



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

INTEGRACIÓN DE POLYLLA Y ACTUALIZACIÓN DE PLUGIN
MEJORAMIENTO DE MALLAS DE UNIDADES DE RESPUESTA HIDROLÓGICA
EN QGIS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

JOSÉ IGNACIO PEREIRA MORGADO

PROFESORA GUÍA:
NANCY HITSCHFELD K.

PROFESOR CO-GUÍA:
PEDRO SANZANA

MIEMBROS DE LA COMISIÓN:
MAURICIO PALMA L.
ÉRIC TANTER

Este trabajo ha sido parcialmente financiado por el Fondo Nacional de Desarrollo
Científico y Tecnológico.

SANTIAGO DE CHILE
2026

RESUMEN DE LA MEMORIA PARA OPTAR AL
TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN
POR: JOSÉ IGNACIO PEREIRA MORGADO
FECHA: 2026
PROF. GUIA: NANCY HITSCHFELD K.

**INTEGRACIÓN DE POLYLLA Y ACTUALIZACIÓN DE PLUGIN
MEJORAMIENTO DE MALLAS DE UNIDADES DE RESPUESTA
HIDROLÓGICA EN QGIS**

Este trabajo aborda la generación de mallas poligonales para apoyar la modelación hidrológica en cuencas urbanas y periurbanas, utilizando QGIS como plataforma de trabajo. A partir de requerimientos geométricos asociados a Unidades de Respuesta Hidrológica y de antecedentes previos que automatizaban la triangulación y disolución en QGIS, se identificó como problema principal la inestabilidad del flujo de triangulación, que podía provocar cierres inesperados del Sistema de Información Geográfica (SIG) en geometrías complejas. En este contexto, se plantea integrar el algoritmo Polylla, desarrollado originalmente en C++, y rediseñar el flujo del plugin para hacerlo operativo dentro del entorno de QGIS.

La implementación se centró en dos aportes. Primero, se desarrolló un binding para exponer Polylla desde C++ a Python mediante pybind11 y CMake, permitiendo su uso como librería en Python. Segundo, se diseñó y construyó un nuevo plugin con una arquitectura por etapas: la triangulación se ejecuta como subprocesso externo y genera archivos intermedios (Shapefile), mientras que Polylla se ejecuta dentro de QGIS desde Python, convirtiendo la triangulación a formato OFF y devolviendo la malla poligonal como una capa en memoria. La interfaz se implementó con Qt Designer y se organizó en dos diálogos alineados con estas etapas, incorporando parámetros configurables para triangulación y opciones de suavizado para Polylla.

Como resultado, se obtuvo un flujo funcional que permite generar triangulaciones y mallas poligonales dentro de QGIS de forma estable en los casos demostrativos presentados, sin cierres inesperados del software. Se concluye que la separación de la triangulación como subprocesso y la integración de Polylla mediante un binding a Python permiten operar el proceso completo en un entorno SIG con mayor control y menor riesgo de fallos. Queda como trabajo futuro realizar una evaluación cuantitativa más rigurosa de la calidad geométrica de las mallas mediante métricas como convexidad y factores de forma, además de explorar optimizaciones de rendimiento e integración más profunda del proceso en QGIS.

Dedicado a mi abuelo, Raúl.

Tabla de Contenido

1. Introducción	1
1.1. Objetivos	2
2. Situación Actual	3
2.1. Antecedentes en QGIS	3
2.2. Algoritmo Polylla	3
2.3. Tecnologías y herramientas	4
2.4. Limitaciones de la solución actual	4
2.5. Justificación de la propuesta	5
3. Implementación de la solución	6
3.1. Requerimientos de la solución	6
3.2. Arquitectura general de la solución	7
3.3. Integración de Polylla	8
3.4. Triangulación	9
3.5. Polylla en QGIS	10
3.6. Interfaz de usuario	11
3.7. Alternativas descartadas	13
4. Resultados: demostración de funcionamiento	14
4.1. Caso 1: Figura simple sin agujeros	15
4.2. Caso 2: Figura compleja sin agujeros	16
4.3. Caso 3: Figura simple con agujeros	17
4.4. Caso 4: Figura compleja con agujeros	18
4.5. Observaciones generales.	19
5. Conclusiones y trabajo futuro	20
5.1. Conclusiones	20

5.2. Trabajo futuro	20
Bibliografía	22

Índice de Ilustraciones

Figura 1	Ventana de diálogo para la triangulación de una capa poligonal.	12
Figura 2	Ventana de diálogo para la aplicación de Polylla sobre una capa triangulada.	13
Figura 3	Caso 1 (a): capa poligonal de entrada, sin triangular.	15
Figura 4	Caso 1 (b): capa triangulada ($\text{min_angle}=20$, $\text{max_area}=1000$).	15
Figura 5	Caso 1 (c): capa poligonal resultante tras aplicar Polylla.	15
Figura 6	Caso 2 (a): capa poligonal de entrada, sin triangular.	16
Figura 7	Caso 2 (b): capa triangulada ($\text{min_angle}=20$, $\text{max_area}=1000$).	16
Figura 8	Caso 2 (c): capa poligonal resultante tras aplicar Polylla.	16
Figura 9	Caso 3 (a): capa poligonal de entrada, con agujeros, sin triangular.	17
Figura 10	Caso 3 (b): capa triangulada ($\text{min_angle}=20$, $\text{max_area}=1000$).	17
Figura 11	Caso 3 (c): capa poligonal resultante tras aplicar Polylla.	17
Figura 12	Caso 4 (a): capa poligonal de entrada, con agujeros, sin triangular.	18
Figura 13	Caso 4 (b): capa triangulada ($\text{min_angle}=20$, $\text{max_area}=1000$).	18
Figura 14	Caso 4 (c): capa poligonal resultante tras aplicar Polylla.	18

Capítulo 1

Introducción

La gestión eficiente de los recursos hídricos en zonas urbanas y periurbanas requiere herramientas de modelación avanzadas capaces de representar la complejidad del territorio. Modelos hidrológicos distribuidos, como PUMMA (Peri-Urban Model for landscape Management) [1], basan su análisis en Unidades de Respuesta Hidrológica (URHs) [2]. Estas unidades deben cumplir con requisitos geométricos, como la convexidad y ciertos parámetros de forma, para asegurar condiciones adecuadas para la simulación. En este contexto, la generación automatizada de mallas poligonales a partir de información geográfica es un desafío que combina la ingeniería en computación con la ingeniería hidráulica.

Este trabajo aborda el problema de generar mallas poligonales controlando parámetros de calidad geométrica, utilizando QGIS como plataforma central. Se seleccionó esta herramienta por ser de código abierto y contar con una activa comunidad que aporta soporte y funcionalidades. Además, el desarrollo de plugins en QGIS se realiza mediante Python, lo que otorga flexibilidad para extender el software. Sin embargo, la ejecución de código externo dentro de este ambiente presenta desafíos técnicos. Específicamente, la integración directa de librerías de triangulación ha generado inestabilidad, conflictos de memoria y cierres inesperados (crashes), impidiendo la automatización confiable del proceso en versiones anteriores.

El presente trabajo busca abordar estas problemáticas mediante ingeniería de software e integración de algoritmos. Se propone la incorporación del algoritmo Polylla [3], originalmente escrito en C++, al entorno de Python y posteriormente a QGIS. Para ello, se desarrolló un binding que exporta funcionalidades de Polylla como una librería nativa de Python. Utilizando esta librería y tomando como base el trabajo previo de Sergio Villarroel [4], se diseñó una nueva versión del plugin. Esta versión permite generar mallas ajustando parámetros relevantes del proceso y rediseña el flujo de triangulación para mejorar la estabilidad, desacoplando etapas críticas para reducir fallos en el entorno de QGIS. El

resultado es una herramienta modular orientada a apoyar la preparación de datos para el modelamiento hidrológico.

Finalmente, el informe se estructura de la siguiente manera: en la Situación Actual se revisan los antecedentes teóricos, las herramientas previas y el algoritmo Polylla. La sección de Implementación detalla la creación del binding, la arquitectura del plugin y la solución a los problemas de estabilidad. Posteriormente, en Resultados y Discusión, se presentan pruebas de funcionamiento mediante casos demostrativos en QGIS, mostrando la generación de triangulación y la aplicación de Polylla sobre distintas superficies. El documento cierra con las Conclusiones y líneas de trabajo futuro, donde se plantea como continuación natural un análisis cuantitativo más completo de calidad y rendimiento.

1.1. Objetivos

1.1.1. Objetivo General

Diseñar e implementar una herramienta de software integrada en QGIS que permita la generación de mallas poligonales mediante la utilización del algoritmo Polylla, asegurando estabilidad operativa y corrigiendo errores críticos presentes en versiones anteriores, con control de parámetros relevantes del proceso.

1.1.2. Objetivos Específicos

1. **Desarrollar un binding de Python para Polylla:** Crear una interfaz funcional entre el código original en C++ de Polylla y Python, permitiendo su uso como una librería estándar dentro del ecosistema de desarrollo actual.
2. **Rediseñar la arquitectura de triangulación:** Implementar un mecanismo de ejecución externa para el proceso de triangulación que evite los conflictos de memoria y crashes aleatorios observados al utilizar la librería Triangle directamente dentro del entorno de QGIS.
3. **Desarrollar un nuevo plugin para QGIS:** Integrar la nueva librería de Polylla y el flujo de triangulación corregido en un complemento de usuario final, permitiendo la manipulación de capas vectoriales y la configuración de parámetros.
4. **Demostrar el funcionamiento de la solución:** Ejecutar el flujo completo (triangulación y Polylla) en casos representativos dentro de QGIS, generando y visualizando las capas resultantes. La validación cuantitativa y la comparación con herramientas previas se dejan como trabajo futuro.

Capítulo 2

Situación Actual

Este trabajo de título es la intersección de dos líneas de desarrollo previas: por un lado, la necesidad de preprocesamiento geométrico para la hidrología urbana (Sanzana y Villarroel) y, por otro, la optimización algorítmica de mallas poligonales (Salinas). A continuación, se describen los antecedentes, el marco teórico y las tecnologías involucradas.

2.1. Antecedentes en QGIS

El origen de la necesidad geométrica abordada en esta memoria parte de la tesis de magíster de Pedro Sanzana [5]. Su trabajo se centró en la caracterización y modelamiento de procesos hidrológicos en cuencas periurbanas del piedemonte de Santiago. Para utilizar modelos distribuidos como PUMMA, Sanzana estableció que el análisis espacial requería mallas base compuestas por polígonos irregulares (URHs). Sin embargo, identificó que estas unidades debían cumplir con criterios estrictos de forma, convexidad y orientación para evitar inestabilidades numéricas en las simulaciones hídricas.

Posteriormente, Sergio Villarroel [4] retomó estos requerimientos con el objetivo de automatizar el proceso dentro de un Sistema de Información Geográfica (GIS). Villarroel desarrolló un plugin para QGIS capaz de triangular polígonos y disolverlos basándose en criterios de forma y convexidad planteados por Sanzana. Aunque este plugin validó la factibilidad de la automatización, presenta problemas de estabilidad, ocasionando el cierre del programa y la pérdida del progreso.

2.2. Algoritmo Polylla

Paralelamente a lo anterior, Sergio Salinas [3] propuso el algoritmo Polylla. Este algoritmo, implementado originalmente en C++, ofrece una estrategia para generar mallas poligonales a partir de triangulaciones arbitrarias.

El funcionamiento de Polylla se basa en el concepto de regiones de arista terminal (terminal-edge regions). En lugar de procesar polígonos de forma aislada, el algoritmo

agrupa triángulos adyacentes basándose en la longitud de sus aristas. El proceso consta de tres fases principales:

1. **Etiquetado:** Se clasifican las aristas de la triangulación (como frontera, interna o terminal) según su longitud relativa en los triángulos que las comparten.
2. **Recorrido:** Se construyen los polígonos fusionando los triángulos que pertenecen a una misma región, utilizando las aristas etiquetadas como guía.
3. **Reparación:** Se detectan polígonos «no simples» (que no sean simplemente conexos) y se dividen para asegurar que la malla final sea válida geométricamente.

Según su formulación, Polylla busca generar mallas con menos polígonos y sin insertar puntos adicionales al dominio. Esto permite que la malla resultante respete exactamente los vértices de la entrada original, una característica crítica para mantener la fidelidad de los datos topográficos. Además, su estructura de datos basada en Half-Edge permite una manipulación eficiente de la topología.

2.3. Tecnologías y herramientas

2.3.1. QGIS

QGIS es la plataforma de referencia en sistemas de información geográfica de código abierto. Su arquitectura permite la extensión de funcionalidades mediante plugins escritos en Python (PyQGIS). Este entorno ofrece acceso a librerías geoespaciales como GDAL/OGR, pero impone restricciones en la gestión de memoria y subprocesos, lo que es crítico para la estabilidad de algoritmos computacionalmente intensivos.

2.3.2. Triangle

Para la etapa de triangulación, el estándar de facto en la investigación académica es la librería Triangle, desarrollada por J.R. Shewchuk [6]. Esta librería permite generar triangulaciones de Delaunay con restricciones de calidad (ángulos mínimos, áreas máximas).

2.3.3. Pybind11

Dado que Polylla está escrito en C++ por razones de rendimiento y el ecosistema de QGIS opera en Python, es necesario integrar ambos lenguajes. Tecnologías como pybind11 permiten crear bindings que exponen funciones y clases de C++ a Python, manteniendo la eficiencia del código compilado.

2.4. Limitaciones de la solución actual

El plugin desarrollado previamente por Villarroel sufre de inestabilidad («crashes») al ejecutarse sobre geometrías complejas o mallas de gran tamaño.

A partir de pruebas y revisión del flujo, se considera que el problema radicaba en la invocación directa de la librería Triangle dentro del mismo espacio de memoria del proceso principal de QGIS. Al producirse errores de segmentación o manejo de excepciones no

controladas dentro de la rutina de triangulación, QGIS se cerraba abruptamente, provocando la pérdida de trabajo del usuario.

2.5. Justificación de la propuesta

La creación del nuevo plugin se justifica por la necesidad de rediseñar la solución tecnológica existente frente a las limitaciones observadas en el flujo previo. El plugin anterior dependía de métodos de disolución que no siempre entregaban una solución geométrica adecuada en una sola iteración y presentaba problemas de estabilidad al trabajar con geometrías complejas. Estas limitaciones afectan la confiabilidad del proceso desde el punto de vista del usuario y condicionan el uso de las mallas generadas como insumo para modelación hidrológica.

En este contexto, se incorpora Polylla como una alternativa a la lógica de disolución previa. Polylla es un algoritmo determinista orientado a construir mallas poligonales a partir de una triangulación, sin introducir puntos adicionales al dominio. Su integración en el plugin busca ampliar las herramientas disponibles para generar mallas poligonales a partir de un mismo insumo triangulado, manteniendo una representación consistente de los vértices de entrada. En esta memoria se presenta la integración y el funcionamiento del flujo completo, mientras que el análisis cuantitativo del impacto de Polylla sobre la calidad geométrica de las mallas se deja como proyección de trabajo futuro.

Finalmente, la propuesta también responde a un objetivo de estabilidad. Para reducir cierres inesperados asociados a la triangulación, se adopta una estrategia de desacople: la triangulación se ejecuta como un subprocesso independiente del sistema, separando su ejecución del proceso principal de QGIS. Con esto se busca mantener el entorno de trabajo del usuario y permitir que los fallos en la etapa de triangulación se manejen de forma controlada.

Capítulo 3

Implementación de la solución

3.1. Requerimientos de la solución

En esta sección se establecen los requerimientos que guiaron la implementación del plugin y las decisiones adoptadas. Estos requerimientos se definieron a partir de los objetivos del trabajo y de los problemas observados en la solución previa, principalmente en lo relativo a estabilidad durante la triangulación e integración de un algoritmo externo (Polylla) en el entorno de QGIS.

3.1.1. Requisitos funcionales

Los requisitos funcionales describen las acciones que el usuario debe poder realizar mediante el plugin y los resultados esperados en cada etapa. En este trabajo, el flujo se compone de una etapa de triangulación y una etapa de generación de malla poligonal mediante Polylla, con la posibilidad de ejecutar ambas de forma secuencial.

La herramienta debe permitir lo siguiente:

- Seleccionar una capa vectorial poligonal en QGIS como entrada para el proceso de malla.
- Ejecutar la triangulación de la geometría de entrada, configurando al menos ángulo mínimo y área máxima.
- Definir un directorio o ruta de salida para los archivos generados por la triangulación (por ejemplo, en formato SHP).
- Seleccionar una capa triangulada como entrada para la ejecución de Polylla.
- Aplicar Polylla sobre la triangulación seleccionada y generar como salida una malla poligonal como capa en memoria dentro de QGIS.
- Ejecutar el flujo completo de forma secuencial: triangulación y luego Polylla.

3.1.2. Requisitos no funcionales

Los requisitos no funcionales establecen restricciones y propiedades del sistema asociadas al uso dentro de QGIS. Estos criterios son relevantes debido a que el trabajo se ejecuta en un entorno con un proceso principal que puede terminar en un *crash* si ocurre una falla de

bibliotecas externas, y donde los tiempos de ejecución y la claridad de la interfaz influyen directamente en la utilidad práctica del plugin.

- Estabilidad: la ejecución de la triangulación no debe cerrar QGIS de forma abrupta al fallar, incluso ante geometrías no convexas o entradas complejas.
- Aislamiento de fallos: si una etapa falla (triangulación o Polylla), el sistema debe terminar la ejecución de forma controlada y reportar el error al usuario.
- Reproducibilidad: dado un mismo input y parámetros, el sistema debe producir resultados consistentes.
- Mantenibilidad: el código debe quedar separado por componentes (triangulación y Polylla) para facilitar cambios y depuración.
- Usabilidad: los parámetros principales deben estar disponibles en la interfaz, con valores por defecto razonables y validación de rangos.
- Rendimiento: el tiempo de ejecución debe ser razonable para tamaños de entrada típicos.

3.2. Arquitectura general de la solución

En este apartado se describe la arquitectura general del plugin y el flujo de ejecución completo, desde la selección de una capa en QGIS hasta la generación de la malla poligonal final. La arquitectura se diseñó para responder a dos restricciones principales: por un lado, la necesidad de controlar parámetros de calidad durante la triangulación y, por otro, evitar cierres inesperados de QGIS asociados a la ejecución de rutinas de triangulación dentro del mismo proceso del SIG. Para ello, la solución se organiza en dos etapas, donde la triangulación se ejecuta como subproceso externo y Polylla se ejecuta dentro del entorno Python de QGIS utilizando el binding desarrollado.

3.2.1. Flujo de ejecución

El flujo de trabajo comienza con la selección de una capa vectorial poligonal desde QGIS. Esta capa constituye la entrada geométrica del proceso, junto con los parámetros de triangulación definidos por el usuario (ángulo mínimo y área máxima). El plugin valida la existencia de una capa seleccionada, la presencia de geometrías y la coherencia de los parámetros numéricos.

La etapa de triangulación se ejecuta como un subproceso independiente del sistema. El plugin define un directorio de trabajo y ejecuta el script asociado a la triangulación, entregando los parámetros configurados por el usuario. El resultado de esta etapa se materializa como archivos en disco en formato Shapefile (SHP), que representan la triangulación generada. Esta decisión permite que si ocurre un fallo durante la triangulación, el error quede contenido en el subproceso y no en el proceso principal de QGIS. Además, deja un resultado intermedio persistente que puede inspeccionarse o reutilizarse sin repetir la ejecución completa.

En la etapa siguiente, Polylla opera sobre la triangulación ya generada. El plugin carga una capa triangulada en QGIS y ejecuta Polylla utilizando la librería de Python creada

a partir del binding C++/Python. El resultado final corresponde a una malla poligonal que se entrega como una nueva capa en memoria, agregada al proyecto de QGIS para su visualización y uso posterior.

3.2.2. Componentes

La solución se compone de los siguientes elementos:

1. Interfaces de usuario en QGIS: la interfaz de triangulación permite seleccionar la capa de entrada, configurar parámetros y definir rutas de salida para los productos de la triangulación, mientras que la interfaz de Polylla permite seleccionar una capa de entrada y el nombre de la capa de salida.
2. Núcleo del plugin (Python, PyQGIS): implementa la lógica de control del flujo. Sus responsabilidades incluyen validar entradas, preparar directorios, construir comandos de ejecución, manejar archivos intermedios y transformar resultados en capas QGIS.
3. Librería Polylla para Python (binding): trae el código C++ de Polylla al entorno Python. El plugin invoca esta librería para construir la malla poligonal a partir de la triangulación.

3.3. Integración de Polylla

El algoritmo Polylla fue desarrollado originalmente en C++ y su uso dentro de QGIS requiere una interfaz hacia Python, dado que los plugins de QGIS se implementan en PyQGIS. En esta sección se describe el código base reutilizado, el diseño del binding implementado con pybind11, el proceso de construcción del módulo mediante CMake y la API final disponible desde Python.

3.3.1. Descripción del código base

La implementación original de Polylla se reutiliza como base sin modificar su lógica principal. El núcleo del algoritmo se encuentra definido en `polylla.hpp`, donde se declara la estructura `PolyllaOptions` para configurar aspectos del algoritmo y la clase `Polylla`, responsable de ejecutar el proceso de generación de mallas poligonales a partir de una triangulación.

Polylla opera sobre una representación de malla triangulada basada en estructuras topológicas, apoyándose en `triangulation.hpp` y componentes relacionados. Este enfoque permite recorrer adyacencias y relaciones entre elementos, lo cual es necesario para construir regiones y fusionar triángulos. En el código base también se incorpora la medida `m_edge_ratio` mediante `m_edge_ratio.hpp`, utilizada por Polylla para evaluar calidad o suavizado de las mallas resultantes.

Desde el punto de vista de entradas, el proyecto original contempla la construcción de Polylla a partir de archivos externos, incluyendo formatos como `OFF` y el conjunto de

archivos .node, .ele y .neigh, que son extensiones comunes de triangulaciones generadas por triangle. Desde el punto de vista de salidas, la implementación original permite exportar resultados en distintos formatos mediante métodos que escriben archivos, pero para los fines del plugin se utilizó la exportación en formato OFF para representar la malla poligonal resultante.

3.3.2. Binding: Creación de py_polylla

Para habilitar el uso de Polylla desde Python se implementó un módulo con `pybind11`, que se llamó `py_polylla`. Este módulo expone una interfaz acotada, centrada en la configuración del algoritmo y en su ejecución a partir de archivos de triangulación.

El binding expone dos elementos principales:

1. `PolyllaOptions`: se expone como clase `Python` con sus campos configurables mediante `def_readwrite`. Los atributos expuestos corresponden a `smooth_method`, `smooth_iterations` y `target_length`, consistentes con la estructura declarada en `polylla.hpp`. Esto nos permite configurar el comportamiento del algoritmo.
2. `Polylla`: se expone como clase `Python` con constructores que reciben rutas a archivos y una instancia opcional de `PolyllaOptions`. Se hará uso del constructor de archivos en formato OFF.

Finalmente, para construir el módulo `py_polylla` se utilizó `CMake` como sistema de compilación, junto con `pybind11`. En términos generales, el archivo `CMakeLists.txt` localiza el entorno de Python para obtener sus headers y librerías, incorpora `pybind11` como dependencia y compila el archivo del wrapper como una biblioteca. El resultado es un artefacto importable desde Python, que luego puede ser utilizado desde el entorno de QGIS como una librería nativa.

3.4. Triangulación

La triangulación se implementa como un script externo (`triangulation.py`) que recibe una capa poligonal en formato SHP, aplica triangulación con restricciones de calidad y genera como salida un nuevo Shapefile compuesto por triángulos. Este componente se diseñó como una pieza independiente del entorno de QGIS, de manera que pueda ejecutarse como subprocesso y dejar sus resultados en disco para su posterior uso por el resto del flujo.

3.4.1. Algoritmo de triangulación

La triangulación se realiza utilizando la librería triangle. El script transforma las geometrías de entrada, manejadas como objetos `Polygon` o `MultiPolygon` de Shapely, a una representación adecuada para la librería de triangulación. Para ello se construyen segmentos que representan el contorno exterior y los contornos interiores (agujeros) del polígono, junto con puntos auxiliares para indicar regiones internas no triangulables.

El script incluye funciones auxiliares para esta preparación. `build_simple_segments` construye los segmentos cerrados de cada anillo a partir de sus vértices. `polygon_centroid` calcula un punto representativo para cada anillo interior, usado como punto dentro del agujero, lo que permite a la triangulación respetar vacíos internos. Posteriormente, `triangulate_single_polygon` ejecuta la triangulación sobre un polígono individual y devuelve una colección de triángulos representados nuevamente como polígonos.

3.4.2. Parámetros de calidad

El script soporta parámetros de calidad asociados a la triangulación mediante argumentos de línea de comandos. Los parámetros principales son:

- Ángulo mínimo (`--min-angle`): permite imponer una cota inferior sobre los ángulos internos de los triángulos, lo que reduce la generación de triángulos delgados.
- Área máxima (`--max-area`): permite imponer un límite superior al área de los triángulos, controlando el nivel de refinamiento de la malla.

3.4.3. Manejo de entradas y salidas

La ejecución del componente se realiza a través de un punto de entrada estándar (`main`) que parsea argumentos con `argparse`. El script requiere explícitamente dos rutas:

- Entrada (`--in`): ruta al Shapefile de polígonos a triangular.
- Salida (`--out`): ruta donde se guardará el Shapefile resultante de triángulos.

El proceso de lectura y escritura utiliza fiona para operar sobre Shapefiles y Shapely para convertir cada feature a geometría manipulable (`shape`). El flujo considera que la entrada puede contener geometrías `Polygon` y `MultiPolygon`, lo que permite procesar capas donde existan unidades compuestas por múltiples partes. Para cada polígono se genera una colección de triángulos, y estos se escriben como nuevos polígonos en el archivo de salida.

3.5. Polylla en QGIS

Polylla se ejecuta dentro del plugin como una etapa posterior a la triangulación. Su propósito es tomar una capa triangulada, convertirla a un formato de entrada compatible con el binding (`OFF`), ejecutar Polylla desde Python mediante el módulo `py_polylla` y transformar el resultado nuevamente a una capa que QGIS pueda cargar. Esta integración se implementa en el archivo `polylla_integration.py`.

3.5.1. Entradas requeridas

La entrada de Polylla es una capa que contiene la triangulación generada en la etapa anterior. En términos de estructura, el flujo asume que la capa de entrada:

- Está compuesta por features poligonales que representan triángulos.

- Cada triángulo se define por tres vértices, con el primer punto repetido al final (anillo cerrado).
- Comparte un sistema de referencia (CRS).

A partir de esta capa triangulada, el plugin construye la representación necesaria para Polylla en formato OFF. Esto se realiza mediante la función `build_off_from_layer`, que recorre los triángulos y genera dos estructuras: un arreglo de vértices y un arreglo de caras, donde cada cara es una terna de índices hacia el arreglo de vértices. Para evitar duplicación de vértices que representan el mismo punto, se aplica una estrategia de deduplicación basada en redondeo de coordenadas, que cuantiza los valores (x, y) a una cantidad fija de decimales antes de registrar un vértice global.

3.5.2. Ejecución de Polylla

El puente principal de ejecución se implementa en `run_polylla_off`. Este método recibe como entrada un archivo OFF temporal (generado desde la capa triangulada). Con esto se construye una instancia de `PolyllaOptions` y se ejecuta Polylla sobre el archivo OFF de entrada. La salida se produce como un nuevo archivo OFF, que representa la malla poligonal resultante.

La ejecución completa se encapsula dentro de una tarea (`PolyllaTask`) basada en `QgsTask`. Esto permite que el proceso se ejecute sin bloquear la interfaz de QGIS y proporciona un punto de control para manejar errores y estados. El método `run` de la tarea define el flujo completo:

1. Construir la representación OFF desde la capa triangulada.
2. Escribir el OFF a un archivo temporal.
3. Ejecutar Polylla y producir un OFF de salida.
4. Leer el OFF resultante y convertirlo a una capa en memoria.

3.6. Interfaz de usuario

La interfaz se diseñó usando Qt Designer, lo que permite mantener la definición visual en archivos .ui y cargarla desde Python. El plugin separa la interacción del usuario en dos diálogos, alineados con las etapas del flujo.

3.6.1. Diálogo de triangulación

El diálogo de triangulación permite seleccionar una capa poligonal del proyecto y configurar parámetros básicos de calidad. Incluye:

- Selector de capa de entrada (capas vectoriales poligonales).

- Parámetros: ángulo mínimo y área máxima.
- Parámetros asociados a descriptor de forma y umbral (presentes en la interfaz, con habilitación condicional del umbral).
- Selección de directorio de salida mediante un selector de archivos.
- Botones de ejecución y cancelación.

La triangulación se ejecuta invocando el script triangulation.py como subproceso, pasando los parámetros seleccionados y construyendo la ruta de salida en base al nombre de la capa.

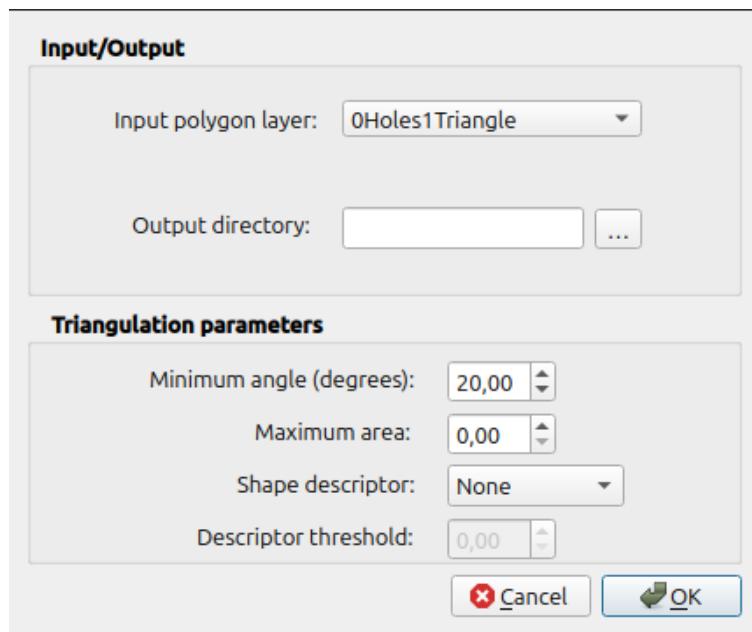


Figura 1: Ventana de diálogo para la triangulación de una capa poligonal.

3.6.2. Diálogo de Polylla

El diálogo de Polylla permite seleccionar la capa triangulada (capas poligonales) y configurar parámetros de suavizado:

- Capa de entrada.
- Método de suavizado: None, Laplacian Edge-Ratio, Laplacian, Distmesh.
- Número de iteraciones.

Este diálogo valida que exista una capa seleccionada antes de permitir la ejecución.

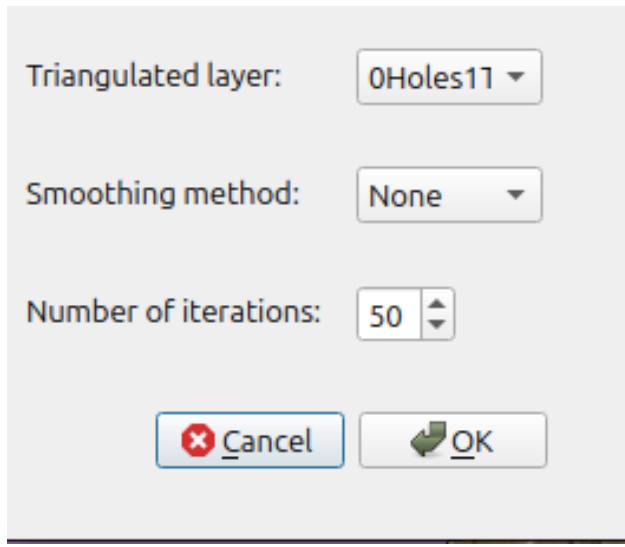


Figura 2: Ventana de diálogo para la aplicación de Polylla sobre una capa triangulada.

3.7. Alternativas descartadas

Durante el desarrollo se evaluaron alternativas para resolver la etapa de triangulación y su ejecución dentro de QGIS. A continuación se resumen los intentos realizados y las razones para descartarlos.

3.7.1. Triangulación con Scipy

Se probó realizar la triangulación usando SciPy con el objetivo de descartar que la librería triangle fuera la causa principal de los cierres inesperados de QGIS. Sin embargo, esta alternativa resultó limitada para el caso de uso del plugin, ya que no permitía controlar parámetros de calidad relevantes, como ángulo mínimo o área máxima. Esto impedía replicar el comportamiento requerido para generar mallas con restricciones y reducía la utilidad de la triangulación dentro del flujo.

3.7.2. Ejecución en QGIS con QProcess

Se exploró el uso de QProcess para ejecutar la triangulación como proceso externo desde QGIS. La intención era desacoplar la ejecución del proceso principal y mantener integración con la interfaz. En la práctica, la llamada no lograba ejecutarse correctamente fuera del ambiente esperado, perdiendo contexto y fallando durante la invocación del proceso.

3.7.3. Decisiones finales

Frente a estas limitaciones, se optó por ejecutar la triangulación mediante una llamada a sistema nativa de Python. Esta alternativa permitió completar la triangulación y generar los archivos de salida de forma consistente. Como consecuencia, la ejecución puede congelar la interfaz de QGIS durante el tiempo que dura el subproceso, pero se privilegió contar con una solución funcional, manteniendo los parámetros de calidad y evitando las restricciones observadas en SciPy y QProcess.

Capítulo 4

Resultados: demostración de funcionamiento

En este capítulo se presenta evidencia visual del funcionamiento del plugin en QGIS. El objetivo es mostrar el flujo completo de trabajo, desde una capa poligonal de entrada, pasando por la triangulación, hasta la generación de una malla poligonal mediante Polylla. Los casos se eligieron para cubrir situaciones típicas del uso esperado, variando la complejidad geométrica y la presencia de agujeros. En cada caso se muestran tres etapas: (i) geometría de entrada sin triangular, (ii) triangulación generada y (iii) malla poligonal resultante tras aplicar Polylla.

4.1. Caso 1: Figura simple sin agujeros

Este caso corresponde a una geometría simple y regular, utilizada como verificación básica del flujo completo.

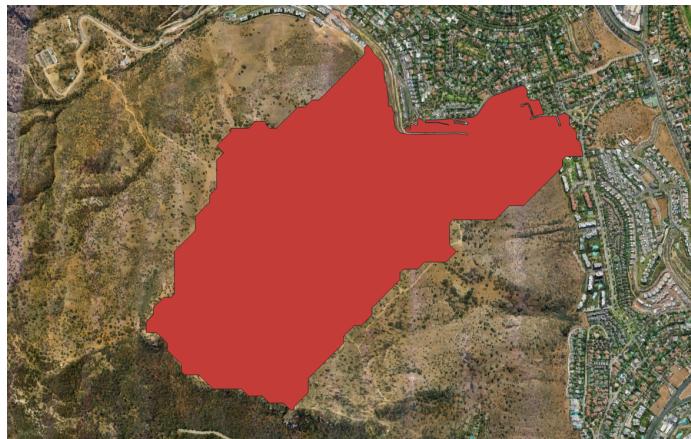


Figura 3: Caso 1 (a): capa poligonal de entrada, sin triangular.



Figura 4: Caso 1 (b): capa triangulada ($\text{min_angle}=20$, $\text{max_area}=1000$).

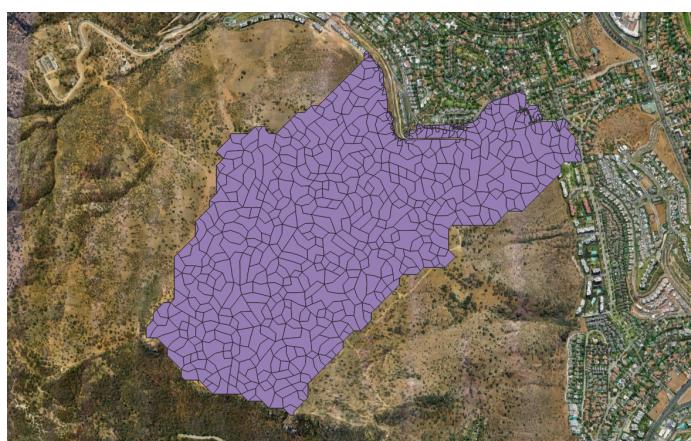


Figura 5: Caso 1 (c): capa poligonal resultante tras aplicar Polylla.

4.2. Caso 2: Figura compleja sin agujeros

Este caso utiliza una geometría sin agujeros, pero con mayor complejidad que el caso anterior debido a los múltiples irregularidades del contorno que lo deforman, para observar el comportamiento del flujo en contornos más irregulares.

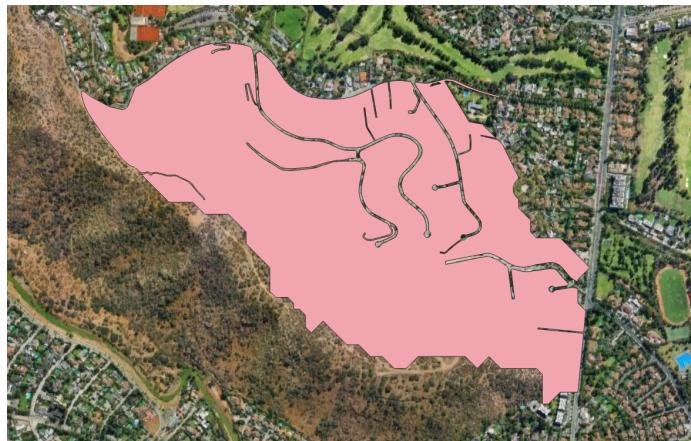


Figura 6: Caso 2 (a): capa poligonal de entrada, sin triangular.

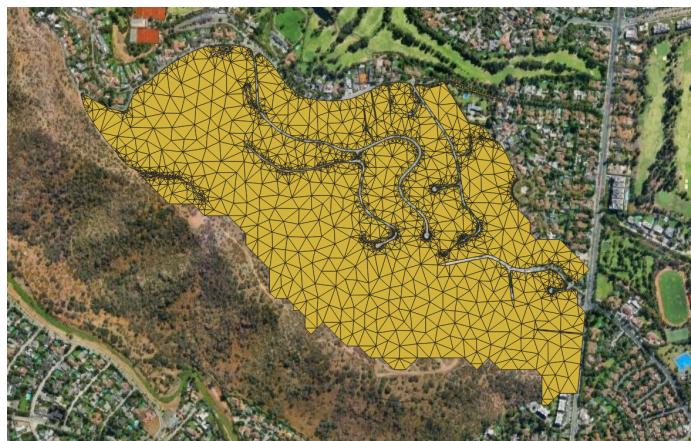


Figura 7: Caso 2 (b): capa triangulada ($\text{min_angle}=20$, $\text{max_area}=1000$).

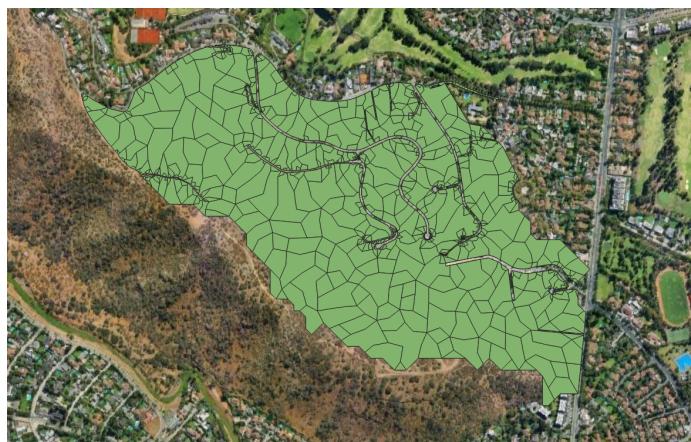


Figura 8: Caso 2 (c): capa poligonal resultante tras aplicar Polylla.

4.3. Caso 3: Figura simple con agujeros

Este caso incluye agujeros en la geometría, lo que requiere que la triangulación incorpore correctamente los vacíos internos para que el resultado final los preserve al aplicar Polylla.



Figura 9: Caso 3 (a): capa poligonal de entrada, con agujeros, sin triangular.



Figura 10: Caso 3 (b): capa triangulada ($\text{min_angle}=20$, $\text{max_area}=1000$).



Figura 11: Caso 3 (c): capa poligonal resultante tras aplicar Polylla.

4.4. Caso 4: Figura compleja con agujeros

Este caso corresponde a una geometría más compleja, con agujeros, caminos y contornos irregulares, representativa de entradas difíciles dentro del flujo del plugin.



Figura 12: Caso 4 (a): capa poligonal de entrada, con agujeros, sin triangular.



Figura 13: Caso 4 (b): capa triangulada ($\text{min_angle}=20$, $\text{max_area}=1000$).



Figura 14: Caso 4 (c): capa poligonal resultante tras aplicar Polylla.

4.5. Observaciones generales.

En los cuatro casos se observa que el flujo completo se ejecuta exitosamente, generando salidas en cada etapa (triangulación y posterior aplicación de Polylla). La triangulación se ejecuta con los parámetros definidos (ángulo mínimo y área máxima), y Polylla genera una malla poligonal a partir de la triangulación sin introducir puntos adicionales al dominio. En los casos con agujeros, el resultado final preserva los vacíos internos definidos en la geometría de entrada. En ninguno de los casos presentados se observaron cierres inesperados de QGIS, mostrando un funcionamiento estable del enfoque adoptado para este conjunto de pruebas.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

En este trabajo se integró el algoritmo Polylla, originalmente en C++, en QGIS mediante la creación de una librería de Python a través de un binding, lo que permitió ejecutar Polylla desde el entorno de un plugin. En paralelo, se rediseñó el flujo de generación de mallas a partir del trabajo previo, separando la etapa de triangulación de la etapa de aplicación de Polylla. Esta decisión permitió aislar la triangulación como un subprocesso externo y reducir los cierres inesperados de QGIS que ocurrían en la solución anterior durante la generación de triangulaciones en geometrías complejas.

Como resultado, se implementó un plugin con un flujo de trabajo completo: selección de una capa poligonal, triangulación con parámetros configurables (ángulo mínimo y área máxima) y posterior generación de una malla poligonal mediante Polylla. La interfaz se organizó en dos diálogos alineados con estas etapas, permitiendo ejecutar el proceso y obtener salidas en disco para la triangulación y en memoria para la malla poligonal final. En los casos demostrativos presentados se observó un funcionamiento estable del flujo, sin cierres inesperados de QGIS.

Durante el desarrollo se intentaron alternativas que no pudieron adoptarse. La triangulación con SciPy se descartó debido a limitaciones para controlar parámetros necesarios del proceso. Asimismo, la ejecución mediante QProcess no funcionó de forma consistente en el contexto del plugin, por lo que se optó por una invocación de subprocesso desde Python. En la integración de Polylla también se decidió no incorporar la funcionalidad de GPU del proyecto original, con el fin de simplificar la implementación y concentrarse en una versión CPU compatible con el flujo del plugin.

5.2. Trabajo futuro

Quedan abiertas varias líneas de trabajo para extender y evaluar la solución. En primer lugar, por limitaciones de tiempo, no se realizó un conjunto de pruebas cuantitativas para caracterizar el efecto de Polylla sobre la calidad geométrica de las mallas en distintos

tipos de cuencas en comparación con la disolución presentada por Villarroel [4] en su memoria. Se recomendaría definir un diseño experimental sistemático y medir métricas como convexidad e índices de forma (por ejemplo, factor de forma), comparando resultados para distintas combinaciones de parámetros de triangulación y opciones de Polylla.

En segundo lugar, también es posible explorar la incorporación de la versión con GPU del proyecto original de Polylla, así como añadir herramientas dentro del plugin para cálculo y visualización de métricas de calidad directamente en QGIS, facilitando el análisis y la comparación de mallas en un mismo entorno.

Bibliografía

- [1] S. Jankowfsky, F. Branger, I. Braud, P. Viallet, S. Debionne, y Rodriguez. F., «Development of a suburban catchment model within the LIQUID framework».
- [2] W. Flugel, «Delineating Hydrological Response Units by Geographical Information System Analyses for Regional Hydrological Modelling Using PRMS/MMS in the Drainage Basin of the River Bröl, Germany.». [En línea]. Disponible en: <https://doi.org/10.1002/hyp.3360090313>
- [3] S. Salinas Fernández, N. V. Hitschfeld Kahler, A. A. Ortiz Bernardin, y H. Si, «POLYLLA, polygonal meshing algorithm based on terminal-edge regions». [En línea]. Disponible en: <https://repositorio.uchile.cl/handle/2250/186136>
- [4] S. Villarroel, «Mejoramiento automático de mallas compuestas de unidades de respuesta hidrológica (URHS) implementado en QGIS». [En línea]. Disponible en: <https://repositorio.uchile.cl/handle/2250/198101>
- [5] P. Sanzana, «Automatización del Procesamiento de Unidades de Respuesta Hidrológica (URHs) con GRASS para un Modelo Hidrológico Distribuido». [En línea]. Disponible en: <https://repositorio.uchile.cl/handle/2250/102771>
- [6] J. R. Shewchuk, *Triangle, Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*.