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. Las triangulaciones de Delaunay 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 triángulos de la malla cuyo circuncírculo contiene alguno de estos puntos. La unión de estos triángulos vecinos forma una **cavidad**, y su cápsula convexa define un nuevo polígono. Este procedimiento permite generar regiones poligonales que pueden servir como base para construir mallas más estructuradas o adaptativas.

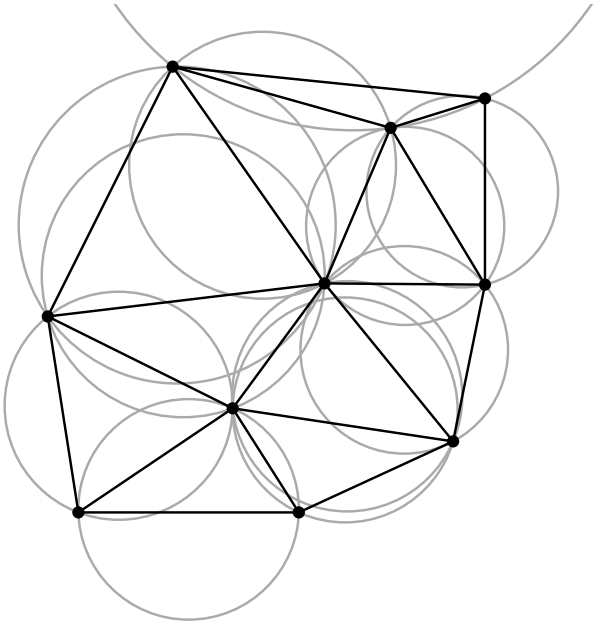

Figura 1: Triangulación de Delaunay con circuncírculos compuesta por 10 vértices.

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

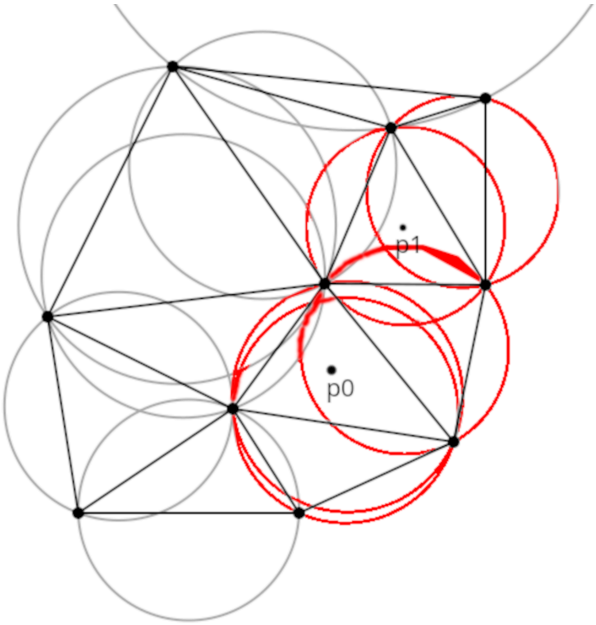

Figura 2: Selección de triángulos para formar un polígono a partir de una cavidad.

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 estos triángulos conforma la cavidad, como se muestra en Figura 3.

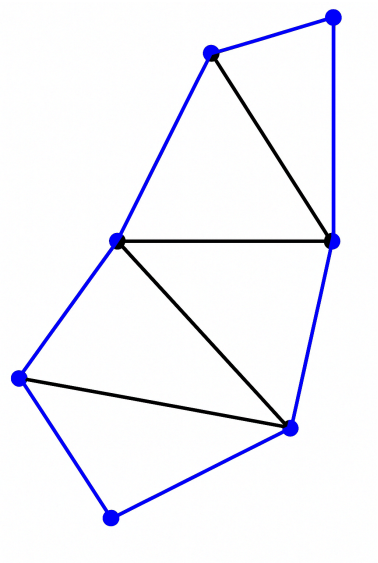

Figura 3: Polígono resultante de una cavidad con capsula convexa en azul

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]. En la actualidad existe poca investigación relacionada con la *generación* de mallas a partir de cavidades, pero sí hay información sobre como *quitar* cavidades durante el proceso de una triangulación inicial de un conjunto de vértices. En el artículo Triangle [1], se definen las cavidades como triángulos que no pertenecen al polígono real cuya malla se desea generar, ya sea porque están fuera de su cápsula convexa (*concavities*) o rellenan un agujero (*hole*) en su interior, los cuales aparecen al intentar la triangulación de los vértices uniendo segmentos entre todos ellos. Cabe destacar que Triangle es capaz de saber de manera sencilla qué triángulos pertenecen realmente al polígono y cuáles no dado que su entrada es un *planar straight line graph* (o PSLG), el cual se define como una colección de vértices y segmentos donde las puntas de cada segmento son los mismos vértices (ver Figura 4).

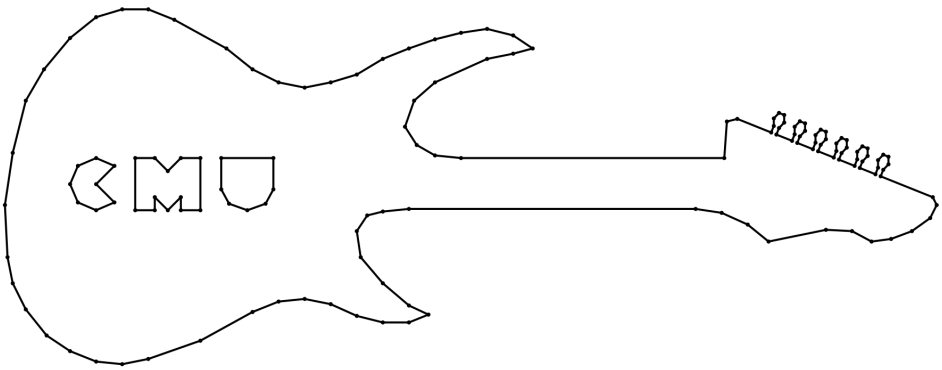

Figura 4: PSLG de una guitarra electrica [1]

No estoy muy seguro de con que más rellenar esta parte… ¿Debería simplemente hablar de como funciona polylla?

3. Objetivos

Objetivo General

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.

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.

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 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 más rápido 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?

4. Solución Propuesta

La solución propuesta involucra adoptar las estructuras de datos de Polylla[5] 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`[6].

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 antihoraria (o CCW del inglés *counter clockwise*), el `halfEdge` anterior de este triángulo y además cuál es su `halfEdge twin`, es decir, el `halfEdge` que une a los mismos vértices en CCW pero para el 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 5.

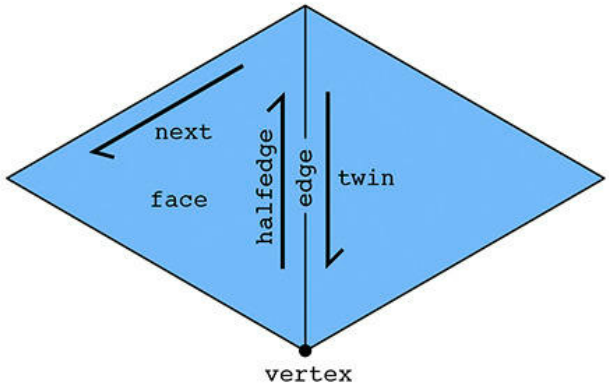

Figura 5: Representación de estructuras `vertex` y `halfEdge`

```
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:

1. 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).
2. A partir de estos archivos, se creará un objeto `Triangulation`[5] encargado de almacenar estos datos en listas de `vertex` y `halfEdge` en vectores de `C++`.
3. Utilizando algún criterio a definir según experimentación, se seleccionará un subconjunto de los triángulos recibidos y se les calculará su circuncentro p_i usando un método por determinar[1]
4. Crear una lista (vector) de índices para almacenar los triángulos que forman parte de la cavidad.
5. Por cada triángulo de la malla, revisar si su circuncírculo contiene a cualquiera de los p_i calculados anteriormente usando un método por determinar[1] y en caso de contener alguno, agregar este triángulo como parte de la cavidad.
6. Retornar el polígono resultante del proceso anterior.

5. Plan de Trabajo (Preliminar)

1. Investigar distintos métodos para calcular el circuncentro y circuncírculo de un triángulo de manera eficiente o en su defecto, algo que permita responder la pregunta *¿El circuncentro del triángulo t_i contiene al punto p_j?* [Abril-Mayo]

[1]Es necesario probar distintos métodos hasta encontrar uno que sea lo suficientemente eficiente

2. Estudiar a cabalidad Polylla para utilizar sus clases y estructuras de datos para la generación de mallas a partir de cavidades. (Trabajo a adelantar hasta aquí). [Junio-Agosto]
3. Programar la implementación de los métodos investigados para formar la cavidad y guardar el registro de los resultados a medida que se escribe el informe final [Septiembre-Octubre].
4. Seleccionar los mejores métodos, comparar su desempeño con la generación de mallas de Polylla y registrar los datos en el informe final [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] S. Salinas y R. Carrasco, *Polylla-Mesh-DCEL*. [En línea]. Disponible en: https://github.com/ssalinasfe/Polylla-Mesh-DCEL/tree/main

[6] 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