Escuela Politécnica Superior de Jaén

Grado en Ingeniería Geomática y Topográfica

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

PLUGIN PARA SOFTWARE GIS FOSS PARA LA ELABORACIÓN DE TRIANGULACIONES CONSTREÑIDAS

Alumno: David López Villegas

Tutor: Prof. D. Manuel Antonio Ureña Cámara
Tutor: Prof. D. Francisco Javier Ariza Lopéz
Dpto: Ingeniería Cartografíca, Geodésica y Fotogrametría

junio, 2019

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Ingeniería Cartografíca, Geodésica y Fotogrametría

Don Manuel Antonio Ureña Cámara y Don Francisco Javier Ariza López , tutores del Proyecto Fin de Carrera titulado: “Plugin para Software GIS FOSS para la elaboración de Triangulaciones Constreñidas”, que presenta David López Villegas, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, junio de 2019

El alumno:

David López Villegas

Los tutores:

Manuel Antonio Ureña Cámara
y
Francisco Javier Ariza López

Índice

1. INTRODUCCIÓN

Este documento representa un Trabajo de Fin de Grado del tipo Trabajo Teórico. Su estructura y estilos siguen las normas establecidas por su institución, la Escuela Politécnica Superior de Jaén, perteneciente a la Universidad de Jaén. Dentro de la modalidad Trabajo Teórico, éste será adaptado para desarrollo de Software de Información Geográfica.

1.1. Objetivos

El objetivo fundamental de este proyecto es desarrollar un plugin para QGIS que permita determinar la triangulación constreñida de Delaunay de un conjunto de puntos de cota conocida.

Los constreñimientos admitidos serán líneas de rotura e islas con el fin de ser aplicados a la generación de levantamientos topográficos.

1.2. Justificación

Para llevar a cabo un Levantamiento Topográfico, en términos de materiales necesarios, se necesita una Estación Total y/o un receptor Global Navigation Satellite System (GNSS), algunas veces un asistente de Topógrafo, y un equipo informático para realizar el procesado de datos de la Estación Total y/o GNSS. Con ello se lleva a cabo el Levantamiento Topográfico, que permitirá modelar la superficie topográfica del terreno, obteniendo el Modelo Digital de Elevaciones (MDE).

Uno de los aspectos donde se puede mejorar la viabilidad económica de este tipo de proyectos es en la elección de un software específico para el procesado de datos. Hoy en día, una de las soluciones en el mercado es AutoCAD, junto con su plugin MDT, o ArcGIS que tiene métodos de creación de triangulaciones. Escoger éstas soluciones fuerza a incluir un gasto en el proyecto asociado al pago de licencias de software. Además, este tipo soluciones para el procesado de datos, no permiten adaptabilidad para casos de uso no reflejados en su diseño, ya que por su licencia de uso no permiten modificaciones en su comportamiento, y están

adaptados para un flujo de trabajo muy concreto, sin posibilidad, en el caso de un Levantamiento Topográfico, de disponer de escalabilidad.

En cambio, el uso de Free Open Source Software (FOSS) es una alternativa en auge por ser gratuita y accesible. Este permite ser ajustado a un determinado proyecto en concreto. Una de las soluciones más comunes es QGIS, éste es un Sistema de Información Geográfica que permite la totalidad de los procesos implicados en un levantamiento topográfico, excepto la resolución de métodos topográficos, y la generación de Triangulaciones de Delaunay Constreñidas (CDT) a partir de una nube de puntos, de Líneas de Rotura e Islas.

El desarrollo de un plugin para generar MDE es un aspecto necesario para que QGIS pueda ser usado como herramienta para la obtención de levantamientos topográficos.

2. ANTECEDENTES

Los avances producidos en el campo del estudio de triangulaciones planimétricas de nubes de puntos tiene su máximo auge en los años setenta y ochenta del siglo pasado. En esa época se desarrollan en gran parte las diferentes metodologías para implementar el Diagrama de Voronoi y la Triangulación de Delaunay. En los años posteriores, las investigaciones se centran en mejorar estos algoritmos para conseguir que sean mas eficientes. Recientemente, los estudios han estado centrados en el desarrollo de nuevas metodologías para incluir constreñimientos a dichas triangulaciones.

Para poder entender el Estado del Arte hasta la actualidad, en primer lugar se ofrecen unos conceptos previos generales para comprender ámbito de aplicación de este tipo de algoritmos.

2.1. Modelo Digital de Elevaciones

Un MDE es una representación finita de una función continua en dos variables. La representación espacial de la altimetría utiliza diferentes modelos, como pueden

ser Grids, Curvas de Nivel, Red de Triángulos Irregulares (TIN) (ver Ilustración 1). Este proyecto usará la aproximación TIN, específicamente en el desarrollo de la Triangulación de Delaunay (van Kreveld, 1997).

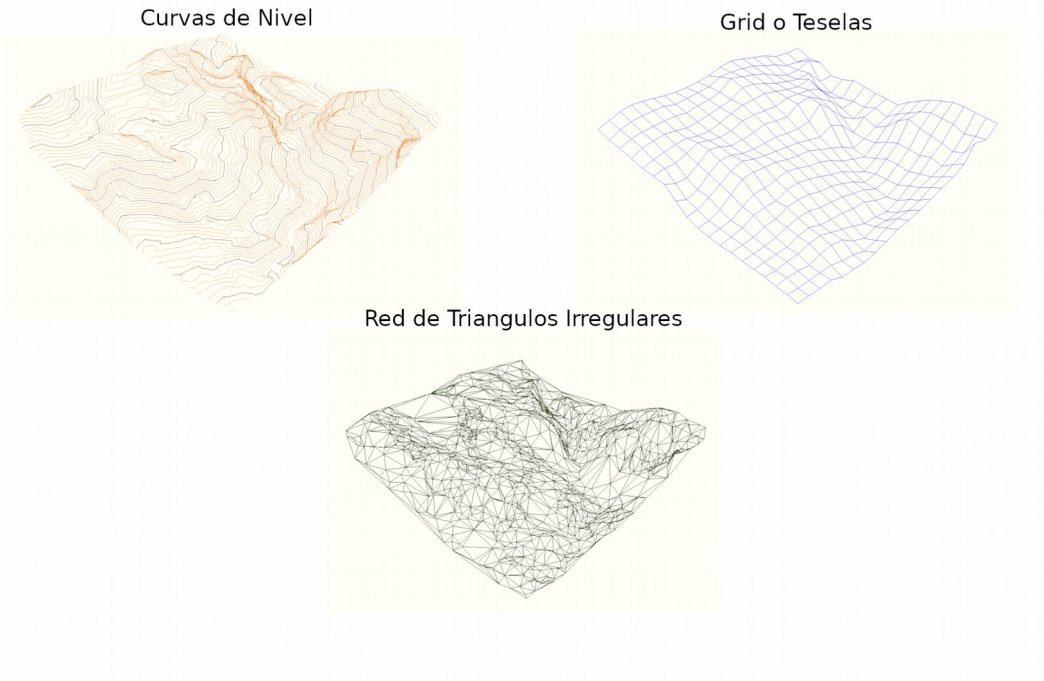


Ilustración 1: Tipos de MDE

2.1.1. Red de Triángulos Irregulares (RTI ó TIN)

Este MDE define una superficie a partir de un conjunto de puntos elegidos convenientemente que pertenecen a esa superficie. En el ámbito de la Topografía, para definir la superficie del terreno mediante una RTI ó Triangulación, solo es necesario medir algunos puntos característicos que pertenezcan a ésta.

Un TIN es una función continua que no es diferenciable en todo el dominio (van Kreveld, 1997). Consiste en un conjunto finito de puntos planimétricos que están almacenados junto con sus elevaciones. Cualquier punto en el dominio que quiera ser observado caerá en un vértice, en un segmento o en un triángulo de la triangulación. Si el punto no cae en una de estas situaciones, entonces su elevación es obtenida mediante interpolación lineal de los tres vértices que componen el

triángulo al que pertenece el vértice.

Una triangulación normalmente comienza con un conjunto de puntos, $p_i \in P$, que son conectados con segmentos, $e_i \in \varepsilon$, de tal modo que estos segmentos forman triángulos, $t_i \in T$. Estos segmentos y triángulos pueden formalmente ser considerados como conjuntos de dos o tres puntos, respectivamente. El objetivo de este proceso es, formar una entidad geométrica unificada de los puntos dados, P (Bærentzen, Gravesen, Anton, & Aanæs, 2012)

Desde un punto de vista matemático, dado un conjunto de puntos $P := \{p_1, p_2, \ldots, p_n\}$, no todos colineales, para poder definir formalmente una triangulación de P, se define la Subdivisión Planar Máxima como una subdivisión S de tal modo que ningún segmento conectando dos puntos puede ser añadido sin que rompa su planicidad. Es decir, cualquier segmento que no está dentro de S intersecta uno de los segmentos existentes (de Berg, Cheong, van Kreveld, & Overmars, 2008).

Una característica de cualquier triangulación es su número de segmentos y de triángulos. Como se puede ver en la siguiente definición, estos dependen completamente del número de puntos que forman parte de la triangulación y el número de puntos que pertenecen a su borde.

Para un conjunto P de puntos en el plano, no todos colineales, y k denota el número de puntos en P que caen dentro de la frontera de la envolvente convexa de P. Entonces, cualquier triangulación de P tiene $2n-2-k$ triángulos y $3n-3-k$ segmentos (de Berg et al., 2008).

Existen muchas triangulaciones para un mismo conjunto de puntos. Dependiendo de cómo se construyen los triángulos, sus ángulos interiores presentan un comportamiento descrito a continuación.

Sea C un círculo, l una linea que intersecta a C en los puntos a y b, y p, q, r, y s puntos que caen en el lado l (ver Ilustración 2). Asumiendo que p y q caen en C,

que r cae dentro de C, y que s cae fuera de C. Entonces $\measuredangle arb > \measuredangle apb = \measuredangle aqb > \measuredangle asb$ (de Berg et al., 2008)

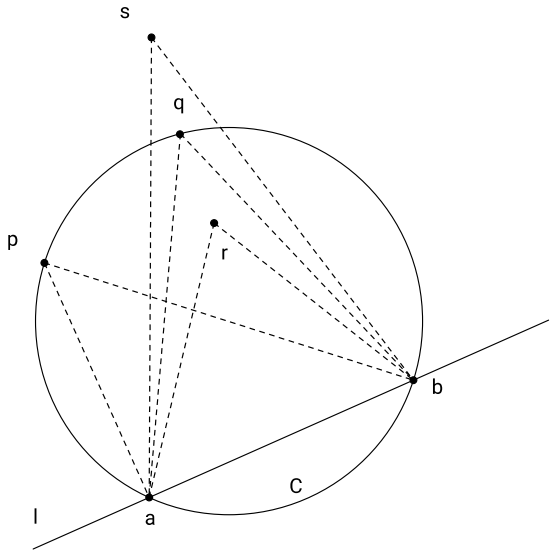


Ilustración 2: Triángulo y su círculo circunscrito, y representación de otro triángulo con la misma base que el anterior, donde se visualiza como se comportan los ángulos de este triángulo dependiendo si el vértice cae dentro, encima o en el exterior del círculo (de Berg et al., 2008).

Además, sea un segmento $\overline{p_i p_j}$ un segmento del triángulo $p_i p_j p_k$ y $p_i p_j p_l$ (ver Ilustración 3), y sea C un círculo a partir de p_i, p_j, y p_k. Si los puntos p_i, p_j, p_k, p_l forman un cuadrilátero convexo y no cae en un círculo común, entonces exactamente uno de los segmentos $\overline{p_i p_j}$ y $\overline{p_k p_l}$ es un segmento ilegal (de Berg et al., 2008).

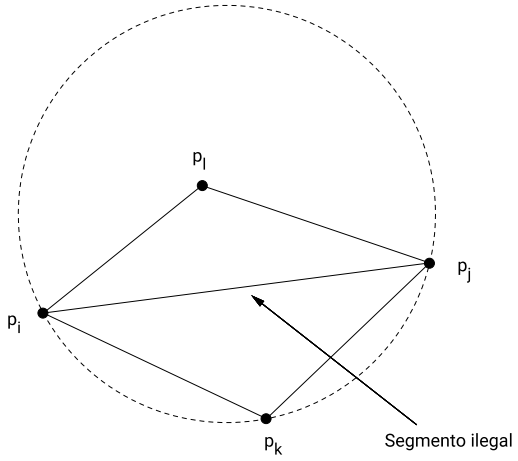


Ilustración 3: Debido a que el punto p_l cae dentro del círculo C, el segmento $\overline{p_i p_j}$ (de Berg et al., 2008).

2.2. Triangulación de Delaunay

La Triangulación de Delaunay es una triangulación que cumple con unas propiedades específicas.

Una triangulación de n vértices, siendo $n \geq 2$ es de Delaunay si y solo si el Círculo Circunscrito (CC), esto es un círculo construido a partir de tres puntos no alineados, de cada polígono está libre de puntos (Guibas & Stolfi, 1985).

Para Bærentzen et al. (1985), esta es una triangulación especial, ya que todos sus triángulos cumplen la condición de que su CC está vació. Establecen que, dado un TIN; si, y solo si cualquier CC de cualquier triángulo, perteneciente a la triangulación, no contiene otro punto dentro, el TIN quedará definido como una Triangulación de Delaunay (Bærentzen et al., 2012).

Para Guibas et al. (1985), dados tres vértices pertenecientes a un polígono de la triangulación y un vértice exterior, se dice que el triángulo es un triángulo de Delaunay si el vértice exterior se encuentra fuera del CC formado a partir del triángulo que está siendo comprobado (ver Ilustración 4) (Guibas & Stolfi, 1985).

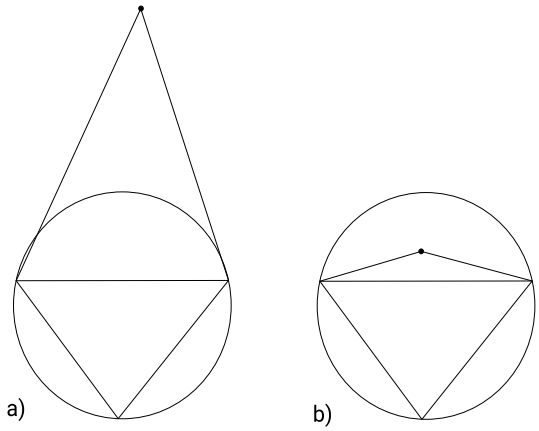


Ilustración 4: Criterio de Círculo Vacío.

En la ilustración 4, la opción a), se corresponde con un triángulo válido, que cumple con el criterio de círculo circunscrito vacío (CCV); por otro lado, la opción b), no lo cumple.

2.2.1. Triangulaciones de Delaunay Constreñidas (CDT)

Para (Bærentzen et al., 2012), una CDT es una triangulación de un conjunto de puntos, con la restricción de que la triangulación debe contener ciertos segmentos, los cuales pueden no cumplir la condición de Delaunay, y el resto de segmentos de la Triangulación de Delaunya (TD) no pueden cruzar los segmentos bloqueados o Lineas de Rotura (LR).

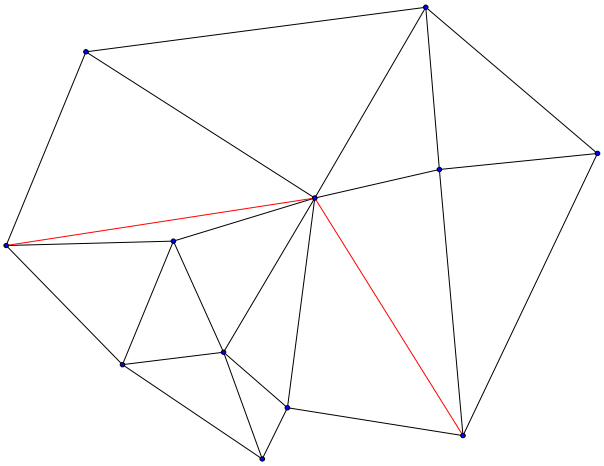


Ilustración 5: CDT cuyos segmentos de color rojo se corresponden con LR.

2.3. Algoritmos

A continuación se exponen diferentes algoritmos que se han desarrollado para abordar el proceso de construcción de una CDT a partir de un conjunto de puntos conocidos. No solo se incluye algoritmos de triangulación, si no también procedimientos específicos en los procesos de inserción de Líneas de Roturas e Islas.

2.3.1. Divide y Vencerás

El tiempo de ejecución es $O(N\log(N))$, siendo el algoritmo asintóticamente óptimo (Lee & Schachter, 1980). En su estructura de datos, este algoritmo usa listas doblemente enlazadas y circulares para la ordenación de la nube de puntos. Para cada punto v_i del conjunto V, se genera una lista de adyacencia ordenada de puntos $v_{i1},\ldots,v_{ik}$, donde (v_i,v_j), $j=1,\ldots,k$, es un segmento de Delaunay. Con esta estructura de datos, para un punto v_{ip}, se dispone de la posibilidad de conocer el punto próximo que aparece en el sentido de las agujas del reloj (CW) inmediatamente posterior a v_{ij} y el punto próximo que aparece en el sentido

contrario de las agujas del reloj (CCW). Por ejemplo, en la Ilustración 6, para obtener el vértice v_5 partiendo de la información en las listas de v_1, se puede determinar indicando que es el punto anterior al vértice v_4, o indicando que es el punto posterior a v_6.

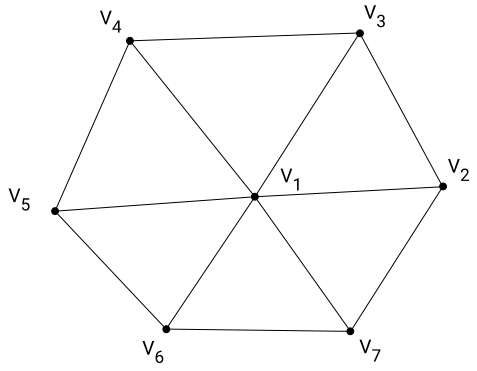


Ilustración 6: Esquema de Lista Circular Doblemente Enlazada correspondiente a v_1 (Lee & Schachter, 1980).

Si el vértice v_i pertenece a la envolvente convexa, entonces su primer valor de la lista de adyacencia es el punto perteneciente a la envolvente convexa en sentido CCW.

A continuación, se ordena renombrando los vértices de forma lexicográfica $v_1 < v_2 < ... < v_n$, de tal modo que $(x_i, y_i) = v_i < v_j$ si y solo si $x_i \leq x_j$ e $y_i < y_j$.

Una vez que el conjunto ha sido ordenado y almacenado en memoria, como en cualquier algoritmo del tipo Divide y Vencerás, se lleva a cabo su resolución en diferentes subprocesos. En este caso, los procesos consisten en dividir el conjunto en dos subconjuntos V_l y V_r, de la forma $V_l = \{v_1, \ldots, v_{\lfloor N/2 \rfloor}\}$ y $V_R = \{v_{\lfloor N/2 \rfloor}, \ldots, v_N\}$, y como el conjunto de datos ha sido ordenado, este proceso se vuelve trivial. Acto seguido se determinan las tangentes superiores e inferiores de los dos subconjuntos (ver Ilustración 7), y por último se construyen los segmentos de Delaunay en el espacio comprendido entre estas dos tangentes. Estos dos últimos procesos, que son explicados a continuación, son ejecutados de forma recursiva hasta que no sea posible realizar ninguna subdivisión más debido a que el número de puntos del subconjunto V_l o V_R sea inferior o igual a 4, cuya solución se vuelve trivial (Lee & Schachter, 1980).

El proceso de determinación de tangentes calcula las tangentes superior e inferior comunes (TSC y TIC) a las envolventes convexas de los subconjuntos V_l y V_r (ver Ilustración 7). Esta función genera dos nuevos segmentos, la TIC y TSC de la envolvente convexa de V_l y V_r ($CH(V_l)$ y $CH(V_r)$). Estas dos tangentes comunes pertenecen a la triangulación de Delaunay final. La determinación de la tangente superior e inferior se llevan a cabo de forma análoga (el pseudocódigo de esta función puede verse en el Anexo 1.1).

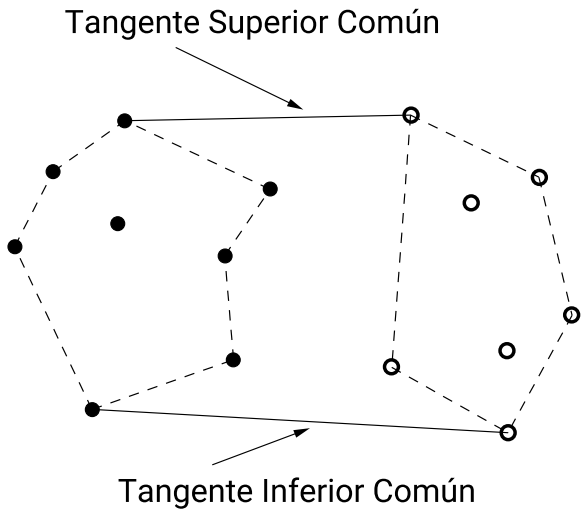


Ilustración 7: Tangentes superior e inferior de dos envolventes convexas.

Para el proceso de Unión comentar que:

- Inicialmente, se evalúan los puntos adyacentes a los puntos de la TIC, que utilizando el criterio del CCV se puede conectar; el punto izquierdo (en V_l) de la TIC, a un punto adyacente del punto derecho (en V_R) de la TIC o; el punto derecho (en V_R) de la TIC, a un punto adyacente del punto izquierdo (en V_l) de la TIC.

- La lista de adyacencia del punto derecho (izquierdo) de la TIC en V_R (V_l) del segmento en consideración, es examinado en sentido CW (CCW) empezando

con el punto izquierdo (derecho) del segmento:

- El procedimiento Insertar (A, B) inserta un punto A dentro de la lista de adyacencia de un punto B y el punto B es insertado dentro de la lista de adyacencia del punto A:

- El procedimiento Borrar (A, B) elimina un punto A de la lista de adyacencia de un punto B y elimina B de la lista de adyacencia del punto A:

- la función Qtest (H, I, J, K) comprueba el cuadrilátero formado por los vértices H, I, J y K en sentido CCW; devuelve Verdadero si el CC del triángulo formado por los vértices HIJ no contiene a K, de lo contrario devuelve Falso.

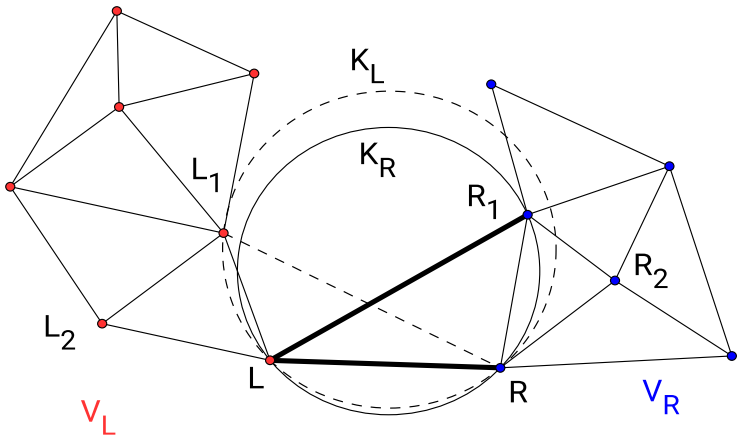


Ilustración 8: Proceso algoritmo Unión (Merge). Las líneas en negrita se corresponden con lados de Delaunay introducidos por el algoritmo de Unión (Lee & Schachter, 1980).

Por último, el proceso Unión, genera segmentos de Delaunay utilizando el criterio del CCV empezando desde la TIC (ver Ilustración 8). Sus datos de entrada son $CH(V_l)$, $CH(V_r)$, TIC y TSC. Une las dos triangulaciones empezando desde TIC, zigzagueando verticalmente hasta que TSC es encontrado. Todos los nuevos segmentos insertados se corresponden con segmentos de Delaunay, y no vuelven a ser evaluados.

2.3.2. Iterativo

Este algoritmo presenta un tiempo de ejecución empírico $O(N^{3/2})$, cuando los datos son uniformemente distribuidos, pero podría tomar $O(N^2)$ en el peor de los

casos (Lee & Schachter, 1980).

Dado un conjunto V de N puntos, la envolvente convexa rectangular es utilizada para definir la región. Los vértices de la región son también incluidos dentro de la triangulación. El rectángulo es separado aproximadamente en $\sqrt{N}$ celdas; los puntos son reordenados por celdas (ver Ilustración 9); e iterativamente, se triangula cada celda, empezando en alguna celda, siguiendo con las celdas vecinas según el esquema de ordenamiento escogido (ver Ilustración 9), hasta que se completa toda la región.

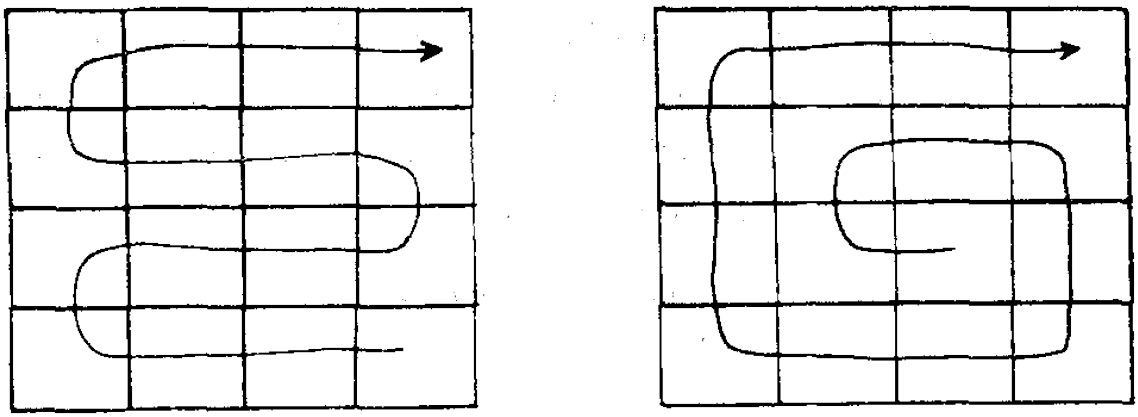


Ilustración 9: Dos posibles esquemas de ordenamiento de celdas (Lee & Schachter, 1980).

Para cada celda, el proceso consiste en insertar un punto en su interior; conectar este punto a las cuatro esquinas de la celda, haciendo una triangulación inicial. El resto de puntos en la celda es insertada una a una. Cada punto insertado en un triangulo genera hasta cuatro cuadriláteros estrictamente convexos que serán evaluados por el criterio del ángulo Máx-Min, criterio que consiste en la maximización de los ángulos mínimos de un triángulo, evitando ángulos demasiado agudos. Cuando el punto cae en un segmento, se producen cuatro cuadriláteros.

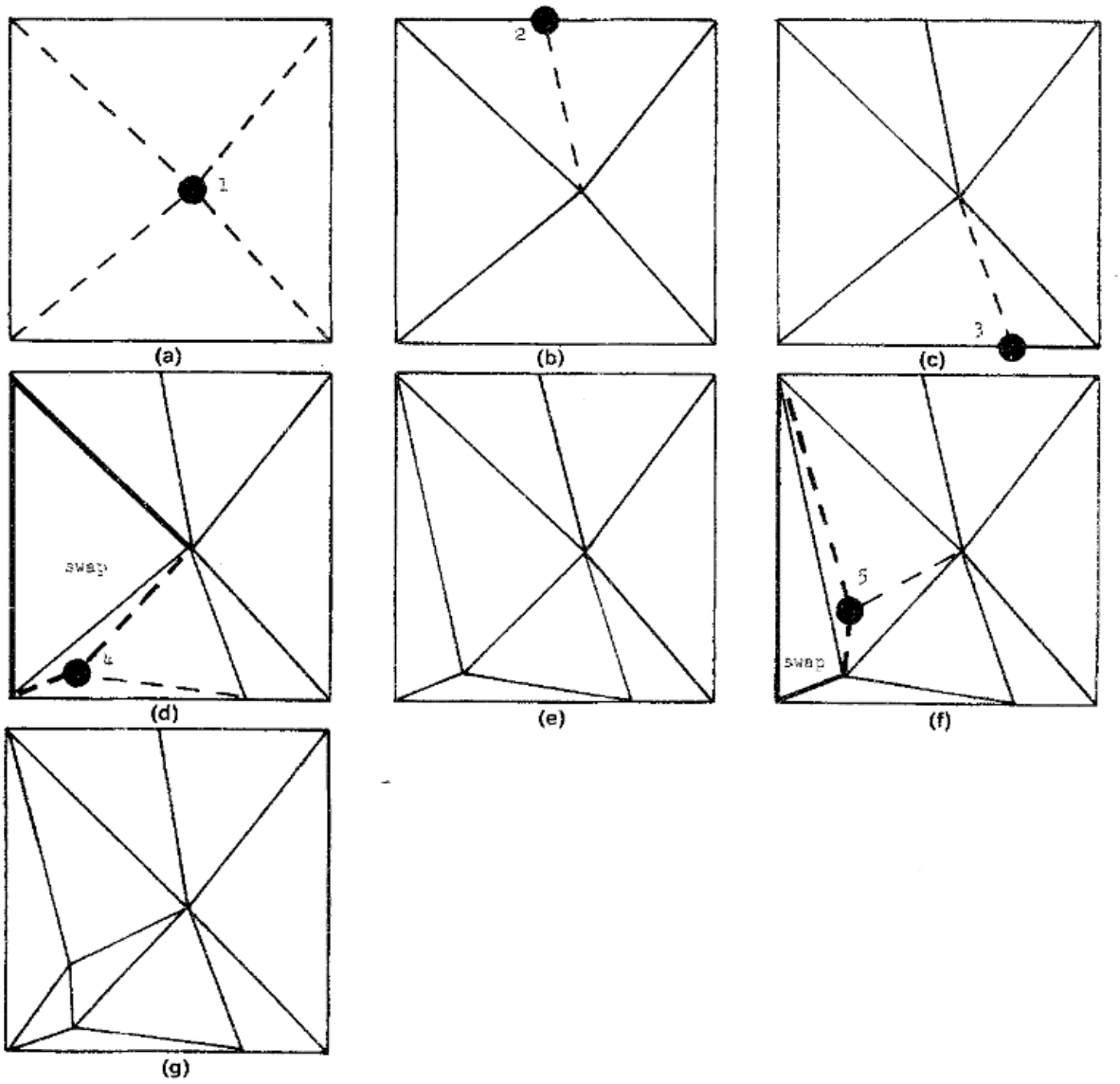


Ilustración 10: Ejemplo mostrando una Triangulación de Delaunay a partir del método iterativo. Los nuevos puntos son mostrados con círculos rellenos (Lee & Schachter, 1980).

Cada cambio realizado debido al criterio del ángulo Máx-Min podría resultar en un nuevo cuadrilátero que necesita ser validado. El procedimiento de cambio puede propagarse además hacia fuera, pero siempre termina. (Lee & Schachter, 1980)

2.3.3. Algoritmos de Líneas de barrido

Fortune (1987) inventó otro esquema para construir la Triangulación de Delaunay en tiempo de ejecución del orden $O(n\log(n))$, que es el método más

eficiente planteado hasta el momento. Éste está enfocado a la obtención del diagrama de Voronoi. En este algoritmo se hace el seguimiento del estado de dos conjuntos. El primero es una lista de bordes llamados “frontera” del diagrama. Estos segmentos son un subconjunto del diagrama de Delaunay y forman un recorrido por el exterior de la triangulación incompleta. El algoritmo utiliza una “cola de prioridad”, llamada “evento de cola”, para el seguimiento de lugares donde la linea de barrido debe parar. Esta cola almacena dos tipos de eventos llamados eventos de “círculo” y eventos de “vértice”. Los eventos de vértice suceden cuando la linea de barrido alcanza un vértice, y el evento de círculo sucede cuando llega al final del círculo formado por tres vértices adyacentes en la frontera. El algoritmo barre con una línea el conjunto de vértices en el sentido ascendente de la coordenada y, procesando cada evento que es encontrado (Fortune, 1987).

En la implementación de Fortune, la frontera es un lista simple enlazada de medios-bordes, y la localización del punto es llevado a cabo utilizando un esquema de zonas de acumulación de valores. La coordenada x de un punto que va a ser insertado es guardado en una zona de acumulación para estar cerca al segmento de frontera correcto. Este segmento es situado en una de las zonas de acumulación donde el punto ha caído, de esta manera los puntos futuros que estén cerca pueden ser localizados con rapidez. Este algoritmo funciona bien en cuanto la consulta de los puntos y los bordes están bien distribuidos entre las zonas. Una zona de acumulación de valores es también usado para representar la “cola de prioridad”. Los miembros de la cola son almacenados en zonas de acumulación de acuerdo a sus prioridades, así que encontrar el mínimo implica buscar primero en las zonas de acumulación no vacías y sacar el elemento mínimo (Su & Drysdale, 1997).

El tiempo de ejecución de Fortune es proporcional al costo de búsqueda y actualización del evento de cola y la estructura de datos en la linea de barrido. Para Biniaz y Dastghaibyfard (2012), el algoritmo de Fortune muestra un tiempo de ejecución del orden $O(n\sqrt{n})$, pero la redefinición del código de Fortune utilizando la estructura de datos basado en array del tipo Montículo para representar la “cola de prioridad” en lugar del “esquema de cubeta”obtiene un tiempo de ejecución del

orden $O(n\log(n))$, mejorando el rendimiento en grandes conjuntos de datos (Biniaz & Dastghaibyfard, 2012) (Biniaz & Dastghaibyfard, n.d.).

Existen otras implementaciones para la elaboración de la triangulación de Delaunay de forma directa, sin calcularla a partir del diagrama de Voronoi como en Fortune, entre ellas se encuentra el algoritmo de Zalik.

Para facilitar la explicación de este algoritmo, se definen una serie de conceptos para el algoritmo ya inicializado, considerando que once puntos han sido introducidos y nos encontramos en el ciclo que inserta el punto doce, con la correspondiente triangulación generada hasta el momento (ver dibujo inferior de la Ilustración 12). La polilínea formada por los segmentos de triángulo que se encuentran en el borde superior de la triángulción (ver linea discontinua superior del dibujo superior de la Ilustración 12) se corresponde con la línea de avance y separa los puntos barridos de los no barridos. La línea discontinua inferior del dibujo superior de la Ilustración 12 se denomina Envolvente Convexa Inferior y comparte con la línea de avance los vértices nombrados izquierda y derecha (Zalik, 2005).

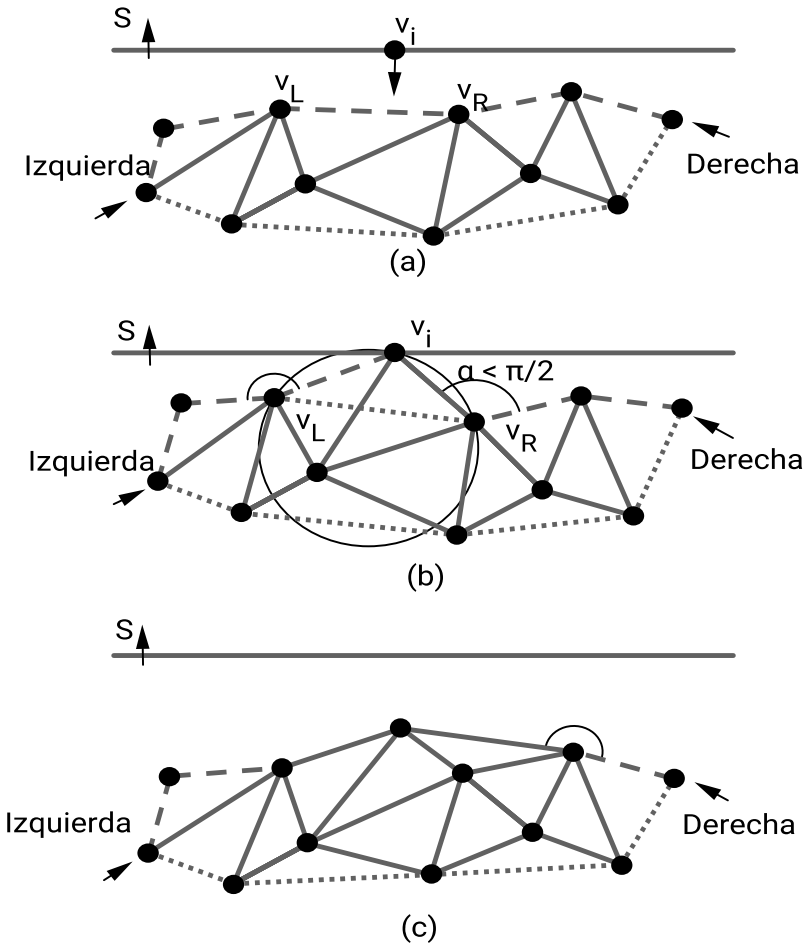


Ilustración 12: Los segmentos frontales de avance son simbolizados con línea discontinua y la envolvente convexa esta representada mediante línea de puntos: proyección vertical del siguiente punto que es alcanzado por la linea de avance (a), nuevo triangulo construido y validado, línea de avance es actualizada, desde el punto v_i hacia la derecha (b), y después de ello se detiene (c) (Biniaz & Dastghaibyfard, 2012).

Una vez definidos los conceptos anteriores, el proceso que se lleva a cabo en un ciclo consiste en:

- Escoger el siguiente punto que no haya sido barrido.
- Proyectar el punto.
- Si la proyección del punto cae en la Linea de Avance. La línea de proyección intersecta un segmento cuyos vértices son uno a la izquierda (V_L) y otro a la

derecha (V_R). Y el triangulo $\Delta_{i,R,L}$ es creado y recursivamente comprobado para cumplir el criterio de Delaunay. Propuesto por Zalik, con el fin de heurísticamente evitar la creación de polígonos astilla, en el proceso de resolución del lado izquierdo relativo al punto V_L, el ángulo entre los vectores de vértices v_i, v_R, v_R+ (ver Ilustración 13), es observado, y si el ángulo es menor a $\pi/2$, el triangulo $\Delta_{i,R+,R}$ es generado. Si no es así, el proceso hacia el lado izquierdo es detenido. El nuevo triángulo es comprobado por el criterio de Delaunay con sus dos triángulos vecinos $\Delta_{i,R,L}$ y $\Delta_{R+1,l,R}$. El vértice v_R es eliminado de la linea de avance y el proceso se repite para v_i, v_R++, v_R+ hasta que el ángulo entre es mayor que $\pi/2$. Los triángulos obtenidos de esta forma no tienden a violar el criterio de Delaunay con demasiada frecuencia, reduciendo el consumo de tiempo de CPU. El proceso de resolución para el lado derecho es el mismo, ya que el problema es simétrico.

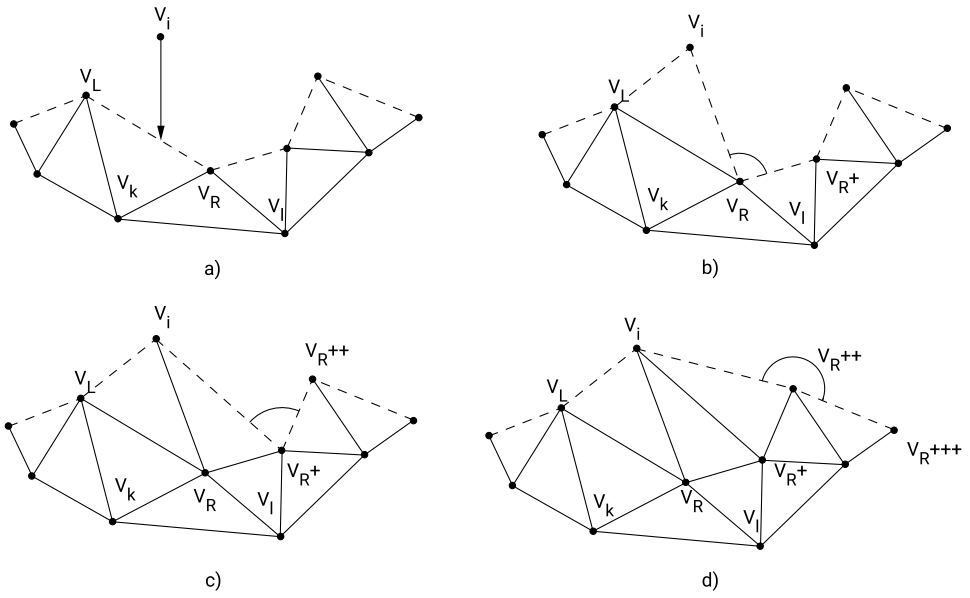


Ilustración 13: La linea de avance es intersectada (Biniaz & Dastghaibyfard, 2012).

- Si la línea de proyección no toca la linea de avance. Para el caso del lado izquierdo (para el lado derecho se convierte en una solución análoga). El triángulo $\Delta_{i,L,-L}$ es generado y es criterio de Delaunay es recursivamente

comprobado (ver Ilustración 14). Los nuevos triángulos en la dirección del lado derecho de la linea de avance es generado con el mismo procedimiento que para el caso en el que la proyección cayera en la Línea de Avance. Los nuevos triángulos también tienen que ser añadidos a la envolvente convexa inferior para ser completada en la parte inferior de la triangulación.

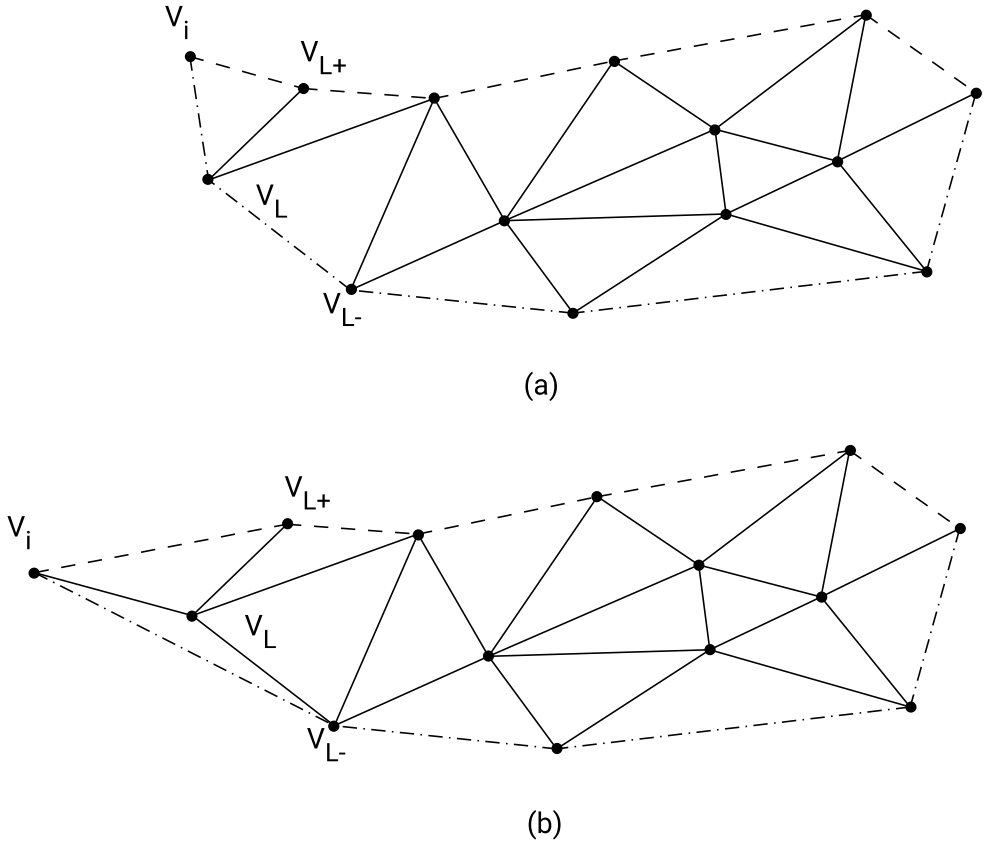


Ilustración 14: La línea de avance no es intersectada

• Si el vértice cae en la línea de barrido. En es caso especial, la solución expuesta por Zalik es sencilla (ver Ilustración 15). Los vértices v_i, v_L y v_R deben tener la misma coordenada y (dentro de una tolerancia e). El triangulo $\Delta_{L,R,k}$ es accedido desde el vértice v_L, y es separado en dos nuevos triángulos $\Delta_{L,i,k}$ y $\Delta_{i,R,k}$. Los triángulos generados han de ser validados y la

linea de avance es actualizada al final.

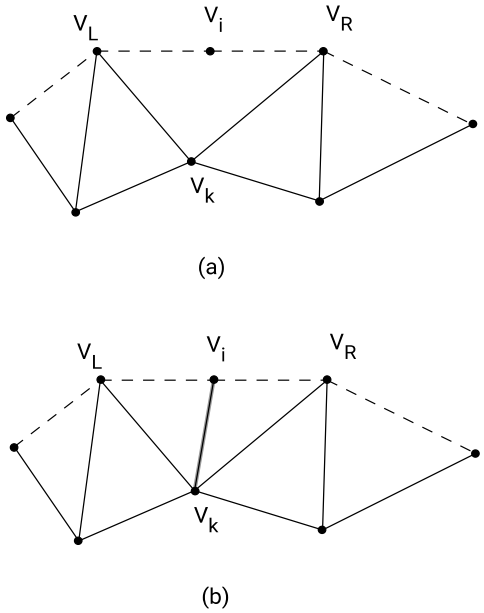


Ilustración 15: Resolviendo el caso especial (Biniaz & Dastghaibyfard, 2012).

- Si el vértice coincide con un vértice de la linea de avance. En este caso V_i es ignorado y el siguiente vértice es cogido inmediatamente.

- Para finalizar, cuando todos los puntos son barridos, algunos vértices de la linea de avance adquiere una distribución cóncava y algunos triángulos no han sido creados todavía. Estos son añadidos procesando la linea de avance, utilizando el algoritmo de Graham para determinar la envolvente convexa. Para ello, tres vértices consecutivos son obtenidos.

2.3.4. Algoritmo Incremental

Probablemente esta clase de algoritmos para la construcción de la triangulación de Delaunay es la más simple de implementar. Estos algoritmos consisten en añadir vértices de uno en uno a la triangulación y actualizarla al finalizar la inserción de este último vértice. Este tipo de algoritmos actualizan el diagrama invirtiendo el sentido de los segmentos que no son validados. Cada

segmento es testeado para su validación a través del criterio del CCV descrito en el apartado 2.1.1. En cuanto se utiliza una inserción de vértices, el número esperado de inversiones de sentido es lineal, no importa como son distribuidos (Su & Drysdale, 1997)

Este algoritmo calcula un triángulo que envuelve a todo el conjunto de triángulos. Básicamente el proceso consiste buscar el triángulo que envuelve el punto a insertar. Esto genera tres nuevos triángulos en la triangulación. Algunas veces, este punto puede caer en un segmento de triangulo; insertar este punto genera cuatro triángulos en su lugar. Para la inserción de este punto, hay que eliminar el segmento donde cae en primer lugar.

Los segmentos generados al insertar un punto en la triangulación son de Delaunay. Sin embargo, los segmentos del triangulo original, pueden no cumplir la condición de Delaunay. Estos segmentos, al ser volteados, pasan automáticamente a cumplir la condición de Delaunay. Los segmentos no son comprobados más de una vez (para cada nuevo punto insertado). Al insertar un punto, los segmentos pueden adquirir un estado, debido al cual, es necesario realizar una comprobación para cerciorar que el segmento cumple la condición de Delaunay. Una vez que todos los segmentos cuyo estado ha cambiado, todos los segmentos en la triangulación pasan el test de Delaunay (Guibas & Stolfi, 1985).

Después de la inserción de un punto, el criterio de CCV evalúa los triángulos generados: estos triángulos son evaluados contra sus triángulos vecinos; estos podrían ser volteados, por lo que se requiere evaluar los triángulos volteados con sus vecinos, definiendo un nivel de adyacencia. Este nivel de adyacencia que será utilizado es establecido por el usuario.

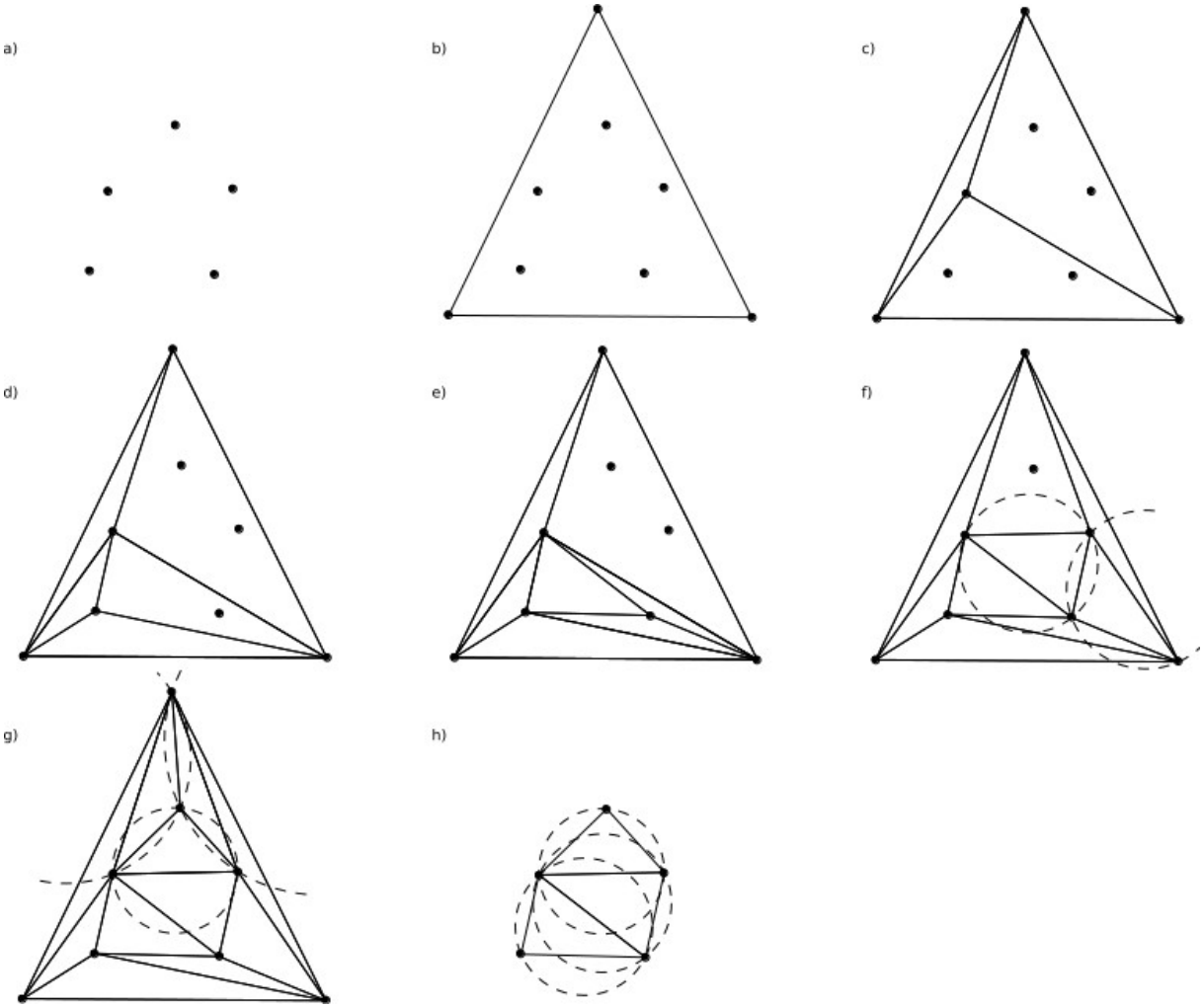


Ilustración 16: Representación de la Triangulación de Delaunay mediante el algoritmo Incremental. La validación de los triángulos han sido representados mediante círculos circunscritos representados mediante líneas discontinuas.

Representación de la Triangulación de Delaunay mediante el algoritmo Incremental. La validación de los triángulos han sido representados mediante círculos circunscritos representados mediante líneas discontinuas.

2.3.5. Algoritmo para la Inserción de Líneas de Rotura en Triangulaciones de Delaunay Constreñidas

En el proceso de inserción de líneas de rotura, en algunos casos, sus segmentos intersectan varios triángulos, como puede observarse en la Ilustración 17.

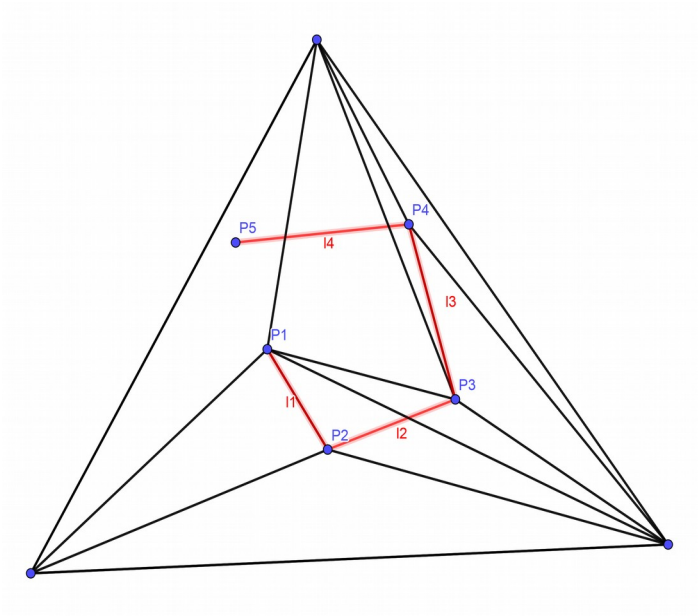


Ilustración 17: Problemática en la inserción de líneas de rotura en una Triangulación de Delaunay Constreñida

En el caso de intersección de segmentos de lineas de rotura, se generan puntos virtuales en el lugar de intersección. Si alguno de los segmentos de linea de rotura es eliminado, el punto virtual también es eliminado. En este algoritmo, se subdividen los segmentos de linea de rotura intersectados, y vuelven a unirse si alguno de los segmentos intersectados es eliminado (Wang, Wu, & Shi, 2007).

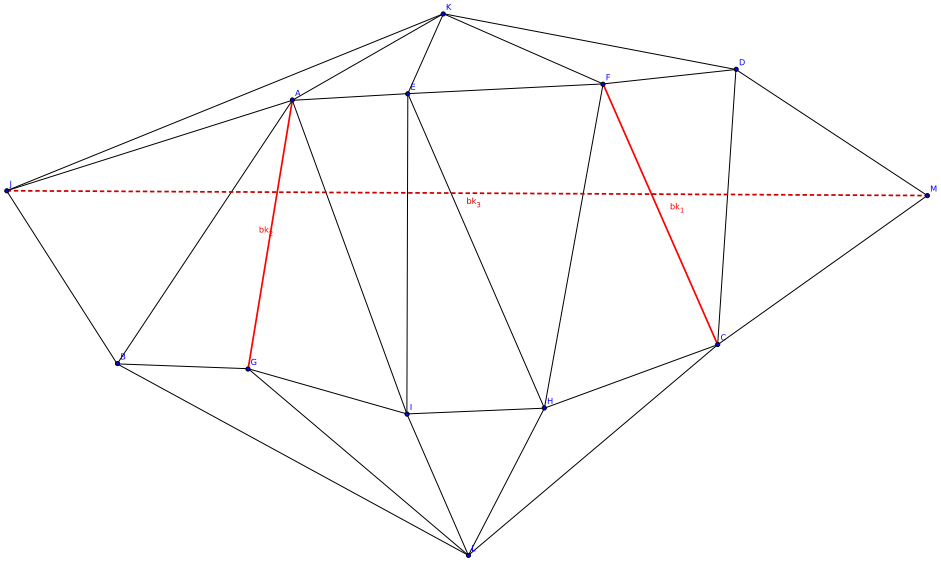


Ilustración 18: Cruce de segmentos de línea de rotura

Para la inserción de lineas de rotura cuyos segmentos cruzan diferentes triángulos, se construye un dominio de influencia de triángulos afectados por el segmento de línea de rotura y estos son eliminados, después se reconstruye la triangulación sobre los vértices de los triángulos eliminados.

El algoritmo funciona de forma recursiva (ver apartado 1.14, 1.15 y 1.16 del ANEXO), dividiendo la zona a triangular en dos partes, la parte superior de la linea de rotura y la parte inferior de la linea de rotura. Para cada ciclo, el vértice que cumple con el criterio de CCV es buscado, y cada vez que se encuentra el punto, la zona de influencia es separada. El criterio de CCV solo es aplicado sobre los posibles vértices utilizados para crear el triángulo. Cada vez que un triángulo es encontrado, el conjunto de puntos utilizados en la búsqueda se divide en dos subconjuntos. El algoritmo termina cuando no quedan más vértices con los que triangular (Vigo, 1997).

2.3.6. Relaciones Geométricas entre Geometrías en Dos Dimensiones

Las metodologías de los algoritmos para la elaboración de TIN mencionados hasta ahora, muestran una descripción general de la metodología, pero no muestran conceptos más elementales con los que han sido necesario trabajar en la implementación de este proyecto. Estos conceptos necesarios para la implementación del algoritmo de inserción incremental y las CDT, definen situaciones topológicas entre los conceptos de punto, segmento y triángulo (geometrías).

En esta sección, cuyo contenido ha sido obtenida de Strobl (2008), se define y enumera el Modelo de Nueve Intersecciones Dimensionalmente Extendido (DE-9IM) que construye las relaciones topológicas entre las geometrías.

En este modelo las geometrías disponen de la siguiente clasificación en función de su definición geométrica:

- Borde: El borde del objeto de una geometría es un conjunto de geometrías de una dimensión inferior.

- Interior: El interior de un objeto geometría se corresponde con los puntos que han caído dentro de la geometría, una vez las geometrías de borde han sido eliminadas.

- Exterior: El exterior de un objeto geometría se corresponde con las geometrías que no están incluidas ni en el Interior ni en el borde.

En la Tabla 1 se puede observar los diferentes tipos de geometrías, junto con sus definiciones de Interior, Borde y Exterior para cada tipo:

Subtipos de Geometría	***Interior (I)***	***Borde (B)***	***Exterior(E)***
Puntos, Multipuntos	Punto, Multipuntos	Conjunto Vacío	Puntos que no caen ni en el Interior ni en el Borde
LineString (Cadena Linea), Line (Linea)	Puntos que quedan al eliminar el Borde	Los dos puntos finales de la geometría	Puntos que no caen ni en el Interior ni en el Borde
Linear Ring (Anillo Lineal)	Todos los puntos del anillo	Conjunto Vacío	Puntos que no caen ni en el Interior ni en el Borde
MultiLineString (Cadena MultiLinea)	Puntos que quedan al eliminar el Borde		Puntos que no caen ni en el Interior ni en el Borde
Polígono	Puntos dentro de anillos	Conjunto de Anillos	Puntos que no caen ni en el Interior ni en el Borde
MultiPolígono	Puntos dentro de anillos	Conjunto de Anillos de sus Polígonos	Puntos que no caen ni en el Interior ni en el Borde

Tabla 1: Definición de Interior, Borde y Exterior para los principales tipos de geometrías descritas por Open Geospatial Consortium.

Para la explicación de la relación topológica de dos tipos de objetos geometría (A y B), cada geometría es representada por su Interior, Borde y Exterior. Todas las posibles relaciones topológicas entre estas dos geometrías pueden ser representadas a partir de una matriz de cuadrada de rango 3. Para construir esta matriz, las filas y columnas se corresponden con la representación de la geometría A y B respectivamente. El valor de un elemento de la matriz se construye a partir de la determinación de la dimensión de la geometría construida a partir de la intersección de dos conjuntos relativos a la geometría A y a la geometría B.

$$DE-9IM(A,B)=\begin{bmatrix} dim(I(A))\cap dim(I(B)) & dim(I(A))\cap dim(B(B)) & dim(I(A))\cap dim(E(B)) \\ dim(B(A))\cap dim(I(B)) & dim(B(A))\cap dim(B(B)) & dim(B(A))\cap dim(E(B)) \\ dim(E(A))\cap dim(I(B)) & dim(E(A))\cap dim(B(B)) & dim(E(A))\cap dim(E(B)) \end{bmatrix}$$

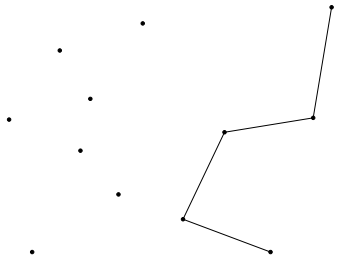
Los Predicados Topológicos son funciones lógicas que son utilizadas para comprobar las relaciones espaciales entre dos objetos geometría. El Modelo de Nueve Intersecciones Dimensionalmente Extendido provee ocho tipos de relaciones espaciales entre puntos, lineas y polígonos.

Predicado Topológico	***Significado***
Igualdad	Las geometrías son topológicamente iguales
Desigualdad	Las geometrías no tienen ningún punto en común
Intersecta	Las geometrías tienen al menos un punto en común (operación inversa a Desigualdad)
Toca	Las geometrías tienen al menos un punto de borde en común, y ningún punto interior en común
Cruza	Las geometrías comparten algunos, pero no todos, los puntos en común, y la dimensión de la intersección tiene una dimensión menor que al menos una de las geometrías
Solapa	Las geometrías comparten algunos, pero no todos, los puntos en común, y la intersección tiene la misma dimensión que las mismas geometrías
Dentro	La geometría A cae en el interior de la geometría B
Contiene	La geometría B cae en el interior de la geometría A (es la función inversa a la función Dentro)

Tabla 2: Predicados Topológicos y sus correspondientes significados, dentro del Modelo de Nueve Intersecciones Dimensionalmente Extendido

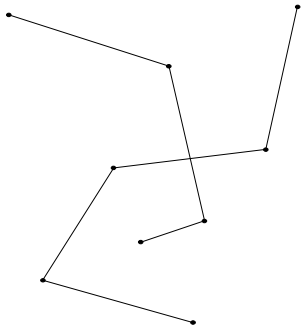
Algunos ejemplos de predicados topológicos son descritos de la siguiente forma:

- Desigualdad: Ejemplo DE-9IM para el caso donde A es una Línea que es desigual a un MultiPunto B. El borde de un punto es por definición el Conjunto Vacío.



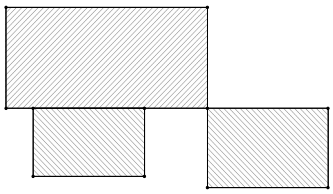
	Interior (B)	Borde (B)	Exterior (B)
Interior (A)	-1	-1	1
Borde (A)	-1	-1	0
Exterior (A)	0	-1	2

- Intersección: Ejemplo DE-9IM para el caso donde A es una Línea que es intersectada por la Línea B. La relación Intersección es inversa a la relación de Desigualdad. Los objetos geometría tienen al menos un punto en común, por lo que la relación Intersección incluye el resto de Predicados Topológicos. Este ejemplo también es válido para el predicado topológico Cruzar.



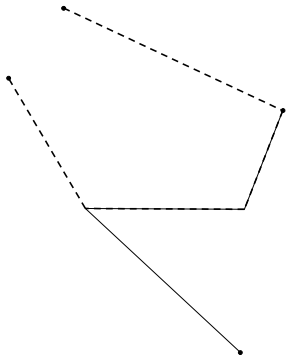
	Interior (B)	Borde (B)	Exterior (B)
Interior (A)	0	-1	1
Borde (A)	-1	-1	0
Exterior (A)	1	0	2

- Tocar: Ejemplo DE-9IM para el caso donde A es un polígono que toca a los polígonos B y C. La relación A-C difiere de la relación A-B en la dimensión de la intersección de borde a borde.



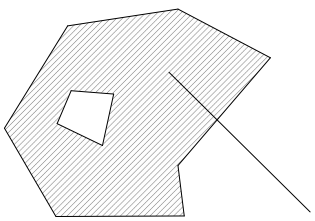
	Interior (B)	Borde (B)	Exterior (B)
Interior (A)	-1	-1	2
Borde (A)	-1	1/0	1
Exterior (A)	2	1	2

- Solape: Ejemplo DE-9IM para el caso donde A es una linea que solapa la linea B. El solape no es conmutativo. La linea A solapa la linea B, sin embargo, la línea B no solapa la línea A. En el DE-9IM se observa esta diferencia en los elementos que comparan el borde de una geometría con el interior de otra.



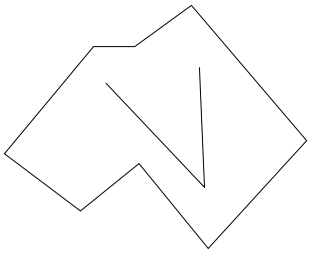
	Interior (B)	Borde (B)	Exterior (B)
Interior (A)	1	-1/0	1
Borde (A)	0/-1	-1	0
Exterior (A)	1	0	2

- Cruzar: Ejemplo para el caso donde A es un polígono y B es una línea que cruza A



	Interior (B)	Borde (B)	Exterior (B)
Interior (A)	1	0	2
Borde (A)	0	-1	1
Exterior (A)	1	0	2

- Dentro: Ejemplo DE-9IM para el caso donde A es una línea que cae dentro del polígono B.



	Interior (B)	Borde (B)	Exterior (B)
Interior (A)	1	-1	-1
Borde (A)	0	-1	-1
Exterior (A)	2	1	2

El patrón de matrices representa el conjunto DE-9IM de valores aceptables para un predicado topológico de dos geometrías. En la siguiente tabla, se puede observar los valores que puede tomar la DE-9IM para los ocho predicados topológicos.

Predicado Topológico	*Patrón que adquiere su matriz*
A.Igualdad (B)	$\begin{bmatrix} T & * & F \\ * & * & F \\ F & F & * \end{bmatrix}$
A.Desigualdad (B)	$\begin{bmatrix} F & F & * \\ F & F & * \\ * & * & * \end{bmatrix}$
A.Intersección (B)	$\begin{bmatrix} T & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \boldsymbol{o} \begin{bmatrix} * & T & * \\ * & * & * \\ * & * & * \end{bmatrix} \boldsymbol{o} \begin{bmatrix} * & * & * \\ T & * & * \\ * & * & * \end{bmatrix} \boldsymbol{o} \begin{bmatrix} * & * & * \\ * & T & * \\ * & * & * \end{bmatrix}$
A.Tocar (B)	$\begin{bmatrix} F & T & * \\ * & * & * \\ * & * & * \end{bmatrix} \boldsymbol{o} \begin{bmatrix} F & * & * \\ * & T & * \\ * & * & * \end{bmatrix} \boldsymbol{o} \begin{bmatrix} F & * & * \\ T & * & * \\ * & * & * \end{bmatrix}$
A.Cruzar (B)	$\begin{bmatrix} T & * & T \\ * & * & * \\ * & * & * \end{bmatrix} \boldsymbol{o} \begin{bmatrix} 0 & * & * \\ * & * & * \\ * & * & * \end{bmatrix}$
A.Solape (B)	$\begin{bmatrix} T & * & T \\ * & * & * \\ T & * & * \end{bmatrix} \boldsymbol{o} \begin{bmatrix} 1 & * & T \\ * & * & * \\ T & * & * \end{bmatrix}$
A.Dentro (B)	$\begin{bmatrix} T & * & F \\ * & * & F \\ * & * & * \end{bmatrix}$
A.Contiene (B)	$\begin{bmatrix} T & * & * \\ * & * & * \\ F & F & * \end{bmatrix}$

*Table 3: Predicados Topológicos y su correspondiente patrón matricial para el DE-9IM. Los valores del patrón matricial se corresponden con la dimensión del resultado de la operación intersección. El dimensión 0 se corresponde con un punto, dimensión 1 se corresponde con una linea, dimensión 2 se corresponde con un polígono, el valor F se corresponde con el con el conjunto vacío, el valor T se corresponde con cualquiera de las dimensiones 0, 1 o 2. El valor * puede tener cualquier tipo de dimensión, incluyendo el conjunto vacío.*

2.4. Soluciones de Software

En el mercado actual existen diferentes soluciones para la elaboración de un Levantamiento Topográfico. Estas utilizan formatos de archivo privados. En ocasiones, la Triangulación de Delaunay Constreñida se incorpora como una herramienta individual, y en otras, ésta queda imbuida en un complemento. Aquí se presentan algunas opciones disponibles.

2.4.1. TcpMDT

Desarrollado por Aplitot S.L (https://www.aplitop.com/), está disponible para diferentes programas de Diseño Asistido por Ordenador (DAO) como AutoCAD, BricsCAD o ZWCAD tienen disponibles el plugin enfocado en Topografía y proyectos de Ingeniería Civil. El plugin puede ser adquirido en dos versiones disponibles: Standard Version, empezando desde el uso de datos brutos permite generar la triangulación, alineamiento horizontal definido por una polilínea, vaciando y rellenando volúmenes, uso de comandos para visualización del terreno, uso de texturas y ortofotos y generación de video, entre otros; Professional Version, ofrece características adicionales como herramientas para el diseño horizontal y vertical de alineamientos, cálculo optimo de altura; y el módulo de Topografía, este es compatible con los otros dos módulos, permite el cálculo y la administración de las medidas obtenidas de una estación total, también funciona con sistemas de coordenadas locales, etc (*TCP - MDT Digital Terrain Model - V7.5*, n.d.).

2.4.2. ArcGis software

ArcGis es un Sistema de Información Geográfica (SIG) no demasiado enfocado a Ingeniería Civil como la solución anterior. Sin embargo, ArcGis tiene una extensión llamada ArcGIS 3D Analyst que ofrece la función de Triangulación de Delaunay Constreñida (CDT), permitiendo el uso de lineas de roturas e islas (*Fundamentals of TIN triangulation in ArcGIS*, n.d.).

2.4.3. QGIS

QGIS es un SIG de Software Libre y de Código Abierto (FOSS). Es bastante completo pero no tiene una herramienta específica para levantamientos de terreno, sino que tiene algún que otro plugin que puede procesar los datos brutos de una estación total. Dispone de la triangulación de Delaunay en 2D, es posible ser ejecutado como un proceso Batch, y no dispone de la posibilidad de utilizar un polígono de contorno donde se generará la triángulación. Esto último no tiene que ser un problema si se procesan correctamente los datos de entrada con anterioridad. Pero por otro lado, no dispone de la posibilidad de determinar la triangulación costreñida de Delaunay. (*QGIS API Documentation*, 2019).

3. METODOLOGÍA

En esta sección, se documenta el conjunto de sub-algoritmos necesarios para la elaboración del algoritmo de CDT. Además, se especifican las condiciones que debe cumplir el algoritmo y sus datos de entrada para que el resultado de su ejecución sea exitosa.

3.1. Especificaciones

A continuación se expone que condiciones deben tener los datos de entrada teniendo en cuenta que el algoritmo siempre será ejecutado en superficies con dimensión 2.5. El método de inserción incremental para generar la triangulación será igual al presentado por Guibas y Stolfi (1985). Sin embargo, el proceso de Validación de Triángulos diferirá un poco. Además, se construirá una interfaz de usuario gráfica con la que el usuario podrá interactuar con los diferentes parámetros del algoritmo.

3.1.1. Especificación de entrada de datos

- El algoritmo debe poder importar datos desde una capa de puntos que representa los puntos de relleno, importar datos de una capa de lineas que representa las líneas de rotura e importar datos desde una capa de polígonos que representa polígonos isla.

- Dos puntos, ya sea de los puntos de relleno o cualquier vértice de linea de rotura o polígonos isla no podrán tener las mismas coordenadas planimétricas. Por ejemplo: cuevas, superficies debajo de un puente, etc.

- La salida del plugin son dos archivos del tipo Shape, representando los segmentos de la triangulación, y representando los polígonos de la triangulación. La coordenada Z debe ser incluida en la geometría de cada entidad. Se podrá seleccionar el directorio de salida de los resultados.

- No habrá ningún punto dentro de los polígonos isla. No podrá existir polígonos isla dentro de polígonos isla. Además, todos los polígonos deberán ser anillos únicos.

- Los datos de entrada pueden ser referenciados en cualquier Sistema de Coordenadas de Referencia (CRS) incluida en la librería proj4.
- Las Líneas de Rotura no podrán intersectar polígonos Isla.
- Los polígonos Isla no podrán contener Puntos de Relleno.

3.1.2. Especificación de Interfaz Gráfica

Será necesario incluir:

- Caja de herramientas para establecer la ruta de los puntos de relleno
- Caja de herramientas para establecer la ruta de las lineas de rotura
- Caja de herramientas para establecer la ruta de los polígonos isla
- Caja de herramientas para establecer el nivel de adyacencia
- Caja de herramientas para establecer la ruta de los archivos de salida

3.2. Diseño

En esta sección se presentan las diferentes etapas del algoritmo de triangulación que pueden ver de forma esquemática en la Ilustración 24.

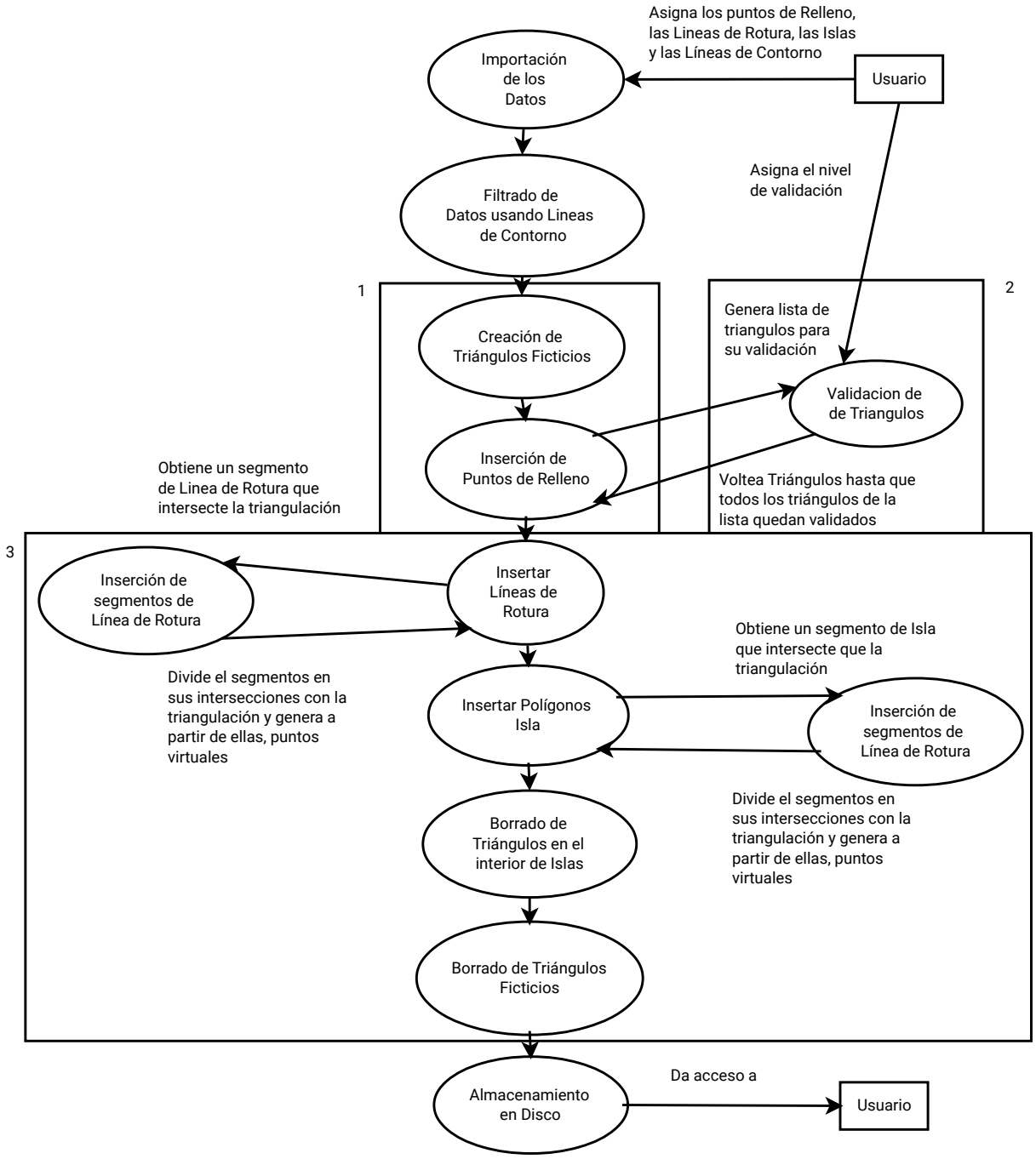


Ilustración 19: Diagrama de Flujo de Datos del algoritmo CDT

3.2.1. Inserción Incremental

El algoritmo elegido que se lleva a cabo para generar la CDT, es del tipo incremental, este tipo de algoritmos, aunque disponen de un tiempo de ejecución del orden $O(n^2)$ presentan algunas ventajas en su implementación.

Primero, al llevar a cabo la inserción de los puntos de manera secuencial y aleatoria, no es necesario ordenar la nube de puntos por coordenada X o Y o ambos, dentro de la cuadrícula, etc. Este ahorro en el tiempo de cálculo no es tenido en cuenta en los algoritmos del apartado 2.

Segundo, este algoritmo permite la edición de la triangulación de forma local, esto quiere decir que en una versión posterior, este algoritmo facilita la implementación de la edición de la triangulación.

Tercero, no es necesario disponer de funciones que permitan la unión de triangulaciones adyacentes, como por ejemplo en los algoritmos de divide y vencerás. En este caso, el proceso que necesita más tiempo de ejecución consiste en buscar el triángulo cuyo punto a insertar queda en su interior.

Cuarto, no es necesario dividir la triangulación en celdas, como se hace en los triángulos iterativos. Esto permite la triangulación generada quede segmentada por la naturaleza de la distribución de los vértices de la nube de puntos, y no quede teselada como ocurre en los algoritmos iterativos.

3.2.2. Validación Incremental de los Triángulos

A diferencia de Guibas y Stolfi (1985), en este proyecto, la metodología para la Validación de Triángulos se lleva a cabo justo al terminar la inserción de un vértice.

Otra posibilidad considerada para validar la triangulación consiste en insertar en primer lugar todos los triángulos mediante el Método Incremental, y generar una triangulación inicial. A continuación, validar los triángulos uno por uno hasta completar toda la triangulación. En este tipo de solución heurísticamente es necesario validar menos triángulos que en el proceso de validación incremental. Sin embargo, su principal inconveniente es que carece de dinamismo a la hora de permitir una edición local dentro de la triangulación.

3.2.2.1. Obtención de la lista de Chequeo

Con el objetivo de trabajar de forma local en este proceso, es necesario

obtener un conjunto de triángulos afectados por la inserción de un vértice de la nube de puntos. Este vértice insertado puede generar tres o cuatro triángulos en la triangulación. A continuación se genera una lista de triángulos de chequeo. Esta lista está compuesta por los triángulos recién insertados y sus triángulos adyacentes. Este nivel de adyacencia establecido por el usuario (ver Ilustracioń 20).

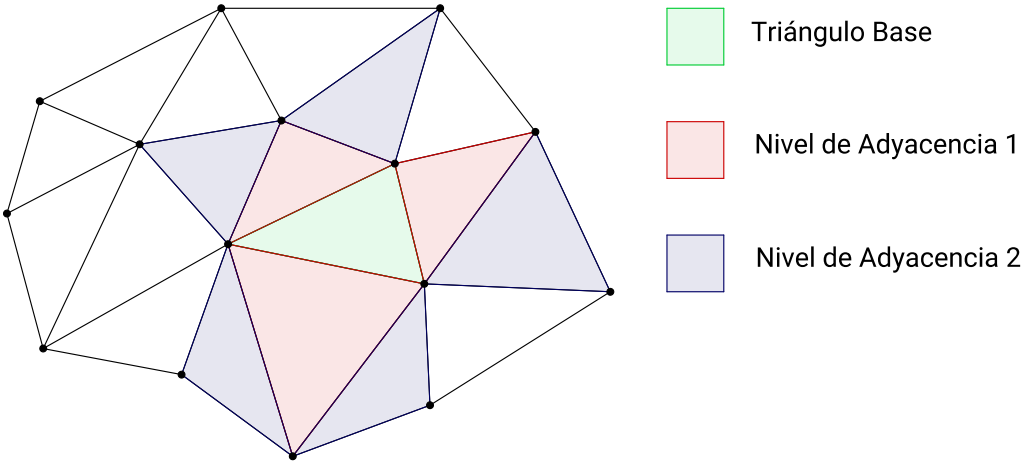


Ilustración 20: Representación de los niveles adyacentes en el proceso de obtención de la Lista de Comprobación. El nivel de adyacencia es elegido por el usuario, este caso, ha sido representado hasta el nivel de adyacencia 2

3.2.2.2. Proceso de Validación de la Lista de Chequeo

El proceso de validación consiste en validar cada triángulo de la lista. Para ello, se obtiene el primer elemento y se comprueba que cumple con el criterio del CCV. Si es así, el triángulo no es insertado de nuevo en la lista de chequeo, en caso contrario, el triángulo es insertado en la cola de la lista. El proceso termina cuando la lista es vaciada completamente. De esta forma, la triangulación queda validada de forma cíclica, chequeando los triángulos implicados en cada iteración.

Debido a que existen nubes de puntos que no disponen de solución en la triangulación de Delaunay, a priori este tipo de subproceso es más propenso a no validar la triangulación completamente de forma homogénea como se hace en el

caso de Validación Incremental. Esto es debido a que el tamaño de la lista de chequeo es inmenso en comparación a una validación local. Lo cual, es muy probable que la zona de los triángulos que han sido volteados sea modificada de nuevo antes de que el bucle permita una nueva edición de la zona afectada, donde poder dar por finalizada su validación.

3.2.2.3. Alternativa para la obtención de la lista de chequeo

Por otro lado, aquí se presenta otra alternativa a la validación de la nube de puntos generando una lista de comprobación utilizando un nivel de adyacencia.

La lista de chequeo se genera a partir de los triángulos introducidos en proceso de Inserción Incremental. A continuación, el primer elemento de la lista de chequeo es seleccionado. Sobre este triángulo, se obtiene su CC y se determina su rectángulo encuadrante. Los triángulos que crucen, o estén dentro del rectángulo encuadrante son añadidos a la lista de comprobación. Por último se ejecuta la validación de la lista de comprobación de forma análoga al método explicado anteriormente (ver Ilustración 21).

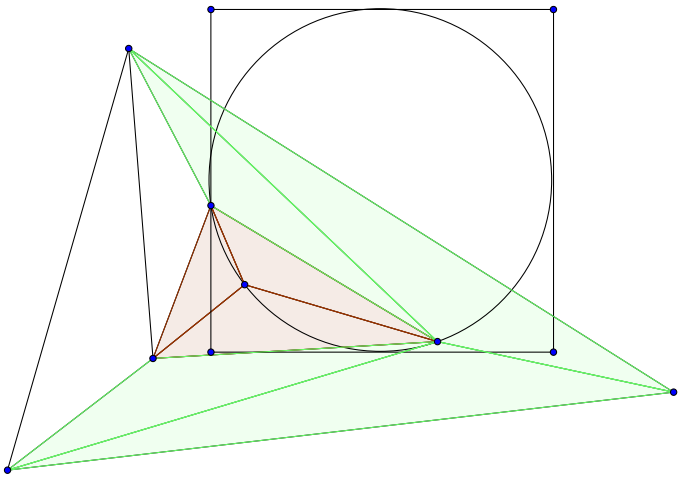


Ilustración 21: Triángulos por defecto en la lista de validación (color rojizo), y triángulos que intersectan o cruzan al rectangulo encuadrante (color verde). Estos últimos son añadidos a la lista de validación.

La principal ventaja de esta alternativa es que no es necesario pedir al usuario

el nivel de adyacencia para ejecutar la validación de los triángulos, estos son determinados de forma automática. Esto permite que los triángulos influenciados por el CC de un triángulo volteado sean añadidos a lista de comprobación para ser evaluados de nuevo. Así se optimiza el tiempo de ejecución, realizando la comprobación solo en los triángulos afectados por los círculos de los triángulos volteados.

3.2.3. Inserción de Líneas de Rotura

En este apartado se estudia la determinación del ámbito de la triangulación que va a ser modificada y varias metodologías para su incorporación a la triangulación.

3.2.3.1. Determinación del ámbito de actuación

Para determinar qué triángulos son afectados por la inserción de un segmento de línea de rotura se parte de los vértices del segmento. A partir de los vértices se determinan los triángulos que los contienen. De forma cíclica, se estudia si los triángulos adyacentes intersectan al segmento de línea de rotura hasta que uno de los triángulos, que contienen un vértice del segmento, es encontrado. En la Ilustración 22 se puede observar cómo evoluciona el bucle hasta que todos los puntos son encontrados:

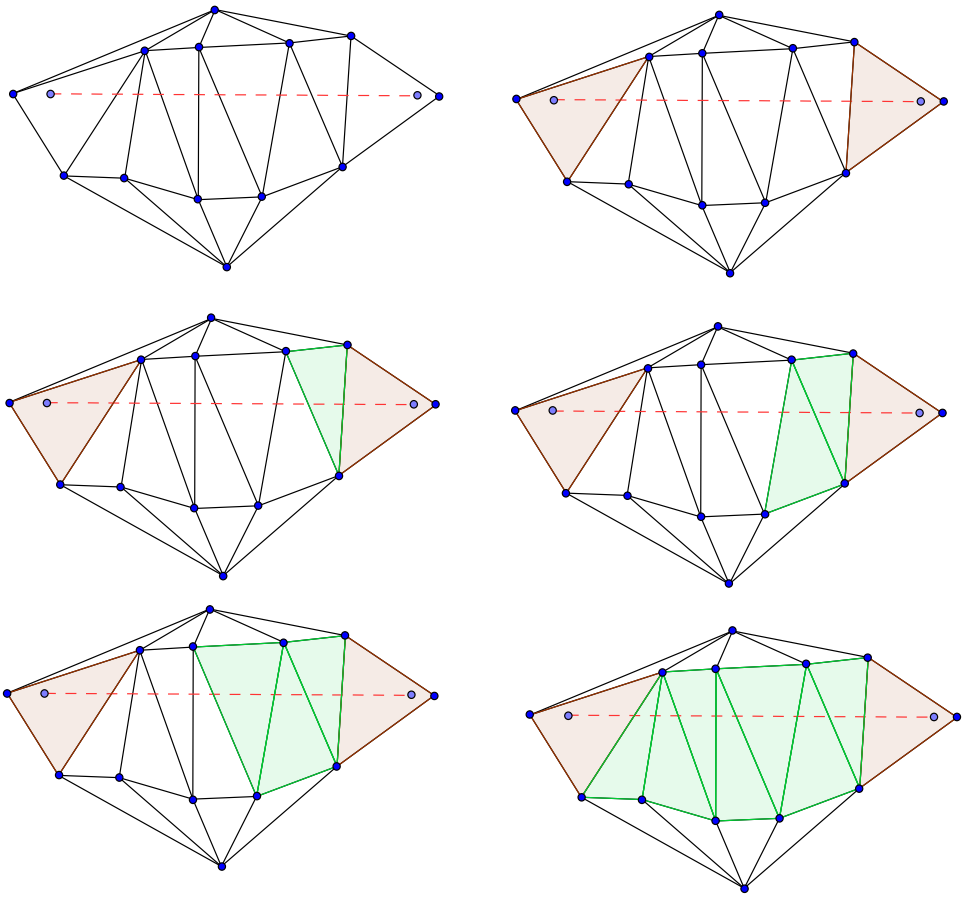


Ilustración 22: Proceso para la determinación de los triángulos implicados en la inserción de un segmento de línea de rotura. Los triángulos de color rojo se corresponden con los triángulos inicial y final (triángulos que contienen los vértices del segmento). Los triángulos de color verde van siendo determinados estudiando la intersección del triángulo con el segmento de linea de rotura. Al encontrar el triángulo final, se detiene el bucle.

Una de las ventajas que presenta esta metodología es que utiliza la información topológica disponible. Esto evita tener que buscar en toda la triangulación cada triángulo que intersecte el segmento de línea de rotura. Esta metodología es totalmente funcional tanto para el método generando puntos ficticios, como para el método de modificación de triángulos adyacentes.

Por otro lado, no se contempla que los vértices de la Línea de Rotura coincidan con algún vértice de los puntos de relleno, ni que esta Línea de Rotura pueda ser modificada en un futuro.

3.2.3.2. Metodología generando puntos ficticios

A diferencia de insertar un punto en una triangulación, en la que solo consiste

en encontrar un triángulo, eliminarlo y generar tres en su lugar, con la sencillez para llevar a cabo este proceso. Insertar una linea de rotura se vuelve un problema topológicamente más complejo y donde existen varias metodologías para solventarlo.

Esta consiste en la inserción de puntos ficticios en el lugar de intersección entre el segmento de línea de rotura y los triángulos que la intersectan. Estos puntos ficticios son añadidos a la triangulación y son insertados como si de un punto de relleno se tratara. Esto generará cuatro triángulos por cada intersección.

A los segmentos de la triangulación que pertenecen a una línea de rotura se les añade una bandera indicando el número de segmento y el nombre de la línea de rotura a la que pertenece. En el siguiente gráfico se puede observar el proceso de inserción.

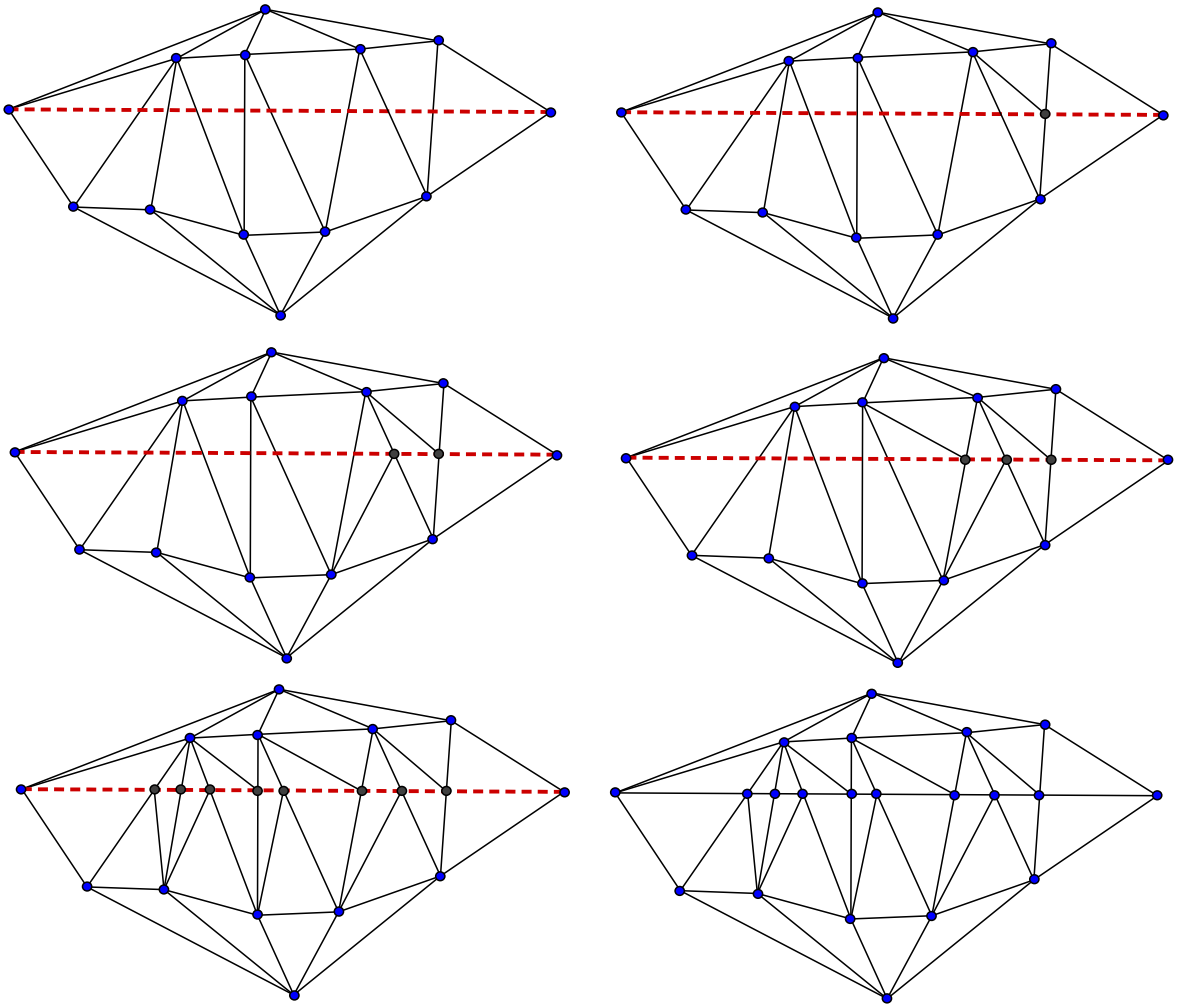


Ilustración23: Proceso de creación de puntos ficticios para la inserción de un segmento de línea de rotura, (color rojo). Aquí los vértices del segmento ya han sido insertados en la triangulación, y no se han representado los triángulos generados que no intersectan el segmento de línea de rotura.

El principal inconveniente en esta metodología es que se incrementa el número de triángulos que se generan. Además, se pueden generar algunos polígonos astilla.

3.2.3.3. Metodología modificando los triángulos adyacentes, sin añadir puntos ficticios

Esta metodología consiste en voltear los triángulos adyacentes al segmento de intersección hasta que no exista ningún triángulo que intersecte al segmento. Esta divide el problema en la zona superior e inferior del segmento de intersección, que son resueltos por separado pero de forma análoga. Partiendo de los triángulos ámbito, por ejemplo en la zona superior, se determina los segmentos de triángulo

que pertenecen al borde del ámbito, estos segmentos se denominan segmentos borde.

Los triángulos ámbito son eliminados, pero sus segmentos son almacenados. El proceso de resolución es un algoritmo del tipo divide y vencerás. Consiste en ir determinando los puntos de inflexión del borde ámbito para separar este en dos. El punto de inflexión pertenece al primer triángulo que va a ser creado, que será compuesto además por los vértices del segmento de línea de rotura. A continuación, se establece como base el segmento generado anteriormente, y se utilizará el primer subconjunto de puntos para determinar un nuevo punto de inflexión. Este proceso se repite hasta que el ámbito está compuesto por un solo punto.

Para determinar los puntos de inflexión, se empieza a recorrer el borde del ámbito utilizando tres puntos. Los dos primeros forman un segmento, y el tercero se estudia si cae a la derecha o a la izquierda y se almacena en una variable. Si el siguiente punto estudiado cae en el lado opuesto al que hay almacenado en memoria, automáticamente se convierte un punto de inflexión.

Si no hay puntos de inflexión en el borde del ámbito, se utiliza el primer punto del borde del ámbito en cuestión (ver Ilustración 28).

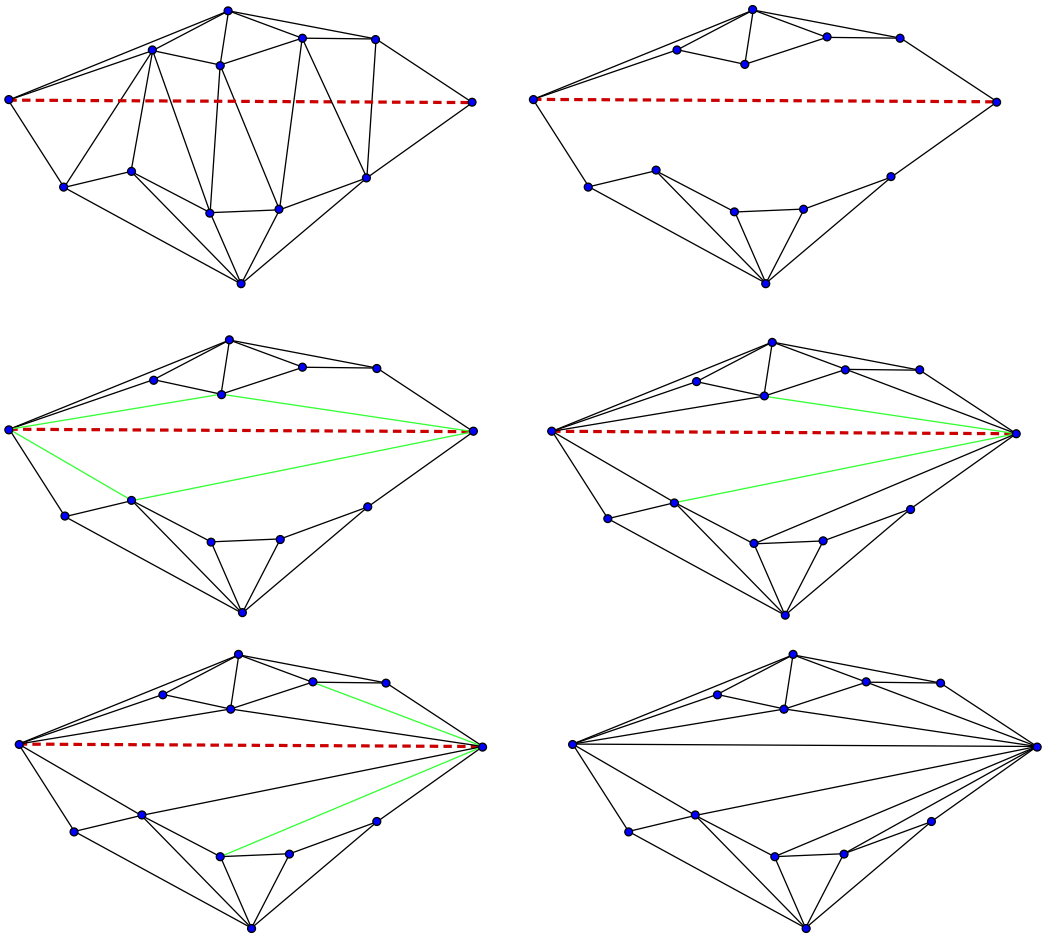


Ilustración 24: Metodología para la inserción de segmento de línea de rotura sin utilizar puntos ficticios

El principal inconveniente que puede presentar esta solución consiste en la creación de triángulos en forma de abanico como se observa en la Ilustración 28.

4. IMPLEMENTACIÓN

En este capítulo se presenta la elaboración del algoritmo, justificando en cada caso que metodologías se han utilizado y posibles problemáticas que hayan podido surgir durante el proceso.

Dada una nube de puntos, la ejecución del algoritmo es organizada principalmente en cuatro fases: importación de los datos, inserción incremental de triángulos, validación de los triángulos insertados, inserción de líneas de rotura y polígonos isla.

4.1. Datos de Entrada

La primera fase de la implementación inserta todos los puntos que la nube de puntos posee, estos puntos son insertados uno a uno.

El formato de los datos de entrada puede ser cualquiera que sea compatible con GDAL-OGR. Los datos de entrada se corresponde a las siguientes capas de información:

- Nube de puntos. Esta se corresponde con los puntos de relleno en un levantamiento. Por las condiciones impuestas en el apartado 3.1.1; no pueden haber dos vértices que pertenezcan a Líneas de Rotura o Islas; no puede haber puntos de relleno en el interior de Polígonos Isla; no puede haber más de un punto con la misma coordenada X e Y.

- Líneas de rotura. Esta capa tiene que ser una capa de geometrías tipo LineString. Por definición; sus geometrías no pueden ser intersectadas entre si; sus geometrías no pueden insertar otro Polígono Isla.

- Polígonos Isla. Esta capa es de tipo Polígono. Por definición; las geometrías Polígono no pueden tener polígonos interiores o exteriores; los polígonos no pueden ser intersectados entre sí; los polígonos no pueden intersectar otras geometrías.

- Líneas de contorno. Delimita la región del espacio donde el algoritmo va a ser ejecutado. Es de tipo polígono, y envuelve a una parte del conjunto de datos. Esto permite seleccionar la cantidad de datos que van a ser operados. Todas las geometrías que intersecten o no estén dentro de este polígono serán ignorados.

- Capa de salida. Se especifica donde van a ser guardados los resultados del procesamiento de los datos en el disco duro. El programa genera dos salidas, una para triángulos y otra para segmentos.

4.2. Recursos Usados

Python ha sido utilizado preferentemente porque existe un entorno muy bien preparado para el desarrollo de complementos. Este entorno permite de forma muy dinámica dotar a QGIS de diferentes funcionalidades. En contadas situaciones, estos complementos pueden llegar a ser parte del paquete de algoritmos oficiales que se ofrecen en distintos módulos del proyecto.

4.2.1. Geospatial Data Abstraction Library (GDAL)

GDAL es una librería para transformación de formatos de Datos Geográficos Ráster y Vectoriales definidos bajo licencia X/MIT. Como librería, presenta un único modelo de datos abstractos ráster y un único modelo de datos abstractos vectorial para todos los formatos soportados. Se divide en diferentes proyectos, uno de ellos es GDAL-OGR cuyas primitivas geométricas son utilizadas vectorialmente. Ésta a su vez está basada en diferentes componentes, entre ellos la librería GEOS, que es una adaptación de la librería Java Topology Suite (JTS) y una implementación de Simple Feature Access (https://www.opengeospatial.org/standards/sfa)

Con el fin de facilitar el proceso de mantenimiento en este plugin, la librería GDAL ha sido utilizada. Esto es así porque con el proceso de actualización que ha sufrido QGIS últimamente, concretamente con la actualización de la versión 2 a la versión 3, su API ha sido modificada en gran medida. Es por ello que el uso de la librería GDAL en la implementación no es afectado en este proceso de actualización.

GDAL ha sido implementada en el lenguaje de programación C++, sin embargo esta ha sido portada a diferentes lenguajes de programación, entre ellos Python. Uno de los puntos importantes a tener en cuenta a la hora de utilizar un estilo de programación, es considerar el grado de compatibilidad que tiene la versión portada de GDAL (https://trac.osgeo.org/gdal/wiki/PythonGotchas). Esta presenta algunos

inconvenientes como por ejemplo:

- Con el propósito de permitir compatibilidad con versiones anteriores, el manejo de excepciones en la versión de GDAL portada esta deshabilitado por defecto.

- En un caso en concreto, como son los objetos Geometry y Features. Estos objetos disponen disponen de una relación fuerte. Esto quiere decir que si dispones de un objeto Feature con su objeto Geometry, y eliminas el objeto Feature, también será borrado el objeto Geometry, el programa arrojará un error fatal. Para solventar este problema, hay dos opciones: o crear una geometría nueva con las mismas características, y por consiguiente borrar nuestra Feature junto con su Geometry; o utilizar como paso de parámetros nuestra Feature, en vez de utilizar solo nuestra Geometry. En cualquier caso, copiar una Geometry solo creará una referencia, y esta producirá un error si la Feature queda eliminada. Si no se tiene en cuenta estos detalles, el programa arrojará un error de segmentación.

- Cuando los objetos Layer son creados, es importante que cuando vayan a ser eliminados, se utilice la función Flush, ya que no liberará los cambios del dataset del que proviene y no se grabarán en disco.

4.2.2. Estructura de Datos

Tres objetos diferentes han sido desarrollados, triangulo, segmento y triangulación (ver Ilustración 25). El objeto segmento guarda la información relacionada con el comienzo y final de los segmentos, que triángulos están situados a la izquierda del segmento y que isla y/o linea de rotura pertenece el segmento. El objeto triángulo guarda la información sobre los identificadores de los triángulos. El objeto triangulación almacena la información sobre identificadores de triángulo, información sobre las lineas de rotura, polígonos isla.

Utilizando esta estructura, el problema sobre referencia cruzadas a la hora de importar diferentes variables globales en Python es evitado, esto muestra cuanta

información relacionada a triángulos son almacenadas dentro de los segmentos y la información almacenada en los segmentos sobre los triángulos. Hay un diccionario de objetos triángulos, y el resto de información puede ser obtenido a través de su objeto triangulo.

Los objetos triángulos disponen de un identificador, que actúa como clave para poder acceder a ellos. El esquema de asignación de nombres para estos triángulos consiste en asignar un número como cadena de texto a los nuevos triángulos. Los triángulos se nombran de forma secuencial por lo que el último número asignado es almacenado en el atributo _max_id_triangles. Cuando un triangulo es eliminado debido a una nueva inserción, su identificador queda libre en el diccionario, pero no es remplazado por otra clave.

En el borde de la triangulación, la información que contiene los segmentos sobre los triángulos que quedan a su izquierda tienen el valor cadena de texto vacío. Acceder al diccionario de triángulos arroja una excepción del tipo error de clave que debe ser gestionada.

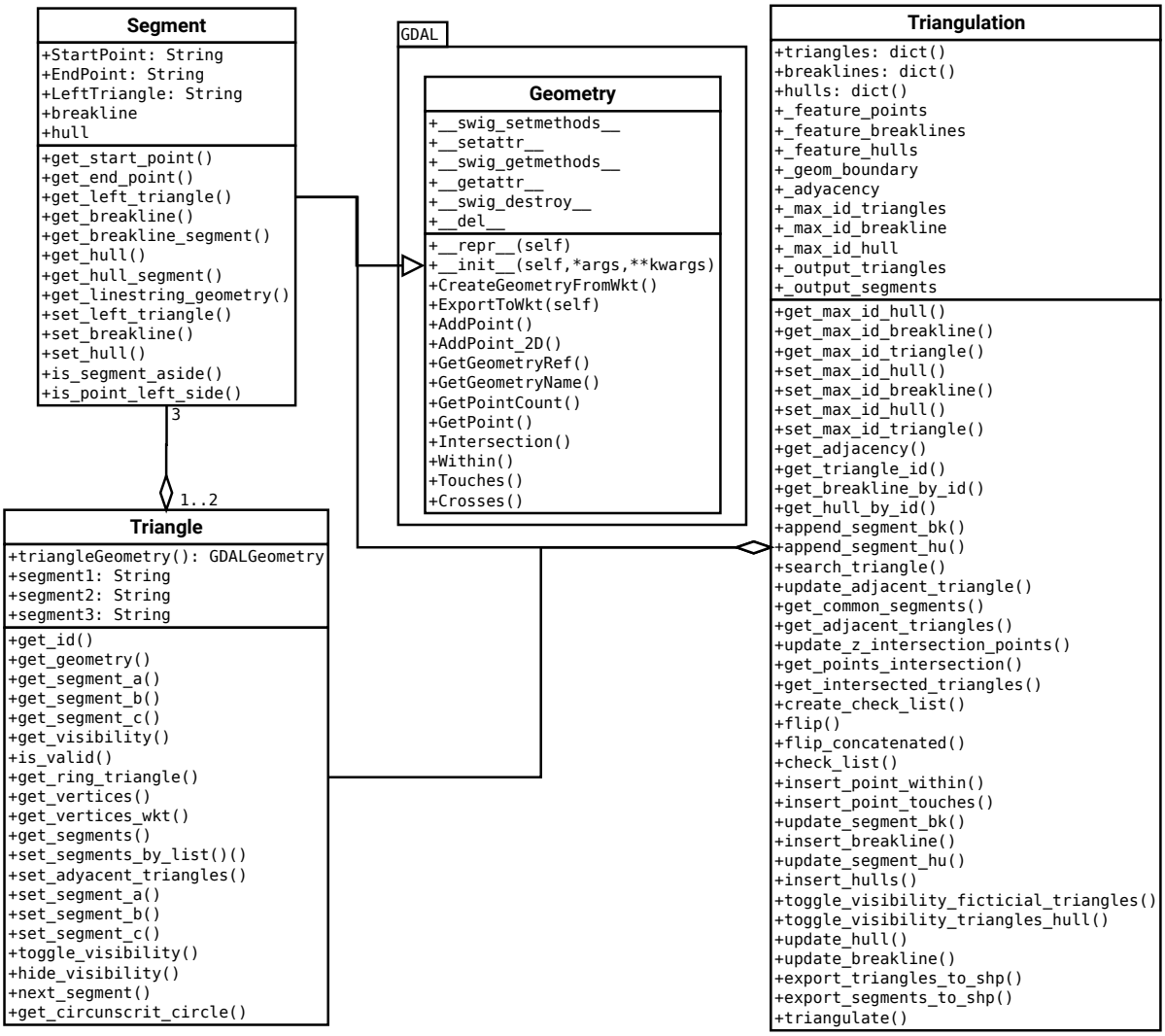


Ilustración 25: Diagrama UML del conjunto de clases del Algoritmo Principal

4.3. Inserción Incremental de Triángulos

4.3.1. Generación de triángulos ficticios

Dado un conjunto de puntos que cumple la especificación de los datos de entrada, un triángulo que envuelve a toda la triangulación es calculado. Este triangulo inicializa el objeto triangulación.

Para este cálculo se determina el rectángulo encuadrante de la nube de puntos. A los vértices se le han aplicado un desplazamiento del doble de la longitud del lado del rectángulo encuadrante. El Pseudocógido en el Anexo 1.3 se puede observar como se han determinado los vértices del triángulo ficticio, este

procedimiento está incrustado en el método __init__ que inicializa la clase triangulation (ver Ilustración 26). En la Ilustración 27 se puede observar un ejemplo gráfico.

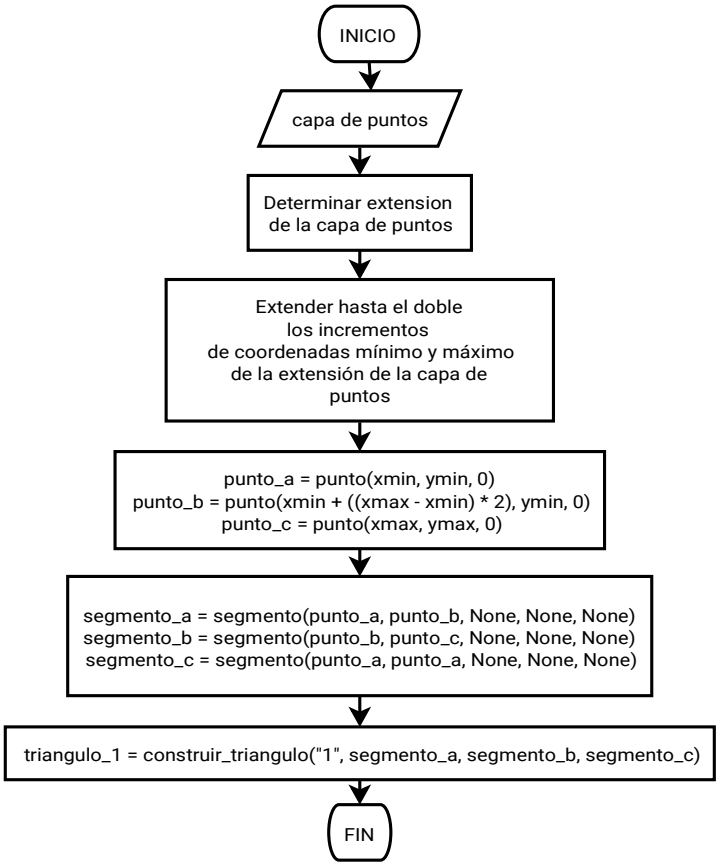


Ilustración 26: Diagrama de flujo del proceso de creación del triángulo ficticio. Este proceso inicializa la triangulación

Cabe mencionar, que los triángulos ficticios no son validados y estos son eliminados al finalizar el algoritmo.

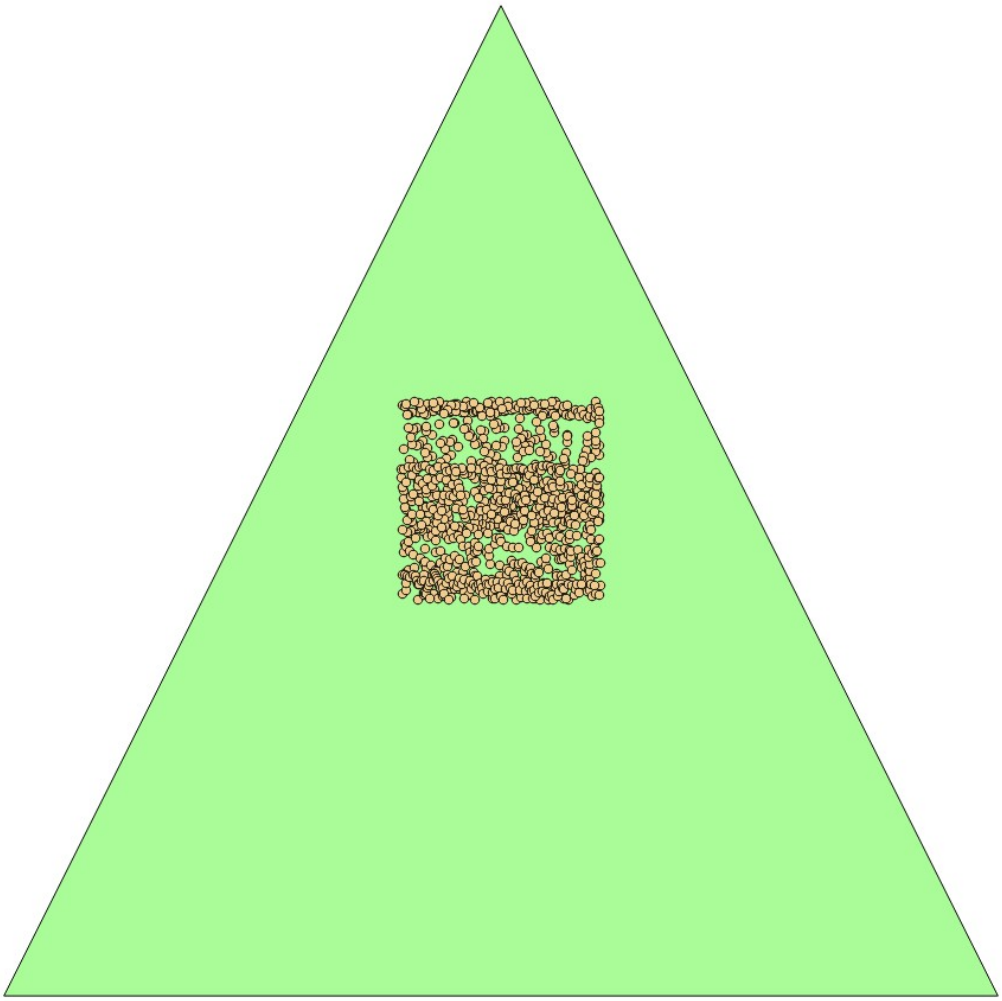


Ilustración 27: Triangulo ficticio que envuelve a la nube de puntos

4.3.2. Búsqueda aleatoria de puntos de relleno

Cogiendo un punto de forma aleatoria, un paso importante en cuanto a tiempo de cálculo es encontrar a qué triángulo el punto pertenece. En esta implementación el triángulo es buscado de manera secuencial, no ha sido utilizado un índice para acelerar el tiempo de búsqueda (ver apartado 5.3), pero las bases y la estructura son establecidas para fácilmente actualizar a esta característica en una futura versión (ver apartado 6.3).

La implementación actual es sencilla y se puede entender en el siguiente Pseudocódigo. Sin embargo esta presenta la utilidad de clasificar si el triangulo que está siendo buscado envuelve al punto, o toca a uno de sus segmentos, en cuyo

caso su triangulo adyacente también será devuelto por esta función (ver apartado del Anexo 1.4).

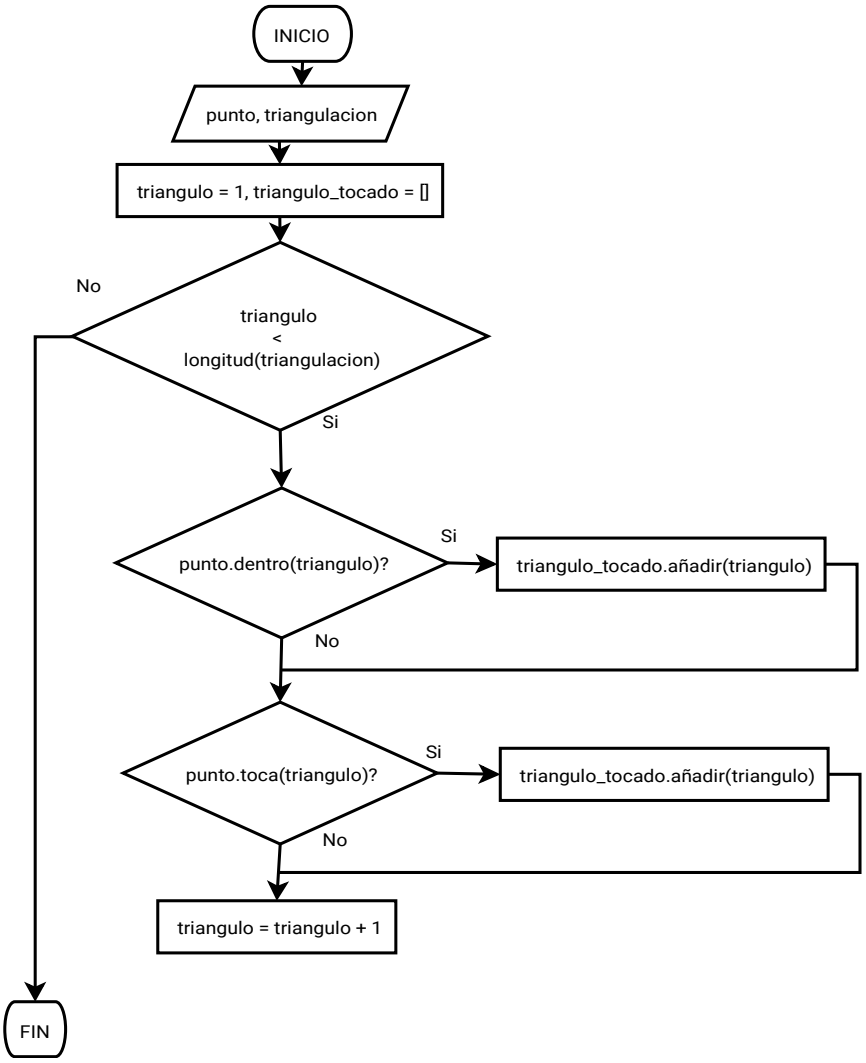


Ilustración 28: Diagrama de Flujo para la búsqueda de en que triángulo cae el punto

4.3.3. Inserción de un punto que cae dentro de un triangulo

Una vez que el triángulo es encontrado, hay dos posibilidades, el punto puede caer en el interior de un punto o caer en un segmento. Debido a la especificación de los puntos de entrada, la inserción de un vértice no puede caer encima de otro

vértice.

En el primer caso, añadir el punto a la triangulación implica crear tres nuevos triángulos y sus tres segmentos correspondientes, tres por cada triangulo. La construcción de los triángulos sigue un orden establecido y este mantiene la orientación de los segmentos en el sentido de las agujas del reloj. Así que, dado dos triángulos adyacentes, el segmento común de un triangulo tiene sentido opuesto a la orientación de su homólogo en el triángulo adyacente.

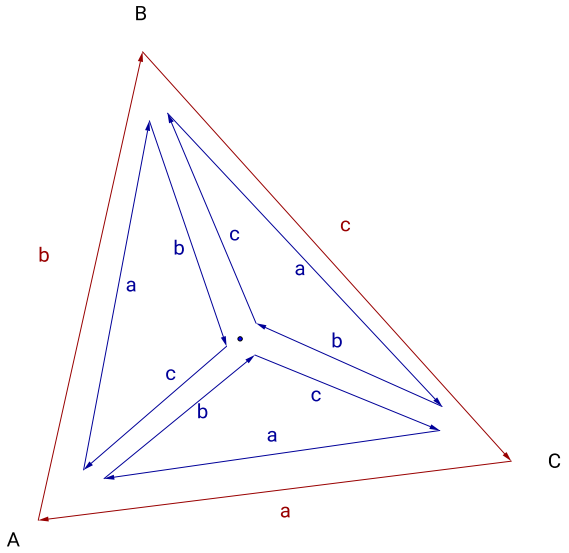


Ilustración 29: Plantilla para la insercion de un punto en la triangulación. Los segmentos de color rojo se corresponde al triangulo que es borrado para dar lugar a tres triángulos (de color azul) que surgen en su lugar

En la ilustración 29 se puede observar como quedan orientados los segmentos cuando tres triángulos son generados cuando un nuevo punto es insertado.

En el Diagrama de Flujo de la Ilustración 30 se corresponde con el método insert_point_within de la clase triangulation (véase ilustración 25). Ver apartado del Anexo 1.5 para ver su Pseudocodigo.

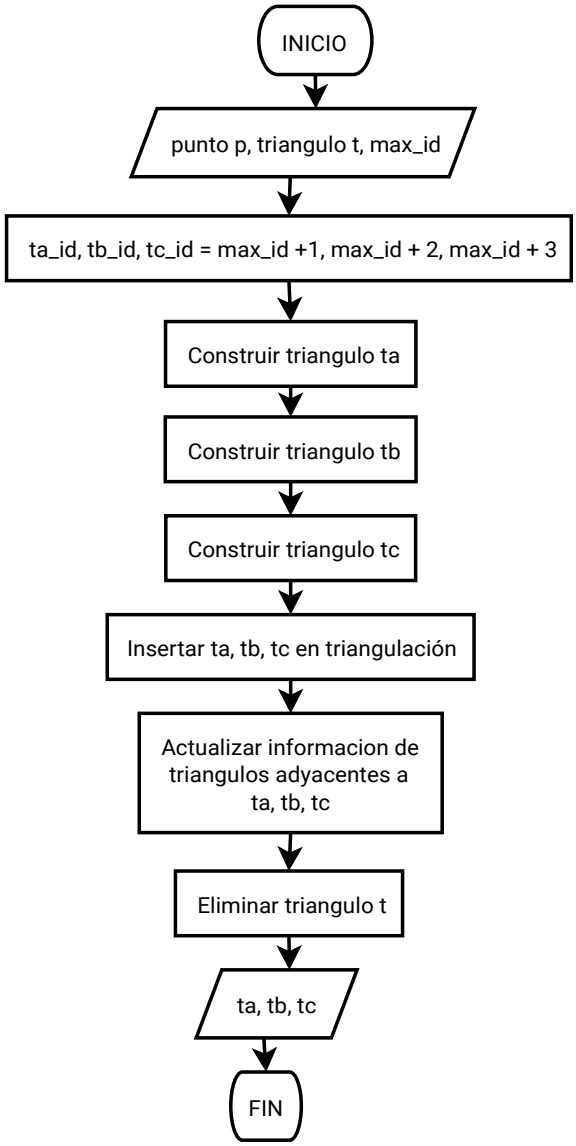


Ilustración 30: Diagrama de flujo del proceso de insercion de un punto en la triangulación cuando el punto cae dentro de un triángulo.

El proceso de inserción de puntos de relleno es anterior al proceso de insertado de lineas de rotura e islas. Ya que no contienen información en este momento, el contenido de los atributos, con respecto a líneas de rotura e islas, de los segmentos es nulo.

4.3.4. Inserción de un punto que cae en un segmento de triángulo

En el segundo caso, añadir cuatro triángulos es necesario. Para poder obtener una orientación de los nuevos segmentos en el sentido de las agujas del reloj, es

necesario especificar a partir de dos triángulos adyacentes; sus segmentos comunes, y los segmentos adyacentes a los segmentos comunes. Para llevar a cabo la orientación de los segmentos es necesario rellenar automáticamente una plantilla donde el segmento común y sus adyacentes son establecidos a partir de la información de los triángulos de partida. Después de ello, cuatro triángulos son generados y los dos triángulos originales son eliminados.

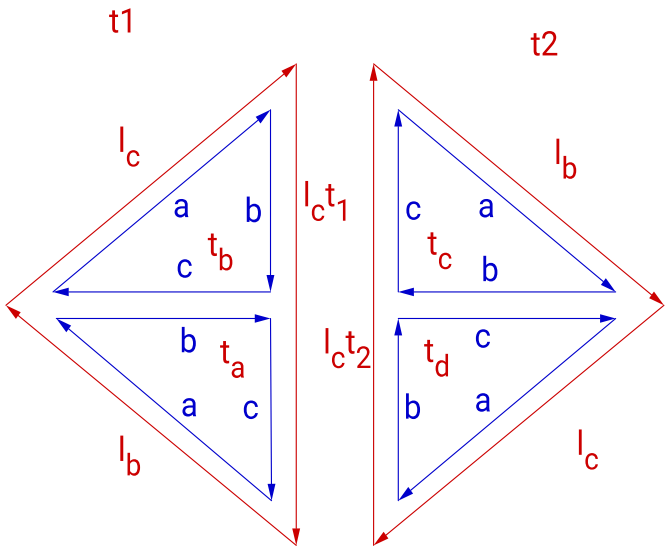


Ilustración 31: Plantilla para la inserción de un punto en la triangulación que cae en un segmento. El segmento se corresponde con el lado común (l_c) de los triángulos t1 y t2 (de color rojo) . Una vez insertados ta, tb, tc, td (de color azul), los triángulos t1 y t2 son eliminados

Para obtener los segmentos comunes de dos triángulos adyacentes, es necesario comparar si cada triangulo adyacente a la izquierda de un segmento de t1, se corresponde con t2. Si es así, el segmento se establece como segmento común. De forma análoga se determina el segmento de t2 que es adyacente a t1. A modo de comprobación, se verifica también que los triángulos de entrada son realemente adyacentes, y que no son el mismo triangulo.

Cómo observación, no importa asignar uno de los triángulos adyacentes como t1 o como t2, en cualquier caso el resultado es el mismo. En la ilustración 32 se puede observar el método utilizado para determinar los segmentos comunes a dos triángulos adyacentes.

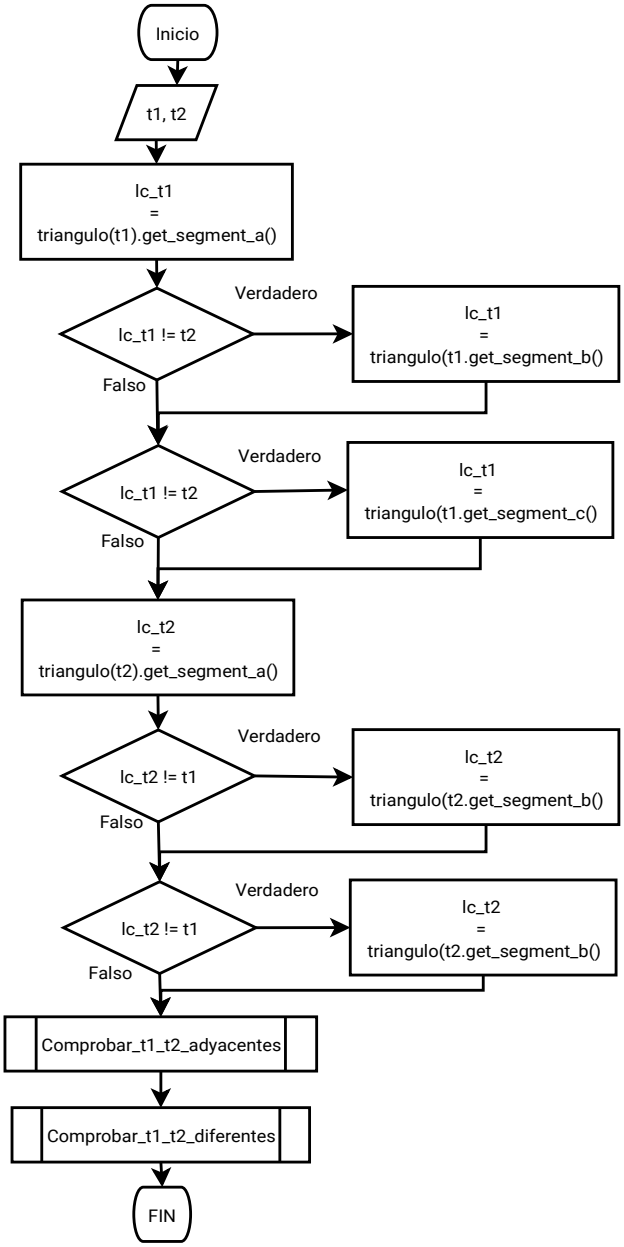


Ilustración 32: Diagrama de flujo del proceso de obtencion de segmentos comunes a partir de dos triángulos, a priori adyacentes.

En el apartado del Anexo 1.6 se puede consultar su Pseudocódigo. Se corresponde con el método insert_point_touches de la clase triangulación (véase ilustración 25):

Cabe destacar la importancia de la consideración de que un punto cae encima de un segmento o no. Por ejemplo en el levantamiento de una carretera, esta es proyectada como una linea recta. Los puntos que la componen en el levantamiento no generan una linea recta, por lo que se forman triángulos cuya relación base y

altura tiende a infinito (ver Ilustración 33).

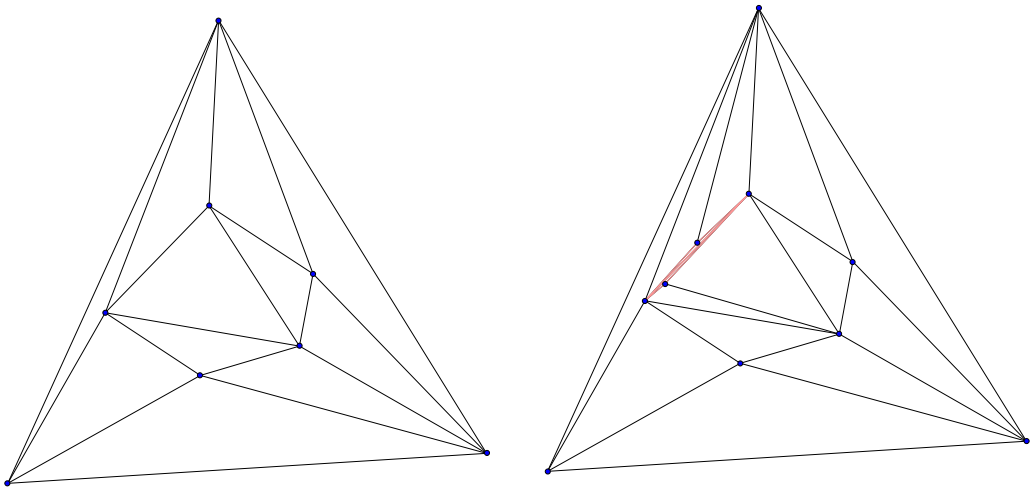


Ilustración 33: Inserción de vértices casi alineados para observar los polígonos astilla generados.

Para mejorar este comportamiento y evitar así la formación de polígonos astilla, sería conveniente utilizar una tolerancia especificada por el usuario. Para esta implementación, se ha prescindido de ella y se ha utilizado la tolerancia que GDAL ofrece por defecto.

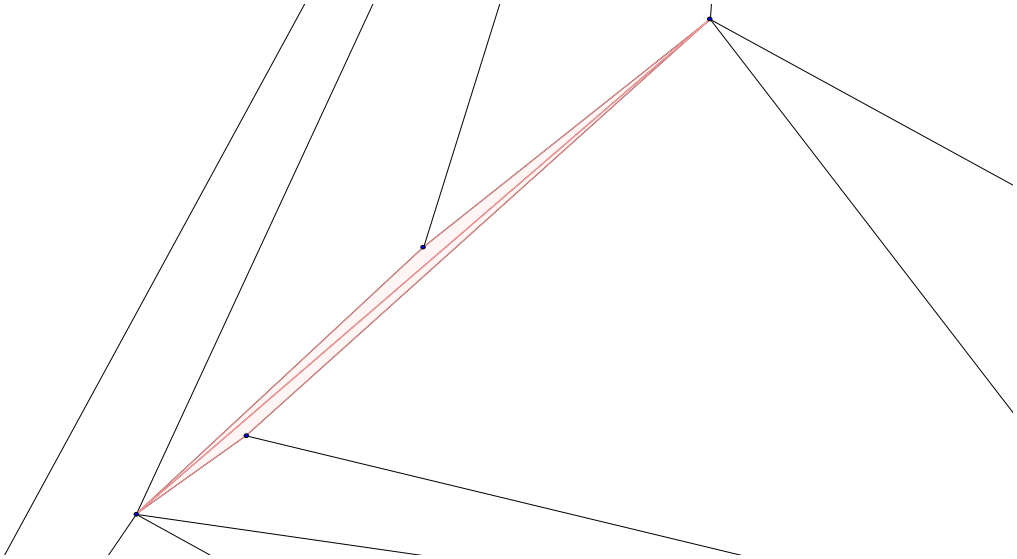


Ilustración 34: Zona de punto alineados. Los polígonos quedan muy astillados.

4.3.5. Inserción de un punto que cae en un vértice de puntos de relleno

Esta situación se encuentra solventada en el momento de importación de los datos de entrada. Esto es así porque no se permite ejecutar el algoritmo cuando existen vértices con la misma coordenada x e y. Véase apartado 3.1.1

4.4. Validación de los Triángulos Insertados

Dentro de las dos soluciones presentadas en el capítulo 3.2.2, se ha optado por escoger la primera solución que consiste en validar una lista de chequeo generada al insertar cada vértice de la nube de puntos. Una de las principales ventajas por la cual se ha optado por esta solución es debido a que este proceso de validación puede ser reutilizado tanto en el proceso de inserción de Líneas de Rotura y Polígonos Isla como en la implementación de una futura versión del programa donde la característica de edición manual del resultado de la Triangulación Constreñida de Delaunay sería añadida, permitiendo la edición local de la triangulación.

El proceso de validación comienza justo después de la inserción de un punto de forma incremental en el método triangulate de la clase triangulation. Es aquí donde se indica que vamos a generar una lista de comprobación y posteriormente la evaluamos.

4.4.1. Creación de Lista de Validación

Para crear la lista de comprobación se utiliza un método de clase para generarlo, los parámetros de entrada son el nivel de adyacencia y los triángulos que han sido generados debido a la inserción de un punto. Para eliminar elementos del conjunto repetidos, se une el conjunto de triángulos con los triángulos adyacentes, acumulando estos en una lista del tipo set (ver Ilustración 35). Así queda una lista de triángulos adyacentes entre sí sin repetir (ver Pseudocógido en el Anexo 1.7).

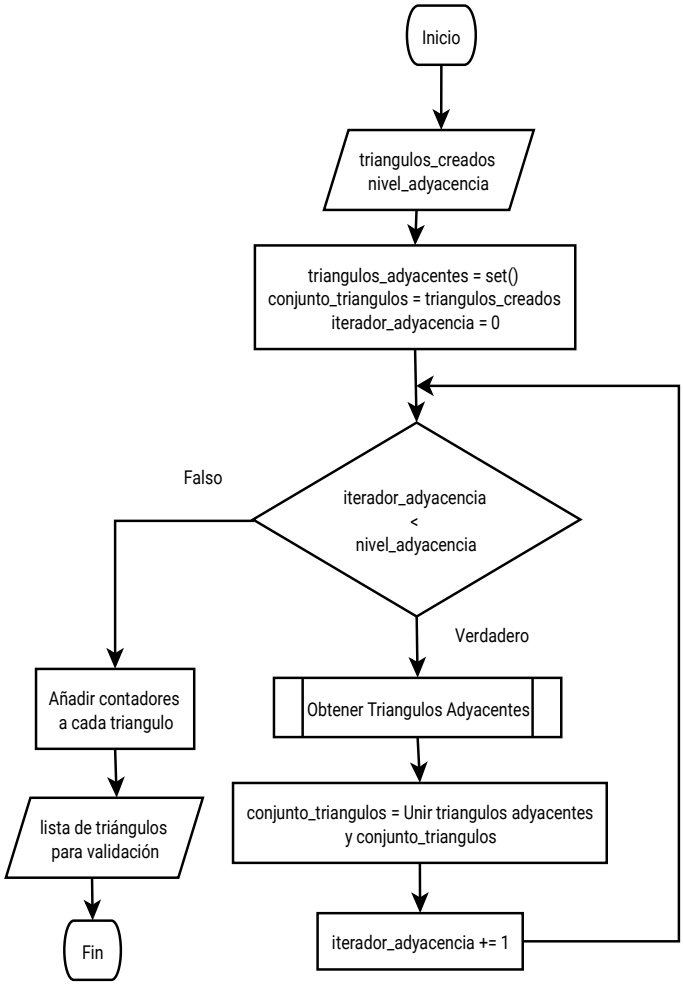


Ilustración 35: Diagrama de flujo correspondiente a la creación de la Lista de Validación

Después, el conjunto de triángulos es transformado a lista de validación. Esta es una lista de listas donde cada elemento incluye uno de los triángulos y un contador de valor cero asociado al número de veces que el triángulo ha sido intentado ser validado.

Existen segmentos cuyo valor del atributo triangle_left contiene una cadena vacía. Estos segmentos se corresponden con segmentos de Polígonos Isla. Para este caso, se espera un Error de Clave a la hora de acceder al diccionario de triángulos. En este caso no se añade ningún contenido a la lista conjunto_triángulos, y se pasa al siguiente ciclo.

4.4.2. Validación de la Lista

Se utiliza el método check_list para validar la lista creada anteriormente. Sus

parámetros de entrada consisten en la lista propiamente y en el nivel de adyacencia. Dispone de varios métodos con los cuales gestionan si los triángulos deben ser volteados o no. El proceso consiste en ejecutar un bucle en donde cada iteración evalúe cada triángulo de la lista. Para detener el bucle, debe de existir convergencia y se establece un límite de repetición cuyos triángulos no han de superar. La convergencia implica que todos los triángulos de la lista de validación siguen cumpliendo la condición de CCV una vez han sido volteados.

En ocasiones, este proceso puede no terminar nunca, ya que puede existir triángulos cuyos volteos modifiquen la triangulación de tal forma que determinados triángulos adyacentes dejen de cumplir la condición de CCV. Si el resultado de las versiones de los triángulos modificados no resuelve el problema, es necesario establecer un criterio para detener el proceso de validación con el menor número de triángulos que no cumplen la condición de CCV, esto es un indicador de que no hay convergencia y termina la validación (ver Ilustración 36).

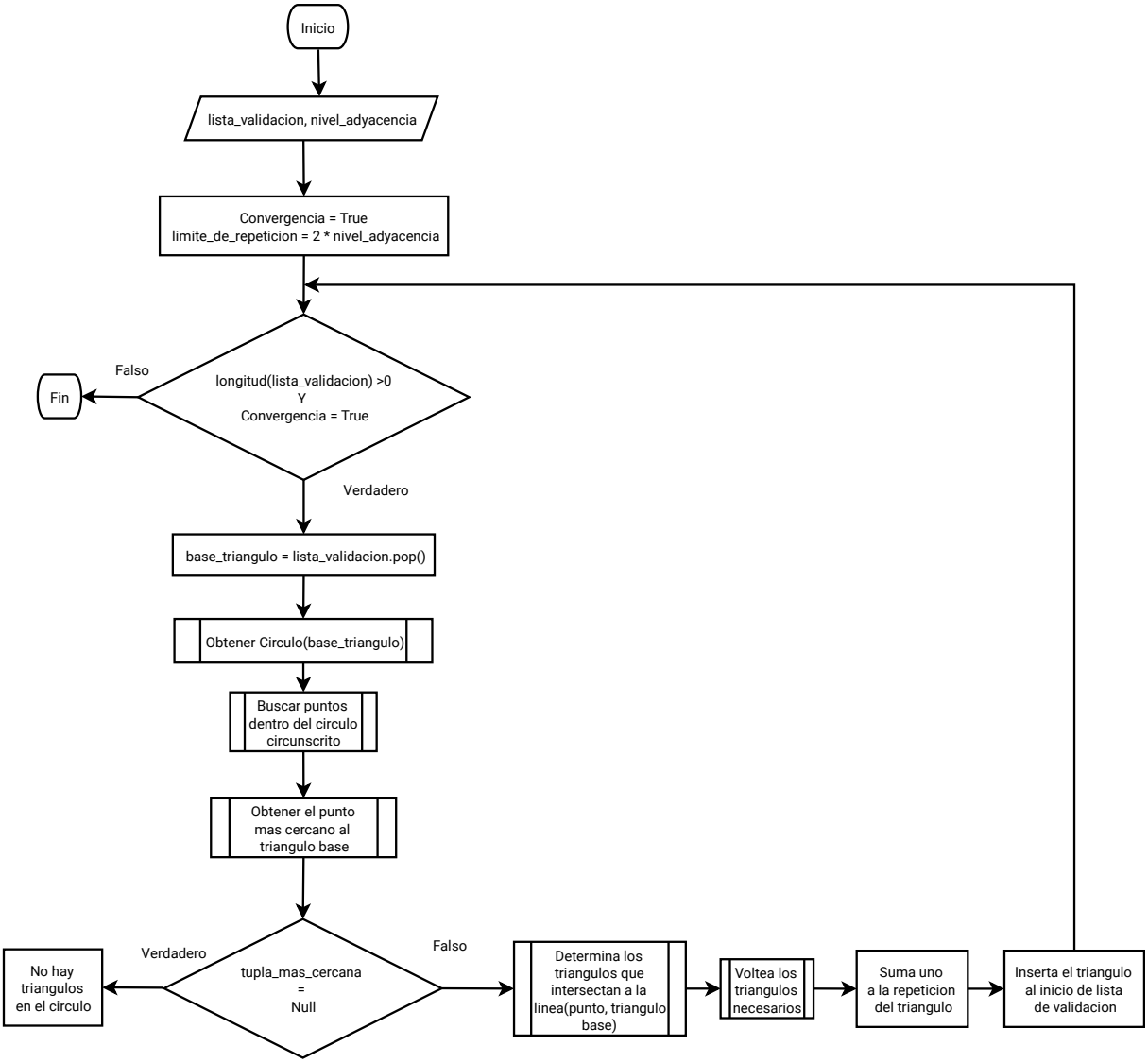


Ilustración 36: Diagrama de flujo correspondiente al proceso de validación de la Lista de Validación

4.4.3. Manejo de nube de puntos que no tienen solución

Con el fin de cumplir con la condición de CCV, esta implementación voltea los triángulos adyacentes al triángulo a validar. Debido a la distribución de los puntos en la nube de puntos, es posible que el proceso de validación de triángulos nunca termine (ver Ilustración 37).

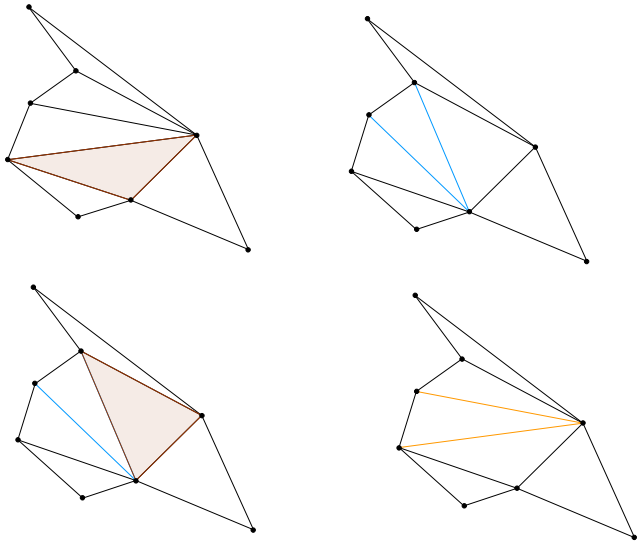


Ilustración 37: Proceso de validación donde se observa el triángulo a validar a la izquierda, y el resultado del volteo a la derecha. Se puede observar como el resultado consecutivo de los triángulos a validar desemboca en el estado inicial, generando un bucle sin fin.

En algunas ocasiones que no se pueden determinar con antelación, voltear dos triángulos adyacentes puede conllevar a que haya nuevos vértices de la triangulación que queden dentro de los CCs de los nuevos triángulos que han sido alterados. Este nuevo grupo de triángulos también tienen un vértice dentro de su CC, por lo que habría que voltearlo de nuevo. Después de que suceda esta situación repetidas veces, es posible que el proceso de validación llegue de nuevo a los triángulos que fueron volteados en primer lugar. Esta situación lleva a un bucle infinito donde nunca se terminaría el procesado de la validación.

En esta implementación, como se puede ver en el Pseudocódigo descrito en el apartado del Anexo 1.8, se utiliza un límite de repetición de validación de triángulos. Este límite es establecido como el doble del nivel de adyacencia elegido para resolver la triangulación. Esto quiere decir, que si el nivel de adyacencia es dos, los triángulos de la lista de validación solo podrán ser volteados cuatro veces para intentar resolver la situación.

4.4.4. Volteado de dos triángulos Adyacentes

Los métodos de inserción de los triángulos siempre generan triángulos orientados en el sentido de la agujas del reloj, véase Ilustración 31. Esto es así ya que han sido generados respondiendo a unas plantillas como se ha explicado anteriormente en el apartado Inserción de puntos de relleno.

En este proceso también ha sido utilizado una plantilla para poder llevar a cabo el volteo, de forma que se siga respetando la topología de los triángulos. En primer lugar hay que determinar qué lados de los triángulos a voltear son adyacentes, y acto seguido, se van nombrando cada elemento del triángulo respetando un patrón que permite la definición de la orientación de los segmentos de los triángulos en el sentido de las agujas del reloj (ver Ilustración 39).

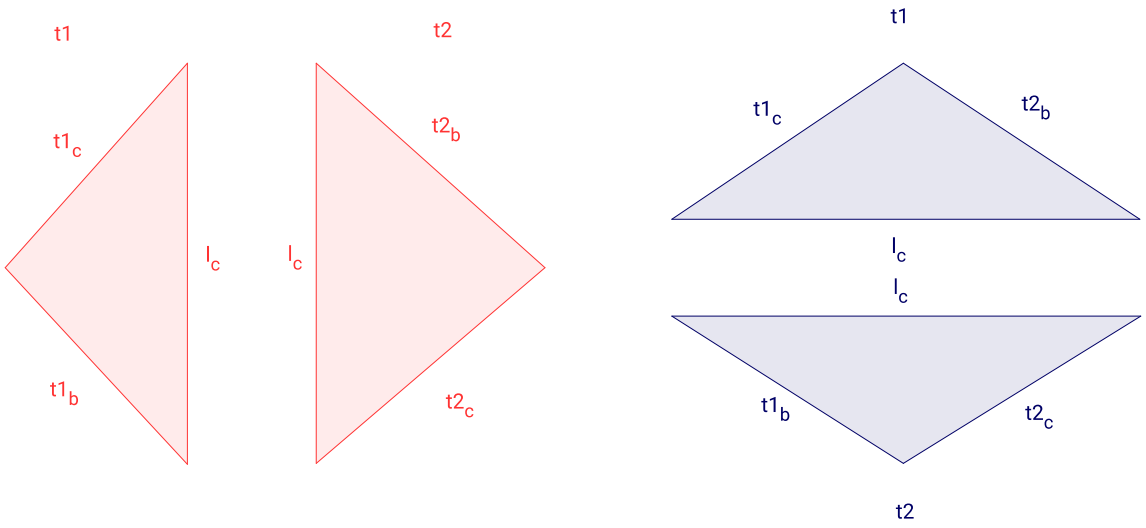


Ilustración 38: Esquema para representar el volteo de dos triángulos adyacentes. Los triángulos del mismo color han sido separados para facilitar su comprensión

Una vez definida la plantilla, cada segmento de los triángulos quedan aislados. El momento del volteo consiste en un intercambio de coordenadas entre los vértices de los lados adyacentes y sus vértices opuestos. Hay segmentos que pasan a formar parte del triángulo volteado y otros se mantienen. En la Ilustración 38 se puede observar el proceso de volteo. Se corresponde con la plantilla antes y

después del volteo, los segmentos t1_b, t1_c, t2_b, t2_c no necesitan ninguna modificación. En el Pseudocódigo descrito en el apartado 1.9 del Anexo, se puede observar como tienen lugar las modificaciones realizadas a los segmentos l_c de ambos triángulos.

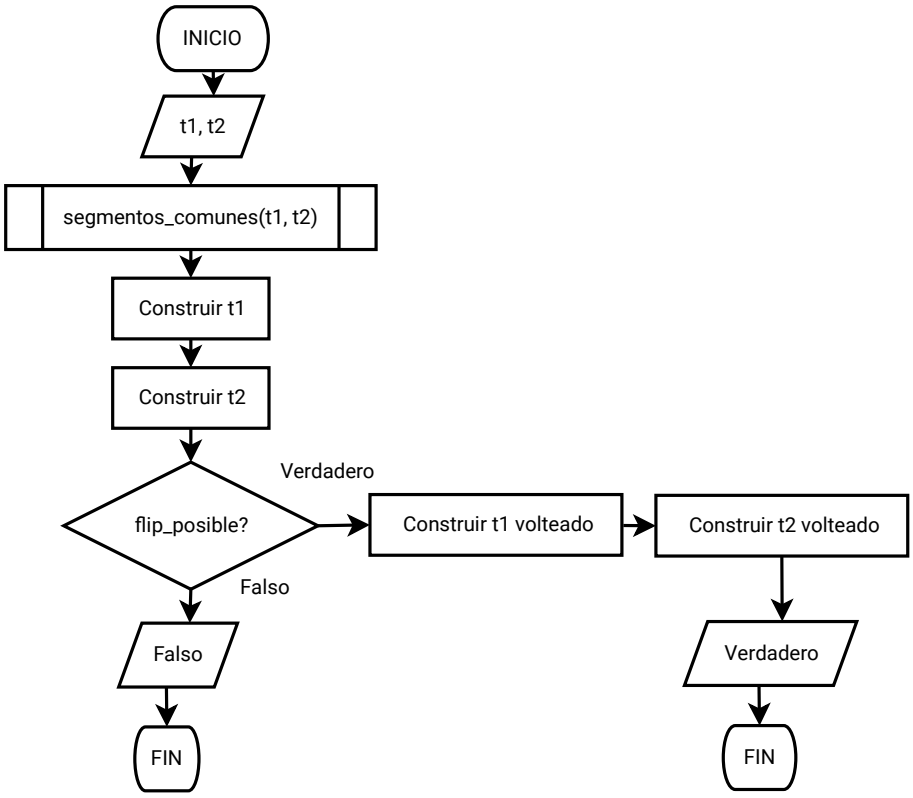


Ilustración 39: Diagrama de flujo para el volteo de dos triangulos adyacentes

Hay ocasiones donde dos triángulos adyacentes no pueden ser volteados por este método. Es en el caso donde el punto opuesto a un triangulo está a la izquierda del segmento b, y a la izquierda del segmento c (ver Ilustración 40 para observar esta situación con más detalle).

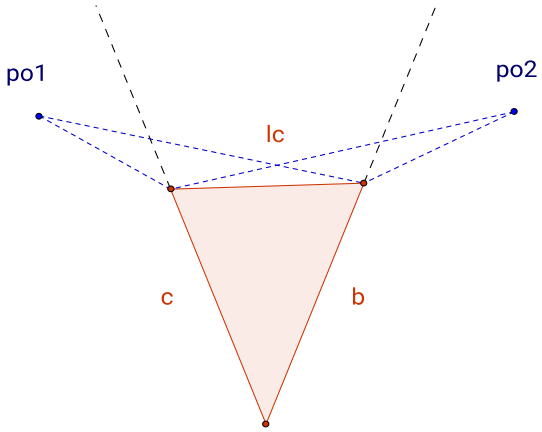


Ilustración 40: Representación de geometrías no aptas para ser volteadas. Los triángulos compuestos por Punto Opuesto 1 (po1) y Lado Común (lc) y Punto Opuesto 2 (po2) y Lado Común (lc) son geometrías nos aptas para ser volteadas.

Para solucionar este problema, se ha determinado qué punto del triangulo adyacente es el vértice opuesto: se ha implementado si un punto se encuentra a la izquierda de un segmento, devolverá verdadero. Para que pueda ser volteado, el punto po1 debería estar a la derecha del segmento c, y a la derecha del segmento b. En cualquier otro caso, el método flip no está preparado para solventar esa situación, solo para ignorarlo, ya que los triángulos con esa geometría suelen ser volteados en el proceso de validación de otros triángulos de la lista de validación.

4.4.5. Volteo de triángulos que no son adyacentes

En el proceso de validación de triángulos, hay situaciones en los que existe un vértice dentro del CC que estamos evaluando. Sin embargo, como se observa en la ilustración 41, ninguno de los triángulos a los que pertenece se encuentran adyacentes al triangulo del CC.

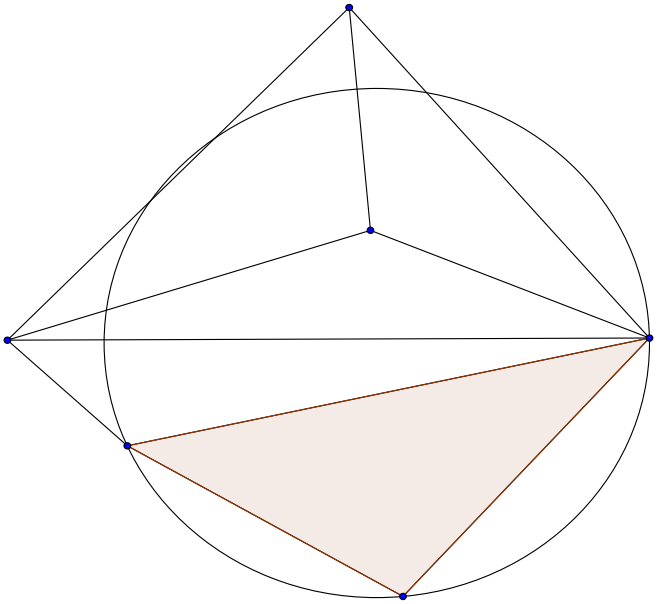


Ilustración 41: Volteo de triángulos no adyacentes. En esta situación, el triángulo con vértice dentro del CC no es adyacente.

El proceso para solventar esta situación consiste en ir volteando de forma consecutiva los triángulos que intersectan el segmento formado por el vértice que está dentro del CC y el centroide del triángulo base. En el diagrama de flujo de la Ilustración 42 se esquematiza el proceso. Para mejor detalle, consultar Pseudocódigo del aparto 1.10 del Anexo. Este proceso se corresponde con el método flip_concatenated, de la clase triangulation (véase ilustración 33).

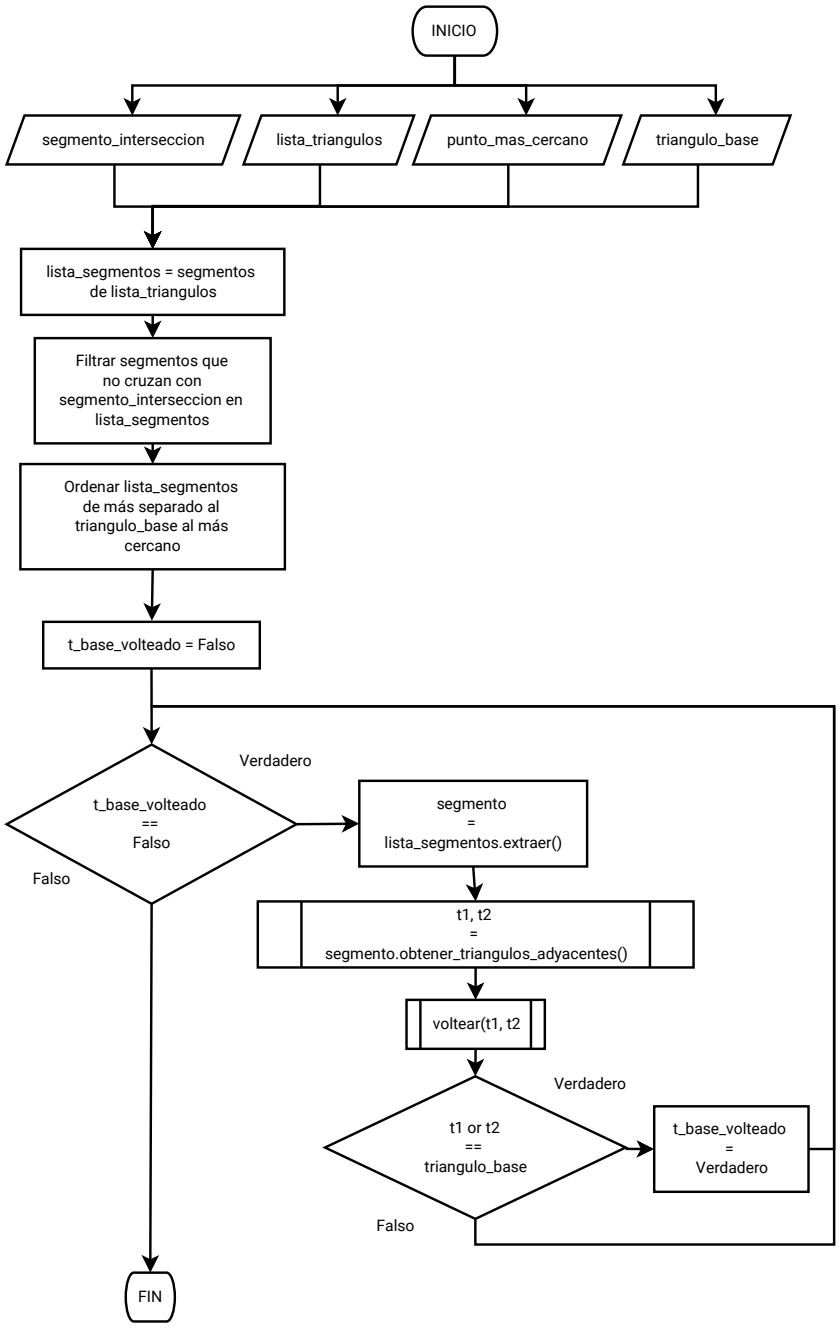


Ilustración 42: Diagrama de Flujo para el proceso de volteo concatenado. El triangulo base es el que esta siendo validado. El punto más cercano es el punto dentro del CC del triangulo base más cercano. El segmento de intersección es el segmento comprendido entre el punto más cercano y el centroide del triángulo base. La lista de triángulos se corresponde con los triángulos que intersectan al segmento de intersección.

Este método es utilizado en el proceso de validación de triángulos (ver apartado 4.4.2). Hay situaciones donde hay más de un vértice dentro de CC. Es de suma importancia que el vértice con el que se trabajará sea el más cercano al

triángulo base. El método flip_concatenated no conseguirá los volteos adecuados entre triángulos a menos que el vértice problema sea el más cercano. En la Ilustración 43 se puede observar el proceso gráfico paso a paso para obtener la validación del triángulo.

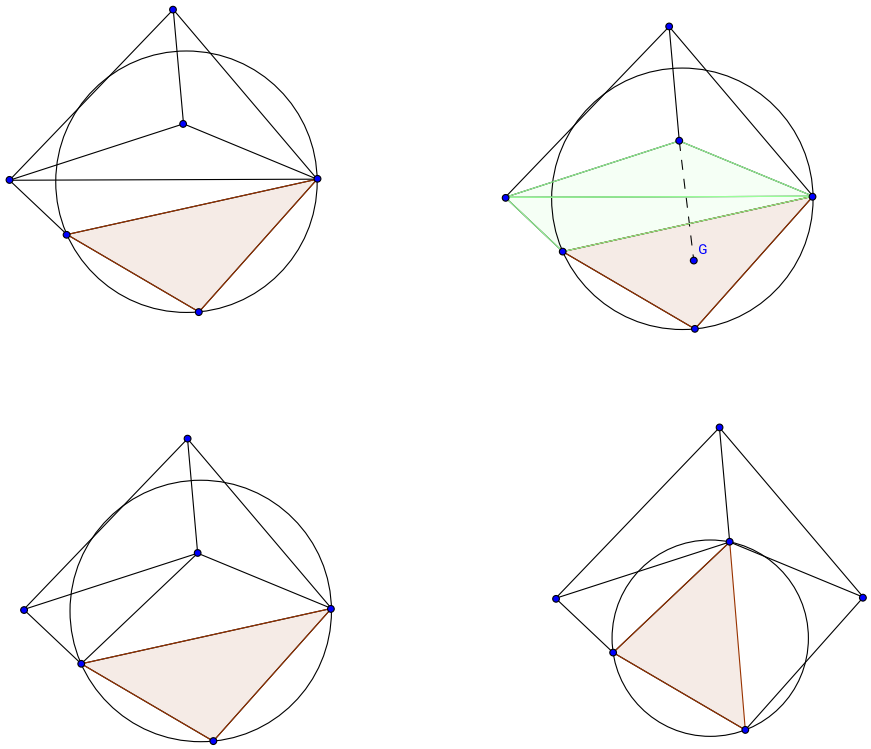


Ilustración 43: Metodología paso a paso en el algoritmo de volteo concatenado

4.5. Inserción de Líneas de Rotura

Los atributos necesarios para la indicación de líneas de rotura, han sido añadidos al objeto segment y triangulation (véase ilustración 25). Estos atributos añaden la posibilidad de acceder a la información de forma bidireccional: a partir de un segmento, determinar a qué linea de rotura, y a qué segmento de línea de rotura pertenece; permite determinar a que isla o segmento de isla pertenece; a partir de una determinada linea de rotura qué triángulos están implicados y por ende, determinar qué segmentos pertenecen a la línea de rotura.

4.5.1. Inserción de Líneas de Rotura después de inserción de Puntos de Relleno

A la hora de implementar las líneas de rotura, estas son insertadas a la triangulación después de la inserción de los puntos de relleno.

Si, las líneas de rotura son insertadas justo después de la inserción del triángulo ficticio y antes de los puntos de relleno, existe una problemática al respecto. Debido al orden de inserción, los polígonos generados en el proceso son muy astillados y esto requiere realizar más validaciones. También es probable que el algoritmo de validación se detenga sin llegar a validar todos los triángulos.

Al insertar la líneas de rotura en segundo lugar, se parte de la existencia de una Triangulación de Delaunay. El hecho de insertar las líneas de rotura en este momento, modifican muy poco la triangulación, por ello, generar esta modificación en un estado avanzado de la triangulación es menos costoso y distorsiona menos la triangulación, ya que se parte de una triangulación bastante parecida a la de Delaunay.

4.5.2. Inserción de Segmentos de Línea de Rotura que intersecta varios triángulos

Al insertar un segmento de Línea de Rotura, se producen problemas topológicos cuando un segmento de línea de rotura intersecta varios segmentos de la triangulación, como se puede observar en la ilustración 18, y aunque la implementación se basa en los procedimientos de inserción de un punto en la triangulación, estos dejan de ser suficientes. La principal diferencia que existe es que para insertar un segmento de linea de rotura, es necesario insertar al menos dos puntos y modificar la topología de los segmentos de triangulación que pertenecen a la línea de rotura.

Una de las problemáticas en la implementación con el uso de GDAL, son el uso de las funciones en las que se llevan a cabo relaciones topológicas. Estas funciones, como por ejemplo, la función intersección, la función dentro y la función cruzar hay que utilizarlas con mucho cuidado (ver apartado 2.3.6). Es importante utilizar éstas

funciones con geometrías de la misma categoría, ya que así se puede obtener un resultado apropiado. Por ejemplo, utilizar la función intersección entre un LineString y un Polygon devuelve un LineString.

Más importante aún, durante la implementación se ha detectado en una ocasión, que cuando se determina la geometría de intersección entre dos segmentos, no siempre la geometría punto resultado toca a los dos segmentos (ver ejemplo en el apartado 2.1 del Anexo y ver la Ilustración 44 con la representación gráfica del ejemplo).

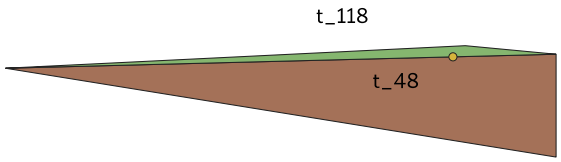


Ilustración 44: Ejemplo gráfico para el resultado del proceso Intersección. El punto está dentro de t_48 y no toca ni a t_118, ni a t_48.

Para esta implementación se ha optado por utilizar la metodología que incluye puntos virtuales (véase apartado 3.2.3.2). El principal motivo ha sido por su sencillez a la hora de administrar la topología de la estructura de datos y el gran grado de reutilización de los procedimientos y métodos existentes. En la Ilustración 45 se muestra el Diagrama de Flujo del proceso general, y en el apartado 1.11 del Anexo se muestra su Pseudocódigo. Muestran el método insert_breakline de la clase triangulation para la inserción de líneas de rotura (ver Ilustración 25).

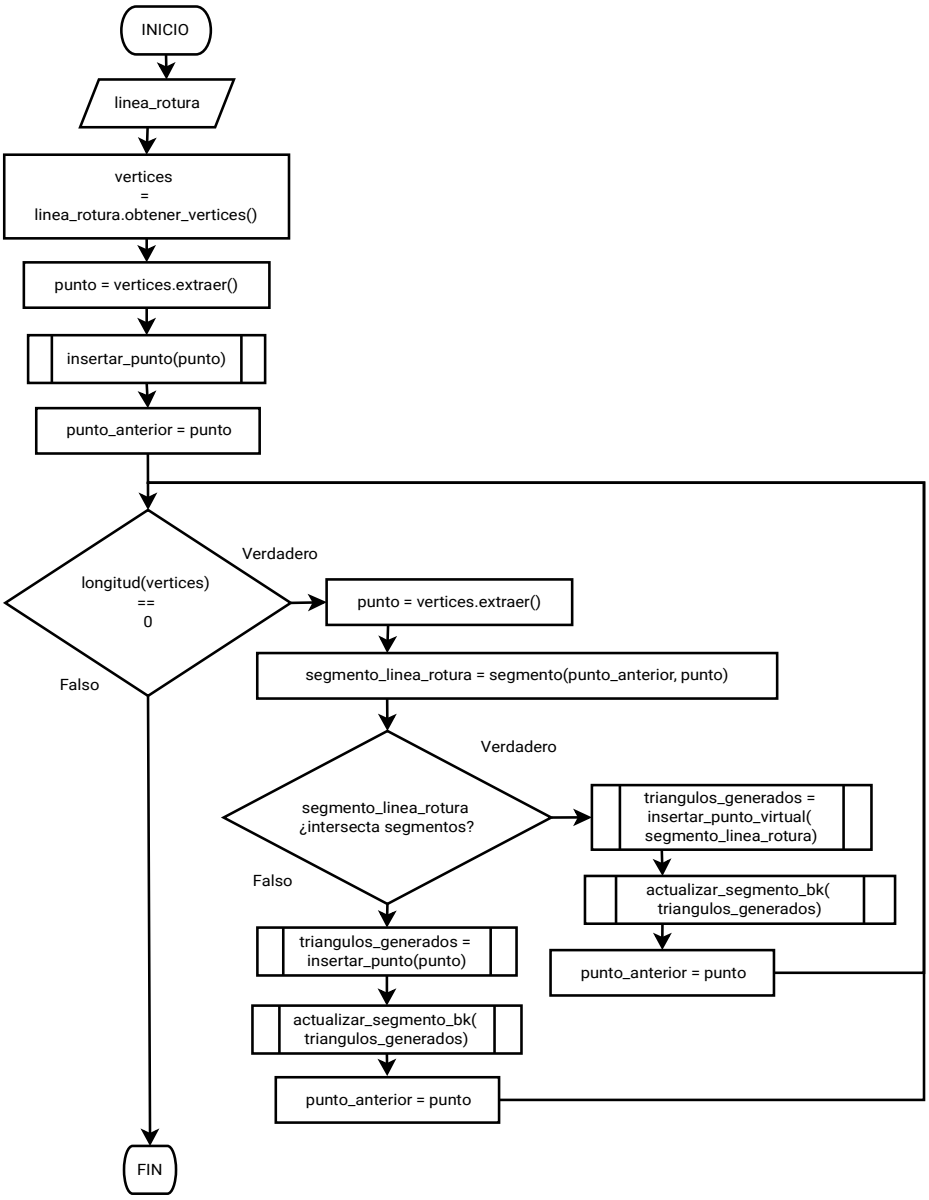


Ilustración 45: Diagrama de flujo para la inserción de lineas de rotura

4.6. Inserción de Polígonos Isla

Para la inserción de polígonos isla, partimos de la implementación utilizada en la inserción de Líneas de Rotura. Básicamente es el mismo procedimiento, sus complejidades son similares. Excepto que para un Polígono Isla el punto inicial y final del polígono es el mismo.

Este proceso se realiza después de la inserción de las Líneas de Rotura. El

orden de implementación entre la inserción de los Polígonos Isla y las Líneas de rotura no responde a ningún motivo. Solo es necesario que la implementación de la inserción de los Polígonos Isla sea después de la inserción de los Puntos de Relleno, ya que el motivo es el mismo que para la inserción de Líneas de Rotura (véase apartado 4.5.1).

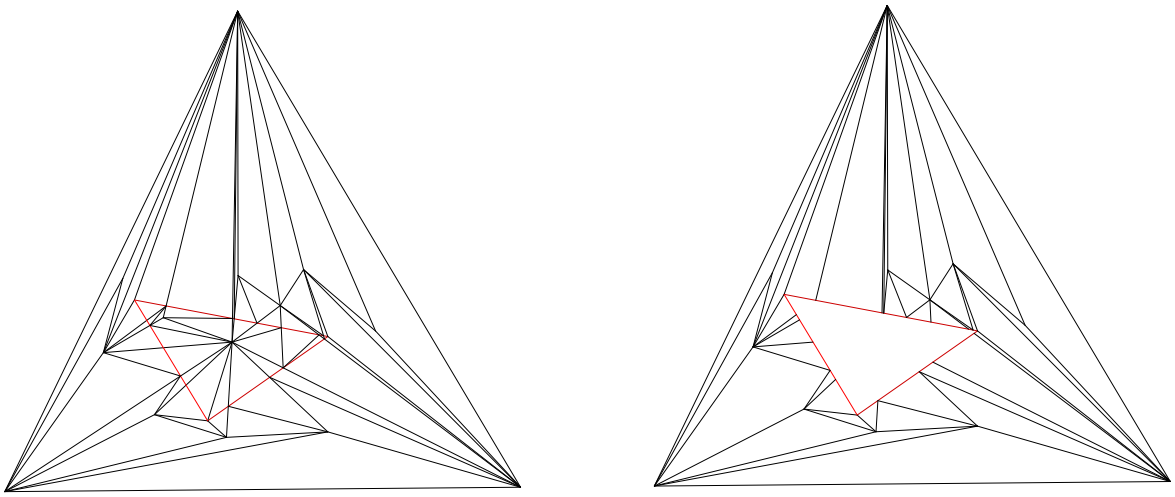


Ilustración 46: Inserción de polígonos isla. Este es el resultado de utilizar la misma metodología de inserción de segmentos que en el apartado 3.2.3.2

Aunque a priori parece un proceso sencillo, se han añadido algunas modificaciones para evitar un problema de actualización en la estructura de datos, debido a que la implementación se basa principalmente en un proceso de inserción de un punto en la triangulación y no en la modificación de relaciones a partir de un punto.

Cuando se inserta el último segmento del polígono Isla, el punto final ya existía en la triangulación, por lo que no puede reinsertarse de nuevo. Se puede ver estas modificaciones en la Ilustración 47, o en el Pseudocógido del apartado 1.12 del Anexo, donde se muestra el método insert_hulls de la clase triangulation (véase ilustración 25).

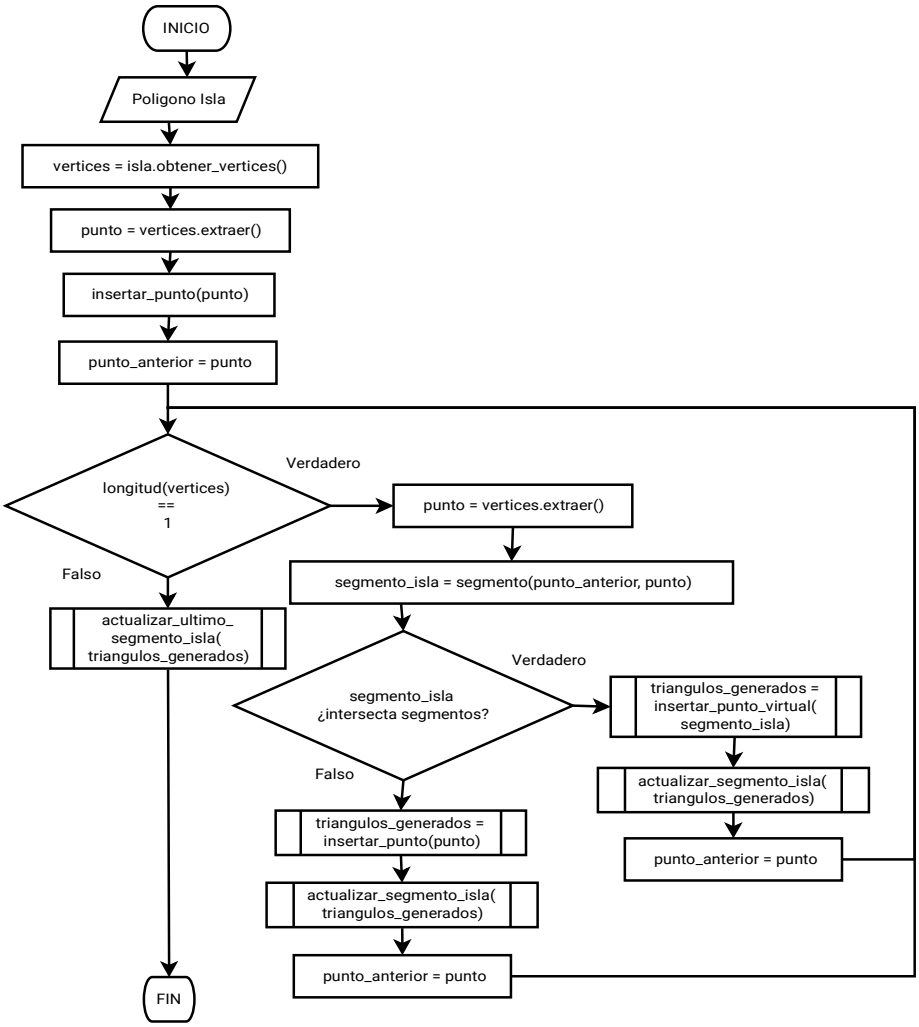


Ilustración 47: Diagrama de Flujo para la inserción de Polígonos Isla

Esta modificación consiste en detener el ciclo de inserción de puntos cuando este encuentra que el punto a insertar es el mismo que el punto inicial. Es en esa situación cuando en vez de insertar un nuevo punto, se actualiza los segmentos de los triángulos insertados del último Punto Virtual, que contendrán el punto inicial.

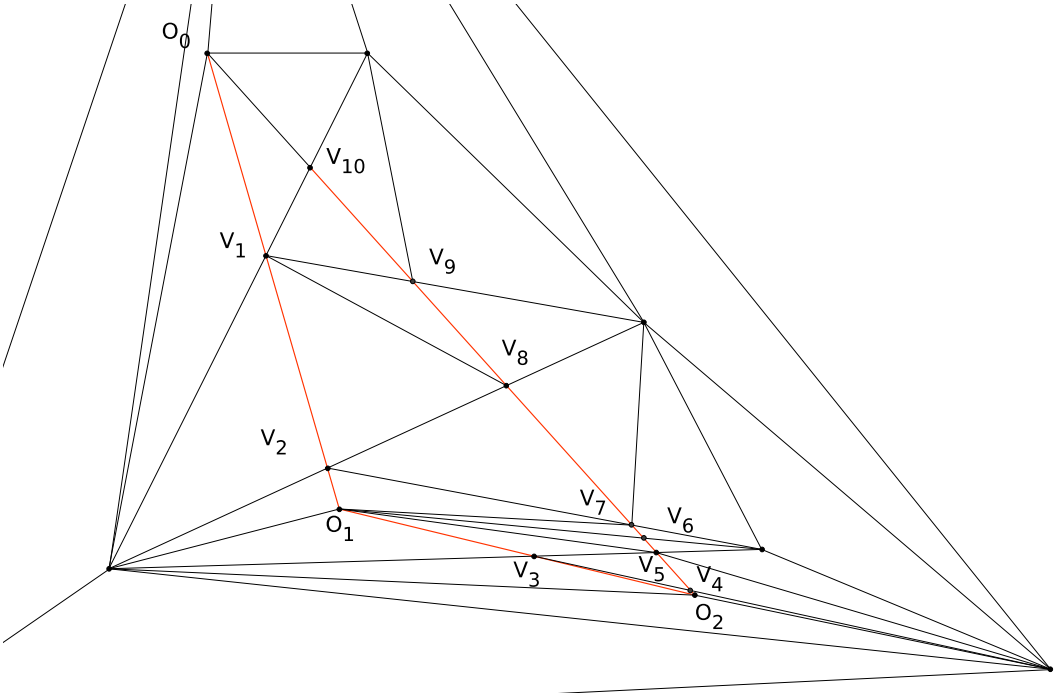


Ilustración 48: Inserción del último segmento de un polígono Isla. Los vértices O representan los vértices del polígono Isla, y los vértices V representan Puntos Virtuales.

En la ilustración 48, acaba de insertarse el punto virtual V_{10}, y se observa que el siguiente punto que aparece en el bucle es O_0 (correspondiente al primer punto del polígono). En esta ocasión, se parte de los segmentos de los triángulos generados al insertar V_{10}. Los segmentos cuyo punto inicial y final (o viceversa) se correspondan con V_{10} y con O_0 actualizan sus atributos para que pertenezca al Polígono Isla.

4.7. Vaciado de Polígonos Isla

Para no disponer de contenido en el interior de los polígonos Isla (como se observa en la ilustración 46), existen varias metodologías. Una de ellas consiste en eliminar la información del interior del polígono de la memoria. La metodología seguida en este proyecto procede de la manera en que la información no es eliminada, si no que es ocultada. Para ello, en la estructura de datos, los triángulos disponen de una bandera (atributo en el objeto triángulo, véase ilustración 25) para disponer si el triángulo será visualizado en los archivos salida o no.

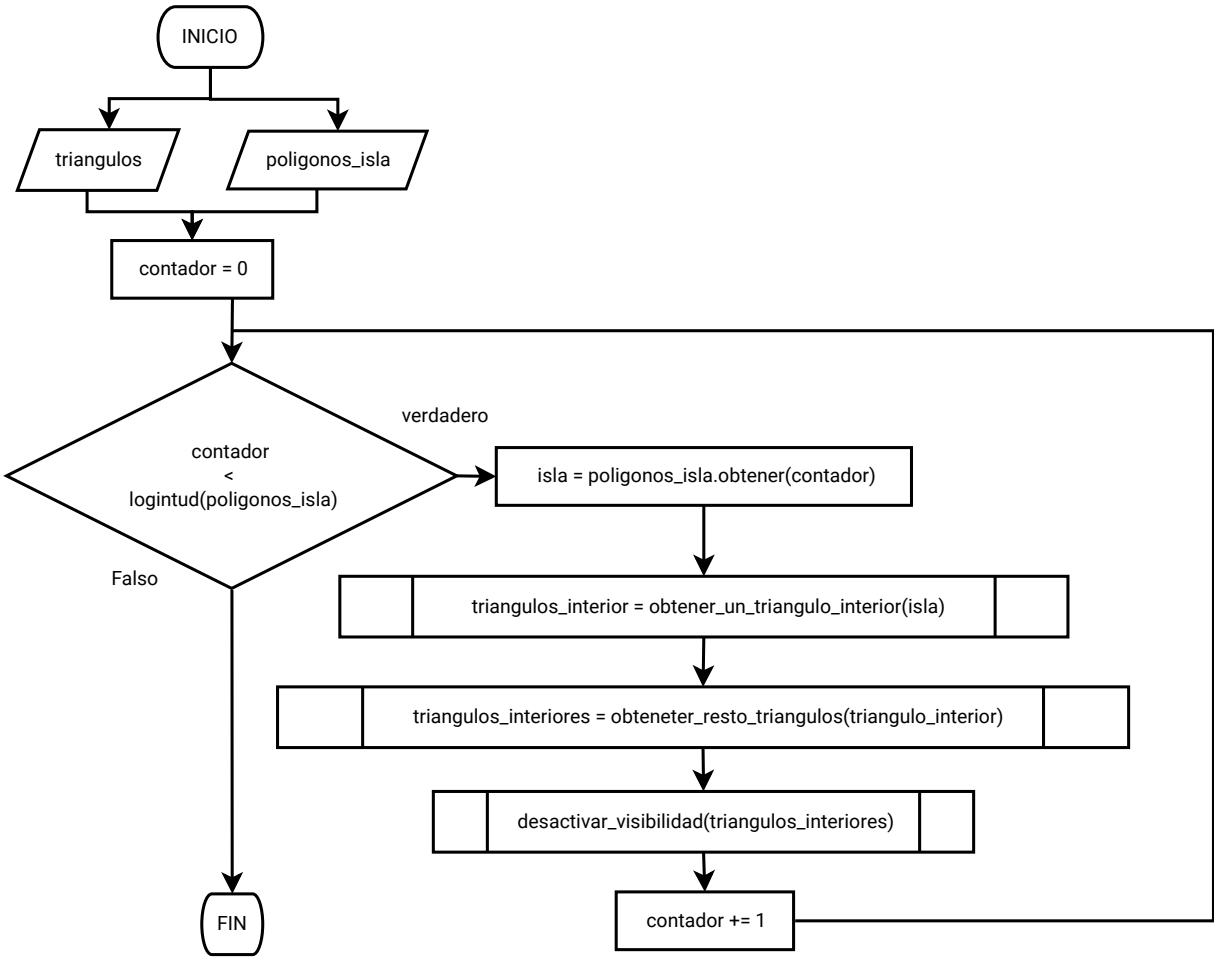


Ilustración 49: Diagrama de Flujo para cambiar el estado, de los triángulos interiores a los polígonos Isla, de visible a no visible.

El proceso general (ver Ilustración 49) consiste en leer la información de un polígono Isla y determinar, a partir de la topología de un segmento del polígono Isla, un triángulo adyacente en el interior del polígono Isla (ver Ilustración 50).

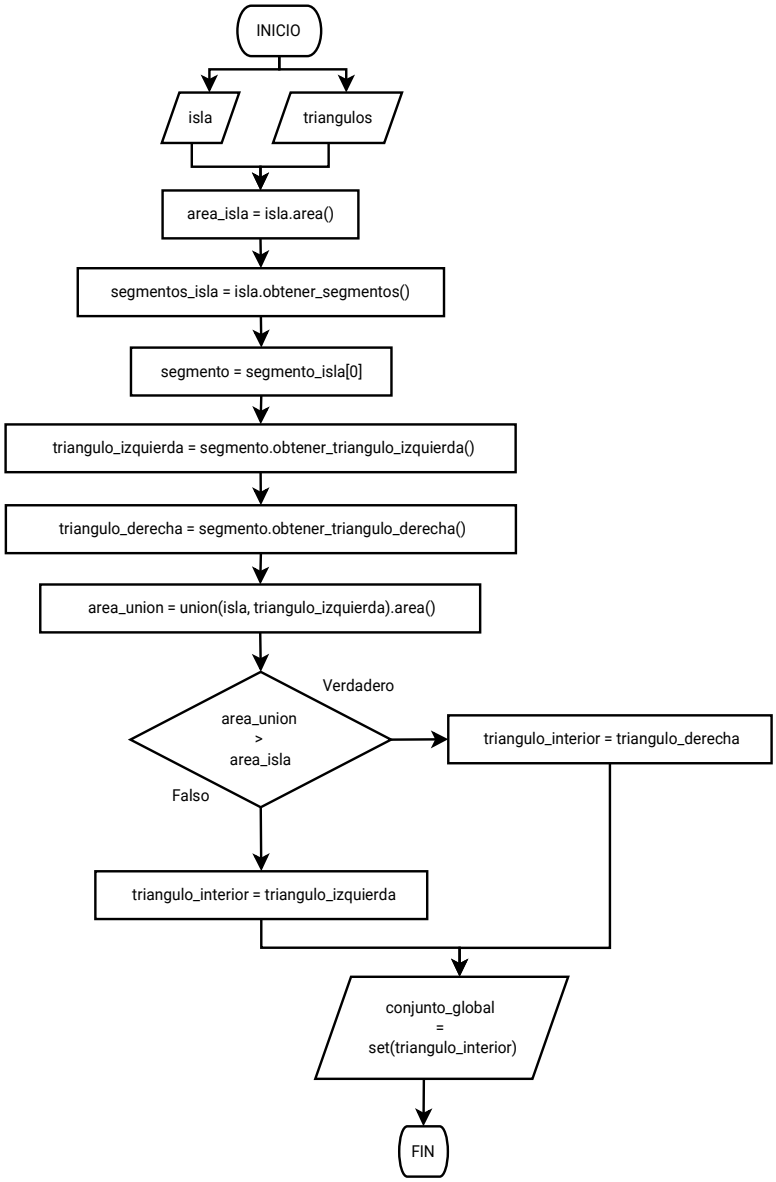


Ilustración 50: Diagrama de Flujo para la obtención de un triángulo interior a un polígono Isla

A continuación, a partir de ese triángulo interior, se determinan de forma cíclica qué triángulos adyacentes se encuentran dentro del polígono isla (ver Ilustración 51).

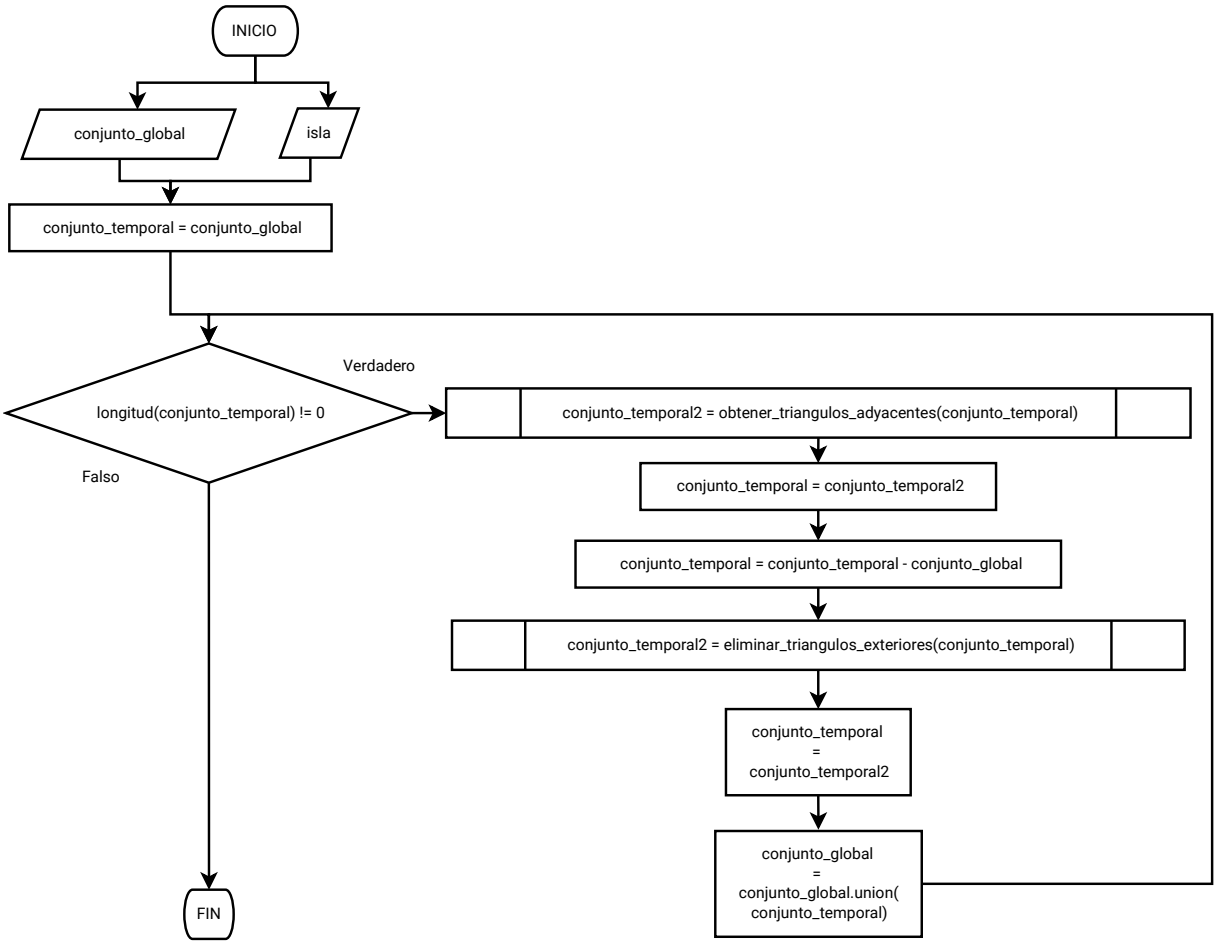


Ilustración 51: Diagrama de Flujo para la obtención del resto de triángulos interiores de un polígono Isla

Este bucle para la obtención de los triángulos interiores del polígono Isla a partir de un triángulo interior consiste en recorrer un conjunto global, el cual parte del triángulo interior al polígono Isla. Para cada ciclo, se determinan los triángulos adyacentes almacenándolos en un conjunto temporal (ver Ilustración 52) y añaden al conjunto global. Cuando exista un ciclo en el cual no se haya podido determinar ningún triángulo adyacente el conjunto temporal permanecerá vacío, y significará que todos los triángulos pertenecientes al conjunto global se corresponden con todos los triángulos interiores al polígono Isla, por lo que el proceso es terminado. Se puede consultar el Pseudocódigo en el apartado 1.13 del Anexo. Este representa el método toggle_visibility_triangles_hull, de la clase triangulation (véase ilustración 25).

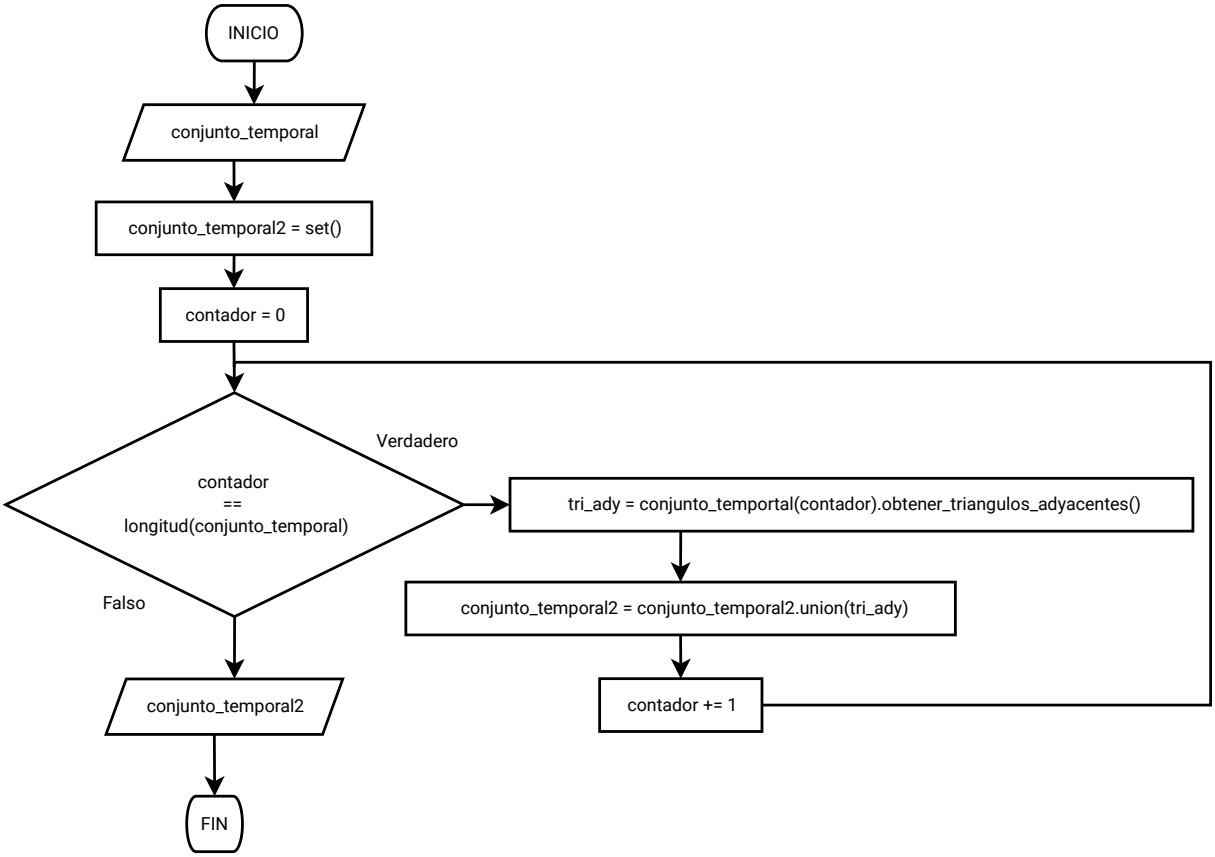


Ilustración 52: Diagrama de Flujo para la obtención de triángulos adyacentes a un conjunto de triángulos

Cabe mencionar (ver Ilustración 52) que el uso del tipo de datos Set de Python nos asegura que, del conjunto de triángulos adyacentes determinados, quedan excluidos los triángulos que ya existen en el conjunto temporal, por lo que el conjunto no contiene triángulos repetidos.

El hecho de haber determinado los triángulos adyacentes al conjunto global en cada ciclo, nos indica que existen triángulos en el conjunto temporal que se encuentran en el exterior del polígono Isla. Para seleccionar solo los triángulos del conjunto temporal que quedan en el interior del polígono Isla hay que evaluar cada triángulo por separado. Esta evaluación consiste en unir la geometría del triángulo en cuestión con la geometría del polígono Isla. Si el área de esta unión es mayor que el área del polígono Isla, quiere decir que el triángulo evaluado se encuentra fuera del polígono Isla y quedará excluido del conjunto temporal (ver Ilustración 53).

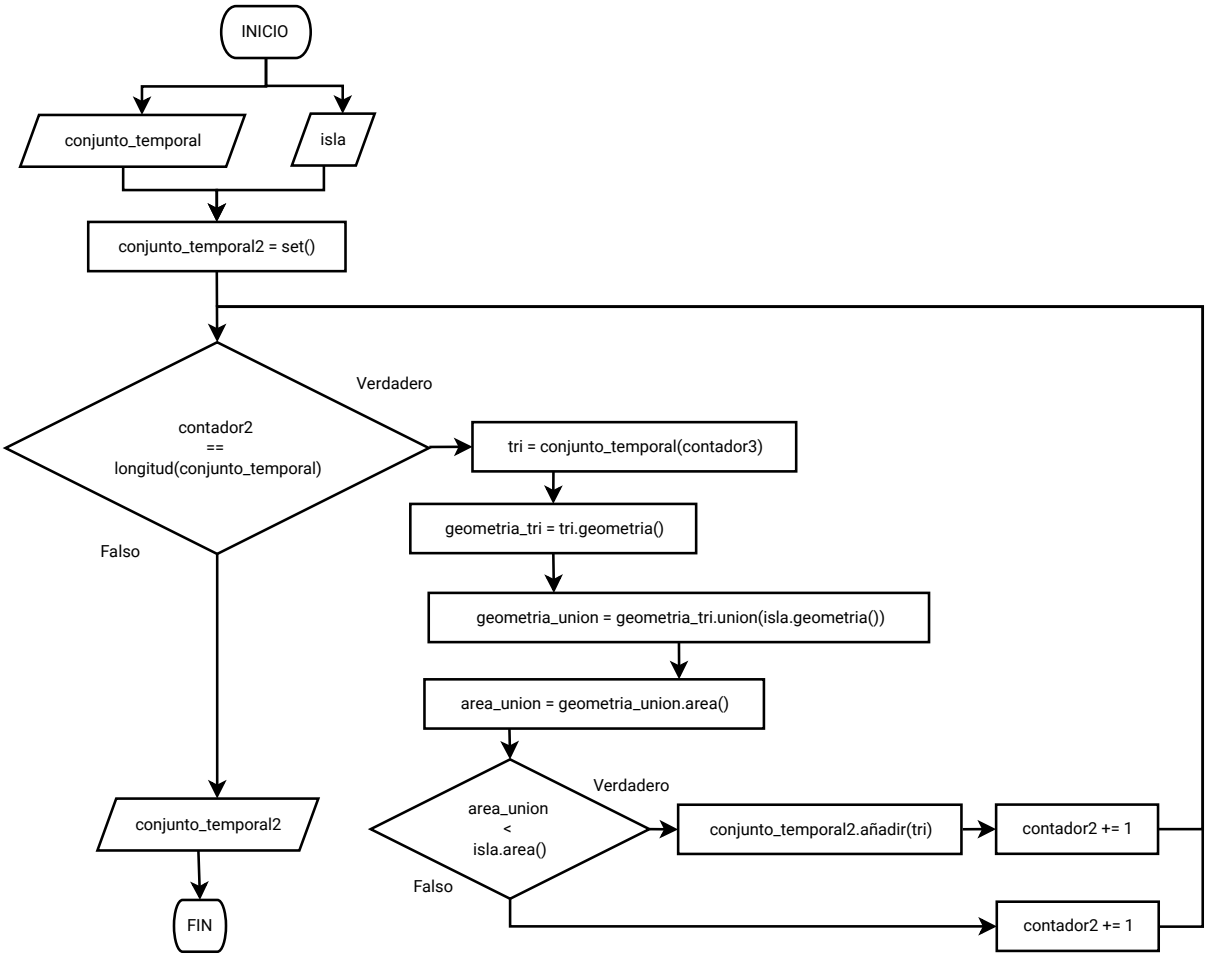


Ilustración 53: Diagrama de Flujo para eliminar los triángulos del conjunto temporal exteriores al polígono Isla

Cuando se determina que un triángulo puede pertenecer a un polígono isla, se compara la unión de la geometría de la Isla y la geometría del triángulo. Al utilizar el método Unión de GDAL, es posible que debido a la precisión de los cálculos, existan triángulos que pertenecen al polígono Isla y sin embargo el área de la Unión se mayor que el área del polígono Isla. Para solventar este problema, se ajusta la precisión del cálculo aumentando el área del polígono Isla un 1/10000 el área origina, siendo suficientemente grande para no dar lugar a este error de precisión. En nuestro caso, la magnitud del aumento se corresponde a la multiplicación del área del polígono Isla por la constante 0.0001, esto confiere cierto dinamismo al computo de la precisión a esta implementación.

5. RESULTADOS

Para la comprobación de la implementación del software desarrollado, se ha utilizado el software Arc GIS Pro y el software Aplitop MDT para comparar el resultado de la ejecuación del programa. A continuación se muestra cómo ha sido llevado a cabo.

5.1. Validación del Algoritmo

Los datos que han sido utilizados para esta comparación se corresponden con tres capas vectoriales. Una capa de puntos de relleno, una capa de líneas de rotura y una capa de polígonos isla. Su extensión se corresponde con xMin,yMin -0.737088,-0.868064 : xMax,yMax 1.436,0.782468 y su Sistema de Coordenadas es Local. Con este Conjunto de Datos Geográficos (CDG) se pretende forzar situaciones pueden producirse y que hay que tener en consideración, cómo por ejemplo, disponer Puntos de Relleno alineados puede desembocar en la situación en la es necesario insertar un punto donde ya existe un segmento. También se han utilizado coordenadas negativas, para tener la certeza de que el algoritmo no se ve afectado por esta situación. Por último, todas las geometrías disponen componente vertical.

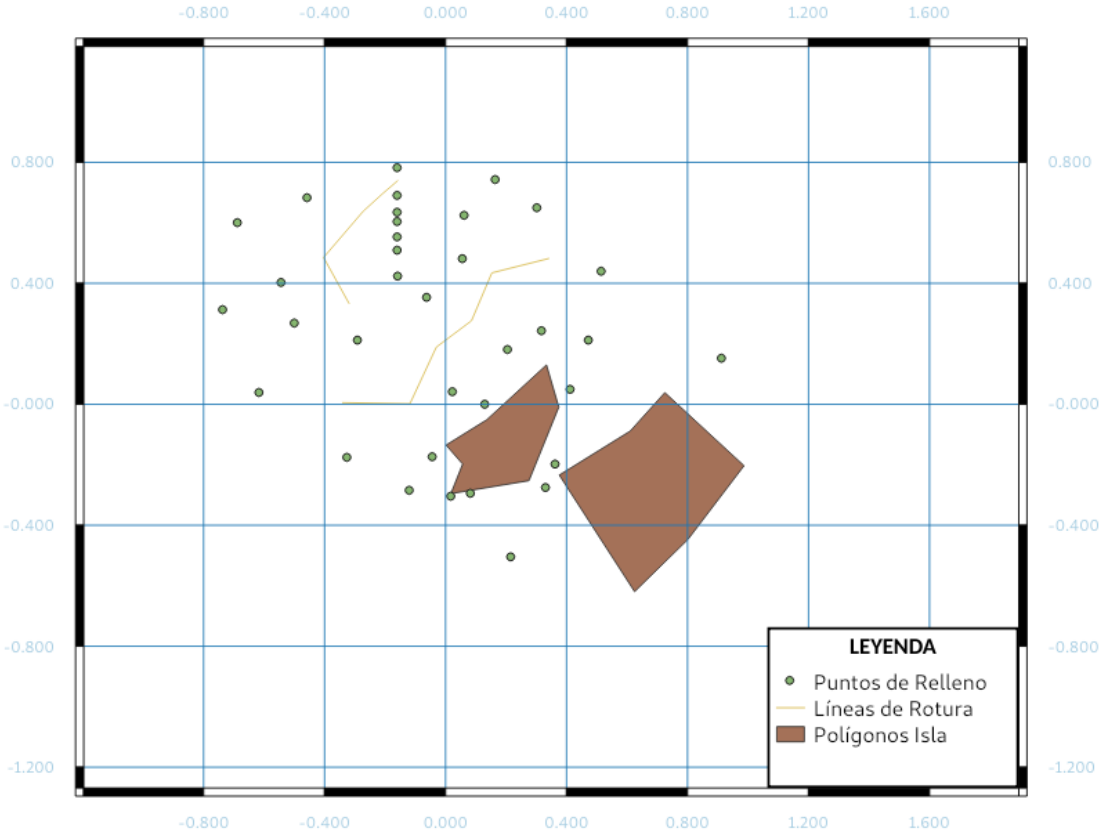


Ilustración 54: Conjunto de datos geográficos de ejemplo como datos de entrada para el algoritmo de Triangulación de Delaunay Constreñida

5.1.1. Resultados con ArcGIS Pro

Utilizando la herramienta Crear TIN del software ArcGIS Pro, utilizando el parámetro que fuerza su comportamiento como Triangulación de Delaunay Constreñida, en la Ilustración 55 se observa el siguiente resultado.

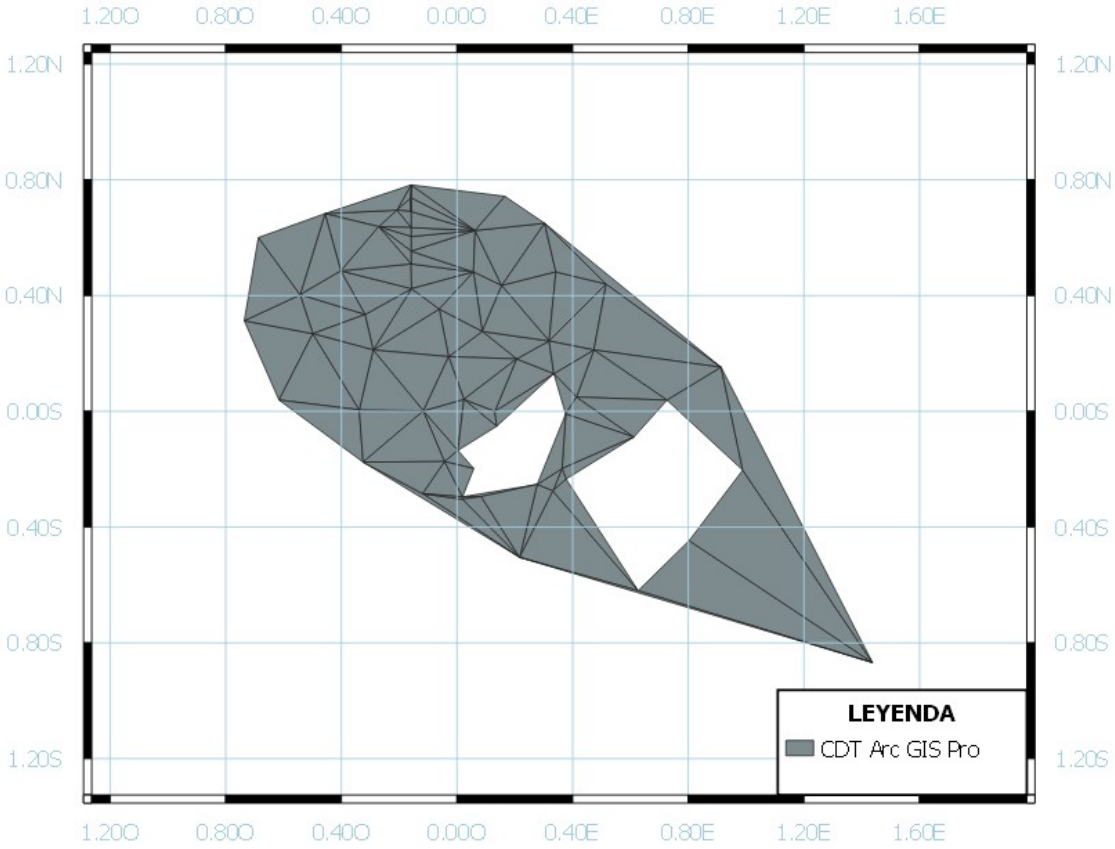


Ilustración 55: Resultado de la ejecución del proceso Crear TIN en el programa Arc GIS Pro

5.1.1.1. Resultados con Aplitop MDT

Para el establecimiento de las líneas de contorno, en Aplitop MDT se ha utilizado la creación de la Línea de Contorno de forma manual.

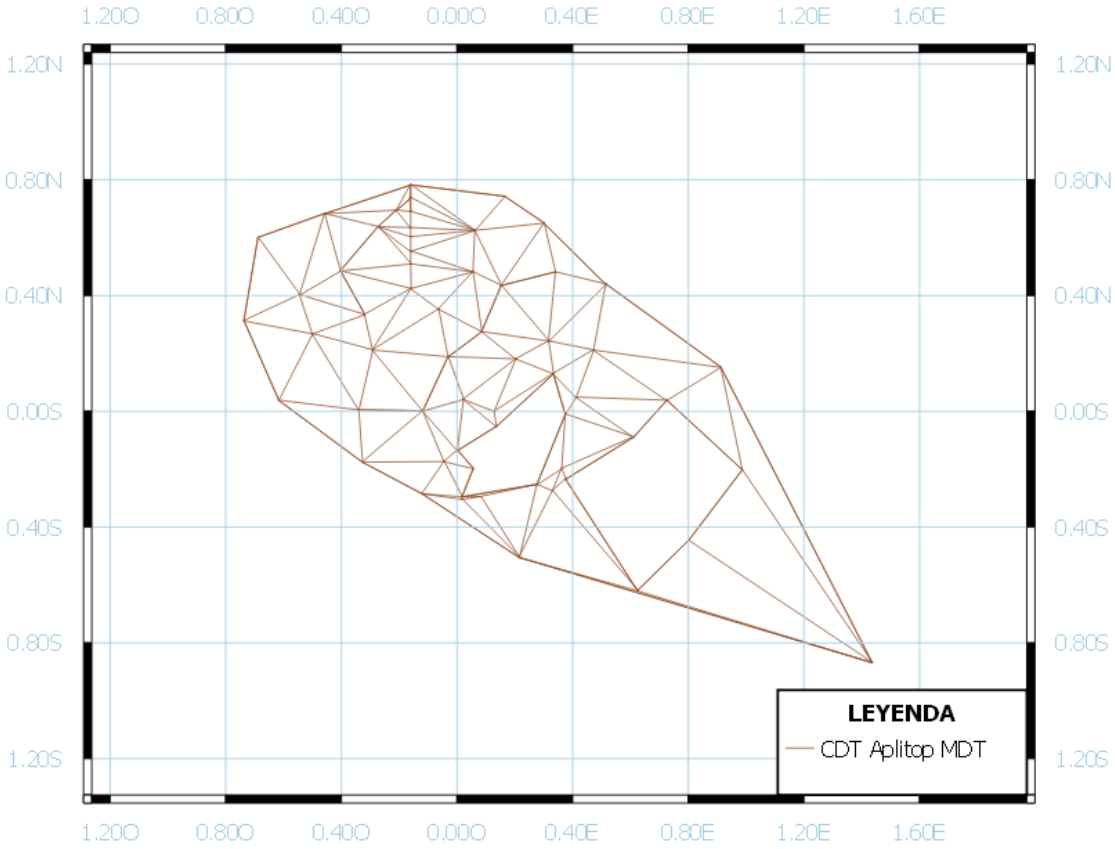


Ilustración 56: CDT construida utilizando Aplitop MDT

Cómo se observa en la Ilustración 56, el resultado obtenido no difiere tanto al obtenido por Arc GIS Pro.

5.1.2. Resultados con el Plugin de QGIS

En la Ilustración 48 se puede observar la interfaz gráfica mostrando los parámetros de entrada para la ejecución del algoritmo, correspondiente al CDG mostrado.

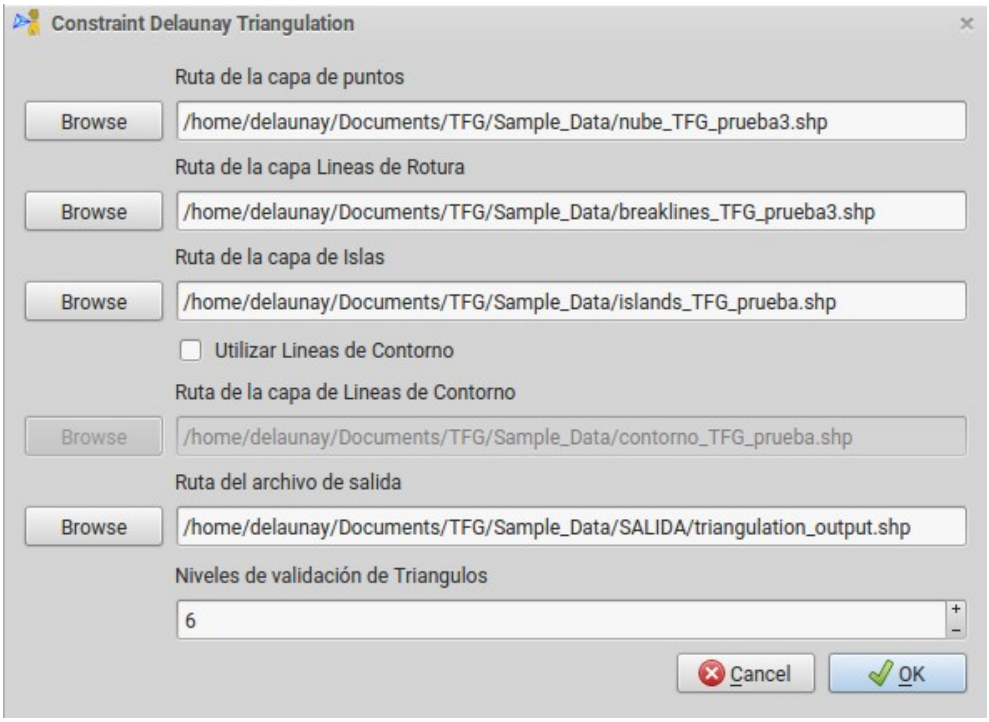


Ilustración 57: Parámetros de Entrada

En la Ilustración 58, se observa el resultado obtenido por el Plugin implementado.

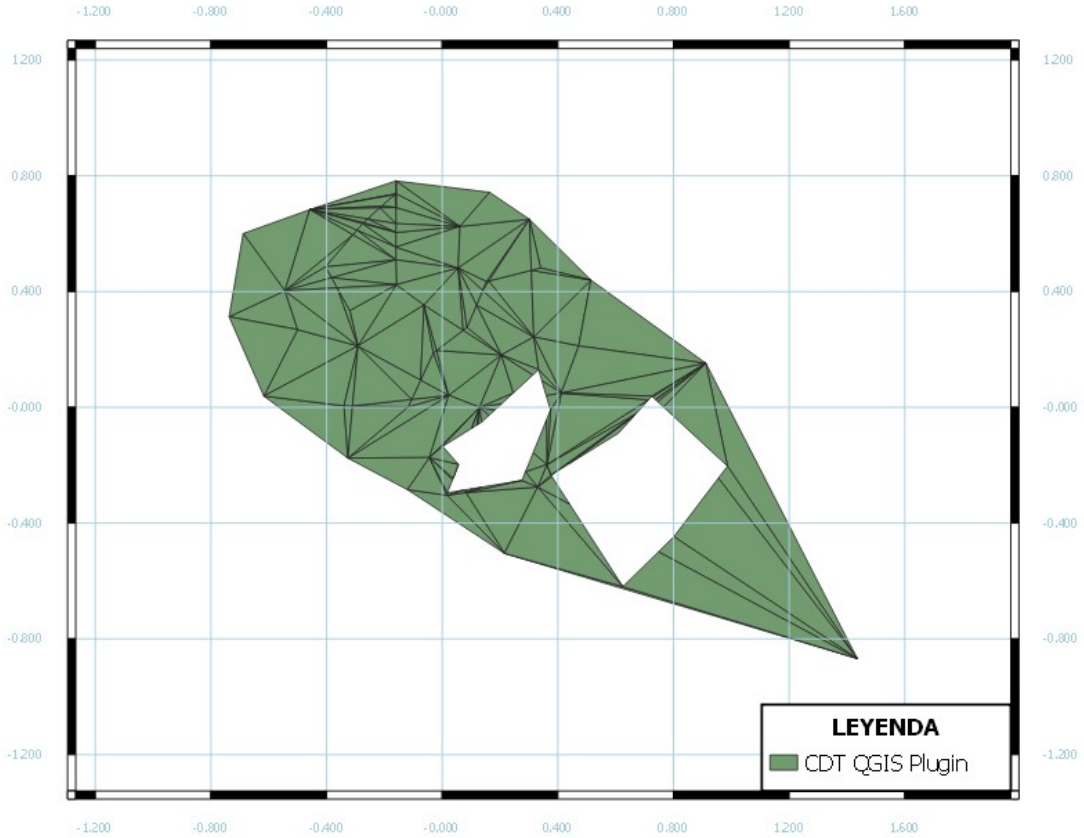


Ilustración 58: Salida de la ejecución de la implementación del algoritmo de Triangulación de Delaunay Constreñida

5.1.2.1. Comparación de resultado

Por último en la ilustración 59, se observa una vista general entre los resultados obtenidos por los diferentes programas. Es necesario comentar que la diferencia visual existente entre el software implementado y el resto reside en dos componentes básica.

La primera de ellas consiste en la metodología utilizada para gestionar las Líneas de Rotura y los Polígonos Isla (ver apartado 3.2.3.2).

En segundo lugar, la implementación de la validación de triángulos que han sido generados o modificados después de la inserción de Líneas de Rotura no ha sido implementada en este momento.

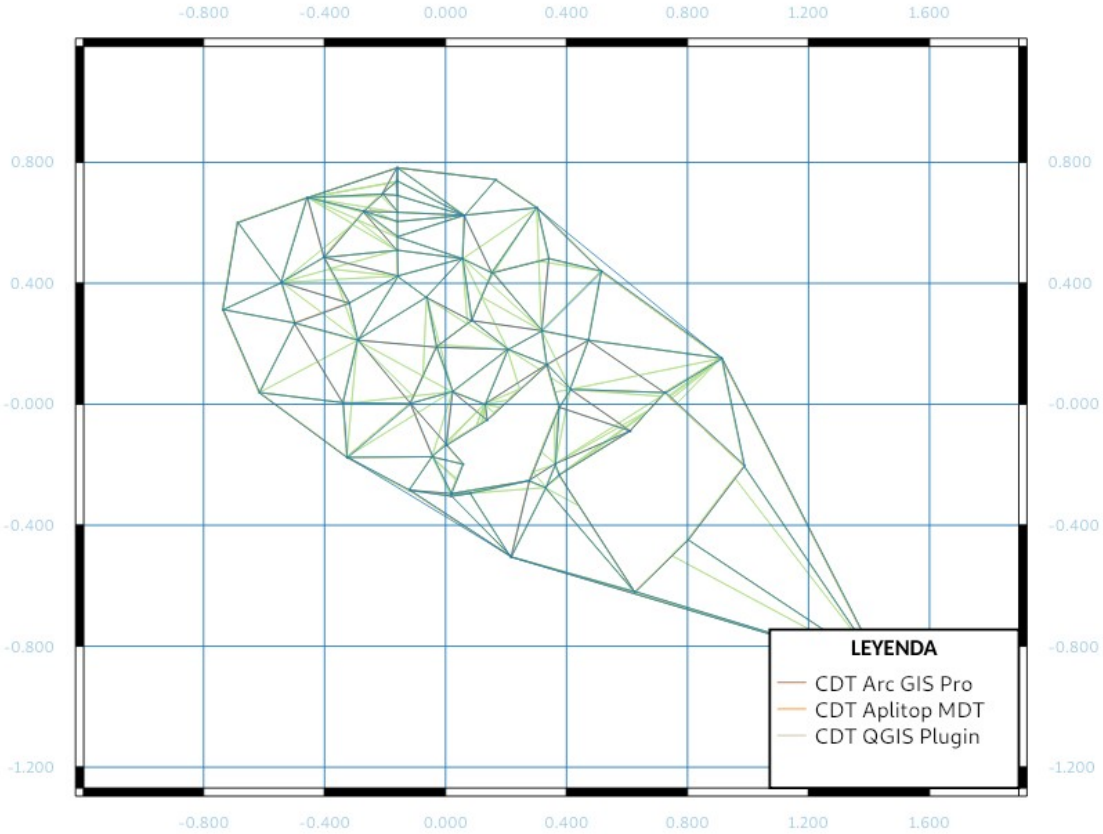


Ilustración 59: CDT construidos desde distintos Software para el CDG utilizado para la validación del Algoritmo.

5.2. Comparativa de Resultados

5.2.1. Descripción del Conjunto de Datos

Nombre	Mínimo Rectángulo Encuadrante (xMin, yMin, xMax, yMax)	N.º Puntos Relleno	N.º Lineas de Rotura	N.º Polígonos Isla	ANEXO 2
Las Huebras	540000.800, 4216001.599, 541998.989, 4217991.570	984	58	19	1
Alcala la Real	418002.720, 4146010.840, 419999.700, 4147998.760	845	91	108	2
Jabalcuz	426002.819, 4176003.600, 427999.679, 4177999.069	999	73	345	3
Moclín	430002.369, 4132005.299, 431999.500, 4133997.419	955	156	55	4
Río Quiebrajano	432001.400, 4176001.470, 433995.710, 4177999.060	999	98	929	5
Colomera	436009.140, 4138000.749, 437999.400, 4139986.649	999	31	20	6

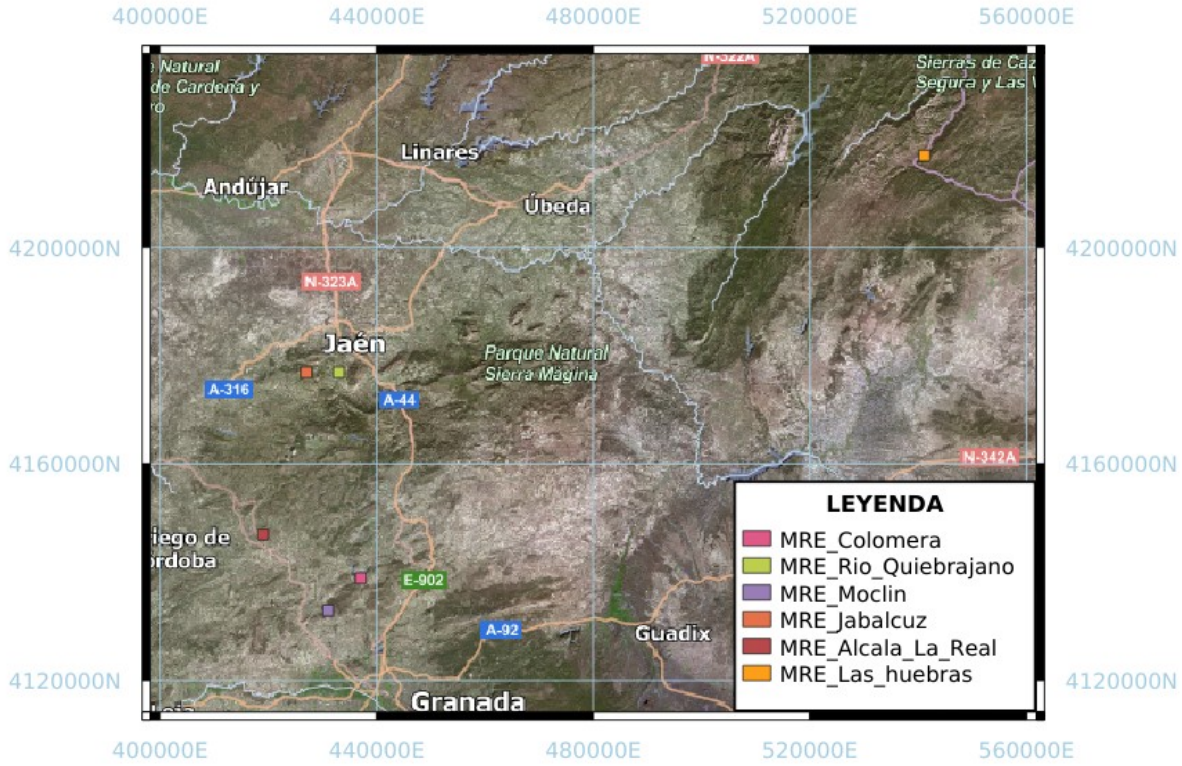


Ilustración 60: Mapa de Situación

5.2.2. Análisis de Resultados

5.2.2.1. Error De Validación De Triángulos En El Proceso De Inserción de Líneas de Rotura y Polígonos Isla

En el proceso de inserción de Líneas de Rotura y Polígonos Isla, no se validan los nuevos triángulos generados (no existe este proceso en el diagrama de la Ilustración 19). Es por ello que los triángulos astilla que son generados no son corregidos(ver Ilustración 61 para ejemplo en la inserción de Líneas de Rotura, ver Ilustración 62 para ejemplo en la inserción de Polígonos Isla).

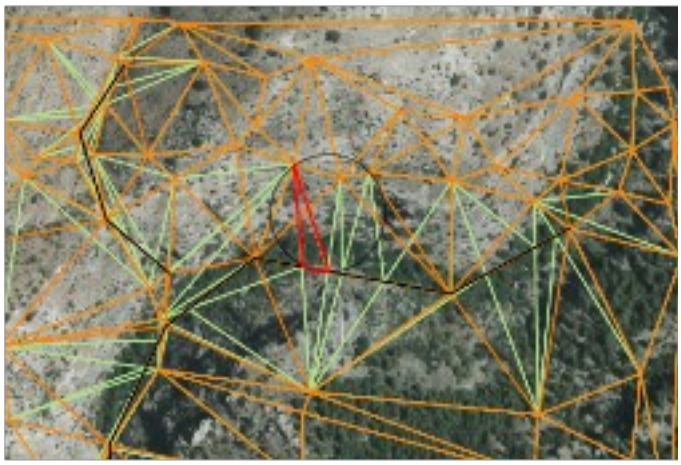


Ilustración 61: Triángulo generado en el proceso de inserción de Lineas de Rotura que no queda validado. Los segmentos negros se corresponden con Líneas de Rotura, la triangulación de color naranja es generada por el software Aplitop MDT. Los segmentos color verde se corresponden a la triangulación generada por el Plugin de QGIS, donde se ha superpuesto la capa debajo de la generada por Aplitop MDT y donde solo se pueden apreciar las diferencias.

Para futuras versiones, se incluirá el proceso de validación durante la inserción de Líneas de Rotura y de Polígonos Isla, como ya está incluido durante la inserción de Puntos de Relleno. Es por ello que los errores mostrados en las Ilustraciones 61 y 62 se produce en todos los ejemplos mostrados en este proyecto.

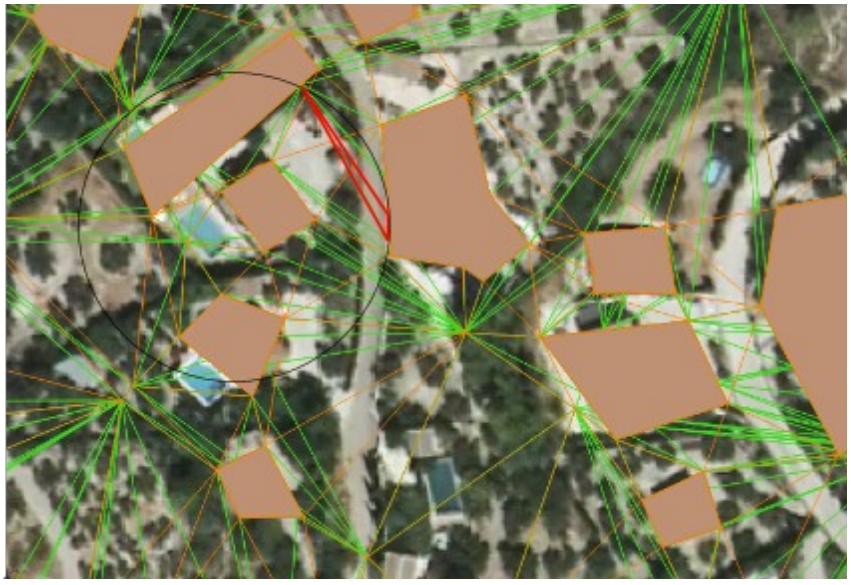


Ilustración 62: Triangulo generado después de la inserción de Polígonos Isla(rojo). En color naranja se muestra la triangulación generada por el software de Aplitop Mdt, en color verde, y superpuesta debajo de la triangulación generada por Aplitop, se muestra la triangulación generada por el Plugin de QGIS. En color marrón claro se muestran los polígonos Isla.

5.2.2.2. Distribución de los puntos de relleno con respecto a los vértices de Líneas de Rotura y Polígonos Isla

Al insertar los segmentos de Línea de Rotura y Polígonos Isla mediante la inserción de puntos ficticios (ver apartado 3.2.3.2), se generan una gran cantidad de polígonos astilla. El número de triángulos generados al usar esta metodología depende, entre otros factores, del número de puntos virtuales(intersecciones) generados, y estos dependen de la longitud de los segmentos de Líneas de Rotura y Polígonos Isla, y la distancia que hay de separación entre los Puntos de Relleno, de la densidad de Puntos de Relleno por metro cuadrado (ver apartado 5.2.1). En la Ilustración 63 se muestra los polígonos astilla generados para cada CDG, en las imágenes sobre Alcalá la Real, Jabalcuz y Río Quiebrajano se pueden observar mayor número de triángulos astilla, mientras que en las imágenes sobre Las Huebras y Moclín se muestran menos polígonos astilla.

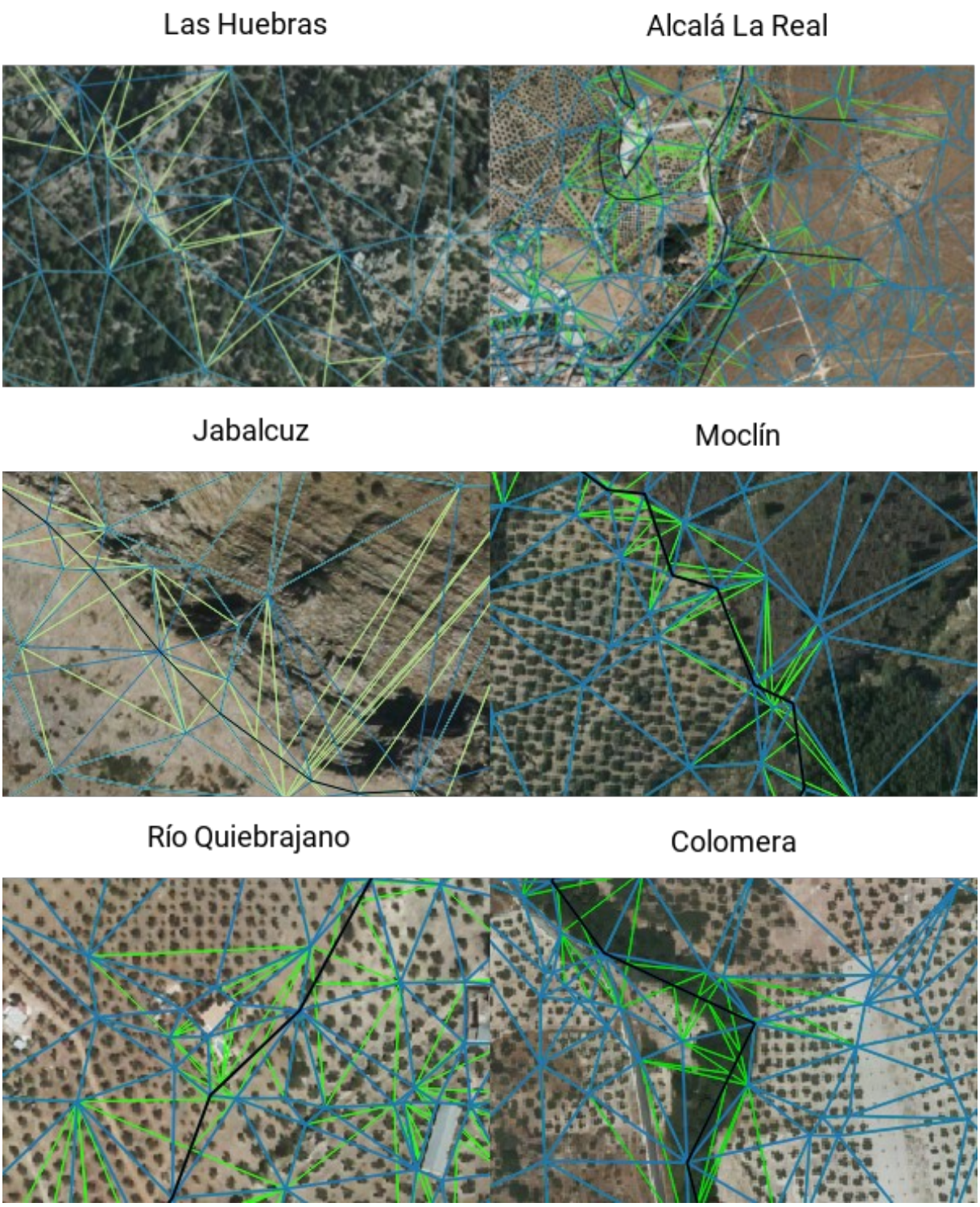


Ilustración 63: Ejemplos de Polígonos Astilla para diferentes CDGs. En azul se muestra las triangulaciones generadas por Arcgis Pro, y en verde y detras superpuesta, se muestran las triangulaciones generadas por el Plugin de QGIS. En negro se muestran las Lineas de Rotura.

En la Ilustración 64 se puede observar en 3D como se distribuyen los polígonos

astilla. Estos pertenecen a una Urbanización donde existe mayor densidad de Polígonos Isla por metro cuadrado. Se puede apreciar la mayor densidad de triángulos astillados en esta zona.

Ilustración 64: Sección de la imagen 3D del CDG del apartado 3 del Anexo 2. En ella se muestran la generación de polígonos astilla en la zona de la Urbanización de Jabalcuz.

5.2.2.3. Discrepancias En La Construcción De Triángulos En Los Bordes De La Triangulación

Existen discrepancias entre los bordes de triangulación generados por el software de Aplitop MDT y el Plugin de QGIS. Para solventar esta situación para el Plugin de QGIS, es necesario editar la triangulación resultado manualmente.

En la Ilustración 65 se pueden observar ejemplos de estas discrepancias en los CDGs pertenecientes a Moclín y Las Huebras (ver apartado 4 y 1 del Anexo 2). En la imagen izquierda del ejemplo 1 (triangulación de Arcgis Pro), y en la triangulación de color verde(Plugin de QGIS) de la imagen derecha del ejemplo 1 no se observan discrepancias. Mientras que en la imagen derecha del ejemplo 1, la triangulación naranja(Aplitop Mdt) difiere de la obtenida por el Plugin de QGIS. Esto es así debido a que el borde de la triangulación en Aplitop Mdt se puede obtener manualmente, antes de construir la triangulación.

Para el ejemplo 2 de la Ilustración 65, se muestra que no hay discrepancias entre los bordes generados por Aplitop Mdt y el Plugin de QGIS. Sin embargo, no es así para el borde generado por Arcgis Pro, el cual muestra un segmento que no es generado ni por Aplitop Mdt, ni por el Plugin de QGIS.

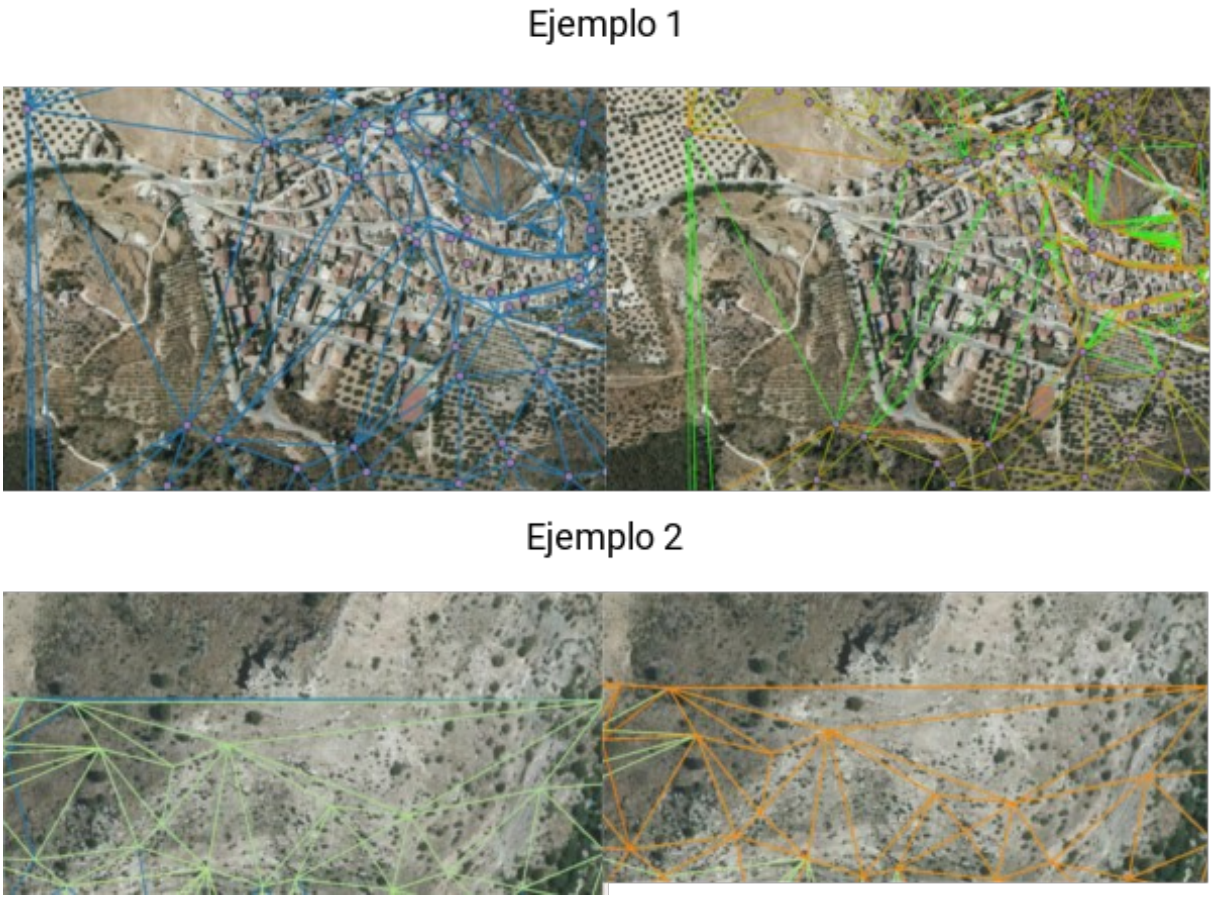


Ilustración 65: Ejemplos sobre el borde de sus triangulaciones. En azul se muestra los segmentos que pertenecen a la triangulación generada por Arcgis Pro. En naranja pertenecen a la triangulación generada por Aplitop Mdt. En verde pertenecen a la triangulación generada por el Plugin de QGIS. Las triangulaciones aparecen superpuestas

5.3. Tiempos de Ejecución

Para la elaboración de la siguiente tabla, se ha utilizado un PC con 8GB de RAM DDR2, procesador Intel QuadCore Q8400 2.66GHz. Teniendo en cuenta que estos tiempos se corresponden a la implementación del Plugin de QGIS, y obviando la comparación con ArcGIS Pro y Aplitop MDT, que muestran un tiempo de ejecución claramente inferior, se arrojan los siguiente resultados.

Ejemplo	Tiempo ejecución (s)
Ejemplo1	5,2
Las Huebras	303,8
Alcalá la Real	368,7
Jabalcuz	436,9
Moclín	351,8
Río Quiebrajano	375,4
Colomera	350,5

6. CONCLUSIONES

Para finalizar, se presenta un conjunto de ideas generales con respecto a los aspectos más relevantes, y los problemas afrontados.

6.1. Comparativa de resultados

Como se puede observar en las ilustraciones 56 y 55 el resultado difiere en como se construye los bordes en la triangulación. En el caso de ArcGis Pro, se utiliza el mismo resultado que produciría procesar la envolvente convexa del CDG correspondiente a los puntos de relleno. Para el caso de nuestra implementación, el borde ha sido construido de forma incremental, y por ello no se corresponde con la envolvente convexa, generando así menor número de polígonos astilla.

Además, se puede observar fácilmente como difieren en el Plugin de QGIS con respecto a ArcGis Pro y Aplitop MDT en la construcción de los triángulos alrededor de las LR y las Islas. Esto se debe principalmente a que no han sido validados los triángulos generados por los procesos de inserción de las LR y las Islas.

Por otro lado, la implementación de este proyecto ha procesado las líneas de rotura y polígonos isla, generando multitud de puntos virtuales y triángulos pequeños

alrededor de los segmentos de línea de rotura y polígonos isla. Aunque el aspecto no es muy homogéneo considero que respeta la geometría que se ha querido representar utilizando las líneas de rotura.

6.2. Objetivos

Acorde con el apartado 1.1 y con las especificaciones establecidas en el apartado 3.1, el complemento desarrollado en este proyecto cumple las especificaciones establecidas.

Como proyecto de softaware, el estado del complemento queda como versión Alfa si se desea su consumo como producto comercial. A lo largo del desarrollo de este proyecto diferentes dificultades han sido encontradas para ser llevado a cabo. Por ejemplo, diferentes situaciones geométricas con las que no se contaban en el inicio del diseño de la implementación, uso de la programación orientada a objetos en el momento que se considera que la implementación lo requiere para un mejor desarrollo, errores encontrados en la librería GDAL; siendo necesario la implementación de soluciones para estos errores. Estos problemas han repercutido en el tiempo de su desarrollo, por lo que ha sido muy complejo llevar a cabo una planificación para su resolución.

6.3. Mejoras Futuras

Para futuras versiones del algoritmo a medio/largo plazo, la mejoras sugeridas principalmente consisten en:

- Verificación de los datos de entrada, para que solo exista la posibilidad de procesar datos para los cuales esta implementación ha sido realizada.
- Implementación de un índice espacial para la búsqueda de triángulos más eficiente. Esto equivaldría en un mejora sustancial en los tiempos de ejecución.

- Implementación de la validación de triángulos generados en el proceso de inserción de Líneas de Rotura o Polígonos Isla.
- Eliminar la condición para las Líneas de Rotura y Polígonos Isla que sus vértices no pueden pertenecer a los Puntos de Relleno.
- Implementación de la metodología expuesta en el apartado 3.2.3.3 como alternativa para la inserción de Lineas de Rotura y Polígonos Isla.
- Implementación de la alternativa expuesta en el apartado 3.2.2.3 para la obtención de la lista de comprobación en el proceso de validación de triángulos.

7. BIBLIOGRAFÍA

Bærentzen, J. A., Gravesen, J., Anton, F., & Aanæs, H. (2012). *Guide to Computacional Geometry Processing*. London: Springer.

Biniaz, A., & Dastghaibyfard, G. (2012). A faster circle-sweep Delaunay triangulation algorithm. *Advances in Engineering Software*, *43* (1), 1–13. http://dx.doi.org/10.1016/j.advengsoft.2011.09.003

Biniaz, A., & Dastghaibyfard, G. (n.d.). *A Comparison of Plane Sweep Delaunay Triangulation Algorithms*.

de Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. (2008). *Computacional Geometry. Algorithms and Applications.* (3rd ed.). Heidelberg: Springer.

Fortune, F. (1987). A Sweepline Algorithm for Voronoi Diagrams. *Algorithmica*, *2*, 153–174.

Fundamentals of TIN triangulation in ArcGIS. (n.d.). Retrieved from desktop.arcgis.com/en/arcmap/10.3/manage-data/tin/fundamentals-of-tin-triangulation.htm

Guibas, L., & Stolfi, J. (1985). Primitives for the Manipulation of General Subdivisions and

the Computation of Voronoi Diagrams. *ACM Trans. Graphics*, *4*, 75–123.

Lee, D. T., & Schachter, B. J. (1980). *Two Algorithms for Constructing a Delaunay Triangulation* (Vol. 9).

QGIS API Documentation. (2019). Retrieved from https://qgis.org/api/3.4/classQgisInterface.html

Strobl, C. (2008). Dimensionally Extended Nine Intersection Model (DE-9IM). In *Encyclopedia of GIS* (pp. 240–245). https://doi.org/10.1007/978-0-387-35973-1_298

Su, P., & Drysdale, R. L. S. (1997). A comparison of sequential Delaunay triangulation algorithms. *ACM Symposium on Computational Geometry*, *7* (5–6), 361–385. https://doi.org/10.1016/S0925-7721 (96)00025-9

Vigo, M. (1997). *An Improved Incremental Algorithm For Constructing Restricted Delaunay Triangulations. 21*, 215–223.

Wang, Y., Wu, L., & Shi, W. (2007). Constrained edge dynamic deleting in CD-TIN based on influence domain retriangulation of virtual point. *Geo-Spatial Information Science*, *10* (3), 208–212. https://doi.org/10.1007/s11806-007-0065-5

Zalik, B. (2005). An efficient sweep-line Delaunay triangulation algorithm. *Computer-Aide Design*, *37*, 1027–1038.

ANEXO 1

1. PSEUDOCÓDIGO

1.1. Determinación de Tangentes Inferiores y Superiores

```
FUNCIÓN Determinacion_Tangentes
DATOS DE ENTRADA
      V_L TIPO Envolvente Convexa
      V_R TIPO Envolvente Convexa
INICIO
      ‘’’
      l (X, Y) denota la línea dirigida desde el punto X al punto Y.
      Para cada envolvente convexa CH (S), se mantienen dos puntos LM (S) y
      RM (S),     estos puntos se corresponden con el puntos más a la
      izquierda y a la  derecha de S, respectivamente.
      ‘’’
A:
      X ← RM (V_L); Y ← LM (V_R);
      Z ← FIRST (Y); Z’ ← FIRST (X); Z” ← PRED (X, Z’)
      SI (Z esta-a-la-derecha-de l (X,Y)) HACER
            Z ← SUCC (Z, Y)
            Y ← Z
      SI_NO
            SI (Z” esta-a-la-derecha-de l (X,Y)
                  Z” ← PRED (Z”, X)
                  X ← Z”
            SI_NO
                  DEVOLVER (X, Y)
            FIN_SI
      FIN_SI
      VUELVE A A:
FIN
```

1.2. Unión de dos triangulaciones

```
FUNCIÓN Unir
DATOS DE ENTRADA
      tangente_superior_comun TIPO Segmento
      tangente_inferior_comun TIPO Segmento
INICIO
      BT ← tangente_inferior_comun
      UP ← tangente_superior_comun
      L ← punto izquierdo de BT
      R ← punto derecho de BT
      MIENTRAS BT no es igual a UT HACER
            A ← B ← falso
            Insertar (L, R)
            R_1← PRED (R, L)
            SI R_1 esta-a-la-izquierda-de l (L, R) HACER
                  R_2 ← PRED (R, R_1)
                  MIENTRAS Qtest (R_1, L, R, R_2) HACER
                        Borrar (R, R_1)
```

```
                  R1 ← R2
                  R2 ← PRED (R, R1)
            FIN_MIENTRAS
      SI_NO
            A ← Verdadero
      FIN_SI
      L1 ← SUCC (L, R)
      SI L1 esta-a-la-derecha-de l (R, L) HACER
            L2 ← SUCC (L, L1)
            MIENTRAS Qtest (L, R, L1, L2) HACER
                  Borrar (L, L1)
                  L1 ← L2
                  L2 ← SUCC (L, L1)
            FIN_MIENTRAS
      SI_NO
            B ← Verdadero
      FIN_SI
      SI A HACER
            L ← L1
      SI_NO
            SI B HACER
                  R ← R1
            SI_NO
                  SI Qtest (L, R, R1, L1) HACER
                        R ← R1
                  SI_NO
                        L ← L1
                  FIN_SI
            FIN_SI
      FIN_SI
      BT ← l (L, R)
FIN_MIENTRAS
FIN
```

1.3. Inicialización de la triangulación

```
PROCEDIMIENTO calcular_triangulo_ficticio
DATOS DE ENTRADA
      capa_puntos TIPO ogr.layer
INICIO
      extension = capa_puntos.GetExtent ()
      xmin = extension[0]
      xmax = extension[1]
      ymin = extension[2]
      ymax = extension[3]

      incremento_x = 2* (xmax – xmin)
      incremento_y = 2* (ymax – ymin)

      xmin = xmin - incremento_x
      ymin = ymin – incremento_y
      xmax = xmax + incremento_x
      ymax = ymax + incremento_y

      punto_a = punto (xmin, ymin, 0)
      punto_b = punto (xmin + ( (xmax-xmin)*2), ymax, 0)
      punto_c = punto (xmax, ymax, 0)
```

```
      segmento_a = segmento (punto_a, punto_b, None, None, None)
      segmento_b = segmento (punto_b, punto_c, None, None, None)
      segmento_c = segmento (punto_c, punto_a, None, None, None)
      t1 = construir_triangulo (“1”, segmento_a, segmento_b, segmento_c)
FIN
```

1.4. Búsqueda del triángulo donde cae un punto

```
FUNCION buscar_triangulo
DATOS DE ENTRADA
      p TIPO ogr.Geometry ()
INICIO
      triangulo_tocado
      PARA triangulo en conjunto_triángulos HACER
            SI punto.Dentro (triangulo) ENTONCES
                  triangulo_tocado.añadir (triangulo)
            SI_NO
                  SI punto.Toca (triangulo) ENTONCES
                        triangulo_tocado.añadir (triangulo)
                  FIN_SI
            FIN_SI
      FIN_PARA

      SI longitud (triangulo_tocado) == 1 ENTONCES
            triangulo_tocado.añadir (None)
            DEVOLVER triangulo_tocado
      SI_NO
            DEVOLVER triangulo_tocado
      FIN_SI
FIN
```

1.5. Insertar punto dentro de triángulos

```
FUNCION insertar_punto_dentro
DATOS ENTRADA
      p TIPO ogr.wkbpoint
      t TIPO string
      max_id TIPO string
INICIO
      ta_id,tb_id,tc_id = max_id (+1, +2, +3)

      # Construir triangulo a
      segmento_a_ta = triangulo (t).get_segment_a ()
      punto_inicio = segmento_a_ta.get_end_point ()
      segmento_b_ta = segmento (punto_inicio, p, tb_id, None, None)
      punto_final = segmento_a_ta.get_start_point ()
      segmento_c_ta = segmento (p, punto_final, tc_id, None, None)
      triangulo (‘ta_id’) = [ta_id, segmento_a_ta, segmento_b_ta,
      segmento_c_ta)
      actualizar_triangulo_adyacente (t, ta_id, segmento_a_ta)

      # Construir triangulo b
      segmento_a_tb = triangulo (t).get_segment_b ()
      punto_inicio = segmento_a_tb.get_end_point ()
      segmento_b_tb = segmento (punto_inicio, p, tc_id, None, None)
      punto_final = segmento_a_tb.get_start_point ()
```

```
      segmento_c_tb = segmento (p, punto_final, ta_id, None, None)
      triangulo (‘tb_id’) = [tb_id, segmento_a_tb, segmento_b_tb,
      segmento_c_tb)
      actualizar_triangulo_adyacente (t, tb_id, segmento_a_tb)

      # Construir triangulo c
      segmento_a_tc = triangulo (t).get_segment_c ()
      punto_inicio = segmento_a_tc.get_end_point ()
      segmento_b_tc = segmento (punto_inicio, p, ta_id, None, None)
      punto_final = segmento_a_tc.get_start_point ()
      segmento_c_tc = segmento (p, punto_final, tb_id, None, None)
      triangulo (‘tc_id’) = [tc_id, segmento_a_tc, segmento_b_tc,
      segmento_c_tc)
      actualizar_triangulo_adyacente (t, tc_id, segmento_a_tc)

      # Eliminar triangulo original
      borrar (t)
      DEVOLVER [ta_id, tb_id, tc_id]
FIN
```

1.6. Insertar punto encima de un segmento de triángulos

```
FUNCION insertar_punto_toca
DATOS DE ENTRADA
      p TIPO ogr.wkbPoint
      t1 TIPO string
      t2 TIPO string
      max_id TIPO string
INICIO
      # Establecer segmentos comunes
      segmentos_comunes = obtener_segmentos_comunes (t1, t2)
      segmento_comun_t1 = segmentos_comunes[0]
      segmento_comun_t2 = segmentos_comunes[0]

      # Establecer t1
      segmento_b_t1 = triangulo[t1].next_segment (segmento_comun_t1)
      segmento_c_t1 = triangulo[t1].next_segment (segmento_b_t1)

      # Establecer t2
      segmento_b_t2 = triangulo[t2].next_segment (segmento_comun_t2)
      segmento_c_t2 = triangulo[t2].next_segment (segmento_b_t2)

      # Crear nuevos identificadores
      ta_id, tb_id, tc_id, td_id = max_id (+1, +2, +3, +4)

      # Construir triangulo a
      segmento_a_ta = segmento_b_t1
      punto_final = segmento_b_t1.get_end_point ()
      segmento_b_ta = segmento (punto_final, p, tb_id, None, None)
      punto_inicial = segmento_b_t1.get_start_point ()
      segmento_c_ta = segmento (p, punto_inicial, td_id, None, None)
      triangulo[ta_id] = triangulo (ta_id, segmento_a_ta, segmento_b_ta,
      segmento_c_ta)
      actualizar_triangulo_adyacente (t1, ta_id, segmento_a_ta)

      # Construir triangulo b
      segmento_a_tb = segmento_c_t1
```

```
      punto_final = segmento_c_t1.get_end_point ()
      segmento_b_tb = segmento (punto_final, p, tc_id, None, None)
      punto_inicial = segmento_c_t1.get_start_point ()
      segmento_c_tb = segmento (p, punto_inicial, ta_id, None, None)
      triangulo[tb_id] = triangulo (tb_id, segmento_a_tb, segmento_b_tb,
      segmento_c_tb)
      actualizar_triangulo_adyacente (t1, tb_id, segmento_a_tb)

      # Construir triangulo c
      segmento_a_tc = segmento_b_t2
      punto_final = segmento_b_t2.get_end_point ()
      segmento_b_tc = segmento (punto_final, p, td_id, None, None)
      punto_inicial = segmento_b_t2.get_start_point ()
      segmento_c_tc = segmento (p, punto_inicial, tc_id, None, None)
      triangulo[tc_id] = triangulo (tc_id, segmento_a_tc, segmento_b_tc,
      segmento_c_tc)
      actualizar_triangulo_adyacente (t2, tc_id, segmento_a_tc)

      # Construir triangulo d
      segmento_a_td = segmento_c_t2
      punto_final = segmento_c_t2.get_end_point ()
      segmento_b_td = segmento (punto_final, p, ta_id, None, None)
      punto_inicial = segmento_c_t2.get_start_point ()
      segmento_c_td = segmento (p, punto_inicial, tc_id, None, None)
      triangulo[td_id] = triangulo (td_id, segmento_a_td, segmento_b_td,
      segmento_d_td)
      actualizar_triangulo_adyacente (t2, td_id, segmento_a_td)

      # Eliminar triángulos originales
      borrar (t1, t2)
      DEVOLVER [ta_id, tb_id, tc_id, td_id]
FIN
```

1.7. Creación de la Lista de Validación

```
conjunto_triángulos = triángulos_creados
PARA iteración en nivel de adyacencia HACER:
      PARA triangulo en conjunto_triángulos:
            obtener triángulos_adyacentes
      FIN_PARA
      unir conjunto_triángulos y triángulos_adyacentes
FIN_PARA
```

1.8. Proceso de Validación de la Lista de Validación

Se corresponde con el método check_list de la clase triangulation (véase Ilustración 33).

```
PROCEDIMIENTO lista_comprobación
DATOS ENTRADA
      lista_validacion TIPO lista
INICIO
      convergencia = True # Se considera que hay convergencia por defecto
      limite_repeticion = 2 * nivel de adyacencia
      MIENTRAS longitud (lista_validacion) > 0 Y convergencia es True
HACER:
```

```
            Obtenemos un triangulo de lista_validacion
            Generamos su circulo circunscrito
            Buscamos vértices de la lista de validacion que estén dentro
            del circulo
            Seleccionamos el punto que este mas cerca al triangulo base
            # Cada punto genera una tupla, cuyos atributos son geometría
            # del punto,
            # un punto del triangulo base, y distancia entre ambos
            SI hay intersecciones ENTONCES
                  voltear los triángulos
                  repeticion_triangulo_volteados + 1
                  insertar triángulos volteados en lista de validacion
            SI_NO
                  No hacer nada
            FIN_SI
            SI repeticion algún triangulo > limite_repeticion ENTONCES
                  convergencia es falso
            FIN_SI
      FIN_MIENTRAS
FIN
```

1.9. Volteo de dos triángulos adyacentes

```
PROCEDIMIENTO flip
DATOS ENTRADA triangulo1, triangulo2
INICIO
      Obtener segmentos_comunes (lc)
      # Construir triangulo1
      segment_b_t1 = t1.next_segment (lc_t1)
      segment_c_t1 = t1.next_segment (segment_b_t1)
      # Construir triangul2
      segment_b_t2 = t2.next_segment (lc_t2)
      segment_c_t2 = t2.next_segment (segment_b_t2)

      SI comprobar_flip_posible (t1, t2) is True ENTONCES
            # Construir triangulo1 volteado
            seg_a_ta = segment (segment_b_t2.obtener_punto_final (),
                                segment_c_t1.obtener_punto_inicial (),
                                lc_t1.obtener_triangulo_izquierda (),
                                None,
                                None)
            seg_b_ta = segment_c_t1
            seg_c_ta = segment_b_t2
            ta = triangle (t1,
                           seg_a_ta,
                           seg_b_ta,
                           seg_c_ta)
            # Construir triangulo2 volteado
            seg_a_tb = segment (segment_b_t1.obtener_punto_final (),
                                segment_c_t2.obtener_punto_inicial (),
                                lc_t2.obtener_triangulo_izquierda (),
                                None,
                                None)
            seg_b_tb = segment_c_t2
            seg_c_tb = segment_b_t1
            tb = triangle (t2,
                           seg_a_tb,
```

```
                        seg_b_tb,
                        seg_c_tb)
            actualizar_triangulo_adyacente (t1, t2, segment_b_t1)
            actualizar_triangulo_adyacente (t2, t1, segment_b_t2)
            DEVUELVE True
      SI_NO
            DEVUELVE False
      FIN_SI
FIN
```

1.10. Volteo concatenado

```
PROCEDIMIENTO volteo_concatenado
DATOS DE ENTRADA
      segmento_interseccion TIPO ogr.wkbLineString
      lista_triángulos TIPO lista
      punto_mas_cercano TIPO ogr.wkbPoint
      triangulo_base TIPO String
INICIO
      # Obtener los segmentos de los triángulos intersectados
      lista_segmentos = []
      PARA triangulo en lista_triángulos HACER
            lista_segmentos.añadir (triangulo.get_segmento_a)
            lista_segmentos.añadir (triangulo.get_segmento_b)
            lista_segmentos.añadir (triangulo.get_segmento_c)
      FIN_PARA
      # Obtener los segmentos que cruzan a segmento_interseccion
      segmento_tupla = list ()
      PARA seg en lista_segmentos HACER
            punto_inicio = seg.get_start_point ()
            punto_final = seg.get_end_point ()
            es_linea_rotura = seg.get_breakline ()
            es_isla = seg.get_hull ()
            SI ( segmento_interseccion.Cruza (seg)
                 Y NO segmento_interseccion.Toca (seg)
                 Y es_linea == None
                 Y es_isla == None
                 Y punto_mas_cercano != punto_inicio
                 Y punto_mas_cercano != punto_final ) HACER
                  punto_interseccion =
                        segmento_interseccion.Intersectar (seg)
                  distancia =
                        punto_mas_cercano.Distancia (punto_interseccion)
                  tupla = (seg, distancia)
                  segmento_tupla.añadir (tupla)
            FIN_SI
      FIN_PARA

      # Voltea los triángulos cruzados, y para cuando triangulo_base es
      volteados
      t_base_volteado = False
      MIENTRAS t_base_volteado es False HACER
            t1_tupla = min (segmento_tuplas, key=lambda item: item[1])
            segmento_tupla.pop (segmento_tupla.indice (t1_tupla))
            t2_tupla = min (segmento_tuplas, key=lambda item: item[1])
            segmento_tupla.pop (segmento_tupla.indice (t2_tupla))
            t1_ = t1_tuple[0].get_left_triangle ()
```

```
            t2_ = t2_tuple[0].get_left_triangle ()
            flip (t1_, t2)
            SI t1_ == triangulo_base OR t2_ == triangulo_base HACER
                  t_base_volteado = True
            FIN_SI
      FIN_MIENTRAS
FIN
```

1.11. Inserción de Líneas de Rotura

```
PROCEDIMIENTO insertar_linea_rotura
DATOS ENTRADA
      linea_rotura TIPO ogr.Geometry
INICIO
      vértices = linea_rotura.GetPoints ()
      # Insertar el primer punto en la triangulación
      punto = vertices.pop ()
      triangulo_contenedor = buscar_triangulo (punto)
      tri_ins = insertar_punto (punto)
      geometria_anterior = punto
      vertice_seg_lr = punto
      # Insertar el resto de puntos en la triangulación
      PARA p en vértices HACER
            punto = p
            segmento_virtual = geometria (vertice_seg_lr, punto)
            p_virtual_insertado = False
            MIENTRAS p_virtual_insertado == False HACER
                  segmento_interseccion = geometria (geometria_anterior,
                                                     punto)
                  # Ver si segmento_virtual intersecta algún triangulo. Si
                  # no intersecta, inserta el vértice en la triangulación.
                  # Si intersecta inserta de forma cíclica puntos
                  # virtuales.
                  p_virt_encontrado = False
                  MIENTRAS (p_virt_encontrado == False
                            AND i < longitud (triángulos_insertados) HACER
                        triangulo = triángulos_insertados[i]
                        seg_tri = triangulo.get_segments
                        PARA seg en seg_tri HACER
                              interseccion =
                              segmento_interseccion.Intersectar (seg)
                              SI ( interseccion.tipo () == “Punto”
                                   Y NO interseccion != geometria_anterior
                                   Y NO interseccion != punto) ENTONCES
                                    v_p = interpolar_z (interseccion,
                                                        segmento_virtual)
                                    tri_ins = insertar_punto (v_p
                                                              tri_ins[i]
                                                        seg.obtener_tri_izq
()
                                                              max_id)
                                    set_max_id (tri_ins[3])
                                    # Obtener los triángulos implicados
                                    tri_imp = list ()
                                    PARA (c=0; 4; c+) HACER
                                          tri = tri_ins[c]
                                          ver = tri.get_vértices ()
```

```
                                        SI (geometria_anterior en
                                            vértices
                                            Y v_p en vértices) ENTONCES
                                              tri_imp.añadir (tri_ins[c])
                                        FIN_SI
                                  FIN_PARA
                                  # Preparar variables para el siguiente
                                  # bucle
                                  actualizar_segmento_bk (tri_imp[0],
                                                          tri_imp[1],
                                                          bk_id)
                                  geometria_anterior = v_p
                                  p_virt_encontrado = True
                                  break_FIN_PARA
                            FIN_SI
                      FIN_PARA
                      i += i
                FIN_MIENTRAS
                SI p_virt_encontrado is False ENTONCES
                      p_virtual_insertado = TRUE
                FIN_SI
          FIN_MIENTRAS
          # Se han insertado los puntos virtual, pero queda un segmento
          # por insertar
          tri_imp = []
          PARA tri en tri_ins HACER
                SI punto.Dentro (tri) ENTONCES
                      tri_imp.añadir (tri)
                FIN_SI
          FIN_PARA
          tri_ins = insertar_punto (punto,
                                    tri_imp[0],
                                    max_id)
          set_max_id (tri_ins[2])
          # Obtener triángulos implicados
          tri_imp = []
          PARA tri en tri_ins HACER
                vértices = tri.get_vértices ()
                SI geometria_anterior en vértices Y punto en vértices
                ENTONCES
                      tri_imp.añadir (tri)
                FIN_SI
          FIN_PARA
          actualizar_segmento_bk (tri_imp[0], tri_imp[1], bk_id)
          # Actualizar geometria anteriores
          geometria_anterior = punto
    FIN_PARA
FIN
```

1.12. Inserción de Polígonos Isla

```
PROCEDIMIENTO insertar_islas
DATOS DE ENTRADA
      isla TIPO ogr.Geometry
INICIO
      anillo = isla.Obtener_Geometria ()
      vértices_isla = []
```

```
      PARA v = 0; anillo.GetPointCount (); i++ HACER
            vértices_isla.añadir (anillo.GetPoint (v))
      FIN_PARA
      isla_id = obtener_max_id_isla () + 1
      establecer_max_id_isla (isla_id)
      # Insertar primer punto en la triangulación
      ver_isla = vértices_isla.pop (0)
      tri_imp = buscar_triangulo (ver_isla)
      tri_ins = insertar_punto (ver_isla, tri_imp, max_id)
      p_anterior = ver_isla
      PARA punto en ver_isla HACER
            ver_isla = punto
            seg_virt = segmento (p_anterior, ver_isla)
            es_punto_virt_insertado = False
            MIENTRAS es_punto_virt_insertado == False HACER
                  seg_interseccion = segmento (p_anterior, ver_isla)
                  p_virt_encontrado = False
                  i = 0
                  MIENTRAS (p_virt_encontrado == False Y i <
                              longitud (tri_ins) HACER
                        t = tri_ins[i]
                        segmentos_t = t.obtener_segmentos ()
                        PARA seg en segmentos_t HACER
                              interseccion =
                              seg_interseccion.Intersectar (seg)
                              SI (tipo_geom (interseccion) == "PUNTO"
                                  Y NO interseccion == p_anterior
                                  Y NO interseccion == ver_isla) ENTONCES
                                    v_p = interpolar_z (interseccion,
                                                        seg_virt)
                                    tri_ins = insertar_punto (v_p,
                                                              tri_ins[i],
                                                        seg.obtener_tri_izq
()),
                                                              max_id ())
                                    establecer_max_id (tri_ins[3])
                                    tri_imp = []
                                    PARA e = 0; e > 4; e++ HACER
                                          t_i = triangulo (tri_ins (e))
                                          vértices = t_i.obtener_vértices
()
                                          SI (p_anterior en vértices
                                              Y v_p en vértices) HACER
                                                tri_imp.añadir (tri_ins[e])
                                          FIN_SI
                                    FIN_PARA
                                    p_anterior = v_p
                                    p_virt_encontrado = True
                                    BREAK_FIN_PARA
                              FIN_SI
                        i += 1
                        FIN_PARA
                  FIN_MIENTRAS
                  SI p_virt_encontrado NO es True ENTONCES
                        es_punto_virt_insertado = True
                  FIN_SI
```

```
            FIN_MIENTRAS
            # Despues de insertar los puntos virtuales, queda insertar el
            # vertice de isla.
            tri_imp = []
            PARA t en tri_ins HACER
                  SI punto.Dentro (t) ENTONCES
                        tri_imp.añadir (t)
                  FIN_SI
            FIN_PARA
            SI primer_p != punto ENTONCES
                  tri_ins = insertar_punto (punto,
                                            tri_imp[0],
                                            tri_imp[1],
                                            max_id)
            FIN_SI
            tri_imp = []
            PARA t en tri_insHACER
                  vértices = t.obtener_vértices ()
                  SI p_anterior en vértices Y punto en vértices ENTONCES
                        tri_imp.añadir (t)
                  FIN_SI
            FIN_PARA
            actualizar_segmento_isla (tri_imp[0], tri_imp[1], max_id_isla)
            p_anterior = ver_isla
      FIN_PARA
FIN
```

1.13. Conmutar visibilidad de triangulos isla

```
PROCEDIMIENTO conmutar_visibilidad_triángulos_isla
DATOS DE ENTRADA
      # En esta ocasión, se modifican los atributos de los objetos, no
      # necesita que
      # ninguna variable sea pasada como parámetro
INICIO
      conjunto_temporal = set ()
      conjunto_global = set ()
      PARA i en islas HACER
            g = obtener_geometria_isla (i)
            area_g = g.area ()
            segmento_isla = i.obtener_segmento ()
            t = segmento_isla[0]
            union_geometrias = g.Union (t)
            SI union_geometrias.area () > area_g ENTONCES
                  # El triangulo esta fuera de la isla
                  conjunto_temporal.añadir (segmento_isla[2])
            SI_NO
                  # El triangulo esta dentro de la isla
                  conjunto_temporal.añadir (segmento_isla[0])
            FIN_SI
            conjunto_global = conjunto_temporal.union (conjunto_global)
            MIENTRAS longitud (conjunto_temporal)!= 0 HACER
                  conjunto_temporal2 = set ()
                  PARA i en conjunto_temporal HACER
                        tri_adyacente = obtener_triángulos_adyacentes (i)
                        conjunto_temporal2 =
                        conjunto_temporal2.union (tri_adyacente)
```

```
                  FIN_PARA
                  conjunto_temporal = conjunto_temporal2
                  # Obtener solo los nuevos triángulos
                  conjunto_temporal = conjunto_temporal – cojunto_global
                  # Obtener solo los que se encuentran dentro de la isla
                  conjunto_temporal2 = set ()
                  PARA i en conjunto_temporal HACER
                        union_geometria = i.Union (g)
                        SI union_geometria.area () < area_g ENTONCES
                              conjunto_temporal2.añadir (i)
                        FIN_SI
                  FIN_PARA
                  conjunto_temporal = conjunto_temporal2
                  conjunto_global =
                  conjunto_global.union (conjunto_temporal)
            FIN_MIENTRAS
            # Cambiar estado visibilidad
            PARA i en conjunto_global HACER
                  SI i.obtener_visibilidad () == True ENTONCES
                        i.cambiar_visibilidad ()
                        triangles[i_id] = i
                  FIN_SI
            FIN_PARA
      FIN_PARA
FIN
```

1.14. Inserción de un segmento de CDT

```
PROCEDIMIENTO AñadirBordeCDT
DATOS DE ENTRADA
T: CDT
ab: Segmento)
INICIO
      Encuentra el triangulo t que pertenece a T que contiene a y es
      cortado por ab
      PU := ListaVacia
      PL := ListaVacia
      v := a
      MIENTRAS b no esta en t HACER
            tseg := TrianguloOpuesto (t, v)
            vseg := VerticeOpuesto (tseg, t)
            SI vseg sobre el segmento ab ENTONCES
                  AñadirLista (PU, vseg)
                  v := Vertice compartido por t y tseg sobre ab
            SI_NO
                  AñadirLista (PL, vseg)
                  v := Vertice compartido por t y tseg debajo de ab
            FIN_SI
            Borrar t de T
            t := tseg
      FIN_MIENTRAS
      TriangularPseudopoligonoDelaunay (PU, ab, T)
      TriangularPseudopoligonoDelaunay (PL, ab, T)
      Reconstituir los triángulos adyacentes de T
      Añadir segmento ab a T
      Seleccionar segmento ab de T como procesado
```

```
FIN
```

1.15. Triangular Pseudopoligono de Delaunay

```
PROCEDIMIENTO TriangularPseudopoligonoDelaunay
DATOS ENTRADA
      P:ListaVertices
      ab:segmento
      T:CDT
INICIO
      SI P tiene mas de un elemento ENTONCES
            c := Primer vertice de P
            PARA cada vertice v que pertenece a P HACER
                  SI v pertenece a CirculoCircunscrito (a,b,c) ENTONCES
                        c := v
                  FIN_SI
            FIN_PARA
            SI P no esta vacio ENTONCES
                  Añade triangulo con vertices a, b, c a T
            FIN_SI
FIN
```

1.16. Añadir Punto a CDT

```
PROCEDIMIENTO AñadirPuntoCDT
DATOS ENTRADA
      stack := StackVacio
      Encuentra triangulo t que pertenece a T que contiene a p
      Divide t en tres triangulos, t1, t2 y t3, dependiendo de p
      Insertar (stack, t1)
      Insertar (stack, t2)
      Insertar (stack, t3)
      MIENTRAS stack no esta vacio HACER
            t := Sacar (stack)
            topo := TrianguloOpuesto (t, p)
            SI segmento compartido por t y topo no es procesado y p
            pertenece al CC (topo) ENTONCES
                  Voltear segmento compartido por t y topografía
                  Insertar (stack, t)
                  Insertar (stack, topo)
            FIN_SI
      FIN_MIENTRAS
FIN_PROCEDIMIENTO
```

2. CÓDIGO PYTHON

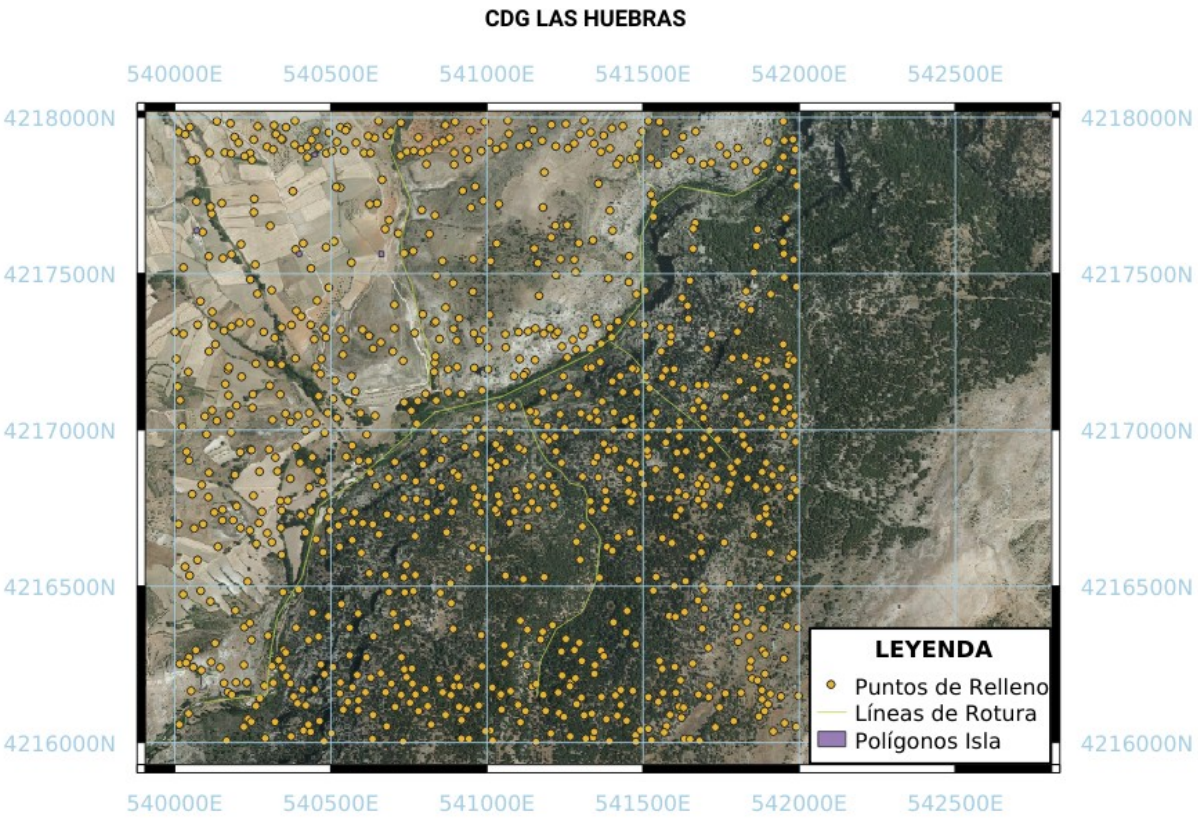
2.1. Script con ejemplo de funciones de GDAL

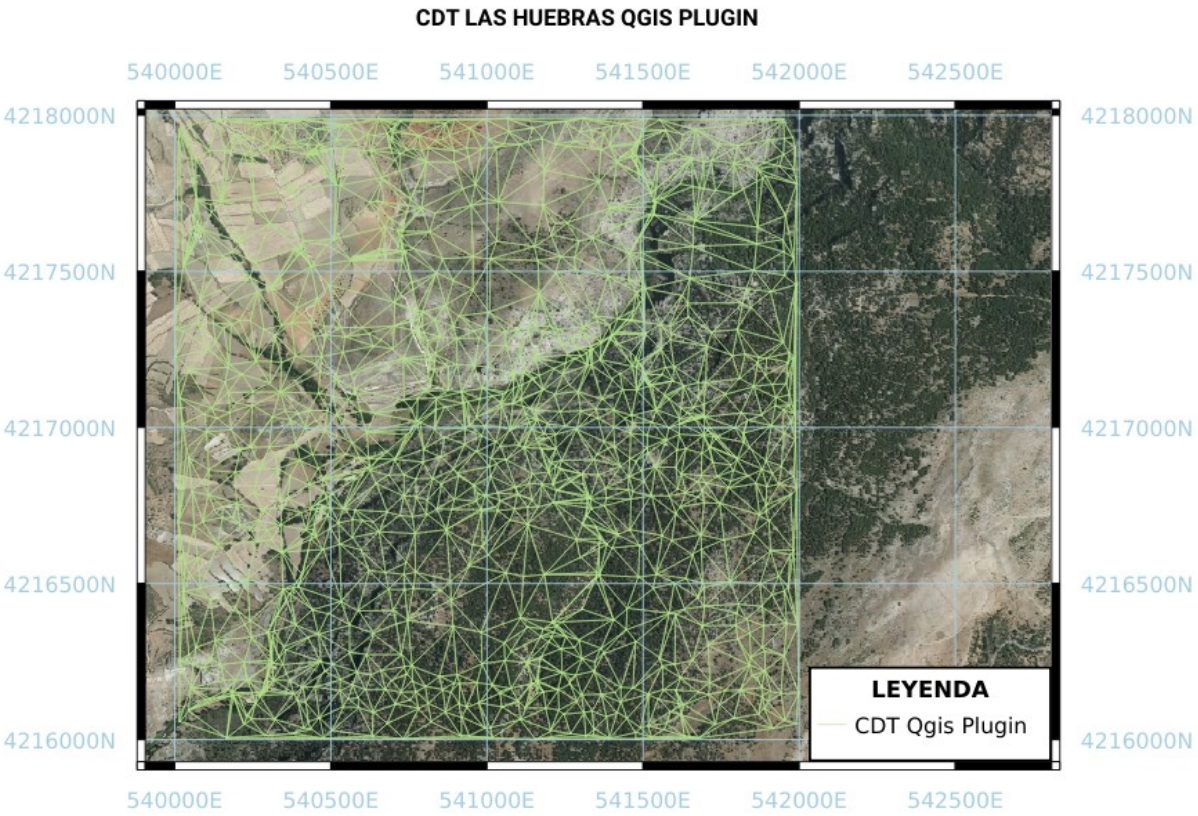
```
# Importación de la Librería GDAL
import osgeo.ogr as ogr

'''
Evaluacion Topologica del punto P con respecto a los triangulos t_48 y
t_118
'''
```

```
p = ogr.CreateGeometryFromWkt ("POINT (-0.215801426203248 0.689316528989509
3.30472453915498)")
t_48 = ogr.CreateGeometryFromWkt ("POLYGON ( (-0.457967278003872
0.683150655929167 0,-0.16 0.690737309450491 0,-0.16 0.635082945196553 0,-
0.457967278003872 0.683150655929167 0))")
intersecion_segmento_t48 = p.Intersection (t_48)
print (intersecion_segmento_t48.ExportToWkt ())

t_118 = ogr.CreateGeometryFromWkt ("POLYGON ( (-0.457967278003872
0.683150655929167 0,-0.209136080447312 0.695368273841282 0,-0.16
0.690737309450491 0,-0.457967278003872 0.683150655929167 0))")
intersecioon_segmento_t118 = p.Intersection (t_118)
print (intersecioon_segmento_t118.ExportToWkt ())
p.Touches (t_48)
p.Touches (t_118)
p.Within (t_48)
p.Within (t_118)

# Misma ejecución para ejemplos más sencillos
t_1 = ogr.CreateGeometryFromWkt ("POLYGON ( (10 10 0, 30 10 0, 20 20 0, 10
10 0))")
p_1 = ogr.CreateGeometryFromWkt ("POINT (10 10 0)")
p_2  = ogr.CreateGeometryFromWkt ("POINT (10 10 10)")

p_1.Touches (t_1)
p_1.Within (t_1)
p_2.Touches (t_1)
p_2.Within (t_1)
```

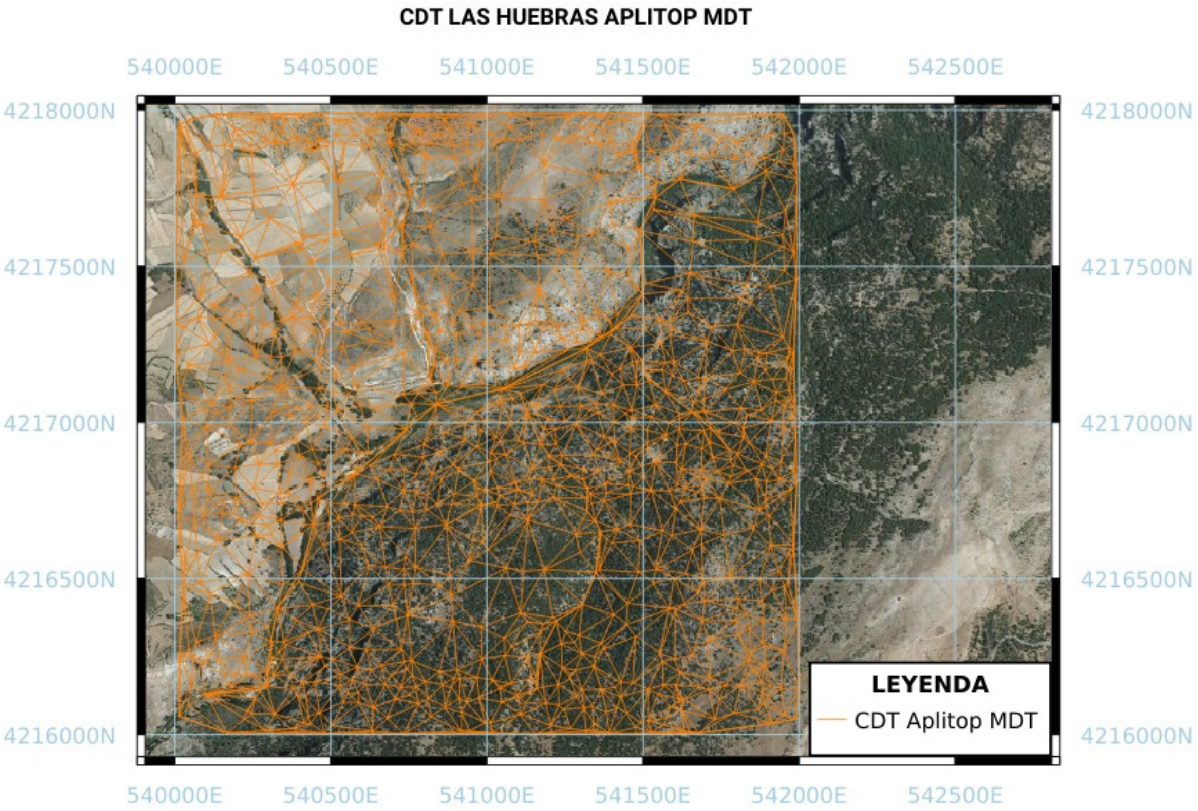
ANEXO 2

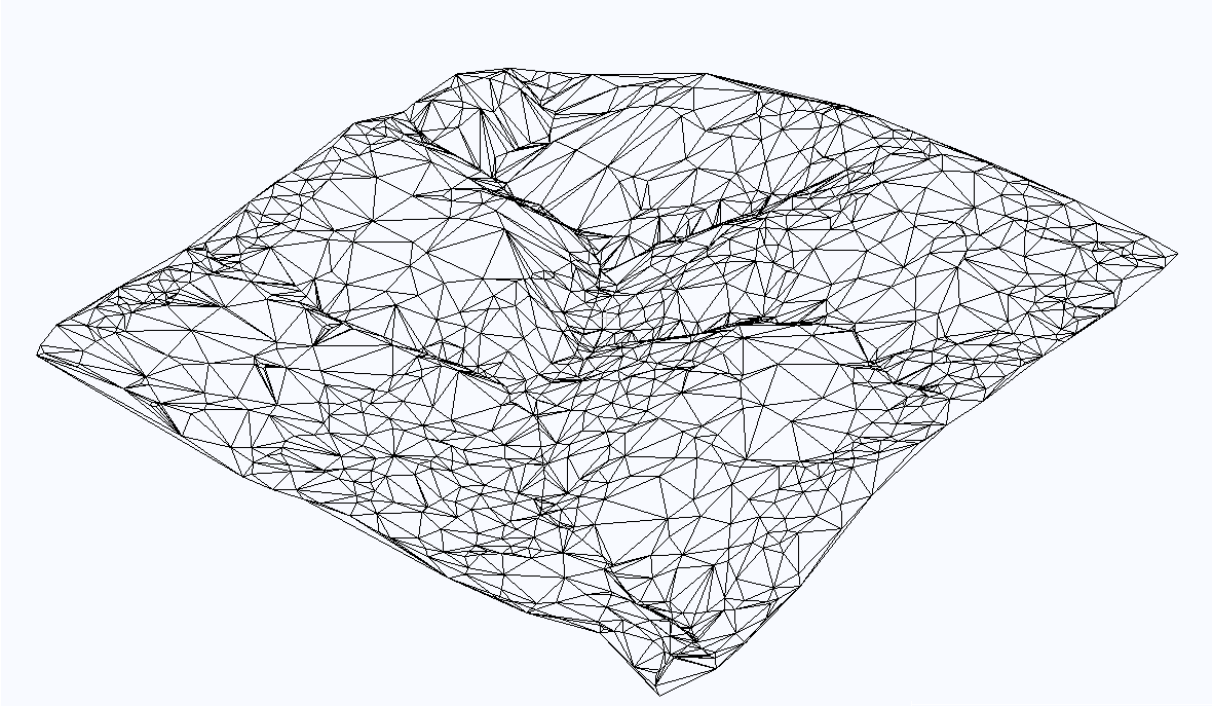
1. LAS HUEBRAS



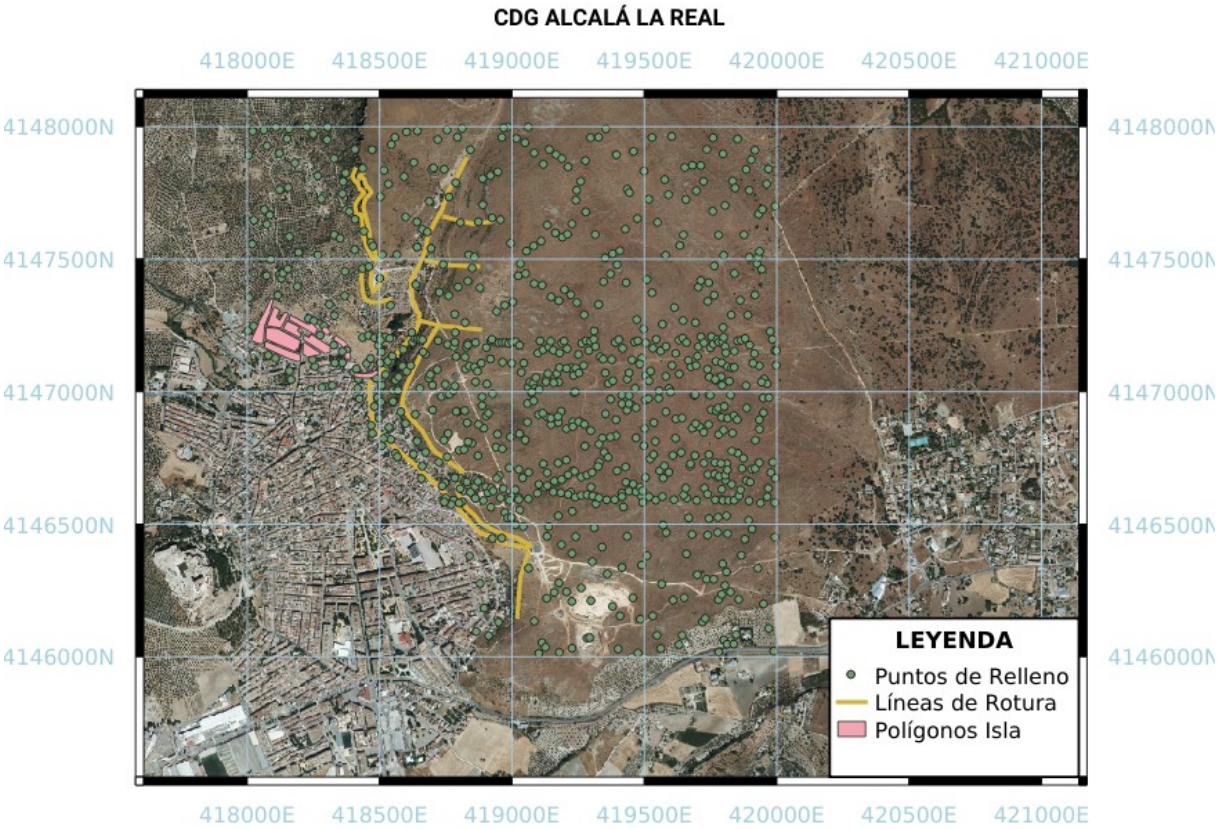

CDT LAS HUEBRAS QGIS PLUGIN
540000E
540500E
541000E
541500E
542000E
542500E
4218000N
4217500N
4217000N
4216500N
4216000N
LEYENDA
CDT Qgis Plugin

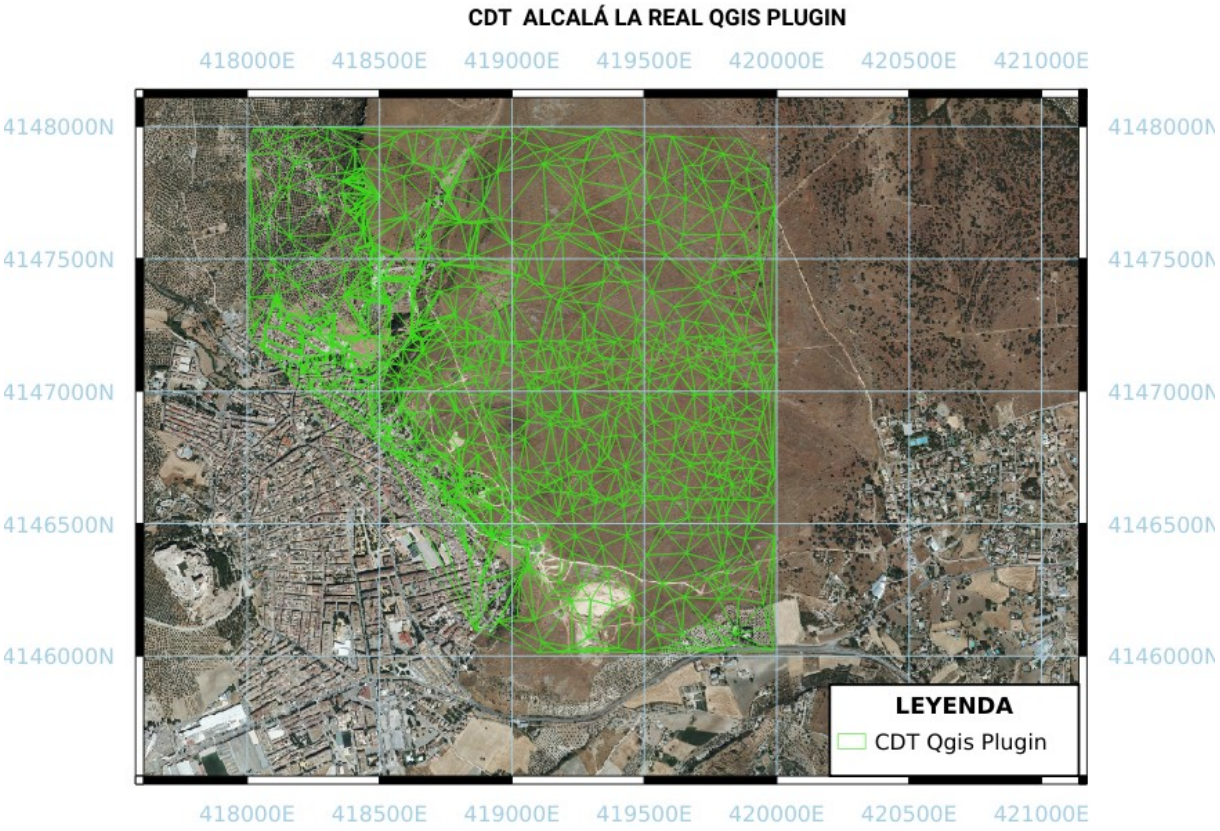

CDT LAS HUEBRAS ARCGIS PRO
540000E
540500E
541000E
541500E
542000E
542500E
4218000N
4217500N
4217000N
4216500N
4216000N
LEYENDA
CDT Arcgis Pro

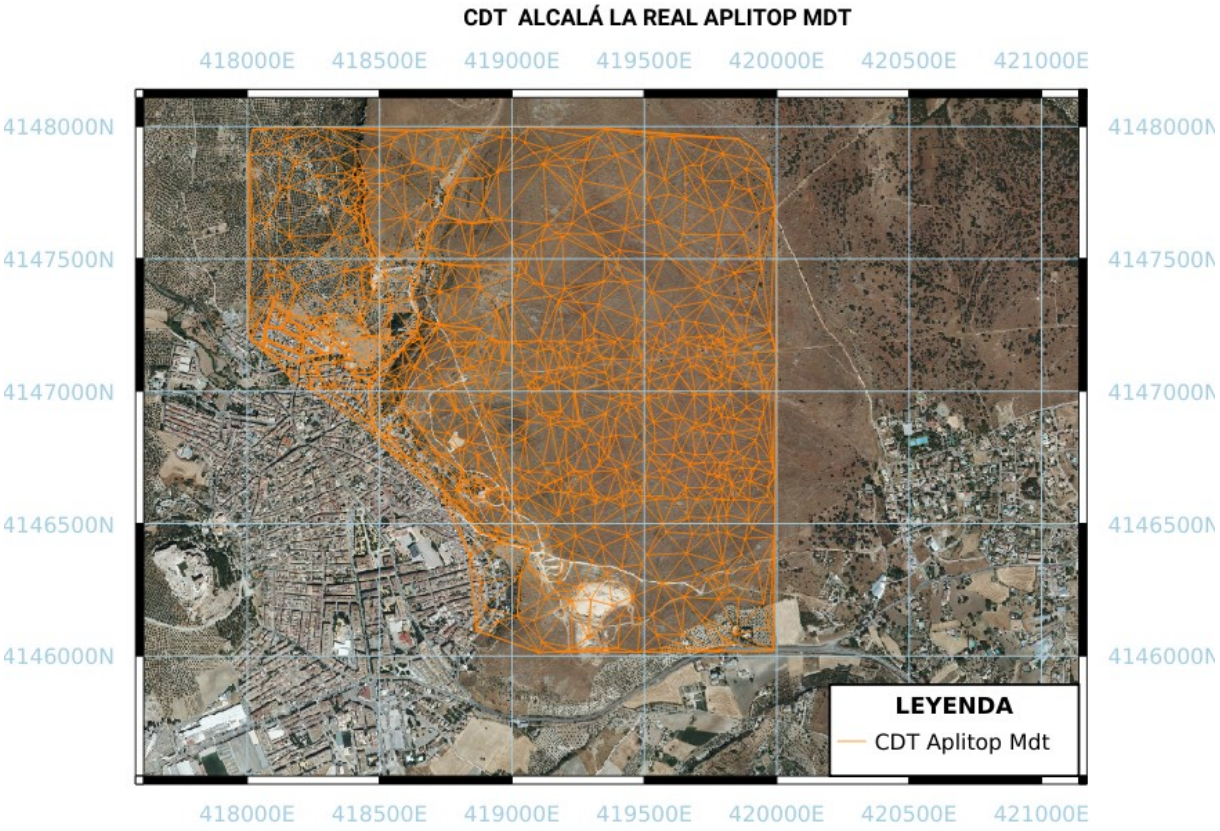

CDT LAS HUEBRAS APLITOP MDT
540000E
540500E
541000E
541500E
542000E
542500E
4218000N
4217500N
4217000N
4216500N
4216000N
LEYENDA
CDT Aplitop MDT

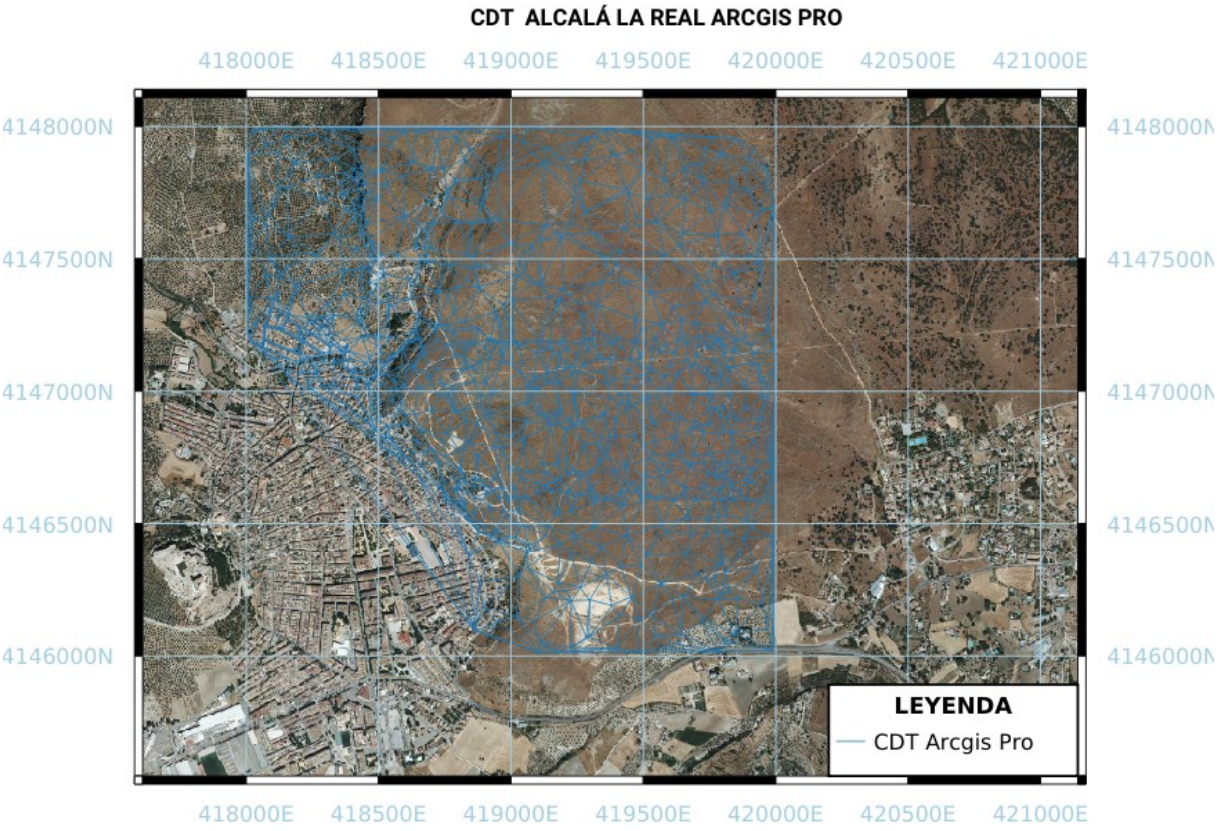


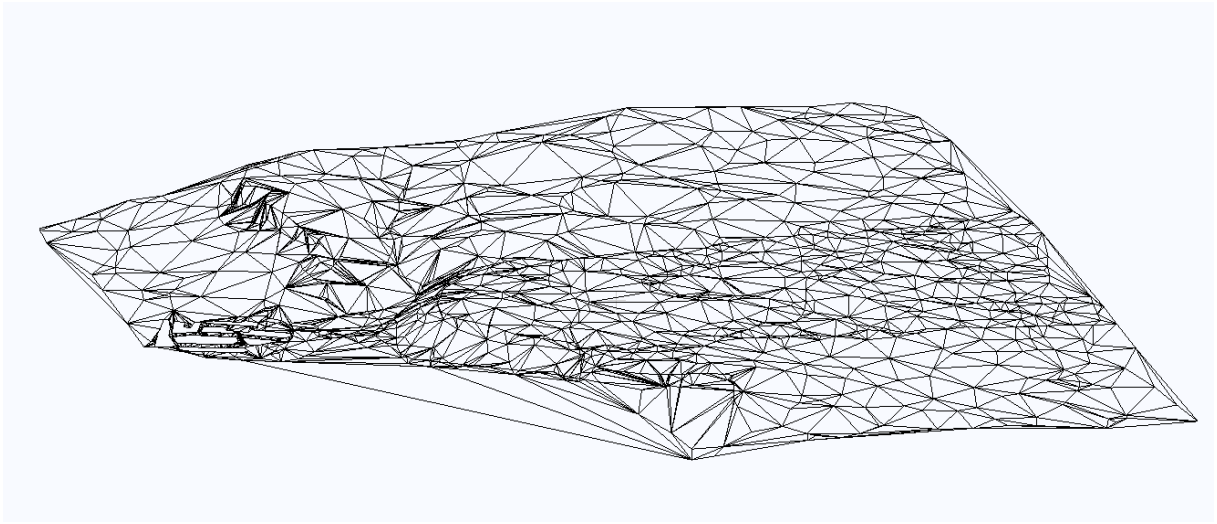
2. ALCALÁ LA REAL



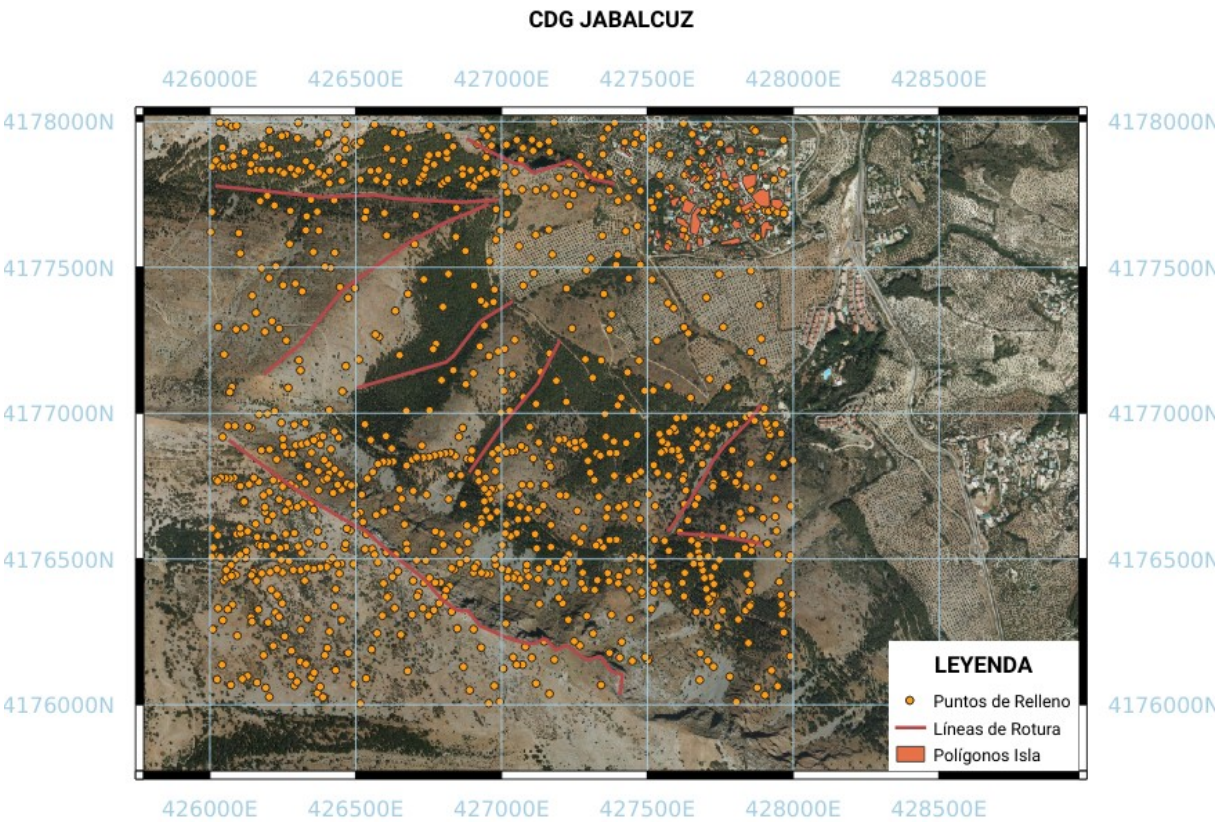

CDT ALCALÁ LA REAL QGIS PLUGIN
418000E
418500E
419000E
419500E
420000E
420500E
421000E
4148000N
4147500N
4147000N
4146500N
4146000N
LEYENDA
CDT Qgis Plugin

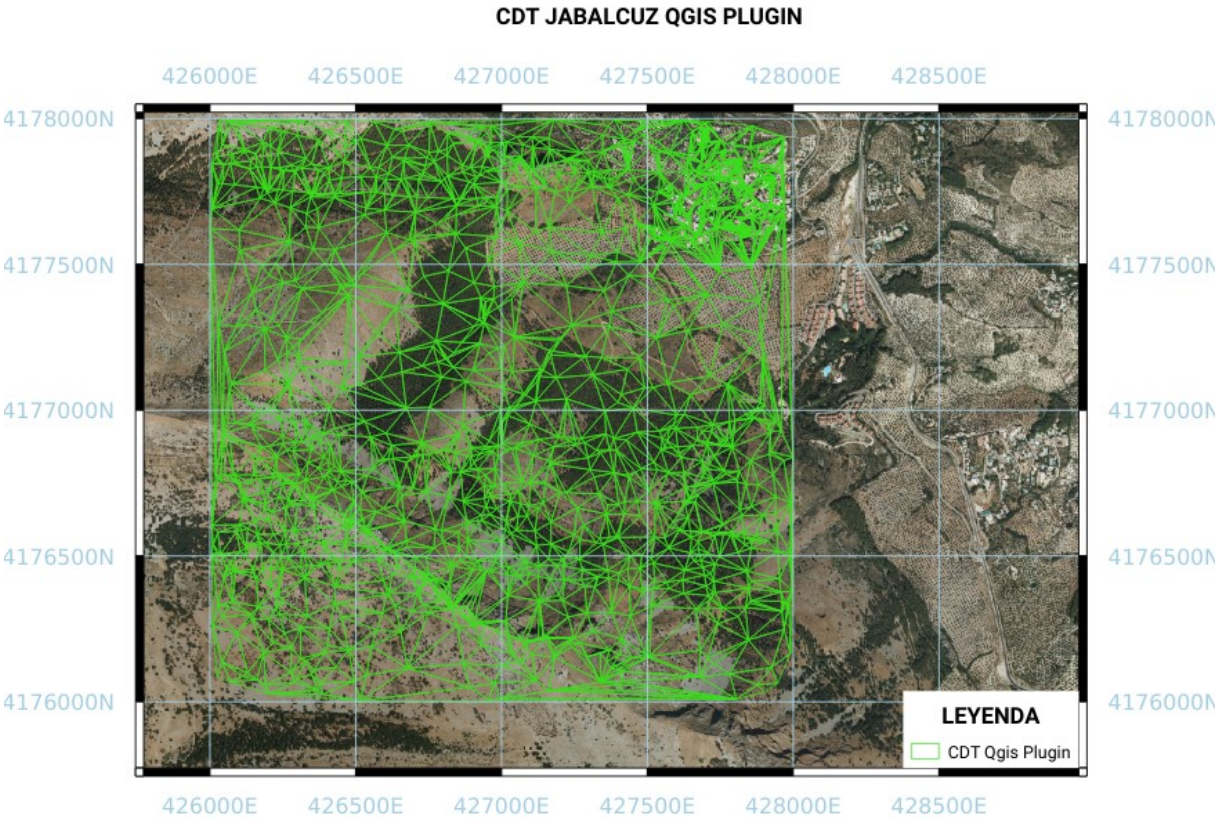

CDT ALCALÁ LA REAL APLITOP MDT
418000E
418500E
419000E
419500E
420000E
420500E
421000E
4148000N
4147500N
4147000N
4146500N
4146000N
LEYENDA
CDT Aplitop Mdt

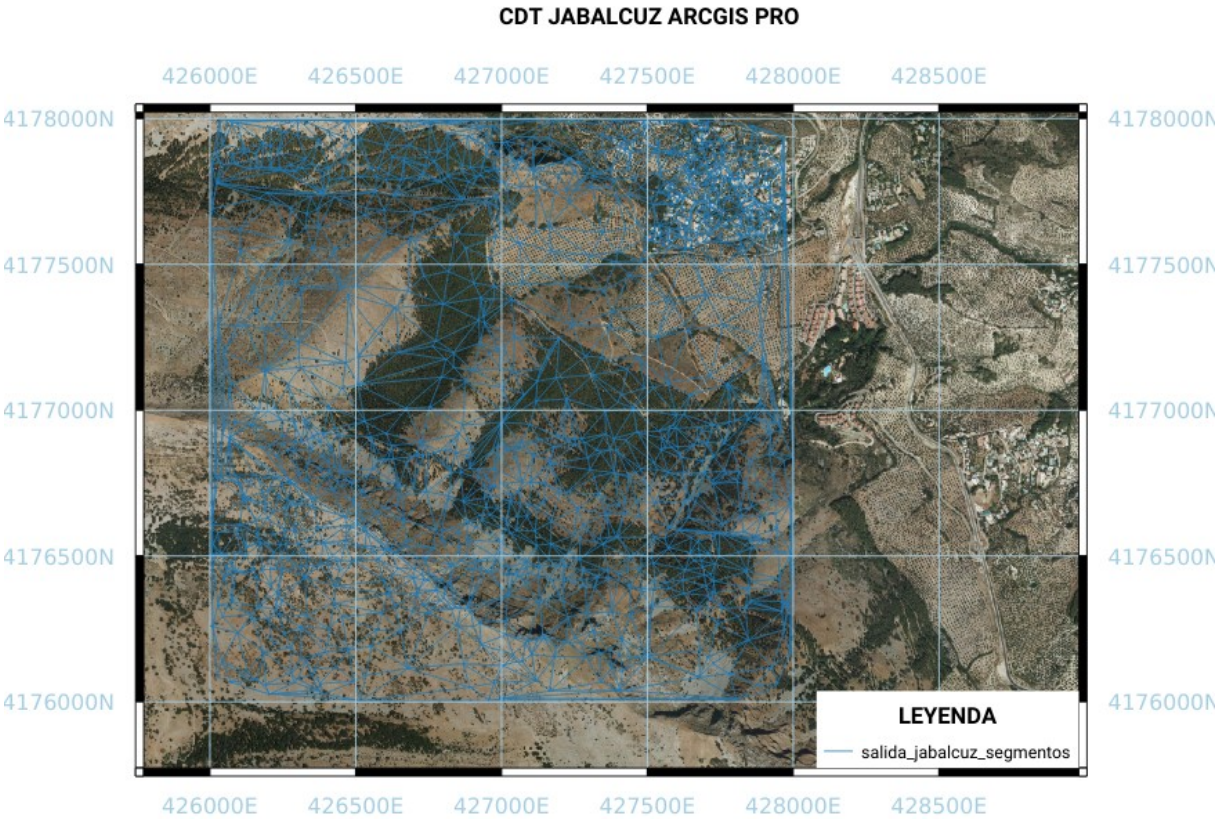

CDT ALCALÁ LA REAL ARCGIS PRO
418000E
418500E
419000E
419500E
420000E
420500E
421000E
4148000N
4147500N
4147000N
4146500N
4146000N
LEYENDA
CDT Arcgis Pro

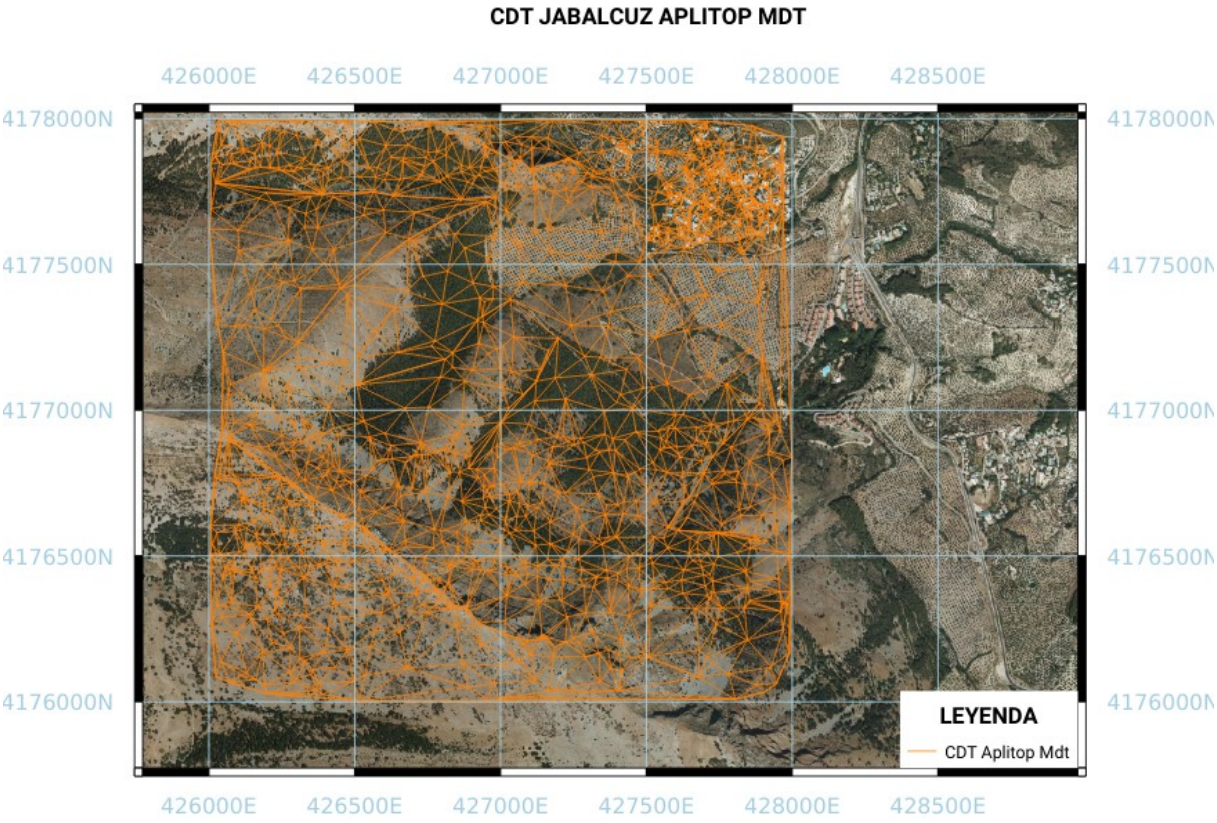


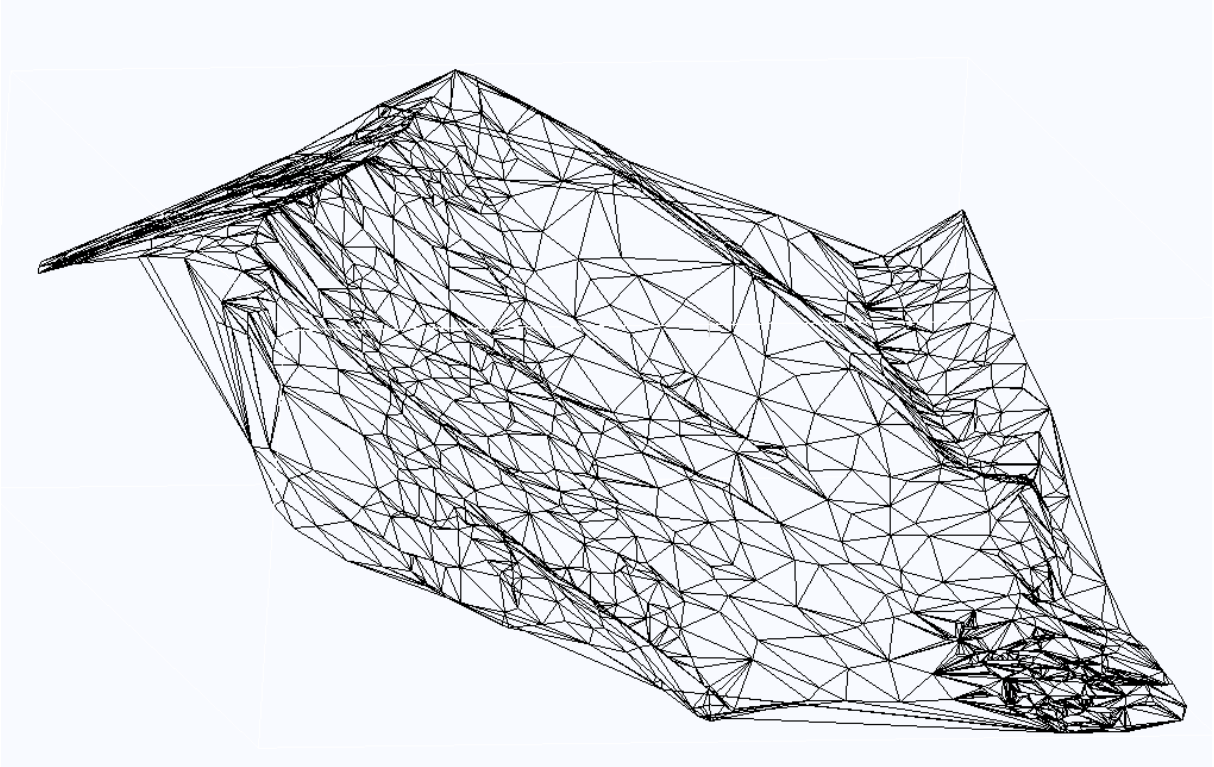
3. JABALCUZ


CDG JABALCUZ
426000E
426500E
427000E
427500E
428000E
428500E
4178000N
4177500N
4177000N
4176500N
4176000N
LEYENDA
Puntos de Relleno
Líneas de Rotura
Polígonos Isla

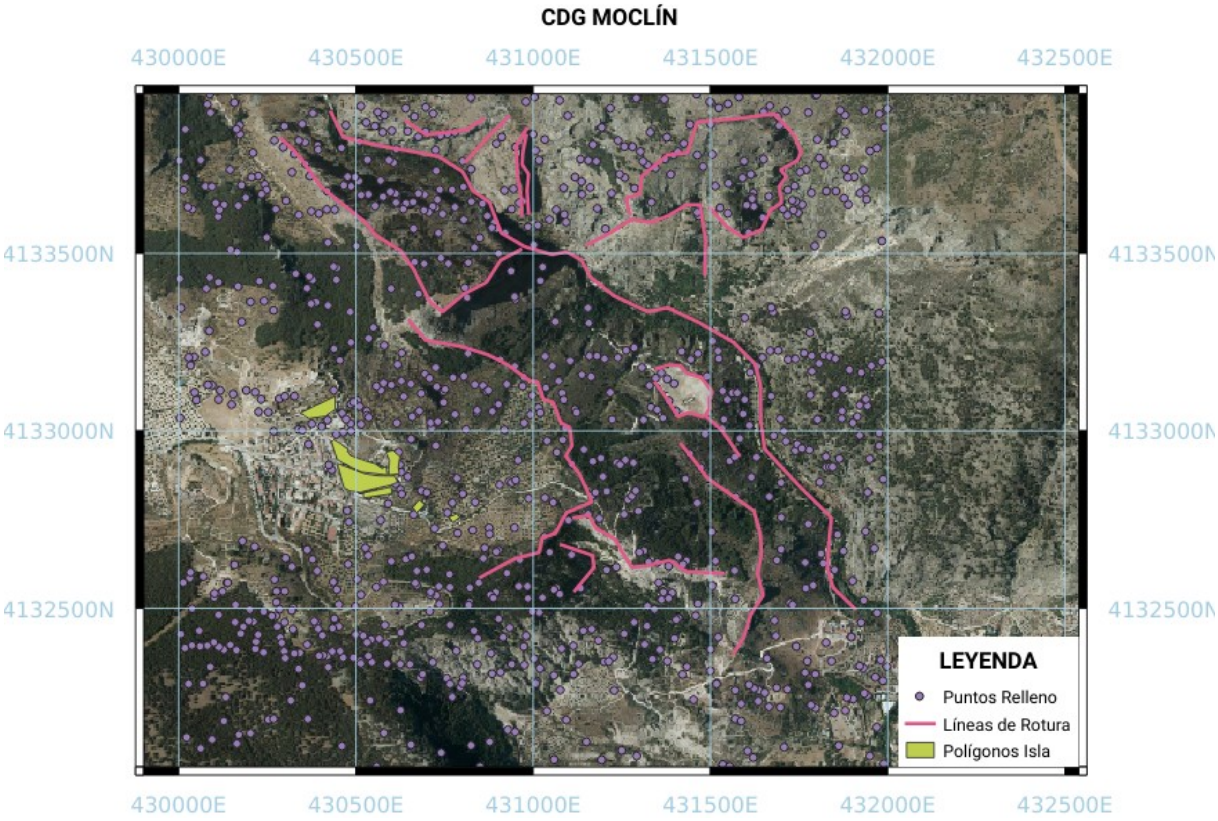

CDT JABALCUZ QGIS PLUGIN
426000E
426500E
427000E
427500E
428000E
428500E
4178000N
4177500N
4177000N
4176500N
4176000N
LEYENDA
CDT Qgis Plugin

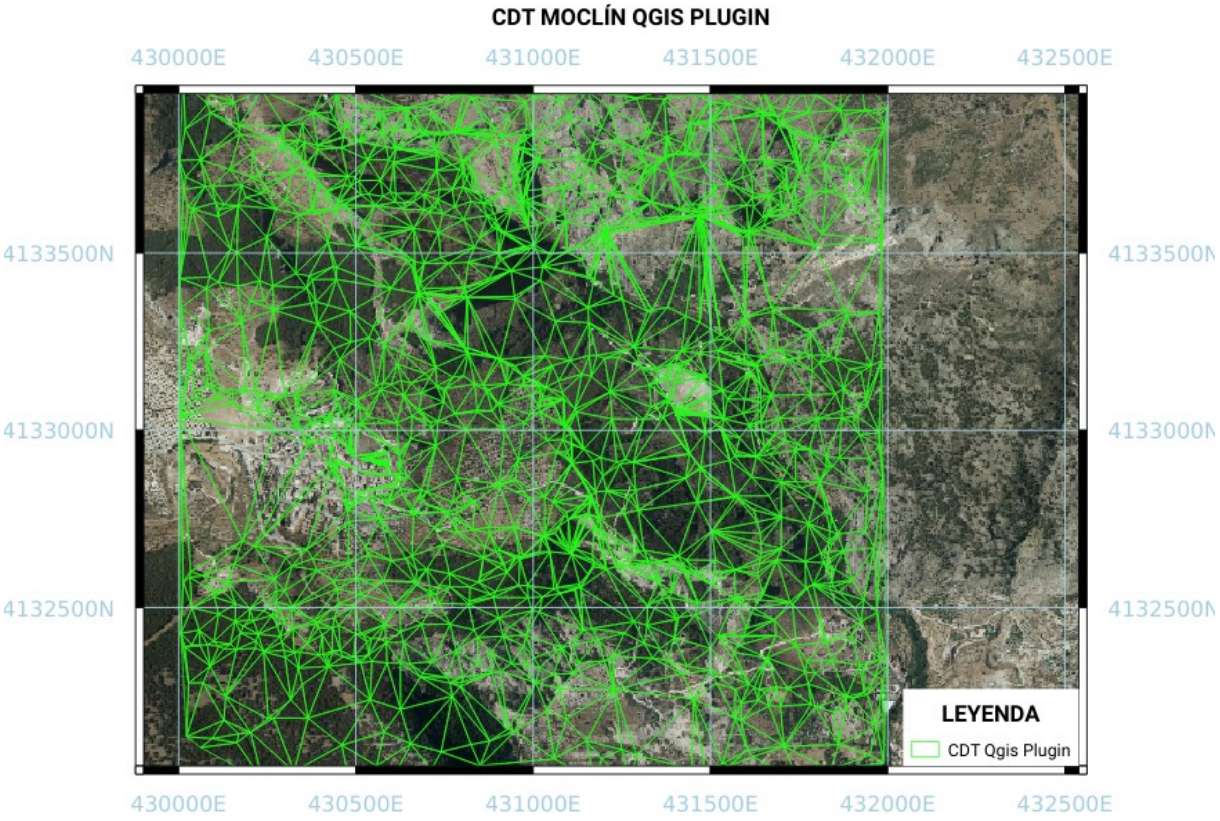

CDT JABALCUZ ARCGIS PRO
426000E
426500E
427000E
427500E
428000E
428500E
4178000N
4177500N
4177000N
4176500N
4176000N
LEYENDA
salida_jabalcuz_segmentos

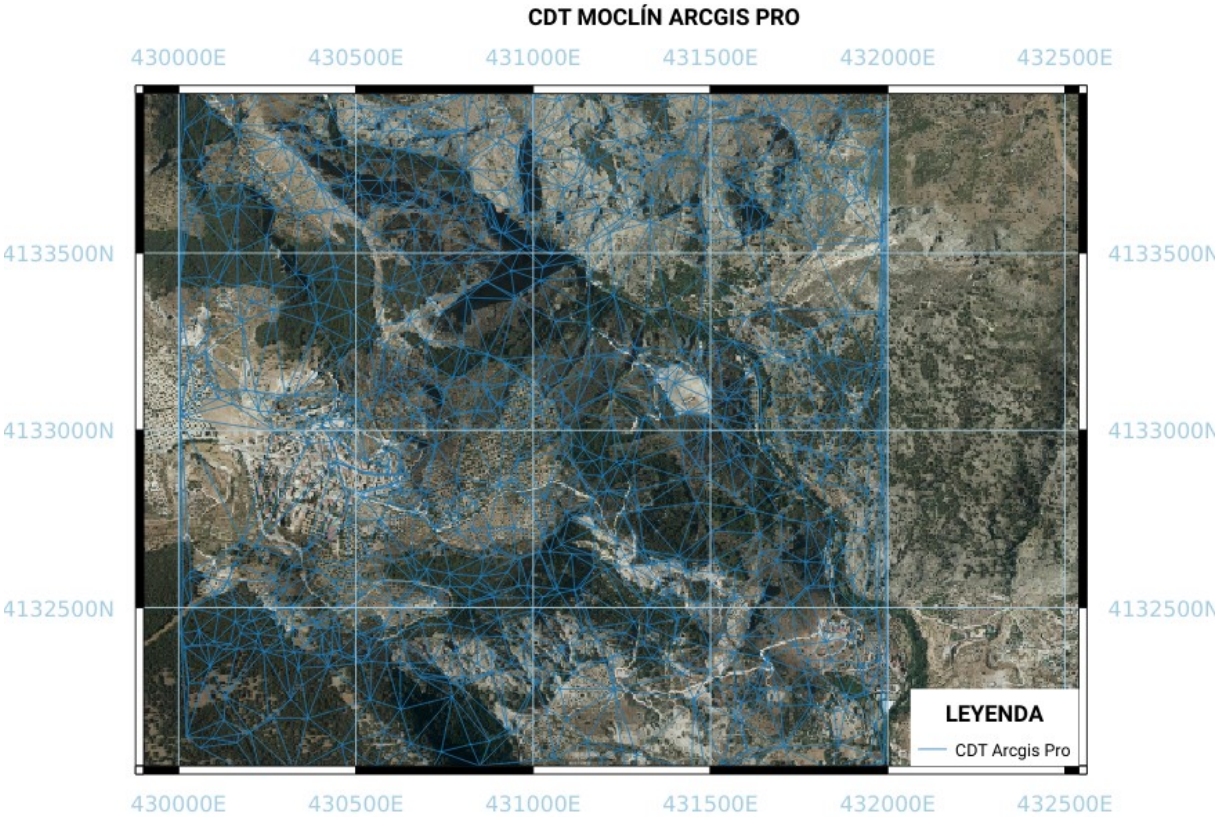

CDT JABALCUZ APLITOP MDT
426000E
426500E
427000E
427500E
428000E
428500E
4178000N
4177500N
4177000N
4176500N
4176000N
LEYENDA
CDT Aplitop Mdt

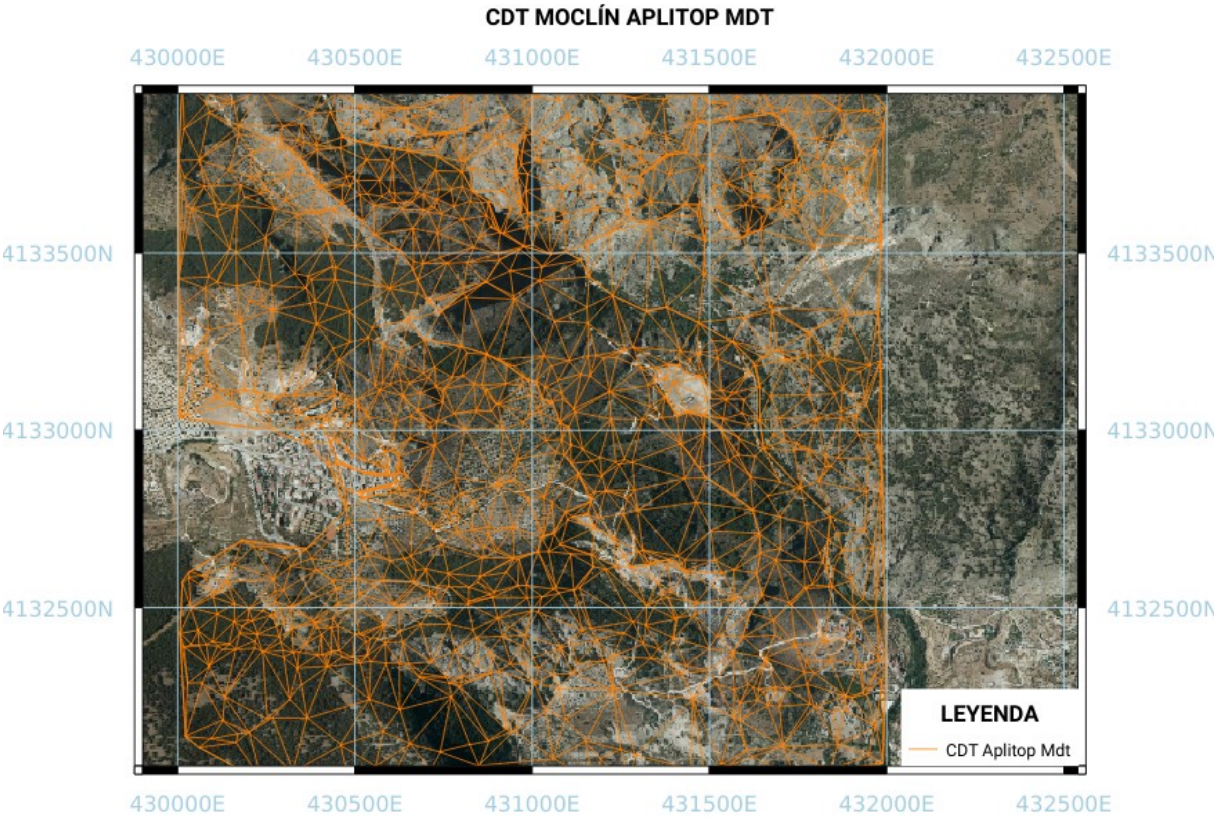


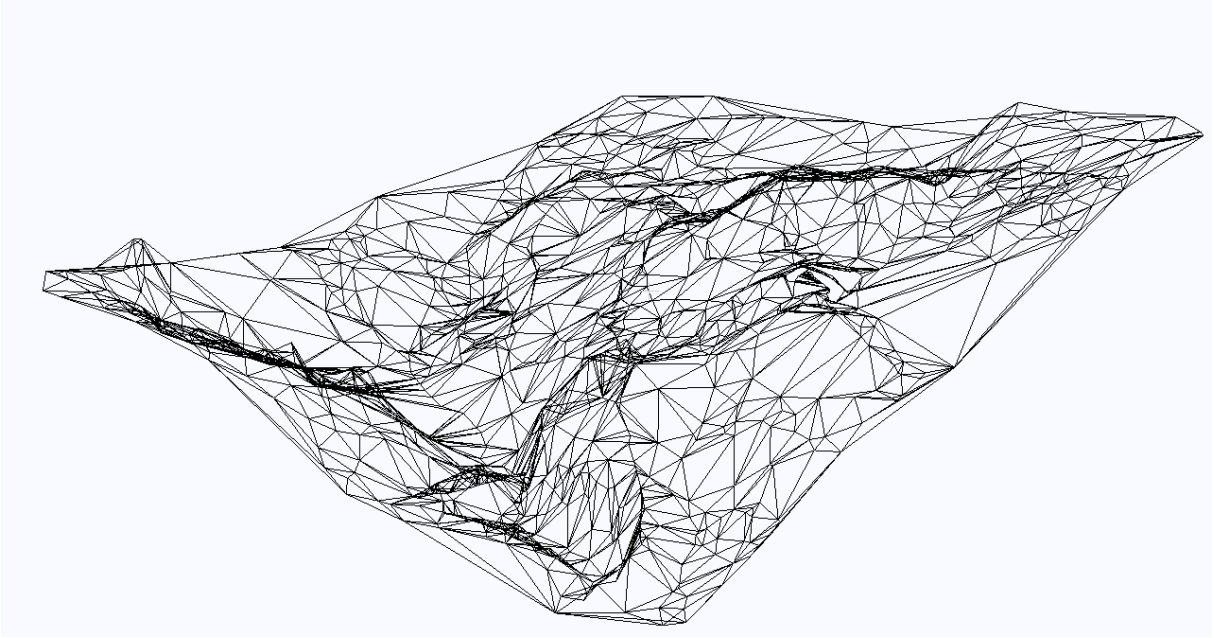
4. MOCLÍN


CDG MOCLÍN
430000E
430500E
431000E
431500E
432000E
432500E
4133500N
4133000N
4132500N
LEYENDA
Puntos Relleno
Líneas de Rotura
Polígonos Isla

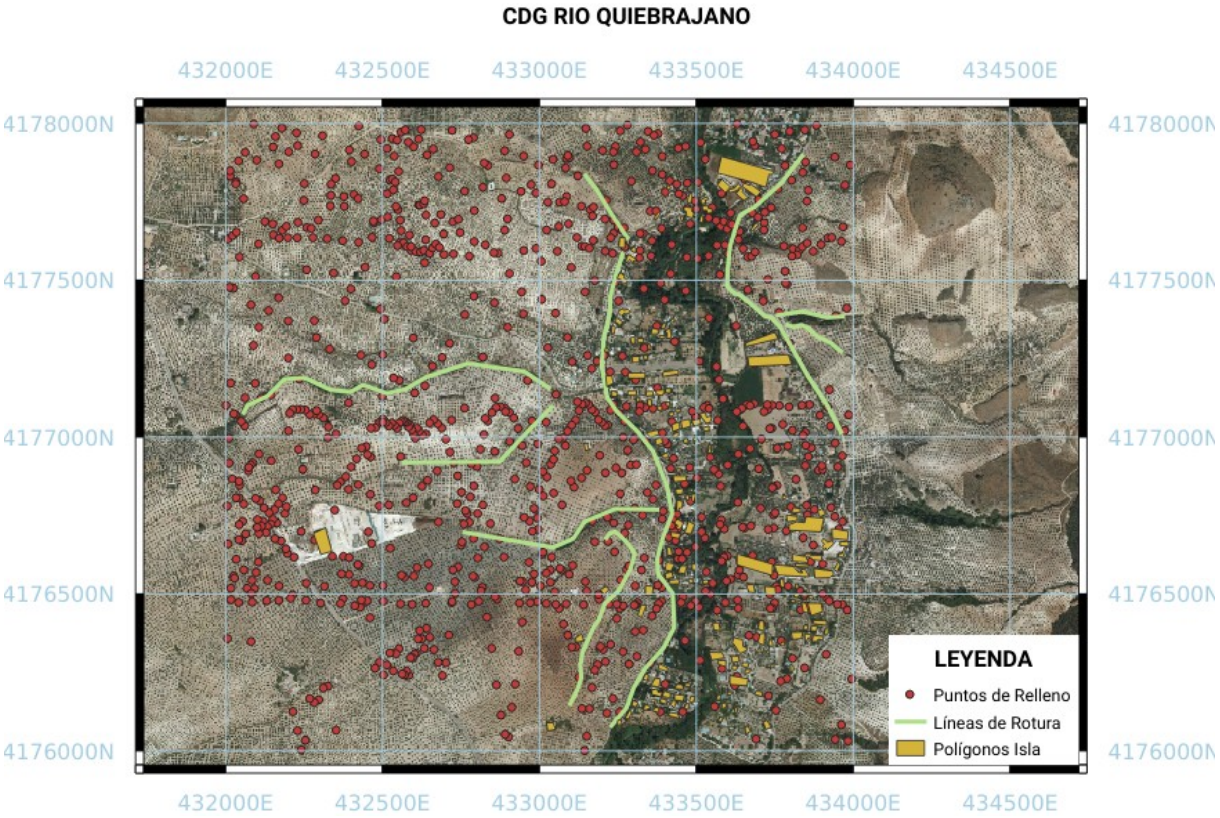

CDT MOCLÍN QGIS PLUGIN
430000E
430500E
431000E
431500E
432000E
432500E
4133500N
4133000N
4132500N
LEYENDA
CDT Qgis Plugin

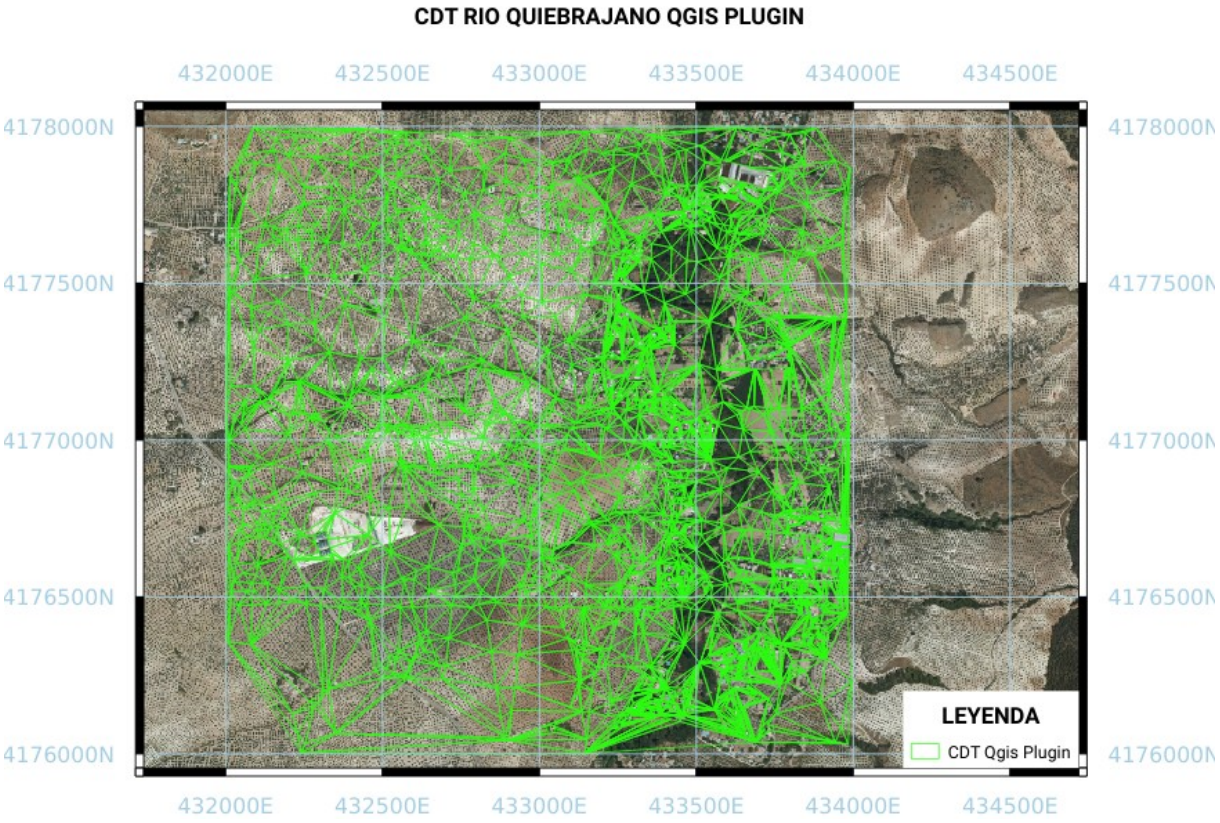

CDT MOCLÍN ARCGIS PRO
430000E
430500E
431000E
431500E
432000E
432500E
4133500N
4133000N
4132500N
LEYENDA
CDT Arcgis Pro

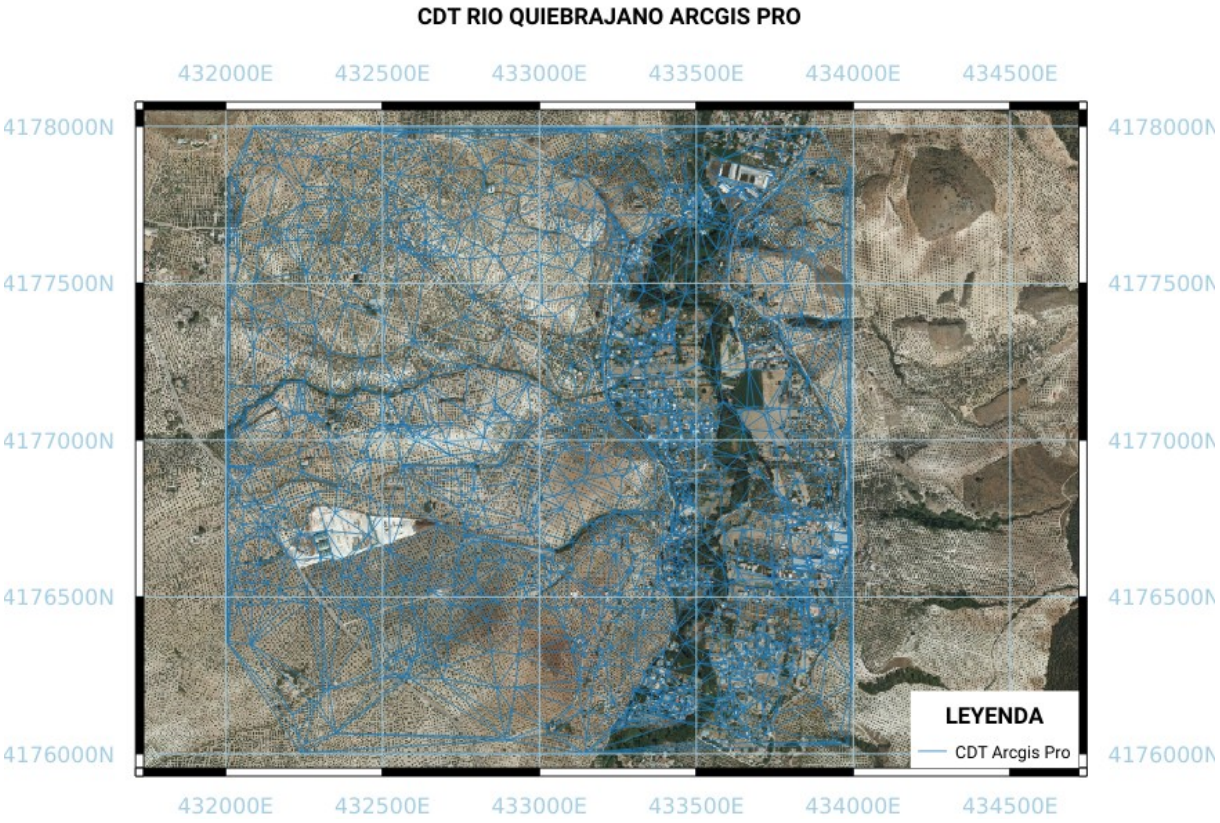

CDT MOCLÍN APLITOP MDT
430000E
430500E
431000E
431500E
432000E
432500E
4133500N
4133000N
4132500N
LEYENDA
CDT Aplitop Mdt

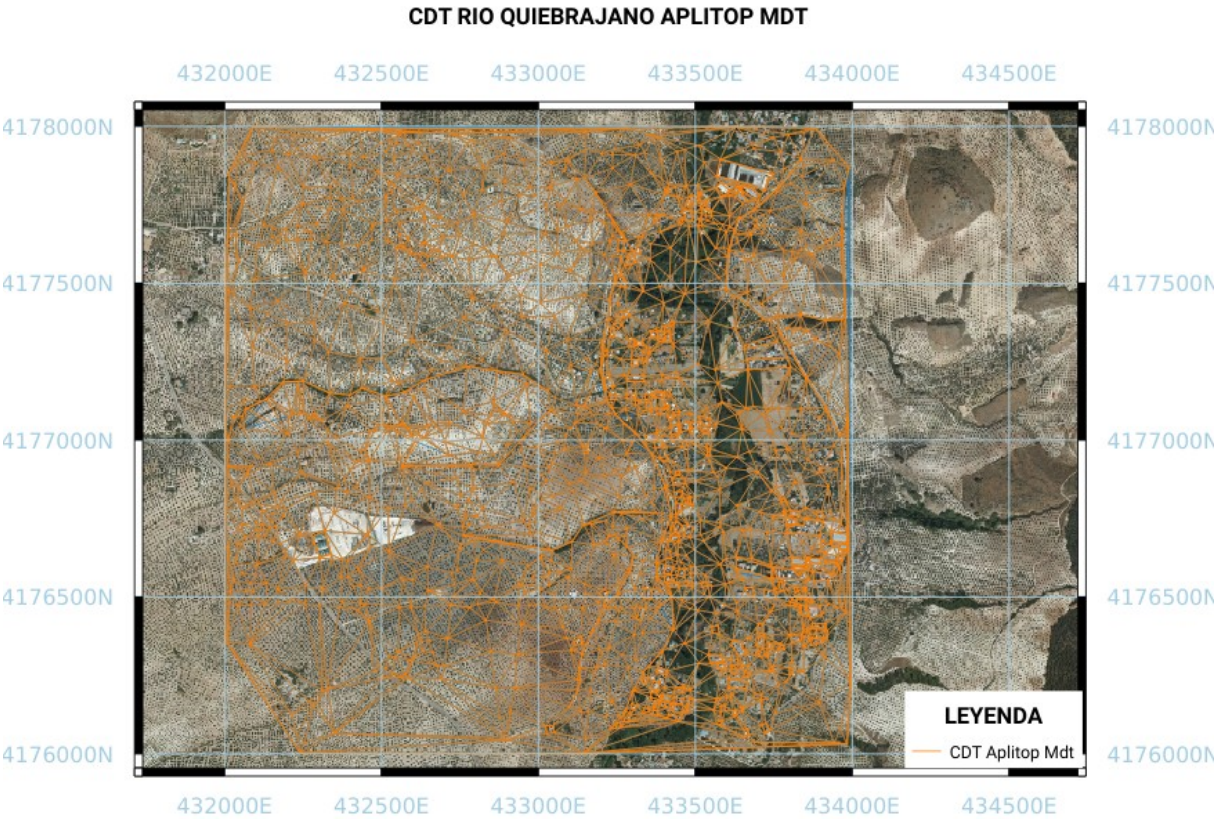


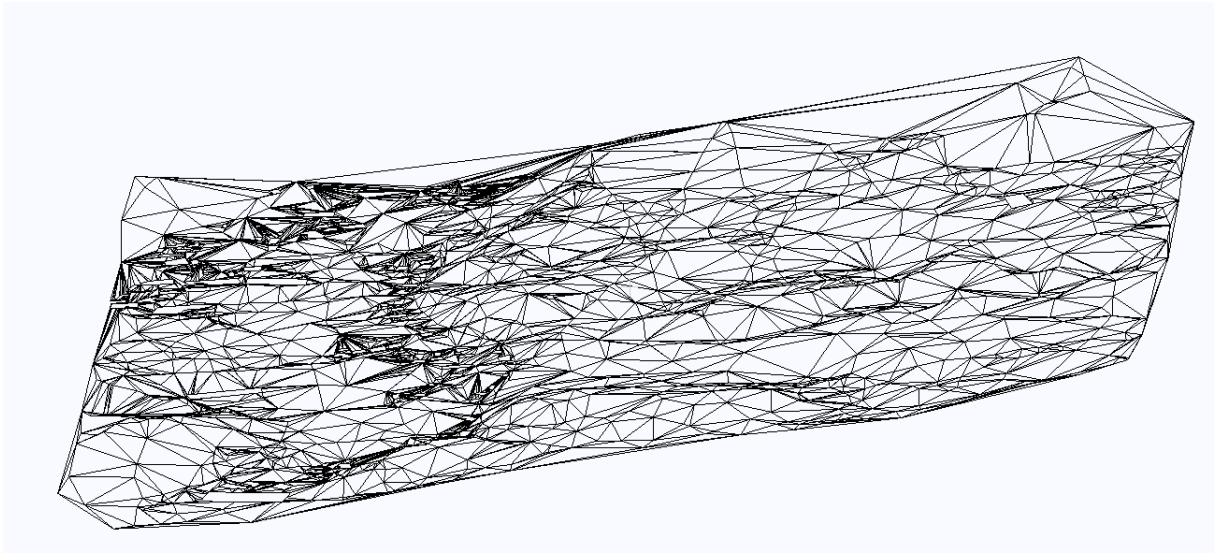
5. RÍO QUIEBRAJANO




CDT RIO QUIEBRAJANO QGIS PLUGIN
432000E
432500E
433000E
433500E
434000E
434500E
4178000N
4177500N
4177000N
4176500N
4176000N
LEYENDA
CDT Qgis Plugin

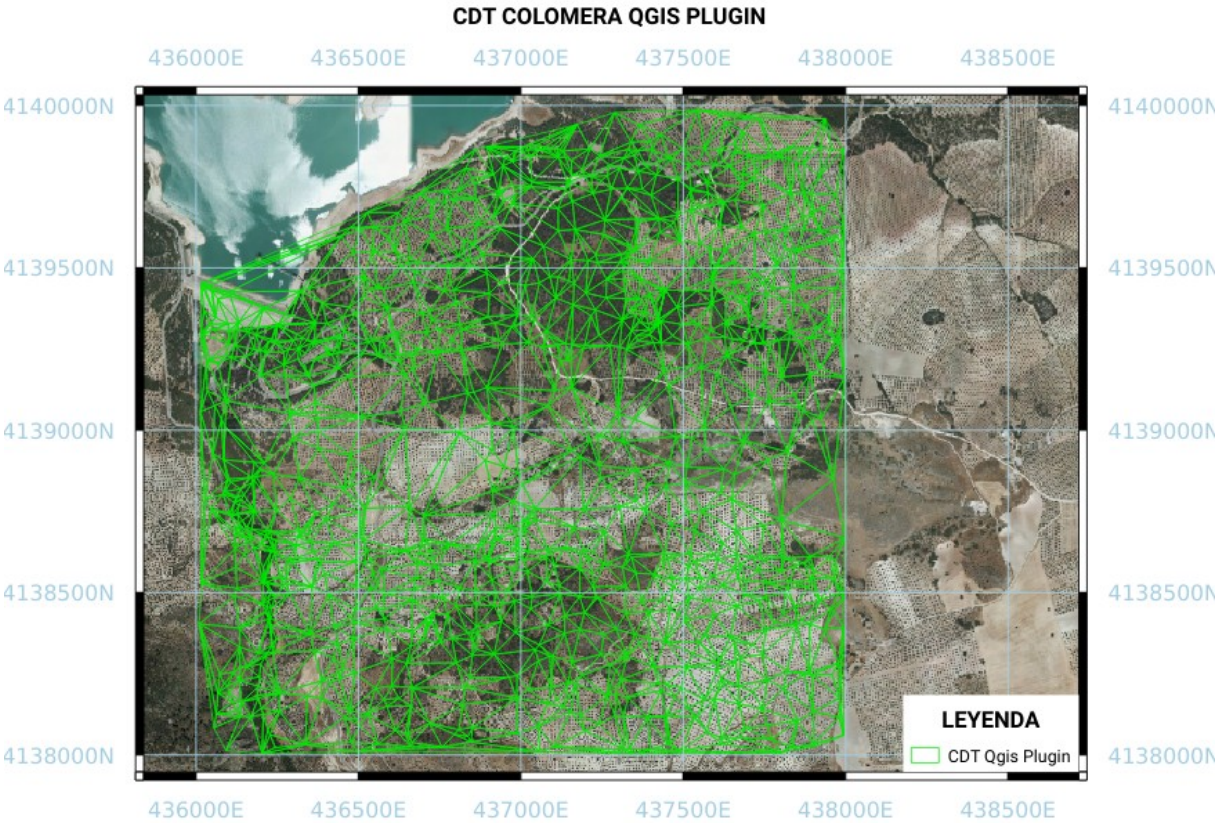

CDT RIO QUIEBRAJANO ARCGIS PRO
432000E
432500E
433000E
433500E
434000E
434500E
4178000N
4177500N
4177000N
4176500N
4176000N
LEYENDA
CDT Arcgis Pro

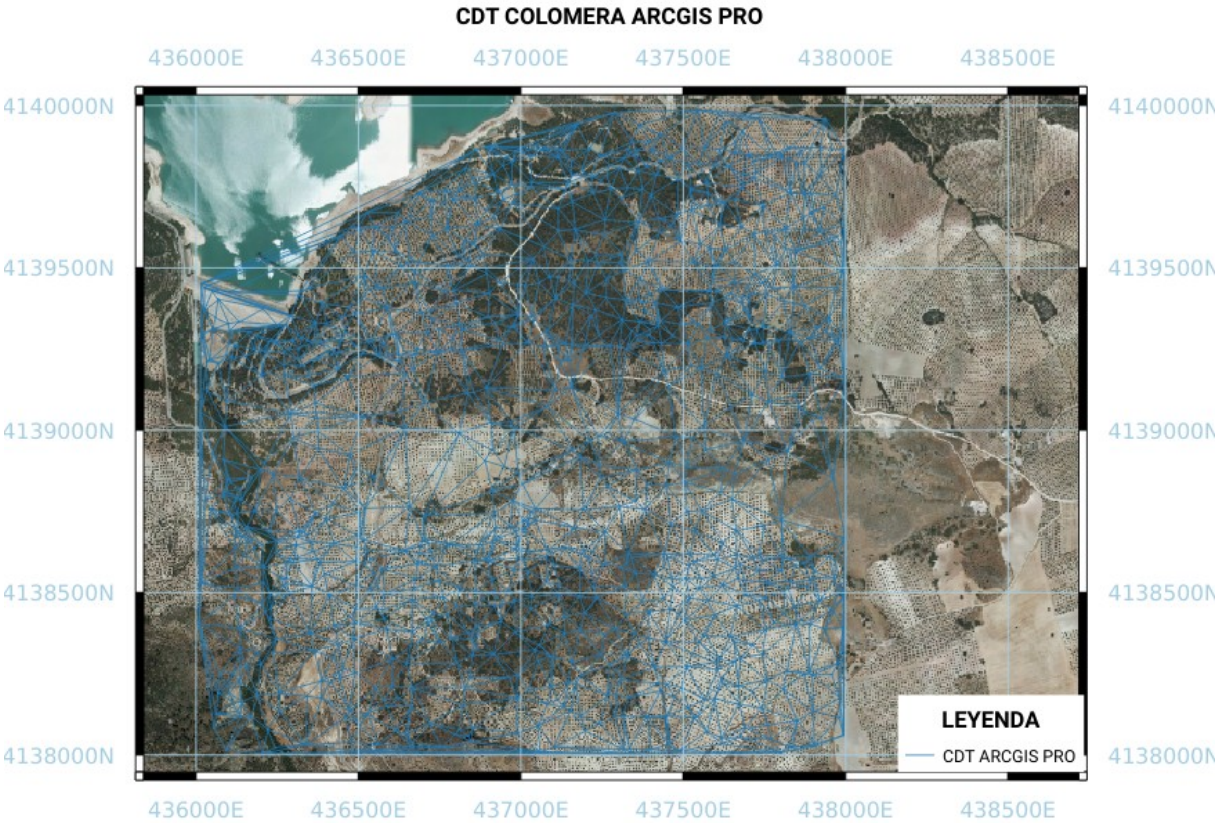

CDT RIO QUIEBRAJANO APLITOP MDT
432000E
432500E
433000E
433500E
434000E
434500E
4178000N
4177500N
4177000N
4176500N
4176000N
LEYENDA
CDT Aplitop Mdt

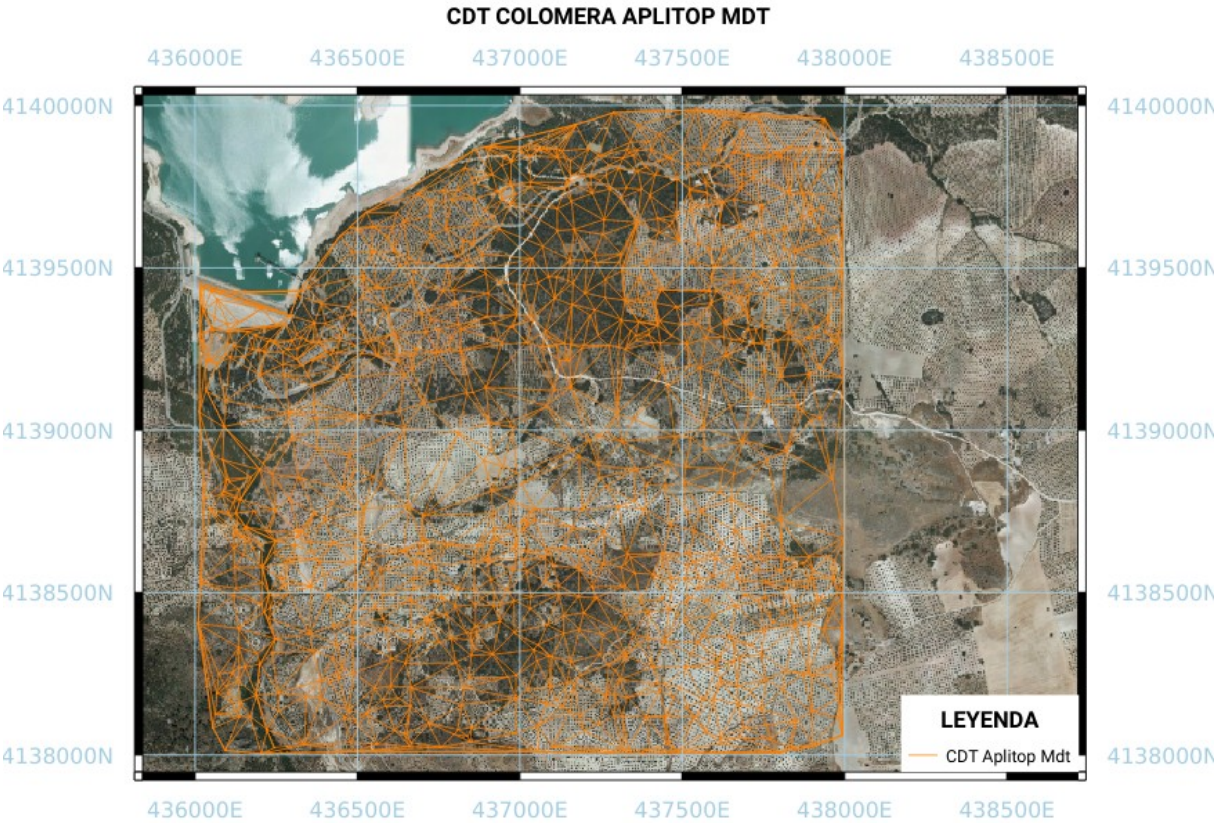


6. COLOMERA


CDG COLOMERA
436000E
436500E
437000E
437500E
438000E
438500E
4140000N
4139500N
4139000N
4138500N
4138000N
LEYENDA
Puntos Relleno
Líneas Rotura
Polígonos Isla


CDT COLOMERA QGIS PLUGIN
436000E
436500E
437000E
437500E
438000E
438500E
4140000N
4139500N
4139000N
4138500N
4138000N
LEYENDA
CDT Qgis Plugin


CDT COLOMERA ARCGIS PRO
436000E
436500E
437000E
437500E
438000E
438500E
4140000N
4139500N
4139000N
4138500N
4138000N
LEYENDA
CDT ARCGIS PRO


CDT COLOMERA APLITOP MDT
436000E
436500E
437000E
437500E
438000E
438500E
4140000N
4139500N
4139000N
4138500N
4138000N
LEYENDA
CDT Aplitop Mdt