sistema moderno que emplea estructuras de datos avanzadas para lograr eficiencia y calidad en la generación de polígonos.

La necesidad de desarrollar nuevas estrategias para la generación de mallas responde a múltiples factores: mejorar la eficiencia de los algoritmos existentes, generar elementos con mejores propiedades geométricas, y adaptar las mallas a dominios complejos sin intervención manual.

## 2. Situación Actual

Existen muchos algoritmos para generación de mallas poligonales, siendo los más relevantes para este trabajo Triangle [1] y Polylla [4].

El algoritmo Triangle [1] se basa en el algoritmo de Ruppert [5] y consta de 4 etapas, de las cuales las primeras 2 son exactamente las mismas que las de Ruppert, estas involucran triangular un *planar straight line graph* (o PSLG), un conjunto de vértices y segmentos que describen un polígono como el de la Figura 4, y posteriormente insertar los segmentos originales de la triangulación como se ve en la Figura 5 y la Figura 6.

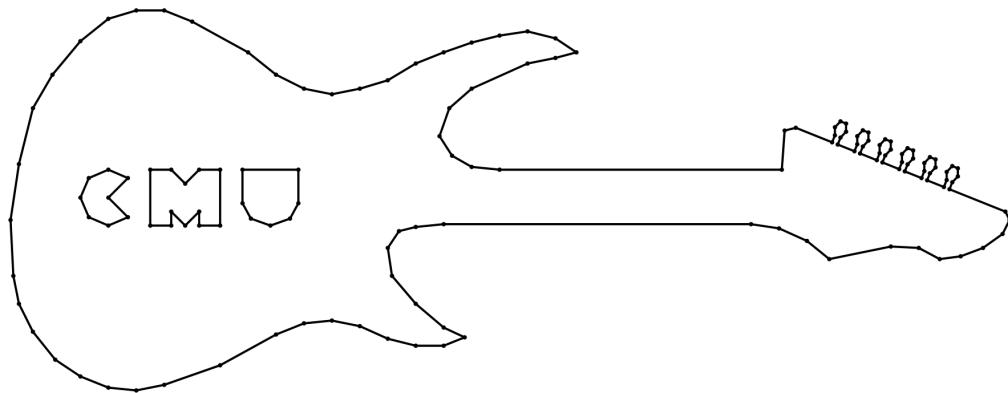


Figura 4: PSLG de entrada una guitarra electrica como se ilustra en Triangle [1]

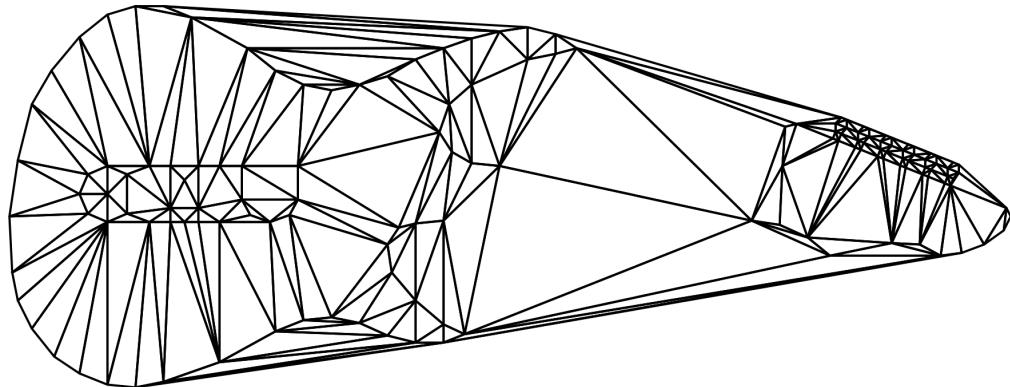


Figura 5: Triangulación del PSLG de la Figura 4 con segmentos originales faltantes [1]

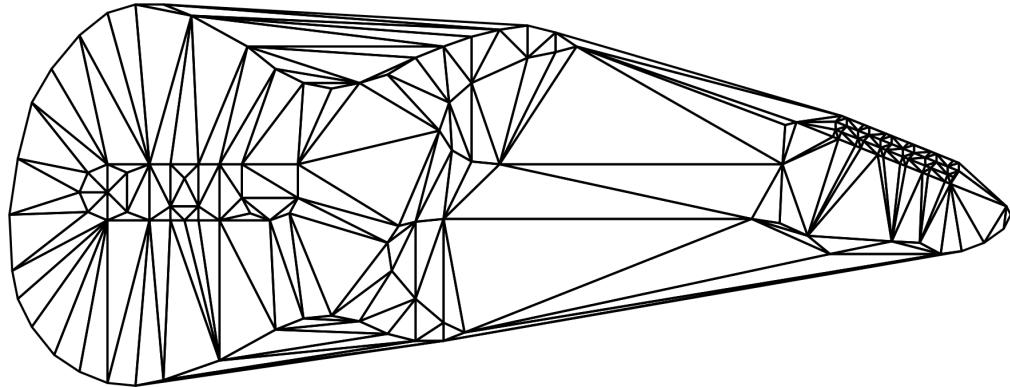


Figura 6: Triangulación restringida del PSLG de la Figura 4 [1]

La tercera etapa difiere del algoritmo de Ruppert y consiste en remover los triángulos extra que están fuera del borde definido por el PSLG original, como aquellos presentes en zonas que originalmente eran no convexas, a las cuales Shewchuk llama concavidades, y los presentes en agujeros (como los del interior del cuerpo de la guitarra en el ejemplo). Esto se ilustra en la Figura 7.

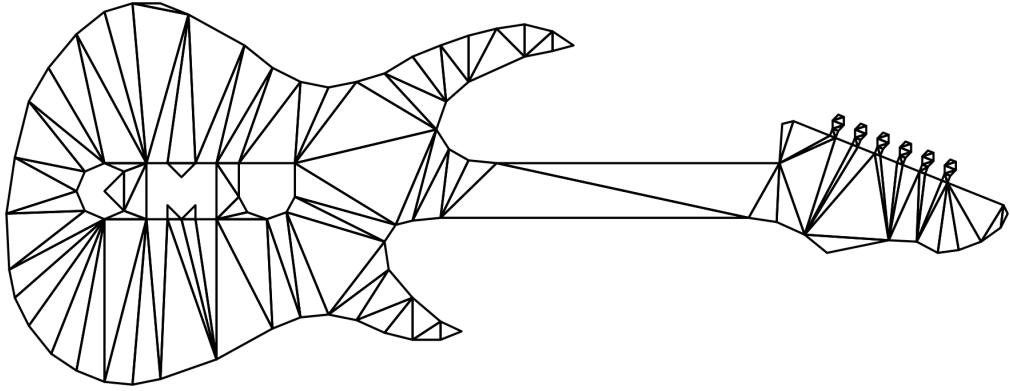


Figura 7: Triangulación restringida del PSLG de la Figura 4 con segmentos extra removidos [1]

La última etapa del algoritmo consiste en el refinamiento de la malla insertando vértices y retriangulando con el algoritmo incremental de Lawson [6] hasta que las restricciones de ángulo mínimo y área máxima de triángulo definidas por el usuario se cumplan. Esta inserción se hace siguiendo 2 reglas, la regla de *segmentos encerrados* y la de los *triángulos malos* dándole siempre prioridad a la primera:

- El *círculo diametral* de un segmento es el círculo único más pequeño que contiene el segmento como su diámetro. Un segmento se dice que está *encerrado* si un punto que no es un extremo del segmento está dentro de su círculo diametral. Cualquier segmento encerrado que aparezca se separa insertando un vértice en su punto medio. Los dos subsegmentos resultantes tienen círculos diamatrales más pequeños y podrían estar o no estar encerrados. El proceso se repite hasta que no queden segmentos encerrados como se ve en la Figura 8.
- El *circuncírculo* de un triángulo es el círculo único cuya circunferencia pasa por todos los vértices del triángulo. Un triángulo se dice que es *malo* si tiene un ángulo que es muy pequeño o un área que es muy grande para satisfacer las restricciones impuestas por el usuario. Un triángulo malo se destruye insertando un vértice en su *circuncentro* (el centro de su circuncírculo). Está asegurado que el triángulo malo será eliminado como se ve en la Figura 9 para mantener la propiedad de Delaunay. Si el vértice insertado encierra un segmento (como se define en la regla del círculo diametral), este será removido deshaciendo la inserción y los segmentos que encerraba se separarán según la regla del círculo diametral.

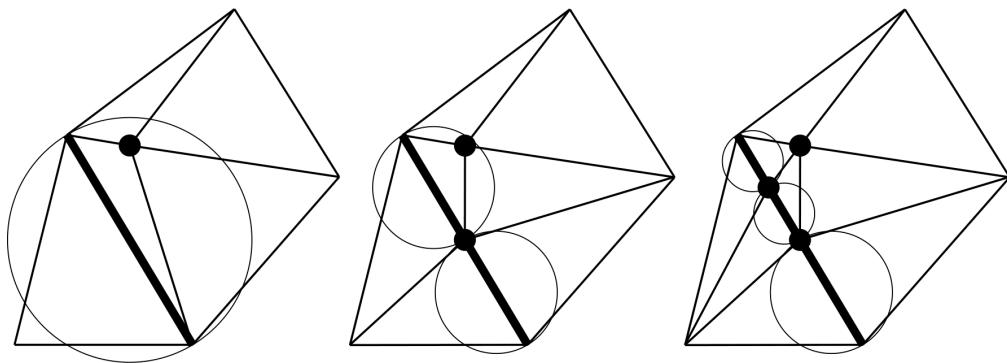


Figura 8: Segmentos divididos según su círculo diametral [1]

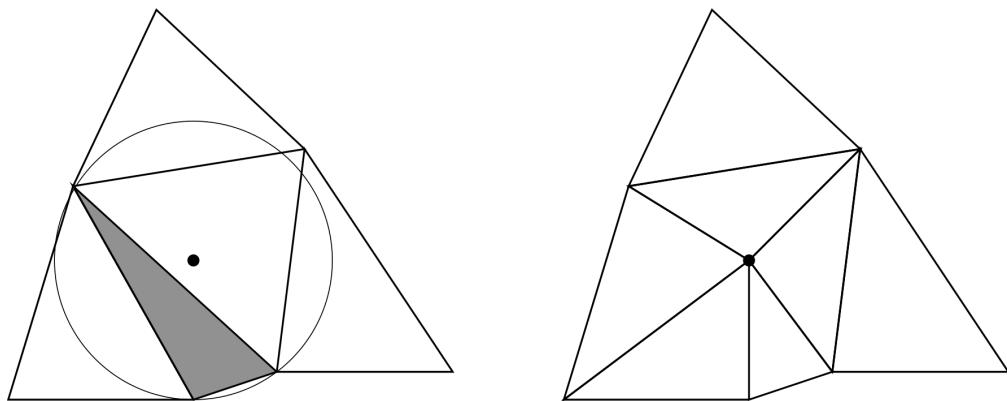


Figura 9: Triángulo separado según su circuncentro [1]

Para el ejemplo de la Figura 4, la malla resultante generada por Triangle es la que se ilustra en la Figura 10.

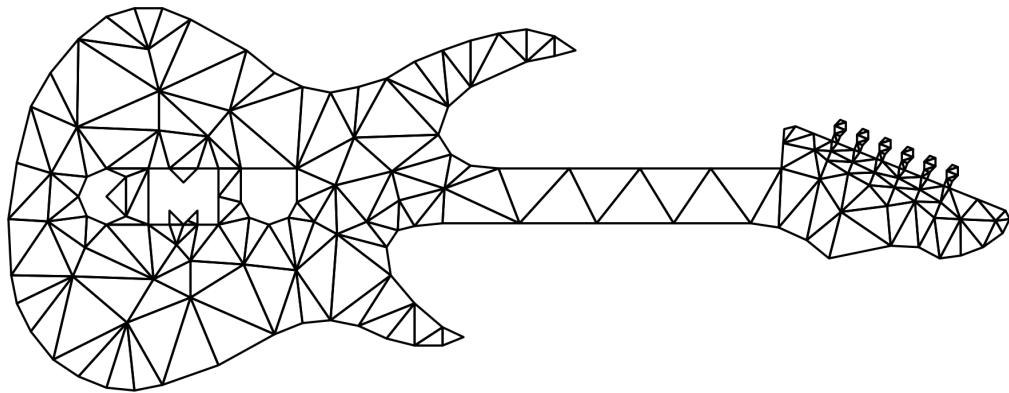


Figura 10: Malla poligonal final resultante de aplicar el algoritmo Triangle [1]

Esta última parte del algoritmo es de particular importancia, ya que el criterio del circuncírculo es análogo al de la cavidad, sin embargo, en este caso solo se utiliza para refinar la malla y re triangular, pero no para generar mallas nuevas. Este trabajo de memoria busca

explorar más a fondo este proceso y utilizarlo para generar mallas nuevas, lo cual en la actualidad no tiene ninguna implementación conocida.

Por otro lado, el algoritmo Polylla, busca generar una malla poligonal a partir de una triangulación arbitraria, usando lo que denomina como *Terminal-edge regions* o regiones de arista terminal, definidas según el *longest edge propagation path* (camino de propagación de arista más larga o *Lepp* [7]) de los triángulos, las cuales utiliza para generar una partición de la triangulación que se asemeja a un diagrama de Voronoi [8]. En la Figura 11 se puede ver un ejemplo de región de arista terminal.

En el contexto de este trabajo, un diagrama de Voronoi se entenderá como una partición de un polígono  $P$  en regiones o “celdas”  $R_i$ , de las cuales cada una contiene un punto  $p_i$  llamado “semilla” y los puntos  $q$  contenidos en  $R_i$  cumplen que  $\|q - p_i\| < \|q - p_j\| \forall p_i, p_j \in P, i \neq j$  donde  $p_j$  es la semilla de cualquier otra celda distinta a  $R_i$ . El diagrama de Voronoi también se le conoce como el *dual* de una triangulación de Delaunay, esto se debe a que, dada una triangulación de Delaunay, se puede obtener su diagrama de Voronoi equivalente si los circuncentros de los triángulos se convierten en semillas para las regiones de Voronoi y viceversa. En la Figura 12 se puede ver una triangulación de Delaunay y su diagrama de Voronoi equivalente.

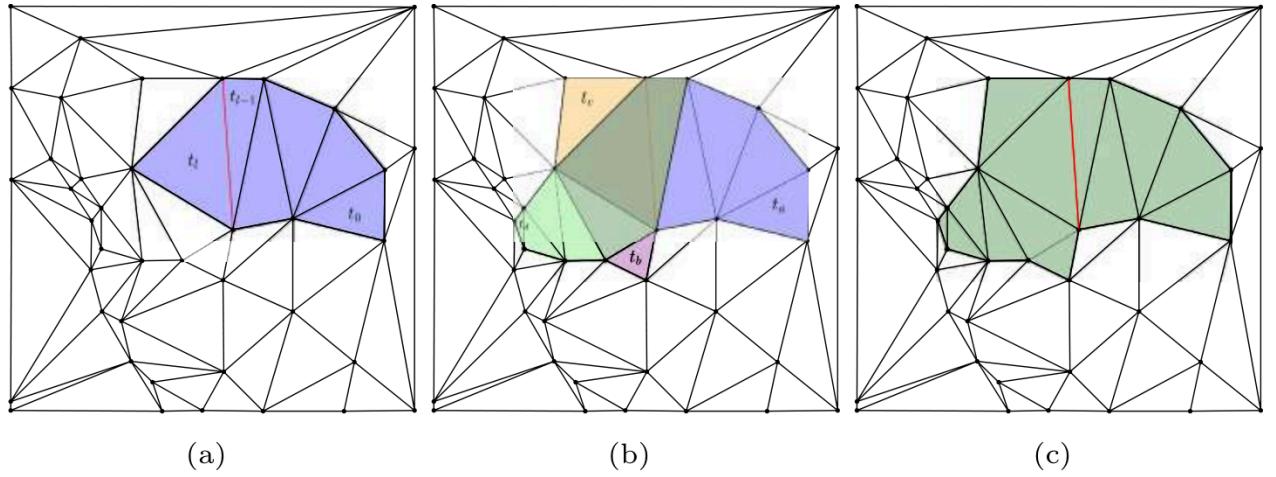


Figura 11: Región de arista terminal. a)  $Lepp(t_0)$  donde la arista roja es la arista terminal. b) Cuatro Lepps con la misma arista terminal:  $Lepp(t_a)$ ,  $Lepp(t_b)$ ,  $Lepp(t_c)$ ,  $Lepp(t_d)$ . c) Región de arista terminal generada por la unión de los Lepp de b) [4]

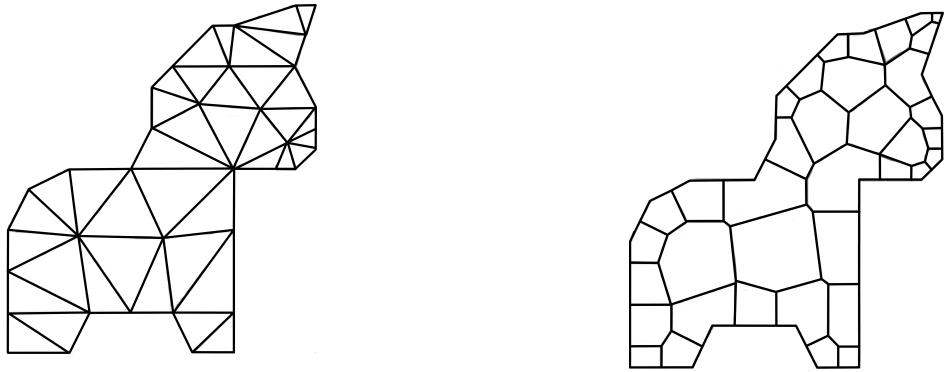


Figura 12: Triangulación de Delaunay y su Diagrama de Voronoi dual [4]

Además de las aristas terminales, Polylla [4] define los siguientes tipos de aristas. Dada una arista  $e$  y dos triángulos  $t_1$  y  $t_2$  que comparten  $e$ :

- *Frontier-edge* o Arista frontera:  $e$  no es la arista más larga ni de  $t_1$  ni de  $t_2$ .
- *Internal-edge* o Arista interna:  $e$  es la arista más larga de  $t_1$ , pero no de  $t_2$  o viceversa.
- *Boundary edge* o Arista de borde:  $e$  pertenece a un solo triángulo. Se manejan como aristas frontera.
- *Barrier edge* o Arista barrera: Arista frontera que queda adentro de una región terminal (y no es el borde).

Polylla consiste de tres fases: Primero etiqueta las aristas de la triangulación de entrada según las categorías anteriores para formar regiones terminales y además designa un *triángulo semilla* en cada región de arista terminal para construir las regiones. Luego, a partir de cada triángulo semilla, hace un recorrido en sentido antihorario o *counter clockwise* (CCW en inglés) de la región de arista terminal para encontrar aristas frontera las cuales formaran la región. Algunas regiones pueden terminar como polígonos no simples, es decir, que tienen puntos colineales o aristas que se intersecan entre sí, por lo que hace una fase de reparación donde las aristas barrera se partitionan en polígonos simples. En la Figura 13 se puede ver una partición de un polígono generada por regiones de arista terminal que presenta una región con un polígono no simple en verde.

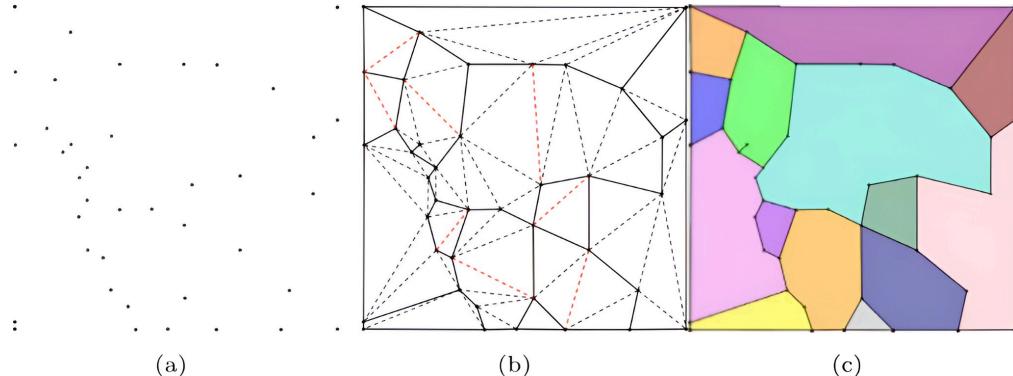


Figura 13: a) Colección de vértices aleatorios, b) Triangulación de Delaunay donde las líneas sólidas son aristas frontera, las líneas punteadas negras son aristas internas y las aristas punteadas rojas son aristas terminales. c) Partición a partir de regiones de arista terminal [4]

En la Figura 14 se puede ver una triangulación de Delaunay y la malla generada por Polylla a partir de ella.

El algoritmo Polylla destaca por sobre otros algoritmos debido a su gran eficiencia en comparación a algoritmos convencionales de construcción de diagramas de Voronoi restringidos, ya que toma bastante menos tiempo en construir con una cantidad de polígonos 3 veces menor y la mitad de vértices que una malla poligonal de un diagrama de Voronoi hecho a partir de la misma triangulación. A esto se le suma también la utilidad de las mallas Polylla en simulaciones de diversos fenómenos que hacen uso del *Virtual Element Method* (VEM) [9] tales como mecánica computacional, dinámica de fluidos, propagación de ondas entre otros problemas que requieren soluciones numéricas de ecuaciones diferenciales parciales. El algoritmo de construcción de mallas basado en cavidades también podría generar mallas para los usos ya mencionados u otros.

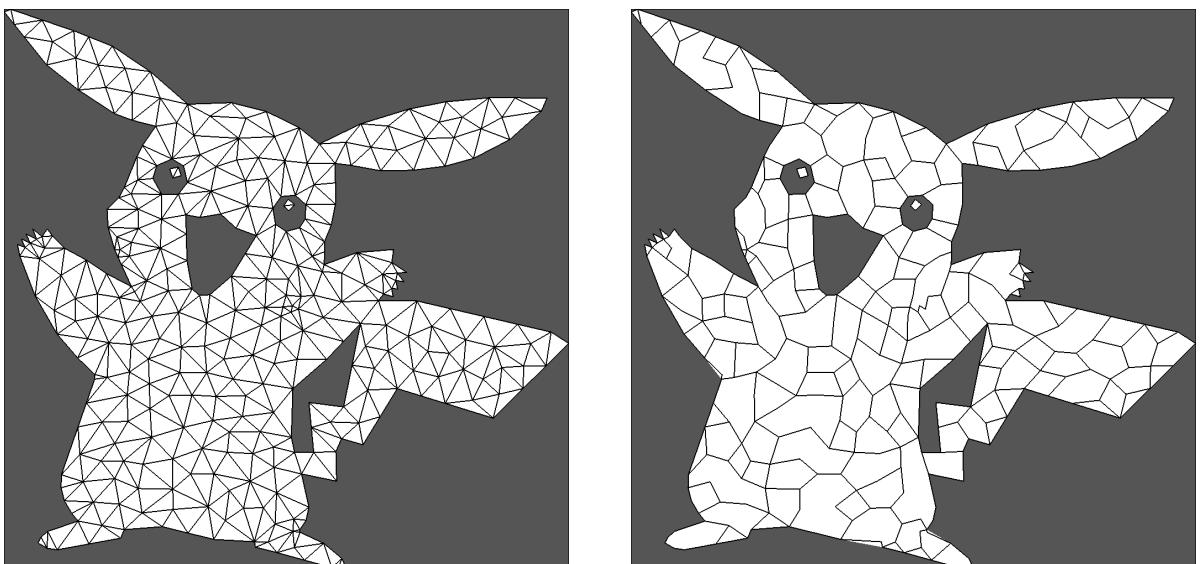


Figura 14: Triangulación de Delaunay y su malla Polylla respectiva [10]

### **3. Objetivos**

#### **Objetivo General**

Diseñar e implementar un algoritmo que permita generar polígonos a partir de una *cavidad* formada por triángulos de una triangulación de Delaunay y comparar su desempeño con el generador de mallas Polylla en cuanto a tiempo, memoria, calidad de las mallas, número de vértices, entre otros criterios.

#### **Objetivos Específicos**

1. Investigar como funciona el algoritmo de mallas Polylla y sus estructuras de datos.
2. Utilizar correctamente las estructuras de datos de Polylla para definir cavidades a través de código.
3. Corroborar que el conjunto de triángulos que conforman la cavidad computada sean los correctos y generar un nuevo polígono a partir de ellos.
4. Comparar el desempeño del algoritmo basado en cavidades con el algoritmo de Polylla basado en regiones terminales en cuanto a eficiencia en tiempo y memoria.
5. Comparar la calidad de las mallas resultantes de ambos algoritmos bajo diversas métricas como calidad de la malla, número de vértices, cantidad de polígonos, etc.

#### **Evaluación**

Para evaluar este trabajo es necesario comparar cualitativamente la malla poligonal resultante del algoritmo basado en cavidades con el algoritmo de Polylla. Se deben evaluar criterios como:

1. La correctitud de la malla poligonal generada. ¿Se ajusta realmente al objeto que se desea modelar?
2. La cantidad de triángulos finales. ¿Genera igual o menos triángulos para representar el mismo polígono?
3. Los ángulos internos de dichos triángulos. ¿Los ángulos de los triángulos siguen cumpliendo con la propiedad de Delaunay?
4. El tiempo de ejecución. ¿Es igual o más eficiente el algoritmo basado en cavidades para generar el mismo polígono que Polylla?
5. El costo en espacio. ¿Utiliza igual o menos memoria que Polylla para generar la misma malla?
6. La calidad de las mallas. ¿La malla generada por el algoritmo basado en cavidades cumple propiedades deseables como triángulos bien distribuidos con ángulos no demasiado pequeños? ¿Es útil para simulaciones?

### **4. Solución Propuesta**

La solución propuesta involucra adoptar las estructuras de datos de Polylla[10] escritas en C++ y aplicarlas a este problema para generar una malla de manera eficiente. Son de especial interés 2 estructuras de datos fundamentales, la estructura `vertex` y la estructura `halfEdge`[11].

La estructura `vertex` (Código 1) define las coordenadas  $x$  e  $y$  de los puntos de la malla poligonal final, además clasifica a los puntos según si son parte del borde del polígono o no, lo cual será útil al momento de representar la cápsula convexa de la malla final. También se encarga de guardar cuál `halfEdge` incide en este vértice.

La estructura `halfEdge` (Código 2) representa las aristas del polígono de una forma particular, no solo es el segmento que une un vértice  $v_i$  con otro vértice  $v_j$ , sino que también guarda información relevante a la hora de recorrer el polígono, como por ejemplo, cuál es su vértice de origen (la “cola” del `halfEdge`), cuál es el siguiente `halfEdge` del triángulo (o cara) actual en orientación CCW, el `halfEdge` anterior de este triángulo y además cuál es su `halfEdge twin`, es decir, el `halfEdge` que une a los mismos dos vértices en CCW pero desde la perspectiva del triángulo vecino. Al igual que los vértices, también guarda la información de si está en el borde o no.

Se puede ver una representación gráfica de estas estructuras en la Figura 15.

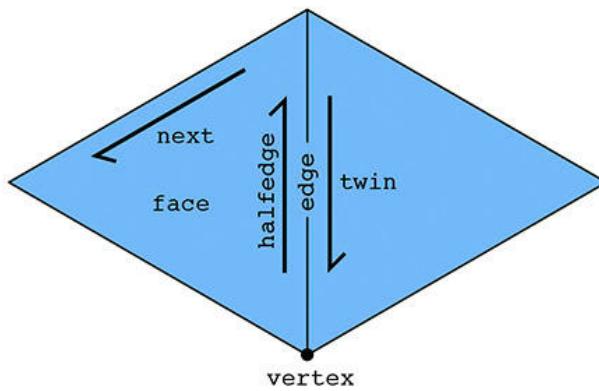


Figura 15: Representación de estructuras `vertex` y `halfEdge`

```
struct vertex {
    double x;
    double y;
    bool is_border = false;
    int incident_halfedge; // <- indice a un array de halfedges
};
```

Código 1: Estructura `vertex`

```
// Todos sus atributos son índices al objeto relevante
struct halfEdge {
    int origin;
    int twin;
    int next;
    int prev;
    int is_border;
};
```

Código 2: Estructura halfEdge

Teniendo estas estructuras de datos se procederá de la manera siguiente:

Similar a Polylla, se recibirá como entrada una triangulación de Delaunay en un conjunto de 3 archivos, un archivo `.node` con los vértices y marcador de borde, un archivo `.ele` con los triángulos de las triangulaciones (qué vértices forman cuáles triángulos) y un archivo `.neigh` con las listas de adyacencia de cada triángulo (sus vecinos). A partir de estos archivos, se creará un objeto `Triangulation`[10] encargado de almacenar estos datos en listas de `vertex` y `halfEdge` en vectores de C++.

Utilizando algún criterio a definir según experimentación, tales como, triángulos que cumplen o no cumplen ciertas restricciones de ángulos o de área, se seleccionará un subconjunto de los triángulos recibidos y se les calculará su circuncentro  $p_i$  y radio  $r_i$  usando un método derivado de determinantes [12].

Conociendo los circuncentros y radios respectivos, estos se almacenan en un vector de tuplas  $(p_i, r_i)$  y luego se creará un *hash map* donde las llaves serán las coordenadas de los circuncentros y los valores serán vectores de índices para almacenar los triángulos que forman parte de cada cavidad. Por cada triángulo de la malla, se revisa si su circuncírculo contiene a cualquiera de los  $p_i$  calculados y en caso de contener alguno, agregar este triángulo como parte de la cavidad que corresponde. Finalmente, se retornan los polígonos resultantes del proceso anterior contenidos en el *hash map*.

## 5. Plan de Trabajo (Preliminar)

1. Estudiar a cabalidad Polylla para utilizar sus clases y estructuras de datos para la generación de mallas a partir de cavidades. [abril-mayo]
2. Diseñar y programar la implementación del método para formar la cavidad a nivel de código como extensión de Polylla [mayo-julio] (Trabajo a adelantar hasta aquí)
3. Probar distintos criterios de selección de triángulos y guardar el registro de los resultados a medida que se escribe el informe final [agosto-septiembre].
4. Seleccionar los mejores métodos de elección de triángulos, comparar su desempeño con la generación de mallas de Polylla y registrar los datos en el informe final [septiembre-octubre].

- Comparar el rendimiento de las mallas generadas en otras aplicaciones frente a la misma malla generada por Polylla [noviembre-diciembre].

## Referencias

- [1] J. R. Shewchuk, «Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator», *Applied Computational Geometry: Towards Geometric Engineering*, vol. 1148, pp. 203-222, may 1996.
- [2] H. Si, «Detri2 is a C++ program for generating (weighted) Delaunay triangulations for (weighted) point sets in 2d.». [En línea]. Disponible en: <https://www.wias-berlin.de/people/si/detri2.html>
- [3] B. Delaunay, «Sur la sphère vide. À la mémoire de Georges Voronoi», *Bulletin of Academy of Sciences of the USSR*, vol. 7, pp. 793-800.
- [4] S. Salinas Fernández, N. Hitschfeld Kahler, A. Ortiz Bernardin, y H. Si, «POLYLLA: polygonal meshing algorithm based on terminal-edge regions», febrero de 2022, *Springer*. doi: 10.1007/s00366-022-01643-4.
- [5] J. Ruppert, «A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation.», *Journal of Algorithms*, pp. 548-585.
- [6] C. L. Lawson, «Software for  $C^1$  Surface Interpolation.», *Mathematical Software III*, pp. 161-194.
- [7] M.C. Rivara, «New Longest-Edge Algorithms for the Refinement and/or Improvement of Unstructured Triangulations», *International Journal for Numerical Methods In Engineering*, vol. 40, 1997.
- [8] G. Voronoi, «Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites», *Journal für die Reine und Angewandte Mathematik*, pp. 97-178, 1908.
- [9] L. Beirão da Veiga, F. Brezzi, G. Manzini, L. Marini, y A. Russo, «Basic principles of virtual element methods», *Math Models Methods Appl Sci*, vol. 23, pp. 199-214, 2013.
- [10] S. Salinas y R. Carrasco, *Polylla-Mesh-DCEL*. [En línea]. Disponible en: <https://github.com/ssalinasfe/Polylla-Mesh-DCEL/tree/main>

- [11] R. Ng y J. Ragan-Kelley, «Computer Graphics and Imaging cs184/284a», University of California. [En línea]. Disponible en: <https://cs184.eecs.berkeley.edu/sp19/article/15/the-half-edge-data-structure>
- [12] J. R. Shewchuk, «Lecture Notes on Geometric Robustness», *Department of Electrical Engineering and Computer Sciences, University of California at Berkeley*, pp. 19-20, abr. 2013.