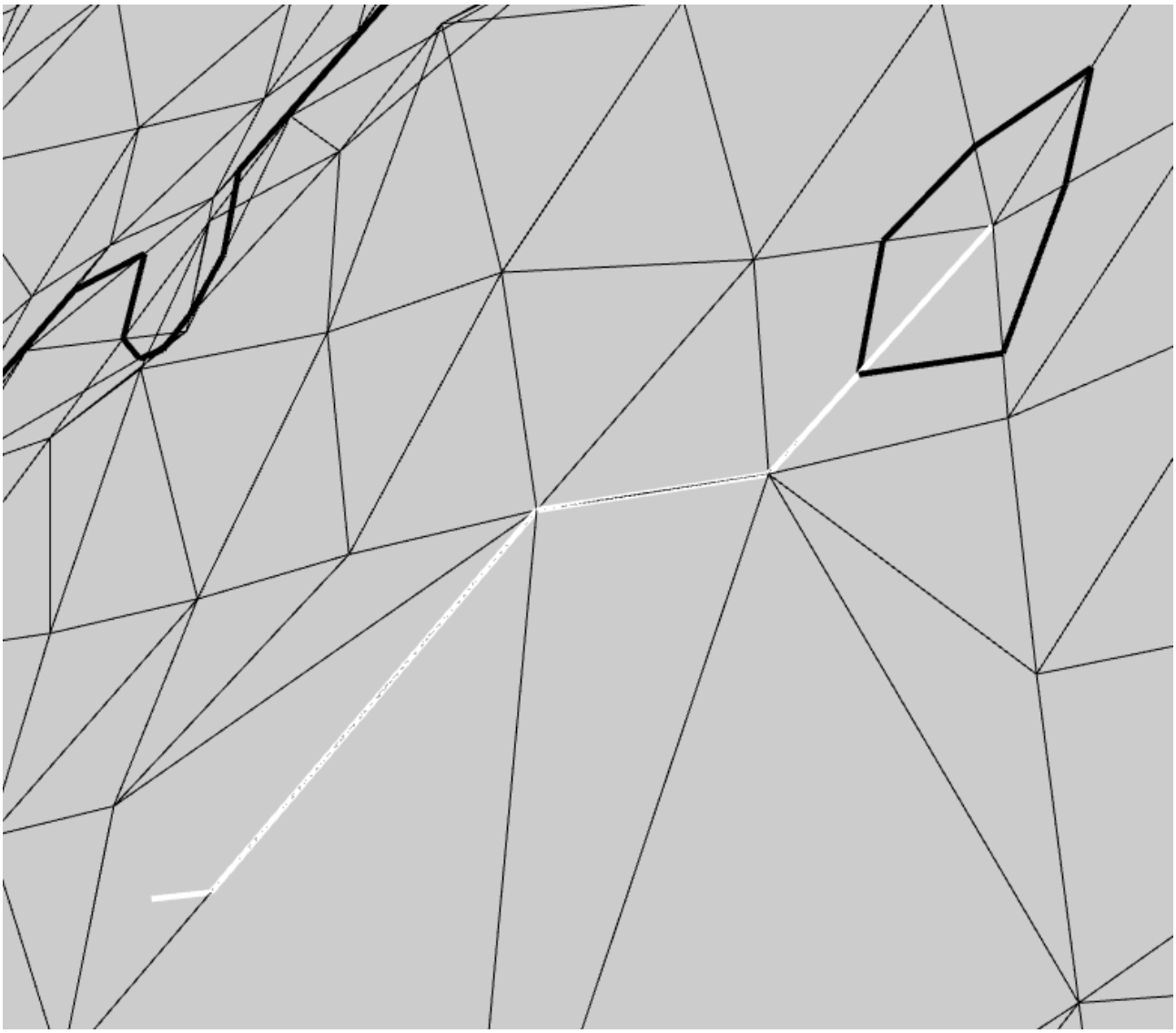
MODELIZACION DE TERRENOS



Guillermo Alonso Núñez
t11m034
Mayo de 2014

Introducción y motivación

Desde hace mucho tiempo, ha suscitado un gran interés al ser humano el estudio de la tierra y sus diferentes propiedades. Como se puede imaginar, realizar un análisis de un lugar era algo costoso, ya que las mediciones debían ser precisas y fiables.

Esto dio lugar a la **cartografía**, rama de la ciencia que se centra en el desarrollo de mapas geográficos.

Según avanzaba la tecnología, cada vez había métodos más precisos y que permitían obtener más información sobre el terreno a estudiar.

Hoy en día, es de gran importancia realizar un buen estudio a la hora de, por ejemplo, estudiar por donde rectificar el flujo de un río para construir una presa. Gracias a la tecnología actual, es posible obtener una gran cantidad de datos precisos de una manera relativamente cómoda, como puede ser con la ayuda de la refracción láser sobrevolando el terreno en un avión, sin embargo, la dificultad reside en cómo manipular esos datos para que nos resulten útiles, es decir, como interpretarlos.

Por tanto, el **problema que se presenta** consiste en, dada una nube de puntos obtenidos de un terreno real, dispuesta en una malla regular, de dimensión 715x746 (533.390 puntos en total), implementar un sistema de simplificación de los datos, implementar una estructura que soporte el modelo del terreno (DCEL) e implementar algunas operaciones sobre el terreno.

Al tratarse de un trabajo en grupo, hubo un reparto de tareas. Para la simplificación de los datos hubo varias propuestas, las cuales se explican más detalladamente posteriormente, el DCEL cada uno intentó implementarlo a su manera, aunque finalmente el que consta en el código es la versión aquí expuesta, y por último cada uno implementó una operación.

- A la hora de programar...

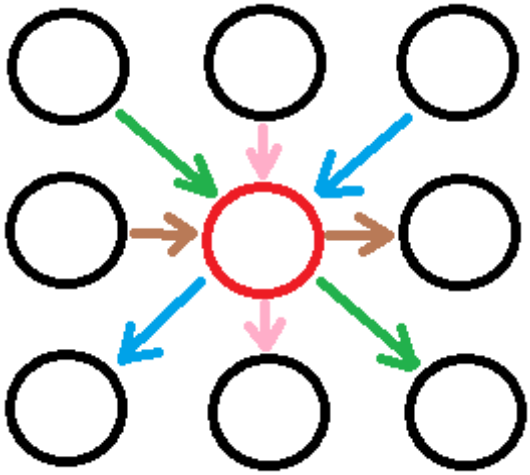
Para la realización de este trabajo, optamos por utilizar la herramienta **Processing**
ya que facilitaba bastante la parte gráfica y está basado en Java, lenguaje que más se ha usado durante la carrera.
También hay que instalar PEASYCAM, una cámara disponible para la visualización en 3D de los resultados.

Simplificación de los datos

Una primera simplificación a considerar podría ser la fuerza bruta: simplemente cogemos uno de cada X puntos y trabajamos con el resultado.
Otra idea a considerar era distribuir los puntos según el intervalo de altura al que pertenezcan (por ejemplo, 500-530, 530-560...) y mirar cuántos hay en cada uno. Después, en función del porcentaje de puntos en cada intervalo, seleccionar 'n' puntos para cada altura de forma aleatoria. Lo malo de ésto, es que no garantizaba un buen resultado siempre, al ser un método "aleatorio".

Al final optamos por hacer una criba según la **8-vecindad**, que consiste en lo siguiente:

- Los puntos del borde de la malla no los borramos, todos formarán parte de la estructura triangular.
- Para el resto de puntos (interiores):



Miramos que la diferencia de alturas para cada dirección, representadas en la figura cada uno con un color, y siempre en el mismo sentido, esté por debajo de un error que aceptamos. Si todas lo están, quiere decir que el punto central casi pertenece a cada una de las 4 rectas que pasan por él, así que lo marcamos para borrarlo después, pero no lo borramos aún porque lo necesitamos para mirar otros puntos que tendrán al actual como vecino.

Después, simplemente borramos los marcados como candidatos a ser borrados.

Por tanto, una vez simplificados los puntos, lo que tenemos es una malla regular con "huecos" en las posiciones donde hemos podido quitar puntos.

- *A la hora de programar...*

Para meter los datos en Processing, lo hacemos mediante una matriz (array de dos dimensiones) de tipo double, *heights*, que se encuentra en el método calcular() de la clase Prueba. Al tener más de medio millón de puntos, Processing daba un error relacionado con la "constant pool", así que esto nos forzó a reducir considerablemente el número de puntos que usábamos como conjunto inicial, tomando sólo uno cada 9 filas y 9 columnas (es decir, 1 de cada 81 puntos).

Por tanto, consideramos el resultado como un nuevo terreno y pese a ya haberlo simplificado por fuerza bruta, lo simplificamos por la 8-vecindad para comprobar que funciona correctamente.

Lo que hacemos es hacer la primera pasada y guardar los borrados en una matriz de booleanos de la misma dimensión, y después a la hora de crear la matriz de puntos (ya no solo alturas; coordenadas x, y, z), guardamos un null en las alturas que podamos borrar.

Ésto se hace con los métodos auxiliares nBorrar(...), update(...) y con un bucle para la creación de los puntos.

Estructura DCEL

Una vez tenemos los puntos simplificados, el siguiente paso es construir una triangulación del terreno que lo represente de forma aceptable. Dicha triangulación no tiene por qué ser única, y dependiendo de lo que queramos saber, nos vendrá mejor una que otra.

Sin embargo, hay una triangulación especial, no necesariamente única para cada conjunto de puntos en la cual nos basaremos, la **triangulación de DELONE** (o Deulonay). Dicha triangulación tiene una propiedad: es la que maximiza los ángulos más pequeños de cada triángulo.

Para llegar a dicha triangulación, lo haremos partiendo de una primera triangulación cualquiera (que construiremos nosotros) y posteriormente iremos intercambiando (o *flipando, del ingles "to flip"*) aristas (detallado más adelante).

A la hora de trabajar con el terreno en sí, tendrá un gran impacto el tipo de datos que usemos para guardarlo. Como nos interesará poder movernos por la triangulación con la menor complejidad posible, optamos por implementarlo según la estructura DCEL (*doubly connected edge list*).

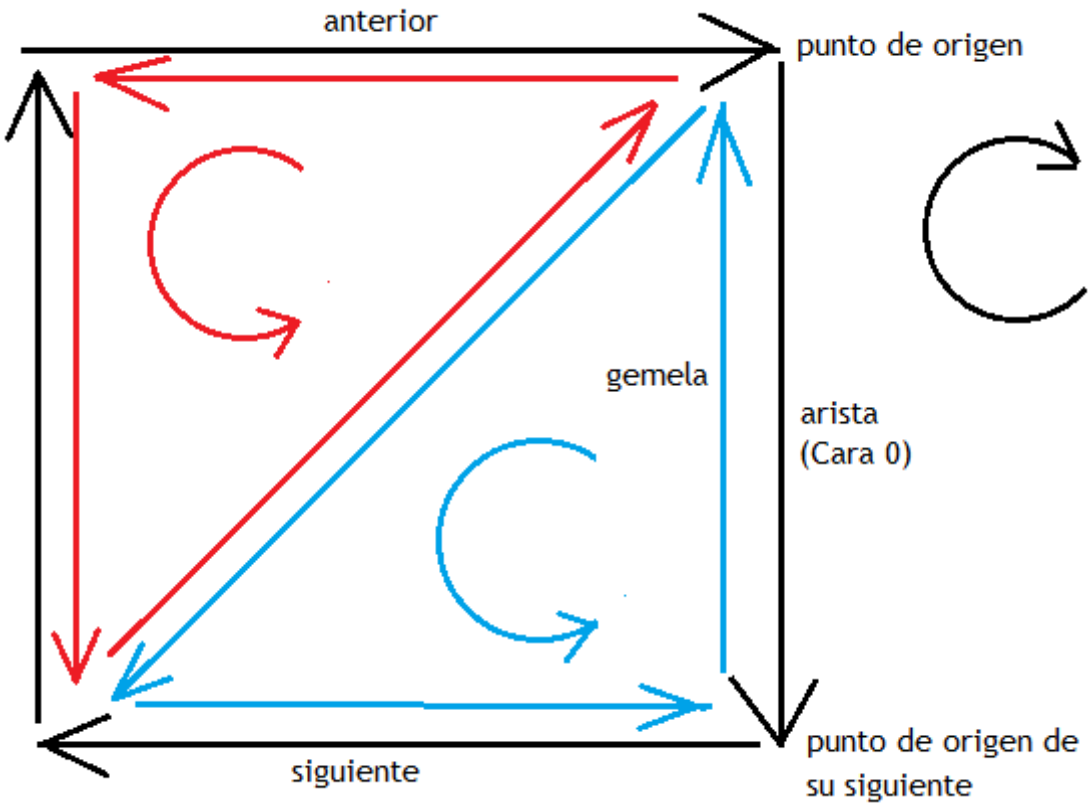
La estructura **DCEL** consiste en lo siguiente:

- Para la unión entre dos puntos, en vez de usar una arista, usamos dos, ambas dirigidas y con sentidos opuestos. Así fijamos un "orden" entre aristas de la misma cara.
- Para cada arista, guardamos la siguiente información:
    - · Punto de origen
    - · Arista anterior
    - · Arista siguiente
    - · Arista gemela (la que une los mismos puntos pero en sentido opuesto)
    - · Cara a la que pertenece
- Consideramos cara a cada triángulo de la estructura, además de la cara exterior, es decir, aristas que no forman parte de ningún triángulo.
- Para cada cara, guardamos una arista representativa.
- Por convenio, hacemos que las caras interiores tengan una orientación positiva, y la exterior (cara 0) sea la única cara con orientación negativa.

Además, esta estructura también nos permite "dibujar" la estructura de forma sencilla, ya que para ello sólo hay que dibujar una línea por cada arista: del punto origen hasta el punto origen de su siguiente arista.

La **complejidad computacional** de esta estructura es:
- O(n) para situarnos en una arista, siendo n el número de aristas.
- O(n) para situarnos en una cara, siendo n el número de caras.
- O(1) para movernos por la estructura a partir de una posición, es decir, complejidad constante.

A continuación se muestra lo que podría ser una triangulación de DELONE de 4 puntos almacenado en una estructura DCEL, con las referencias a la información de la arista vertical situada más a la derecha. Nótese también la diferencia en el sentido de la orientación de las caras interiores respecto a la exterior.



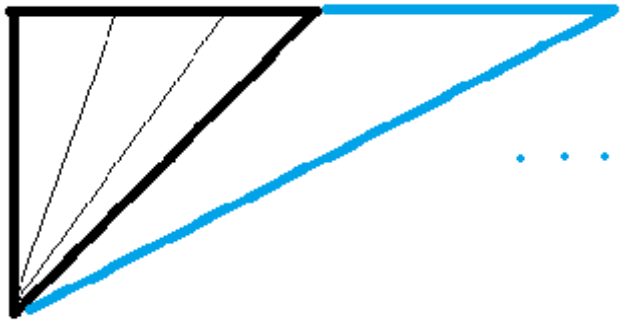
Creación del DCEL

Una vez tenemos ya la matriz de puntos, y aprovechándonos de que sabemos que los puntos del borde de la malla están intactos, procedemos a crear el DCEL.

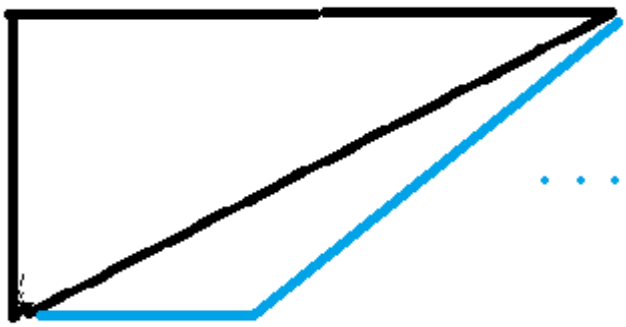
El primer paso es crear un primer triángulo, que siempre estará formado por la esquina superior izquierda de la malla, su vecino derecho y su vecino inferior, todos ellos pertenecientes al borde.

Una vez hecho esto, se distinguen 4 casos que nos crearán la estructura:

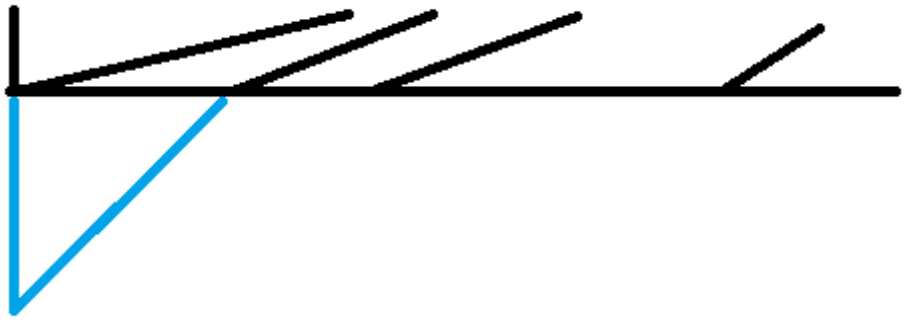
- Caso C1: Para la primera fila, vamos creando triángulos siempre añadiendo un vértice en la fila superior y uniendolo con el primer punto de la siguiente fila, así hasta llegar al final de la fila superior.



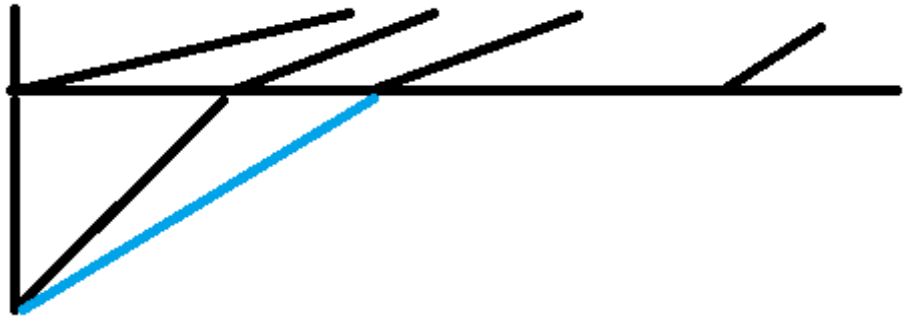
- Caso C2: Para la primera fila, ahora hay que ir creando triángulos por abajo para terminarla y poder cambiar de fila. Repetimos hasta llegar al último punto de la fila inferior



- Caso C3: Al cambiar de fila, tenemos el "techo" ya construido de la fila anterior, así que lo primero que tenemos que hacer es crearnos un primer triángulo para esta nueva fila para poder trabajar sobre ella.



- Caso C4: Para el resto de la fila, rellenamos de manera similar al caso 1 con la excepción de que ahora el techo ya lo tenemos, sólo hay que terminar los triángulos



Se puede comprobar que con este proceso obtendremos una primera triangulación del terreno, aunque esta contendrá triángulos muy alargados y no será por lo general una triangulación de DELONE, si bien esto lo podemos arreglar posteriormente mediante el "flipado" de aristas.

Hay que tener en cuenta varias cosas sobre este proceso:

- El caso C1 solo se usa para la primera fila de toda la malla, ya que necesitamos ir creando el "techo" de los triángulos. Siempre usamos para la creación el primer punto de la segunda fila, el punto más a la derecha de la triangulación de la primera fila, y el siguiente punto a la derecha de éste, hasta llegar al final.

- El caso C2 se usa tanto después del caso C1 como del caso C4, para ir rellenando el "suelo" de cada fila. Los puntos que se toman para este caso siempre son el último de la fila superior, el punto más a la derecha de la fila inferior y el siguiente punto a la derecha, hasta llegar al final.

- El caso C3 se usa en todos los casos excepto para la primera fila, ya que para ese caso creamos un primer triángulo por separado. Siempre usamos el primer punto de la fila superior, el primer punto de la inferior, y el siguiente punto de la fila superior.

- El caso C4 se usa para todas las filas excepto para la primera. Siempre cogemos el primer punto de la fila inferior, y el siguiente a la derecha de la superior, hasta llegar al final.

A la hora de programar...

Para cada fila, un posible orden de los casos, con sus respectivas condiciones para aplicar sólo los que necesitemos, sería **C3-C1-C4-C2**. Nótese que algunos de éstos se aplican varias veces, hasta llegar al final de la fila.

Es importante recordar que para el DCEL, cada arista representada gráficamente son en realidad 2 aristas, una en cada sentido, por lo que eso también hay que tenerlo en

cuenta a la hora de programar. Las aristas se van almacenando según su orden de creación en un array de aristas, cuya dimensión máxima podemos calcular aplicando la fórmula de Euler para grafos (lo mismo para las caras).

También, a la hora de crear cada triángulo, hay que fijarle una cara, una arista que forma esa cara, y para cada arista, fijar cual es su siguiente, anterior, gemela, su cara y su punto de origen.

Para saber qué puntos tiene que seleccionar en cada paso el algoritmo, lo que hacemos es usar una serie de aristas y puntos auxiliares e ir actualizándolos según nos convenga.

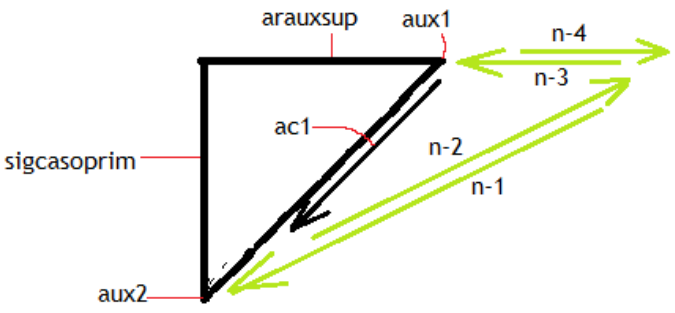
Las aristas/puntos auxiliares usadas en la creación del DCEL son las siguientes:
- arauxsup: arista de la fila superior más a la derecha.
- arauxinf: arista de la fila inferior más a la derecha.
- sigcasoprim: arista que une el primer punto de la fila superior con el primero de la inferior
- ac1: (arista crítica) última arista diagonal creada, perteneciente a la cara exterior.
- ac2: (arista crítica)
- aux1: punto más a la derecha de la fila superior perteneciente a la triangulación.
- aux2: punto más a la derecha de la fila inferior perteneciente a la triangulación.

A continuación, una representación gráfica para ver mejor la creación del DCEL a la hora de programar. Nótese que las aristas creadas tienen asociado un valor que va desde n-1 hasta n-4, donde n representa el número total de aristas hasta el momento, y cada vez que creamos unas aristas nuevas lo actualizamos (es decir, n es un valor que usamos para saber donde hay que crear las últimas aristas dentro del array de aristas).

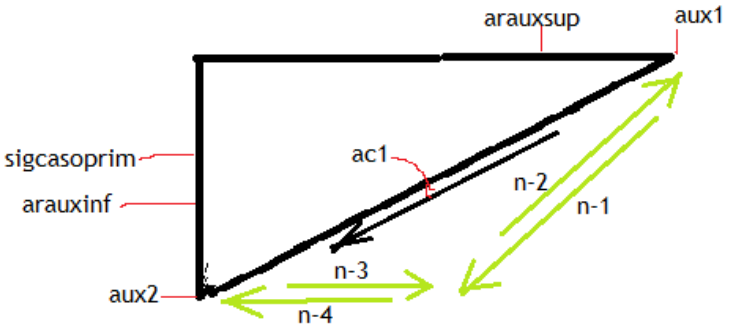
En la imagen los valores de cada cosa apuntan al valor que tienen antes de aplicar cada caso, y viene indicado para cada uno cuales habría que actualizar.

Nótese también que en cada triángulo, hay que fijar bien todo lo perteneciente al DCEL, es decir, modificar atributos de aristas y caras ya creadas anteriormente para que todo sea correcto, cambiar aristas siguientes y anteriores, gemelas...

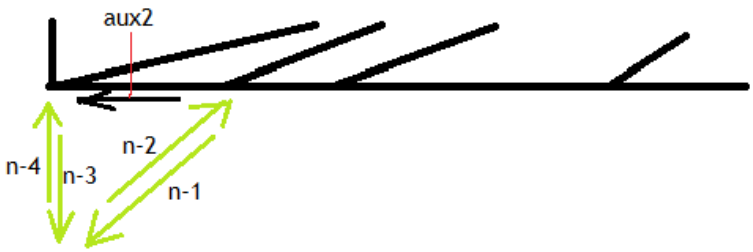
También se actualizan internamente las aristas y puntos auxiliares.



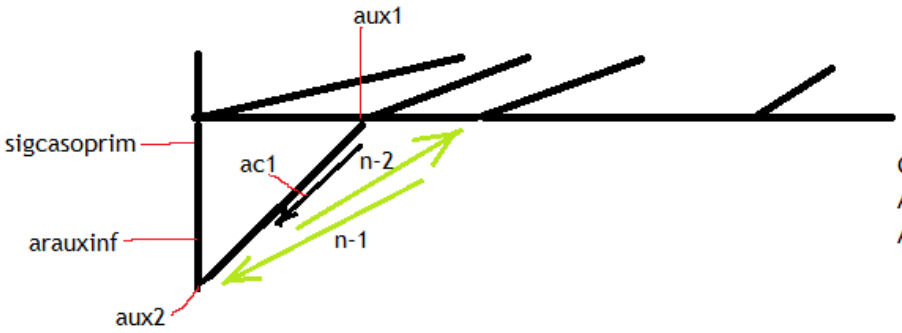
C1:
Actualizar arauxsup
Actualizar aux1
Actualizar ac1



C2:
Si aux2 es el primero, añadimos ac2
Actualizar ac1
Actualizar arauxinf
Actualizar aux2



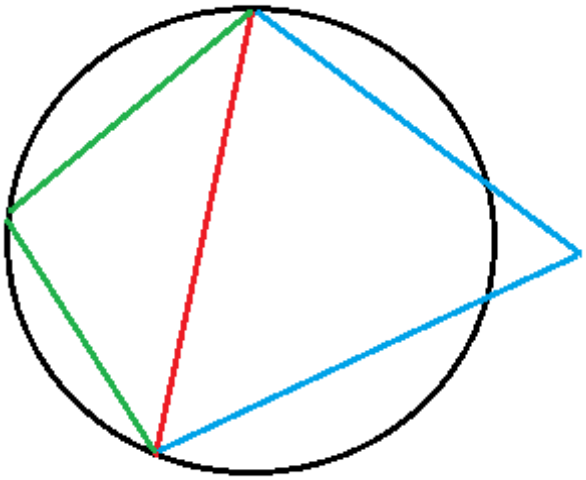
C3:
Actualizar arauxsup
Actualizar sigcasoprim
Actualizar arauxinf
Actualizar aux1
Actualizar aux2
Actualizar ac1



C4:
Actualizar ac1
Actualizar aux1

Una vez tenemos ya creada toda la triangulación, "flipamos" las aristas para obtener una triangulación de DELONE.

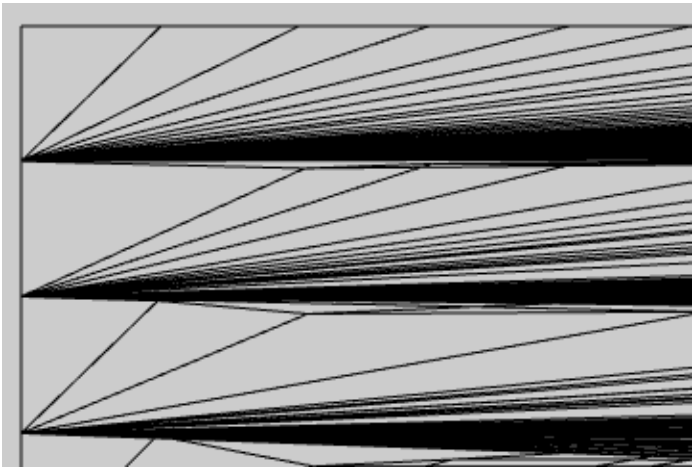
Para decidir si hay que flipar o no una arista (en realidad flipamos pares de aristas), nos fijamos en si para una circunferencia determinada por los extremos de la arista (en rojo) y el vértice que cierra un triángulo por un lado de la arista, contiene al vértice que cierra al triángulo por el otro lado.

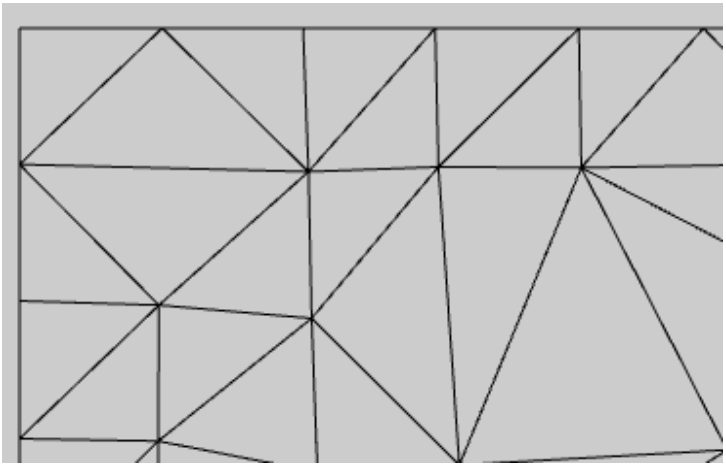


Si lo contiene, hay que flipar la arista. Si no lo contiene, esa arista es considerada adecuada para una triangulación de DELONE.

Nótese que las aristas pertenecientes al cierre convexo del terreno no se "flipan" nunca.

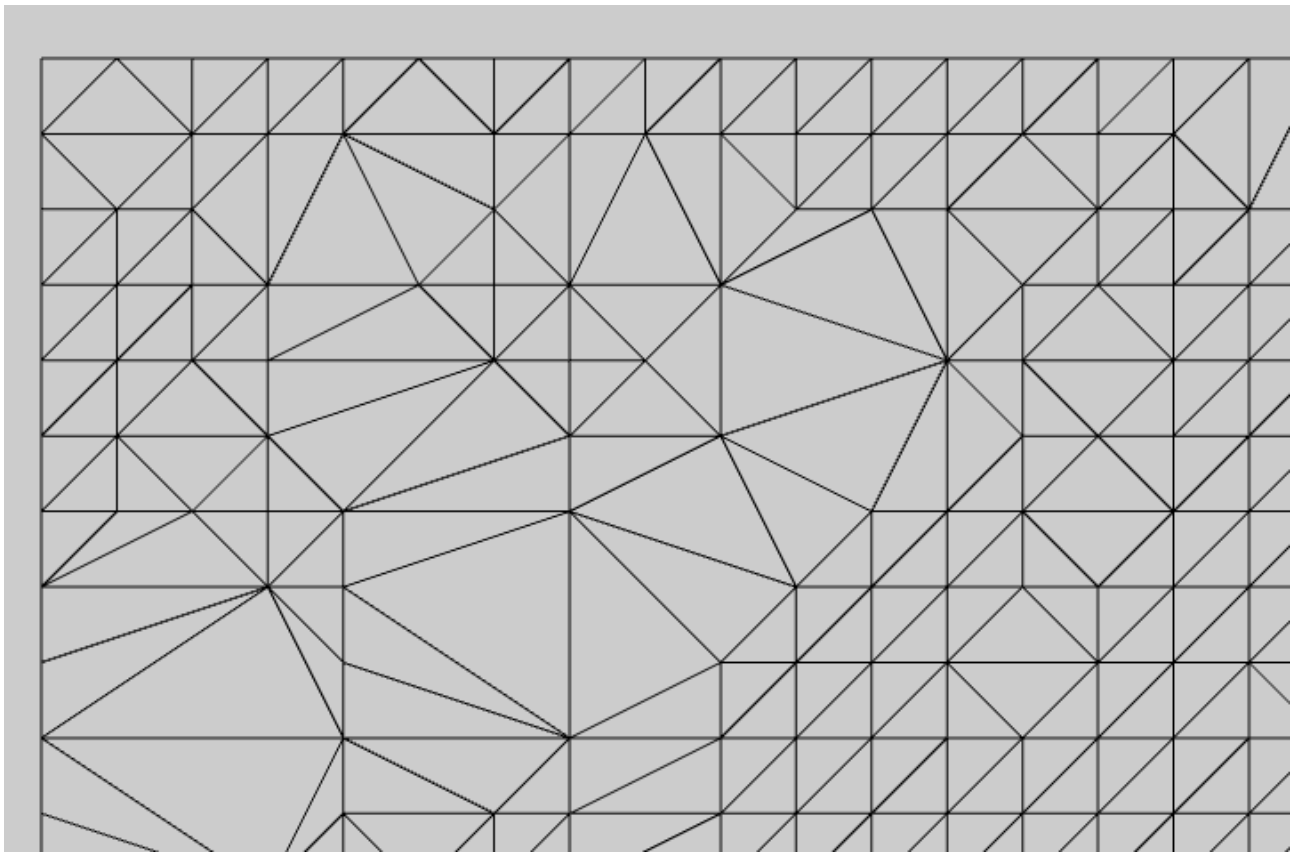
Comparativa de la misma malla, una imagen antes de flipar las aristas y otra después:

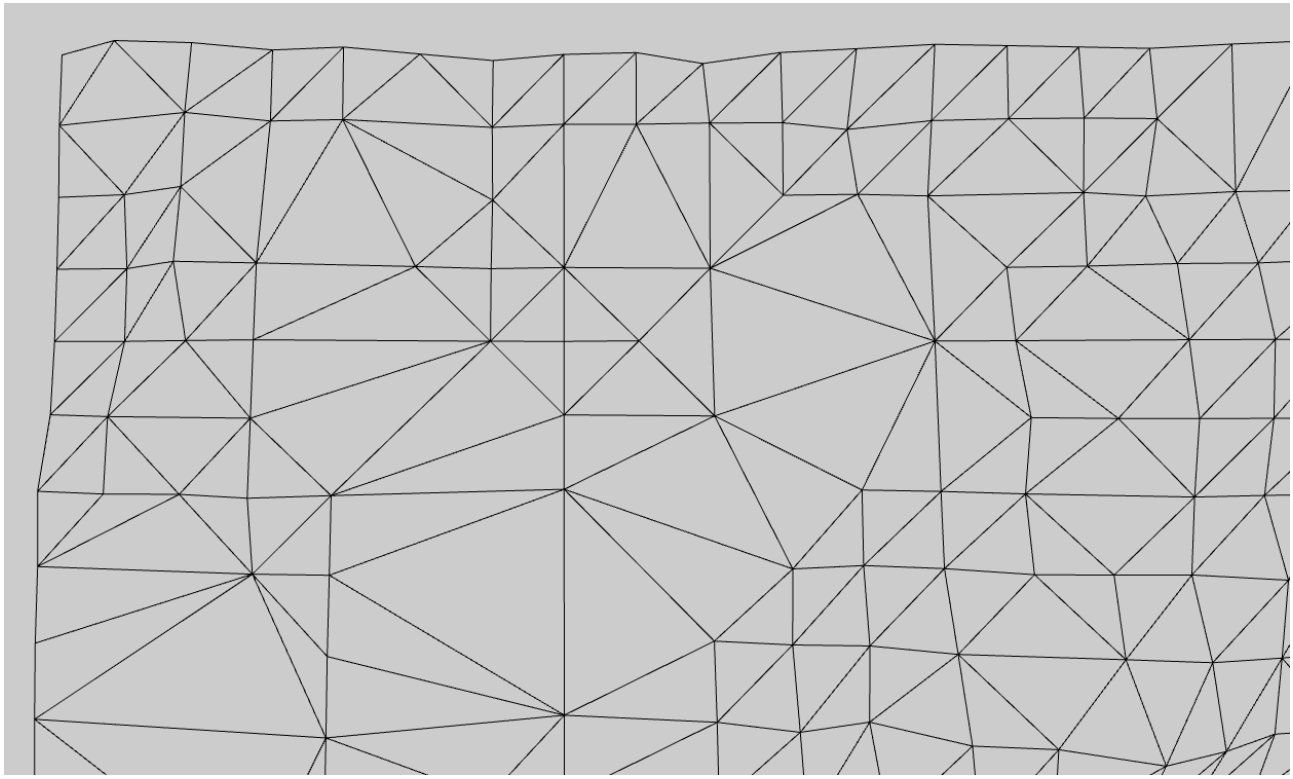




Visualización del DCEL: 2D y 3D

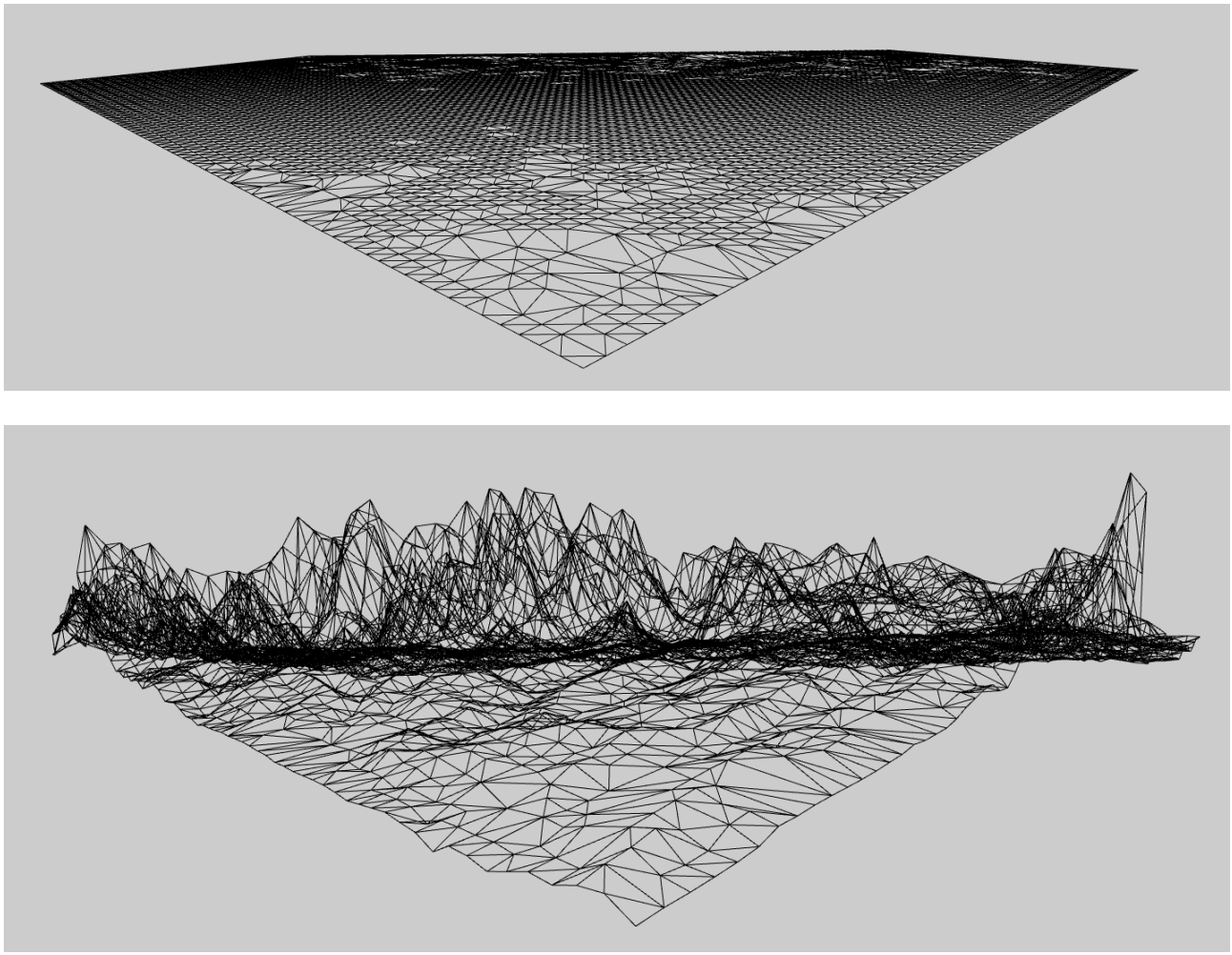
Si deseamos que el programa nos muestre el modelo DCEL, tenemos dos opciones: verlo en 2D y verlo en 3D. Para ello, hay que incluir en el método **draw()** las correspondientes llamadas a funciones de la clase Prueba, **dibujar2D()** y **dibujar3D().**





En la página anterior, se muestra una porción del mismo terreno en 2D y en 3D desde una vista aérea. Como se puede apreciar, la triangulación en 3D se ve algo deformada, como cabría esperar.

A continuación se muestra una comparativa de todo el terreno en 2D y en 3D.



Nótese que no se muestra un cuadrado perfecto en 2D (y en 3D también falta una porción al fondo por representar) porque sólo renderiza hasta una distancia, sin embargo, podemos mover la cámara y observar esa parte del terreno sin problemas.

Trayectorias de desagüe

Calcular la trayectoria de desagüe desde un punto viene a ser equivalente a calcular el camino por el que se desplazaría una gota de agua soltada en ese punto, bien hasta que se estanque o bien hasta que se salga del terreno.

Primero de todo, si el punto pasado está fuera de la malla triangular, alteramos los datos para que esté dentro, y si justo cae en un punto de la malla, calculamos 4 caminos posibles diferentes, ya que si no no podemos determinar bien la cara a la que pertenece (aún así es posible que el punto caiga en una arista, en ese caso seleccionamos la primera cara que encuentre que lo contenga, ya que hay un 50% de probabilidades de que caiga por ahí, pero realmente no sabemos como es el terreno en ese punto así que nada de lo que hiciéramos sería del todo fiable).

Después hay que saber a qué cara de la triangulación pertenecen las coordenadas (x,y) pasadas como parámetro.

Una vez sabemos la cara a la que pertenece el punto, calculamos el vector normal al plano de esa cara, y modificamos la coordenada Z de ese vector para meterlo en el propio plano. Esa dirección es en la que caerá la gota en el primer plano. Ahora hay que mirar con qué arista del triángulo intersecciona esa dirección desde el punto, y hallamos el punto de intersección de ambas rectas, que será el punto hasta el que se mueva la gota en un primer momento, ya que ahí puede haber un cambio de pendiente y modificará su trayectoria.

Como sabemos el punto en el que intersecciona y la arista, nos cambiamos a la cara de su gemela y volvemos a calcular la trayectoria. Repetimos este proceso hasta que nos salgamos del terreno o bien hasta que la gota vuelva a una cara por la que hemos venido.

Cuando esto ocurre, quiere decir que la gota va a quedarse estancada en esa arista, ya que ambos planos la mandan hacia el otro, así que en ese momento pasamos a desplazarnos solo por aristas (es como si la gota estuviera en el pico de una V).

La primera dirección en la que se mueve es hacia el vértice de menor altura extremo de esa arista (de 2 posibles). Una vez ha llegado a ese vértice, puede haber varias aristas que salgan de él, por lo que hay que mirar todas las posibilidades y seleccionar aquella de mayor pendiente, que será por la que la gota se desplazará. Una vez todos los vértices vecinos del actual se encuentran a una altura mayor o igual que actual, la gota se queda estancada y terminamos el método.

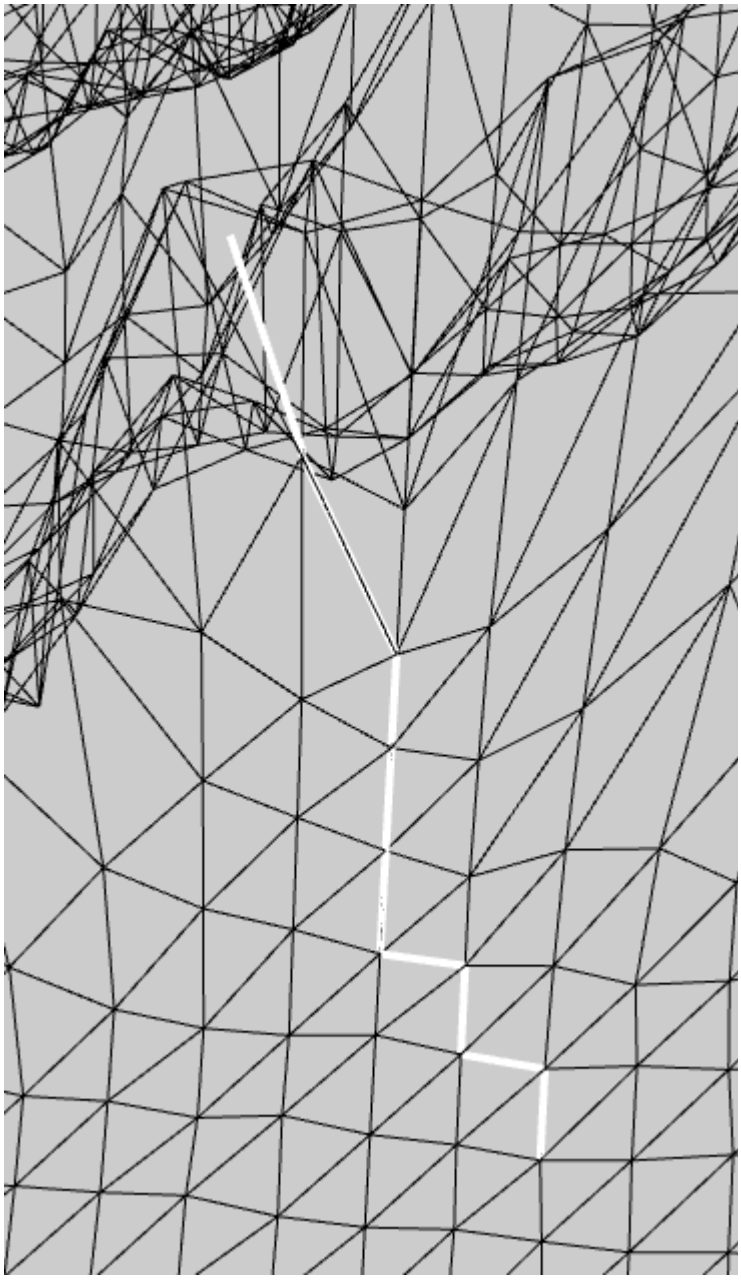
Visualización de trayectorias de desagüe

La visualización de una trayectoria de desagüe se realiza en 3D y en color blanco. Se recomienda visualizarlo a la vez que el modelo DCEL en 3D para tener una referencia visual.

Para llamar a ésta función, hay que incluir en el **draw()** una llamada a la función de la clase Prueba **caminoGota(double x, double y).**

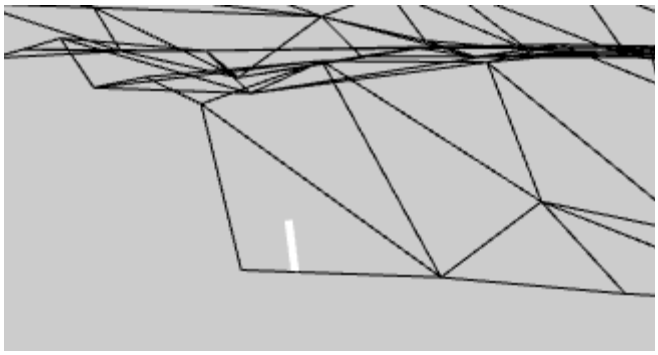
A continuación se muestran una serie de ejemplos con llamadas a esta función:

p.caminoGota(4156, 1134);



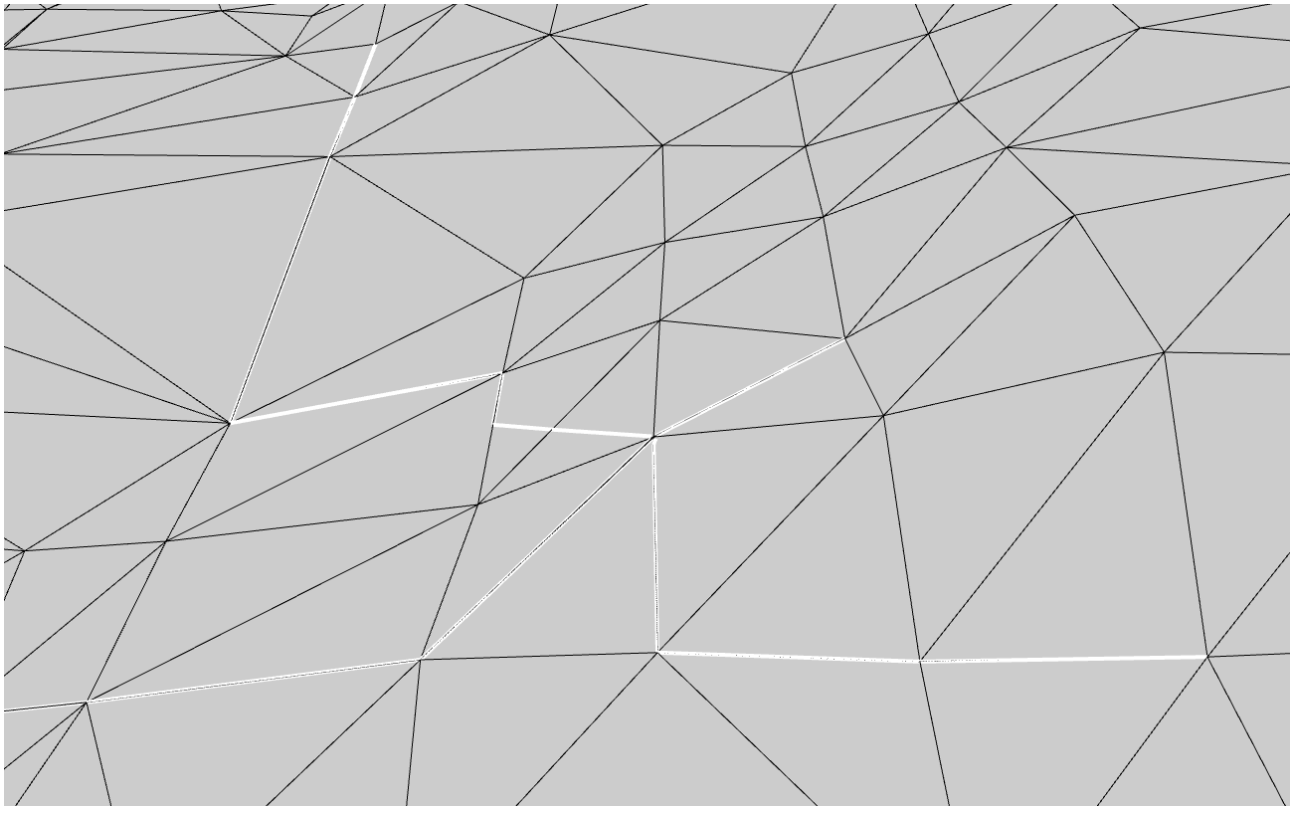
En este ejemplo, se coloca la gota en una zona montañosa y se aprecia su caída hasta que se queda estancada, sin salir del terreno.

p.caminoGota(30, 30);



Aquí, la gota nada más caer por la primera cara se sale de la malla, así que termina el proceso.

p.caminoGota(1000, 1000);



Aquí, la gota cae justo en un punto de la malla (centrado en la imagen), lo cual provoca cuatro posibles caminos (algunas partes se ven un poco mal debido al reescalado de la imagen para encajarla en un documento).

Enlaces de interés

http://www.processing.org/ - Página oficial de la herramienta Processing

http://mrfeinberg.com/peasycam/ - Página del creador de la cámara PEASYCAM

http://www.holmes3d.net/graphics/dcel/ - Página que contiene información acerca del DCEL, en inglés.