CC5502 – Mallas Geométricas y Aplicaciones
Prof: María Cecila Rivara

Presentación
Algoritmo Lepp-Delaunay

Gabriel Sanhueza Sanhueza

Introducción

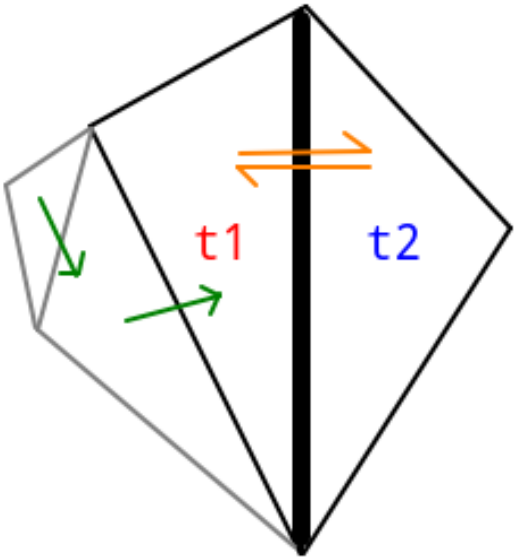
- A menudo se tienen puntos, cuyas triangulaciones son pésimas, en el sentido de no ser lo suficientemente “suaves“.
- Es posible mejorar estas triangulaciones agregando puntos convenientes que la armonicen visualmente.

Solución elegida

- Para ello se implementó el algoritmo de refinamiento por búsqueda de arista más larga (Lepp).
- Este algoritmo recorre la triangulación buscando triángulos cuyo ángulo más pequeño sea menor que cierta *tolerancia* elegida.
- A partir de uno de estos triángulos, va buscando los vecinos por la arista más larga, hasta encontrar con un triángulo terminal o el borde de la triangulación.

Solución elegida

- Se definen triángulos terminales como aquellos donde sus aristas más largas son la misma arista.
- Aquí, se busca hacer intercambio de aristas si los triángulos son localmente Delaunay (o inserción de centroide si no lo son).



- En caso de que la arista más larga sea de borde, se bisecta esta arista por el medio.

Arquitectura y librerías

- Arquitectura ModelView
- Lenguaje: C++
- Interfaz: Qt 5.8

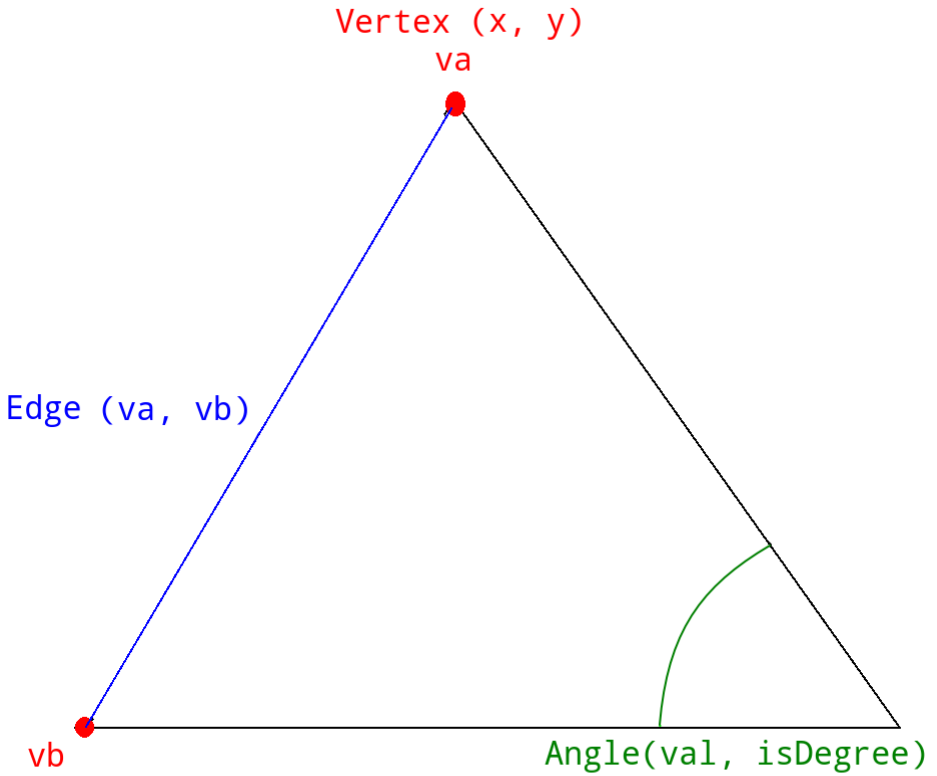
Code less.
Create more.
Deploy everywhere.

Archivos relevantes (.h,.cpp)

- Angle : Abstracción de ángulos
- Canvas : Pantalla de dibujo
- Constants : Alto y ancho de pantalla
- Edge : Detección de arista más larga
- Model : Algoritmos de refinamiento
- Triangle : Triángulos de V*ertexs*
- Vertex : Vértices (Puntos parseados)
- View : Ventana principal
- Main : Archivo para correr programa

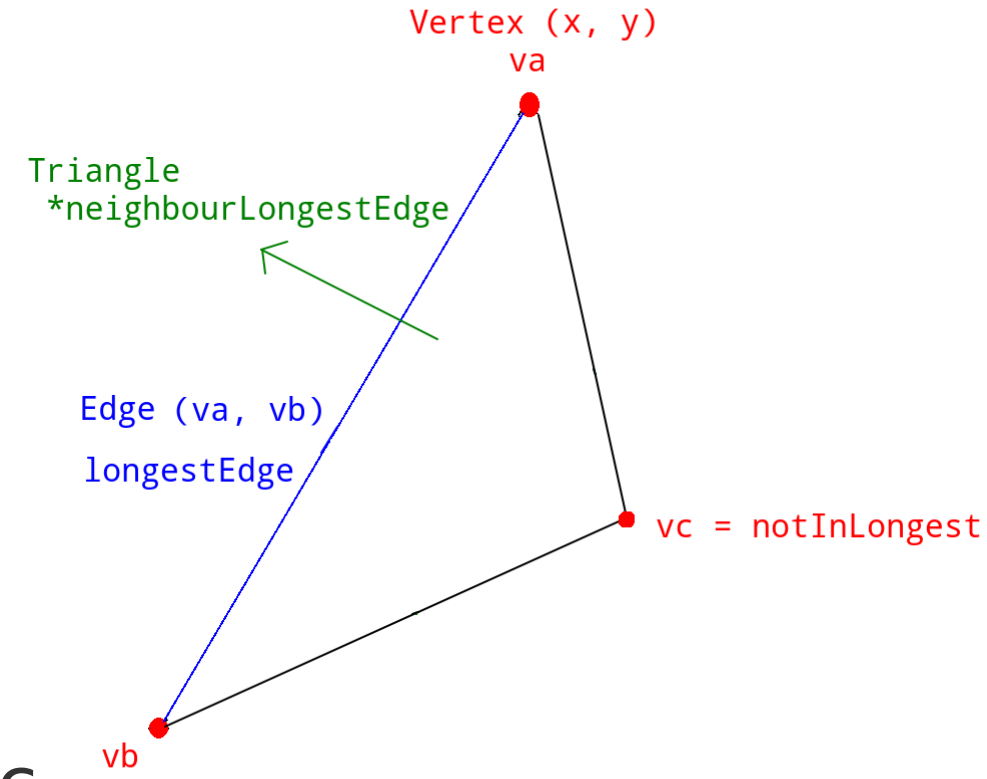
Estructuras de datos

- Vertex
  - int x
  - int y
- Edge
  - Vertex va
  - Vertex vb
- Angle
  - double val
  - bool isDegree



Estructuras de datos

- Triangle
  - Vertex va, vb, vc
  - Vertex notInLongest
  - Edge ab, bc, ca
  - Edge longestEdge
  - Triangle *ta, *tb, *tc
  - Triangle * neighbourLongestEdge

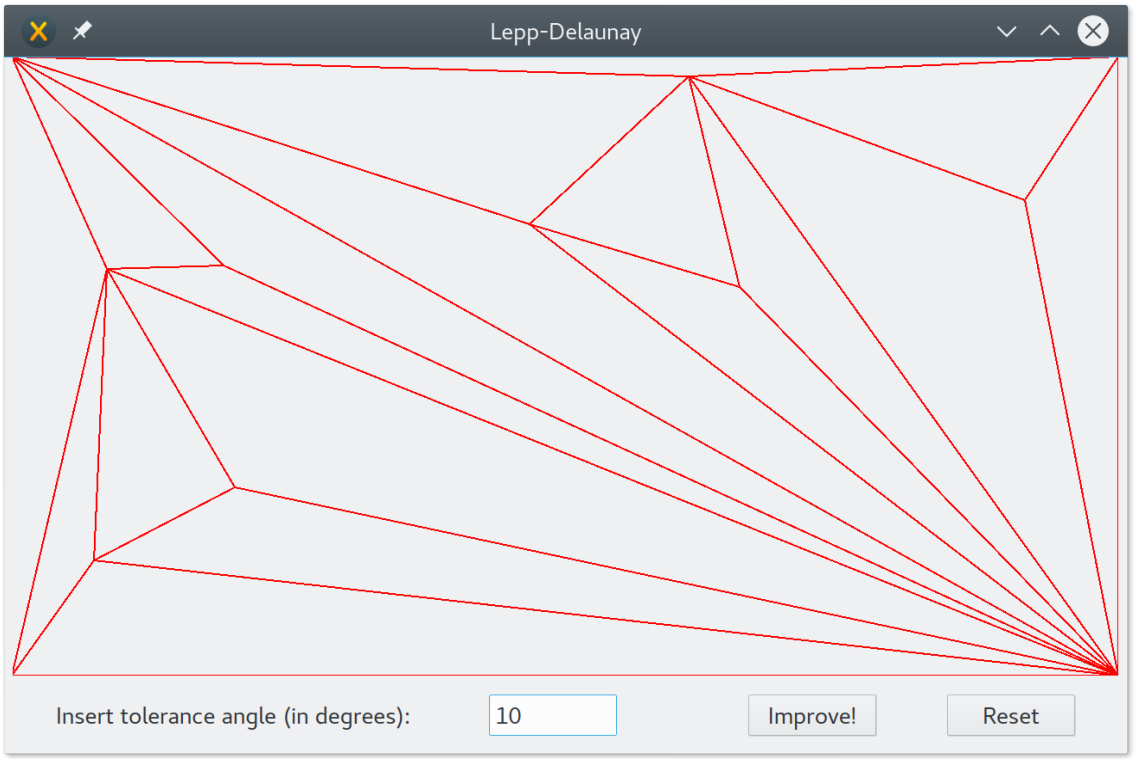


Estado inicial

- Triangulación parseada desde archivo
  - Input de tolerancia por parte del usuario

Triangles1.txt

```
1   0 0 110 473 0 720
2   0 0 95 134 110 473
3   1280 0 95 134 0 0
4   95 134 258 219 110 473
5   1280 0 258 219 95 134
6   110 473 258 219 1280 0
7   110 473 245 477 0 720
8   1280 0 245 477 110 473
9   0 720 245 477 1280 0
10  0 720 783 697 1280 720
11  0 720 599 525 783 697
12  1280 0 599 525 0 720
13  783 697 1172 553 1280 720
14  1280 0 1172 553 783 697
15  1280 720 1172 553 1280 0
16  599 525 842 452 783 697
17  1280 0 842 452 599 525
18  783 697 842 452 1280 0
19
```



Algoritmo base

- Detectar vecinos para cada triángulo
- Encontrar triangulos malos ($O < O_{tol}$)
- Mientras hayan triángulos malos:
  - Tomar uno
  - Crear lista Lepp
  - Insertar:
    - Si borde, en Centro de Edge más largo
    - Si terminales:
      - Flip de diagonales si no son localmente Delaunay
      - Centroide en otro caso
  - Actualizar triángulos malos

Encontrar triángulos malos

- Para cada triángulo de la triangulación:
  - Si ángulo mínimo menor que tolerancia:
    - Agregar a lista de triángulos malos
- Retornar lista

Lista Lepp

- A partir del triángulo recibido:
  - Si vecino es borde
    - Agregar triángulo y marcar borde
  - Si triángulo A lleva a B y el vecino más largo vuelve a A
    - Agregar triángulo y marcar terminal
  - En otro caso
    - Seguir con el vecino de arista más larga
- Retornar lista

Caso Inserción en borde

- Tomar borde más largo del triángulo
- Dividirlo
  - Crear 2 triángulos respecto al dividido
  - Borrar el dividido

Caso Flip de diagonal

- Tomar los 2 triángulos terminales
- Crear 2 triángulos nuevos con la diagonal intercambiada
- Eliminar los 2 triángulos terminales antiguos de la triangulación
- Agregar los 2 nuevos triángulos

Caso Inserción de centroide

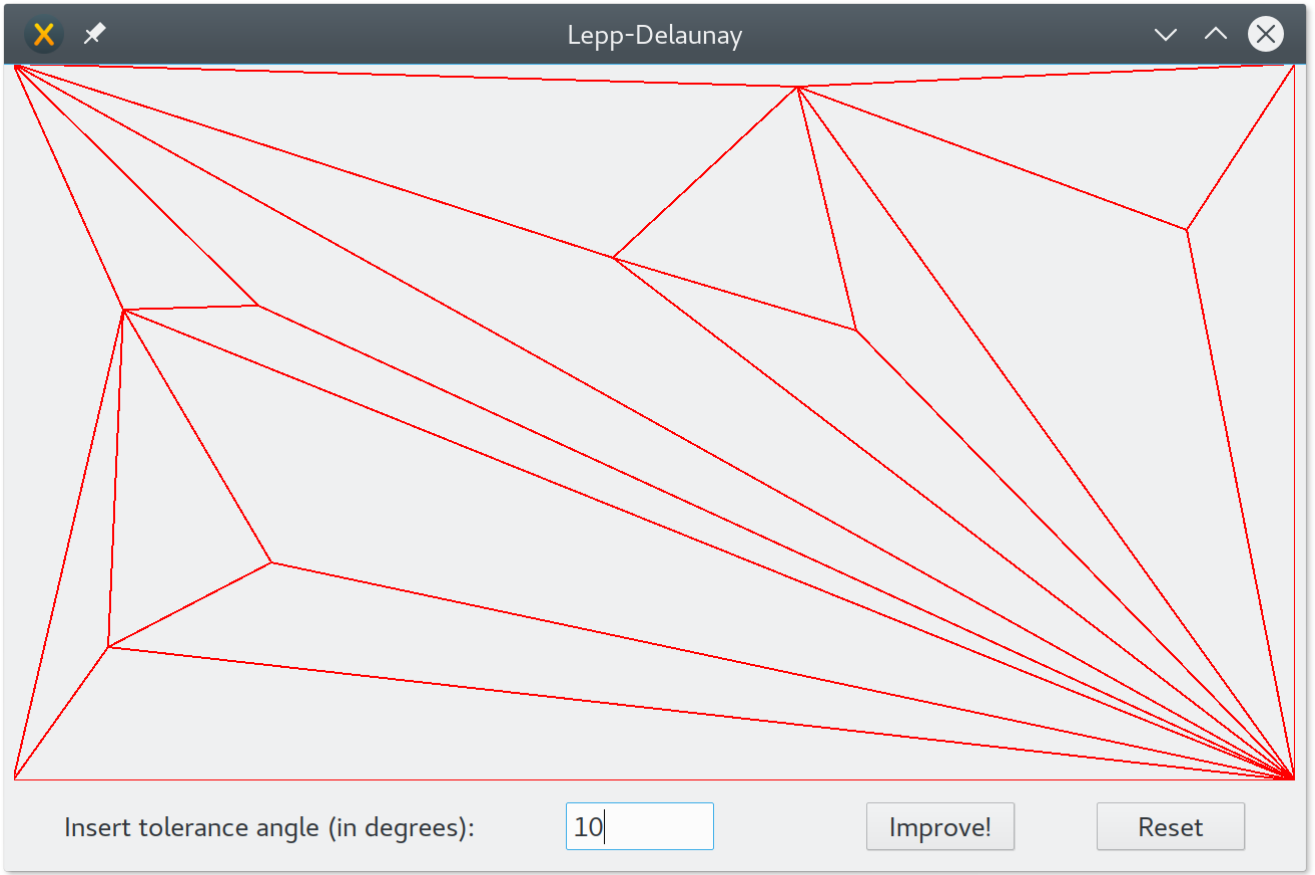
- Tomar los 2 triángulos terminales
- Calcular centroide como el promedio de sus coordenadas
- Crear 4 triángulos con 2 de los Vertex anteriores y el centroide para cada uno
- Eliminar los 2 triángulos terminales antiguos de la triangulación
- Agregar los 4 nuevos triángulos

Actualizar triángulos

- Volver a escanear la triangulación
- Detectar los ángulos mínimos nuevamente

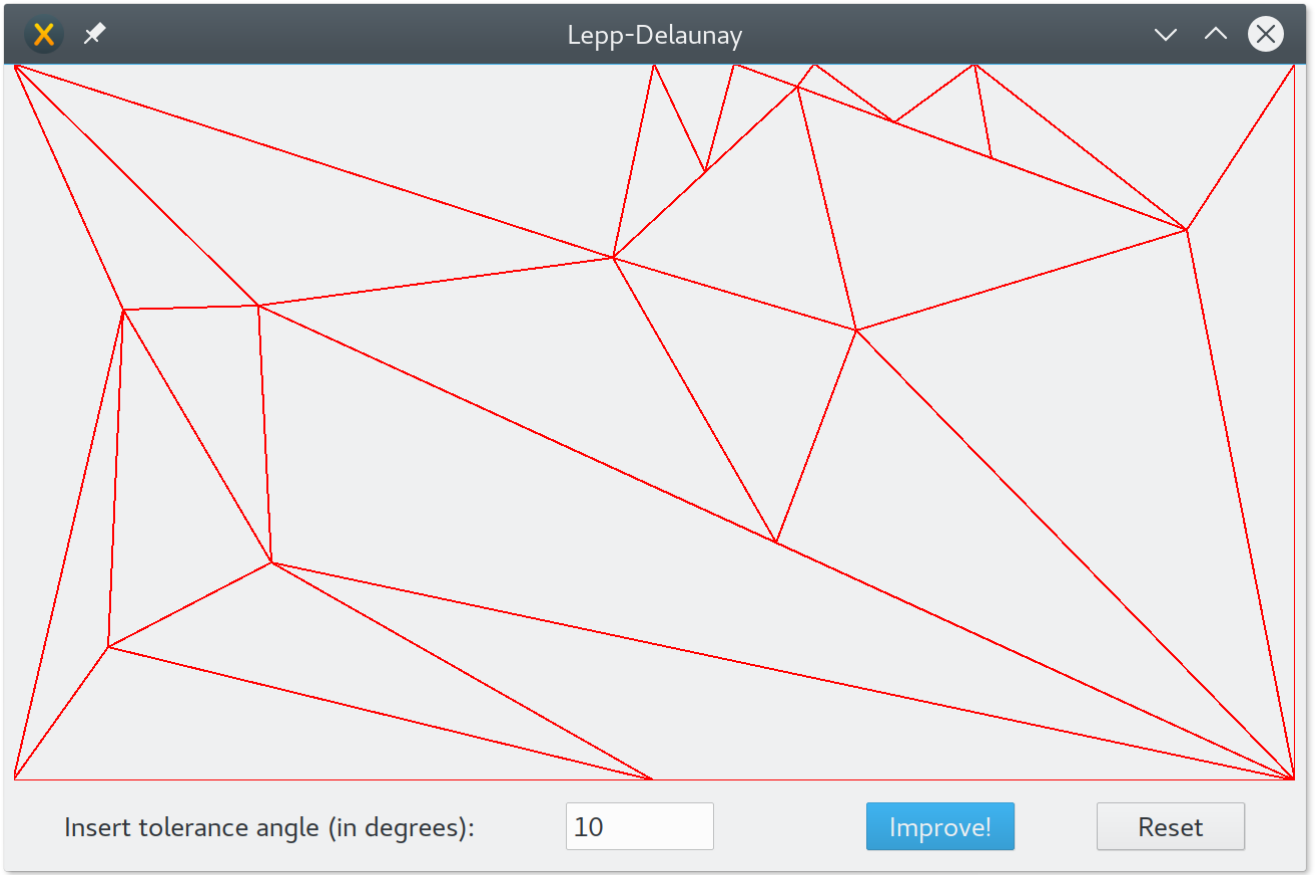
Resultados: Test 1

- Triangulación original



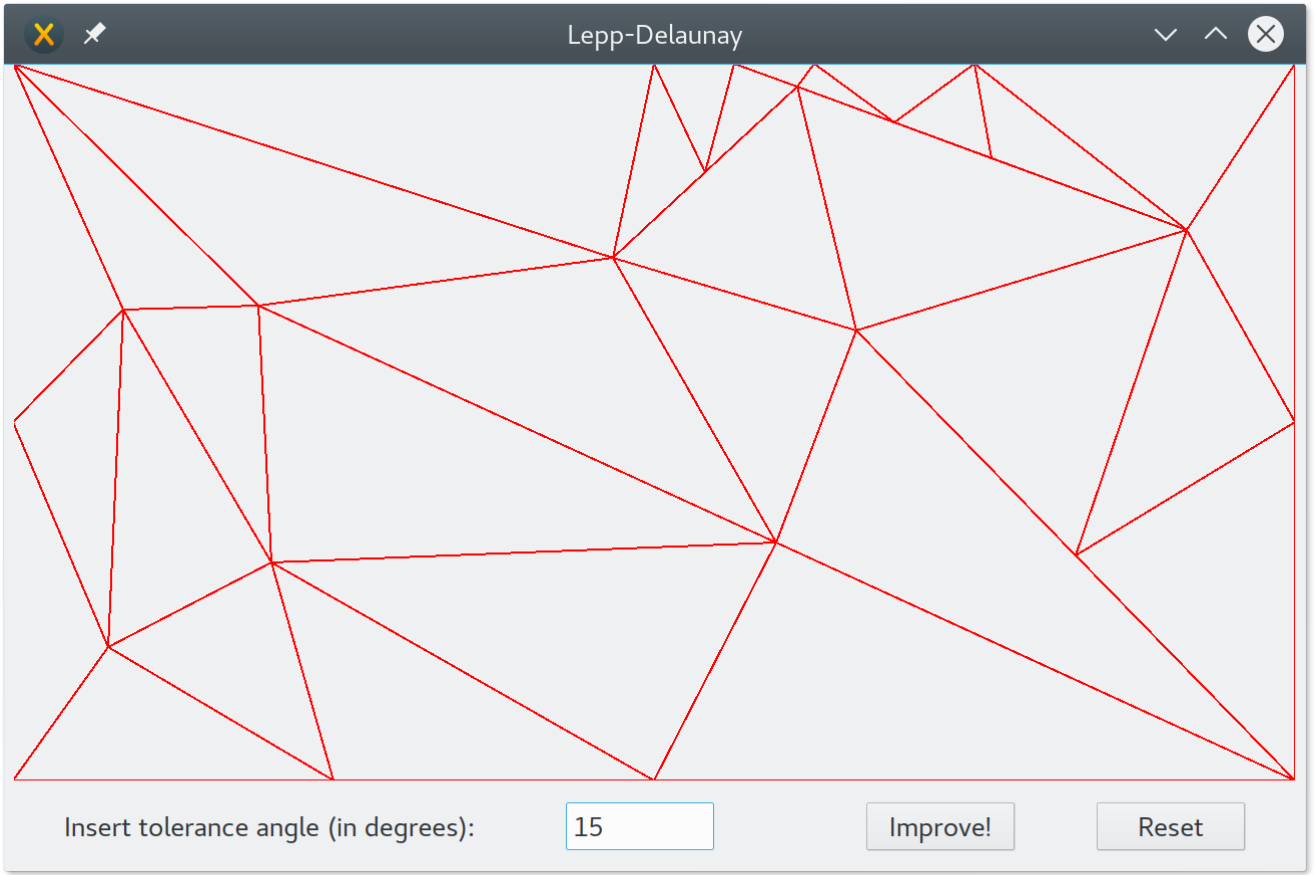
Resultados: Test 1

- Refinamiento con $\alpha_{tol} = 10°$



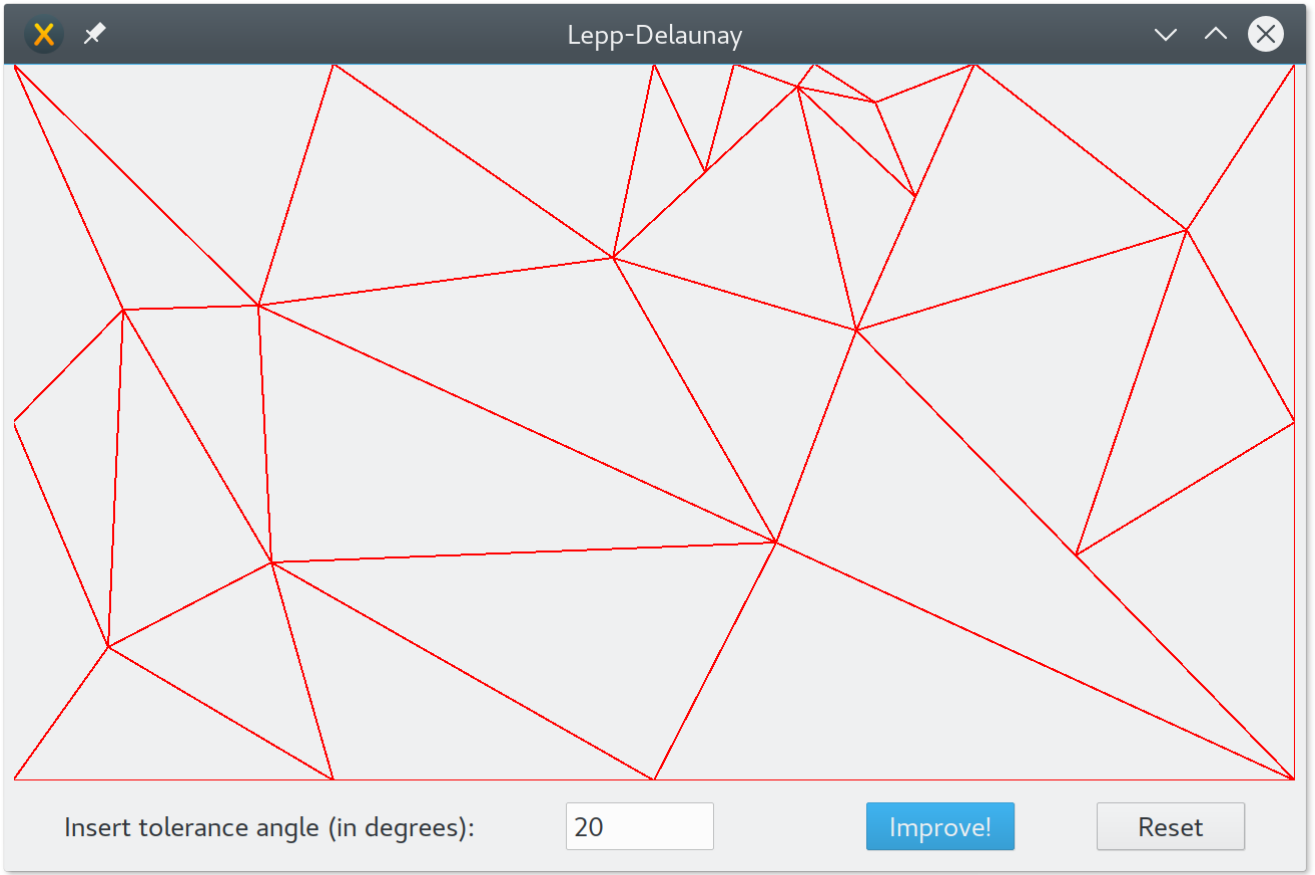
Resultados: Test 1

- Refinamiento con $\alpha_{tol} = 15°$



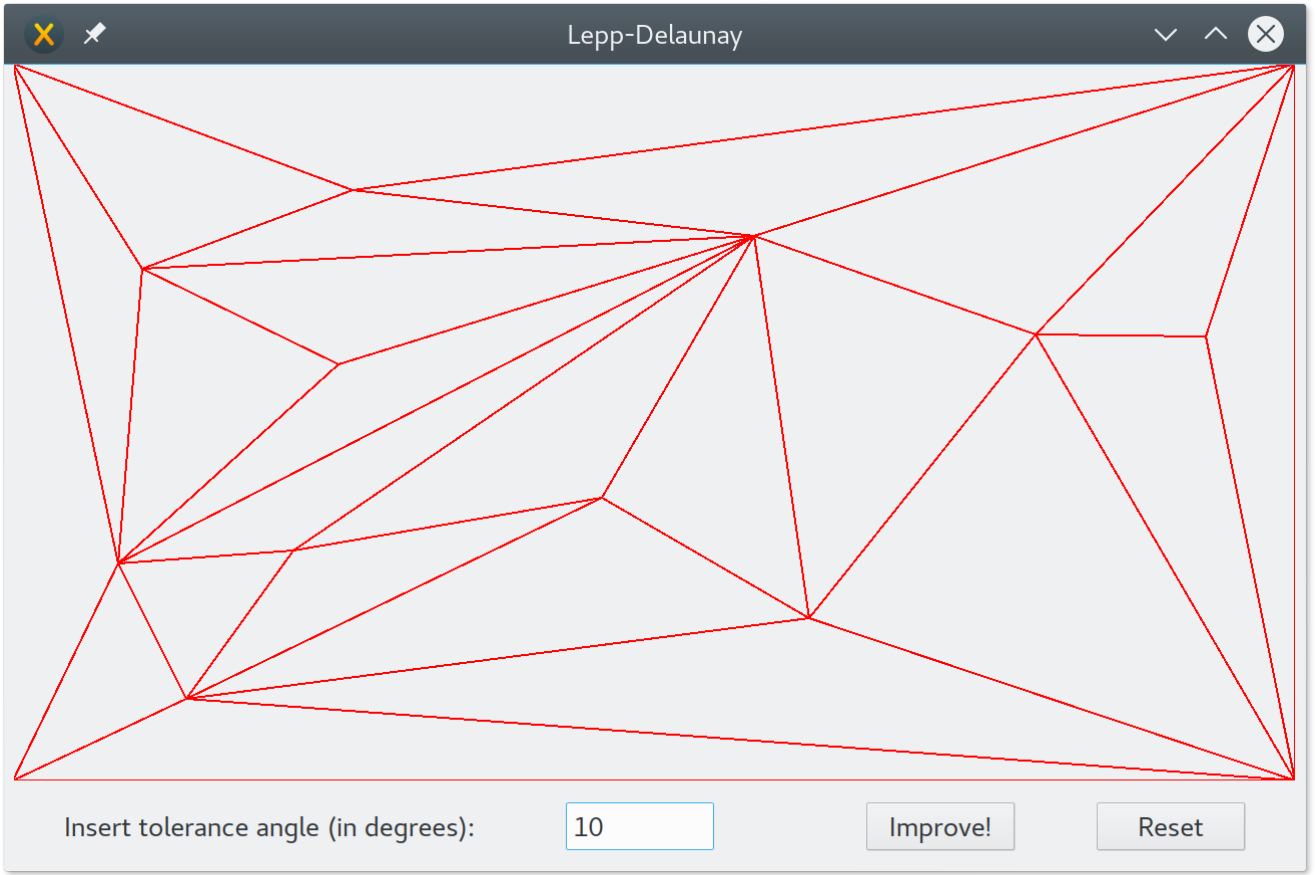
Resultados: Test 1

- Refinamiento con $O_{tol} = 20°$



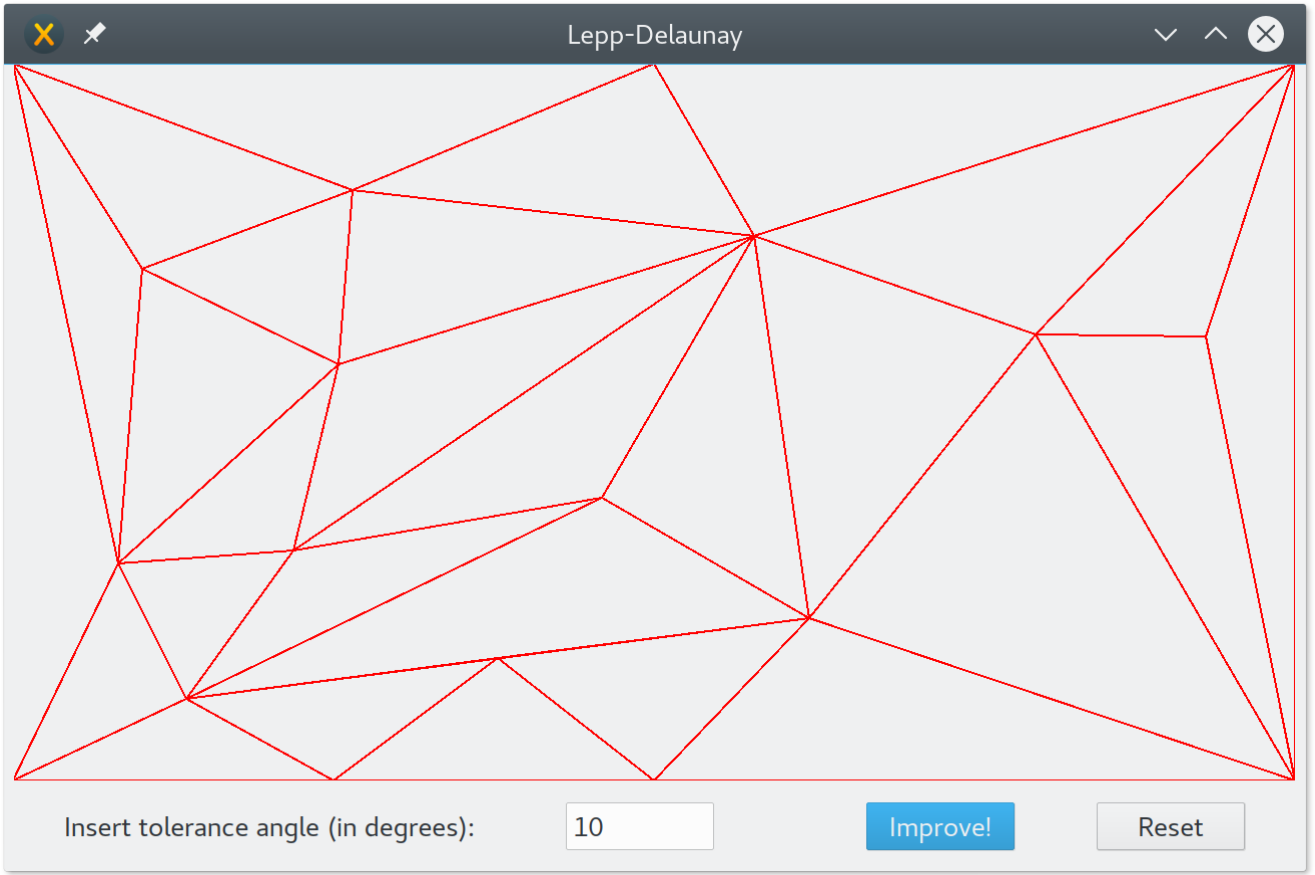
Resultados: Test 2

- Triangulación original



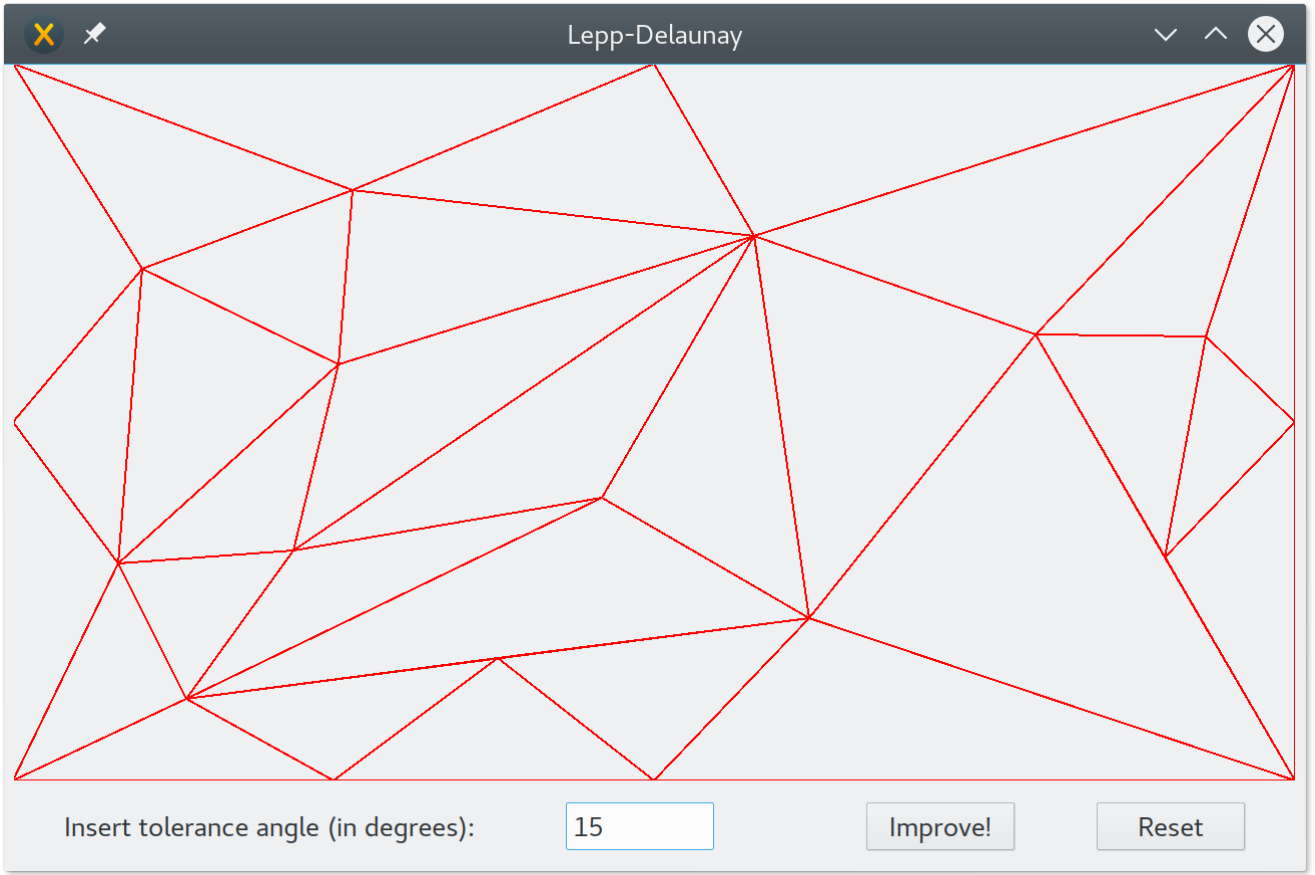
Resultados: Test 2

- Refinamiento con $\sphericalangle_{tol} = 10°$



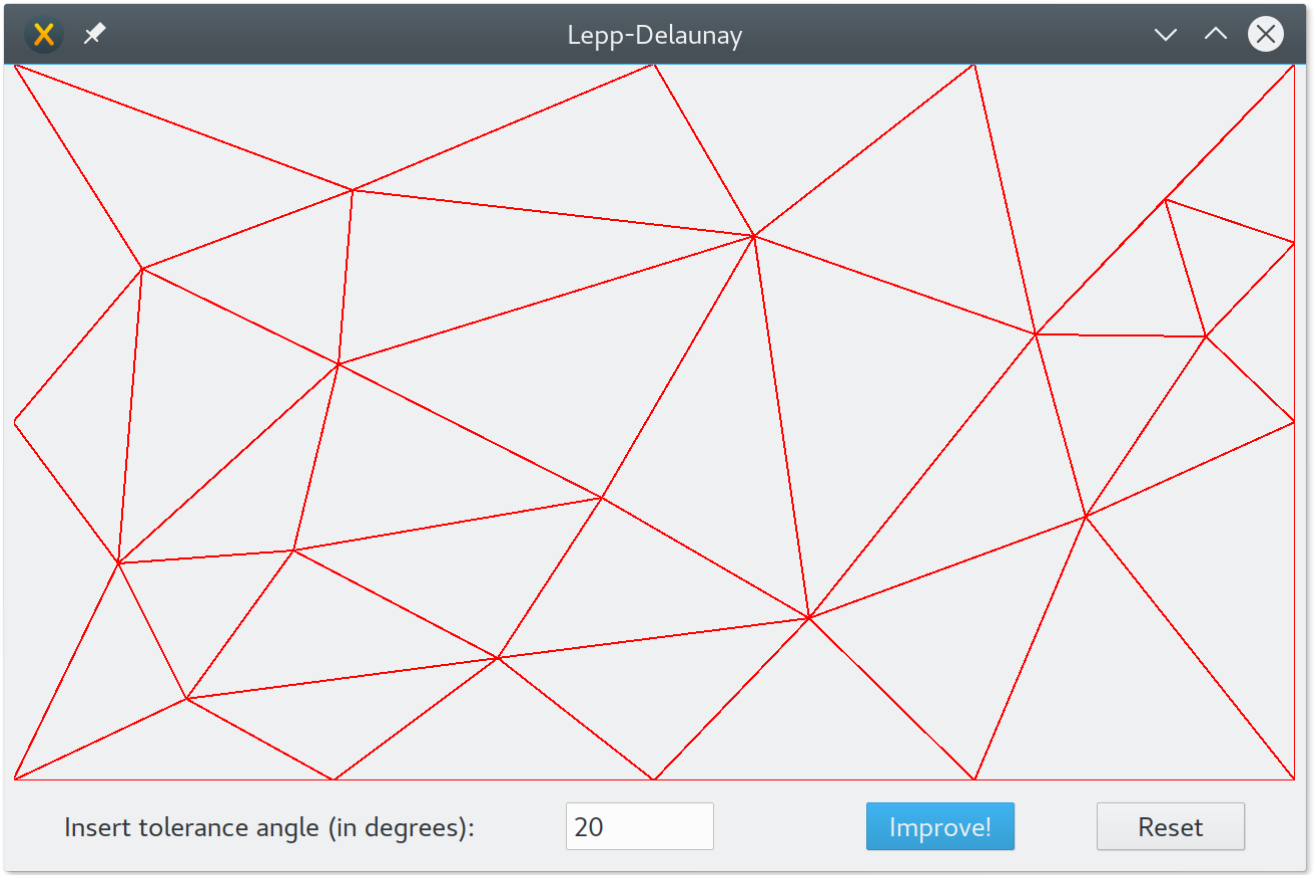
Resultados: Test 2

- Refinamiento con $\sigma_{tol} = 15°$



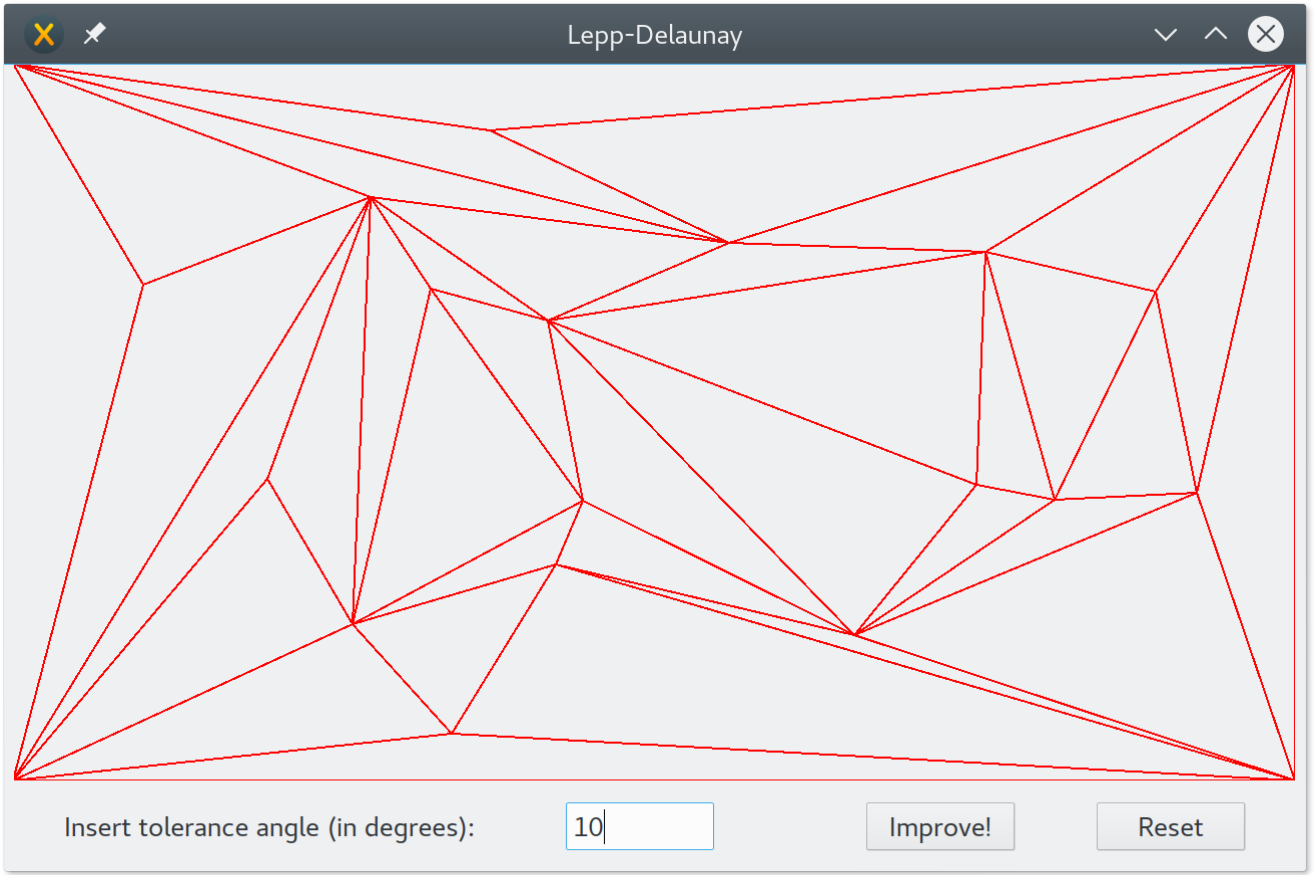
Resultados: Test 2

- Refinamiento con $O_{tol} = 20°$



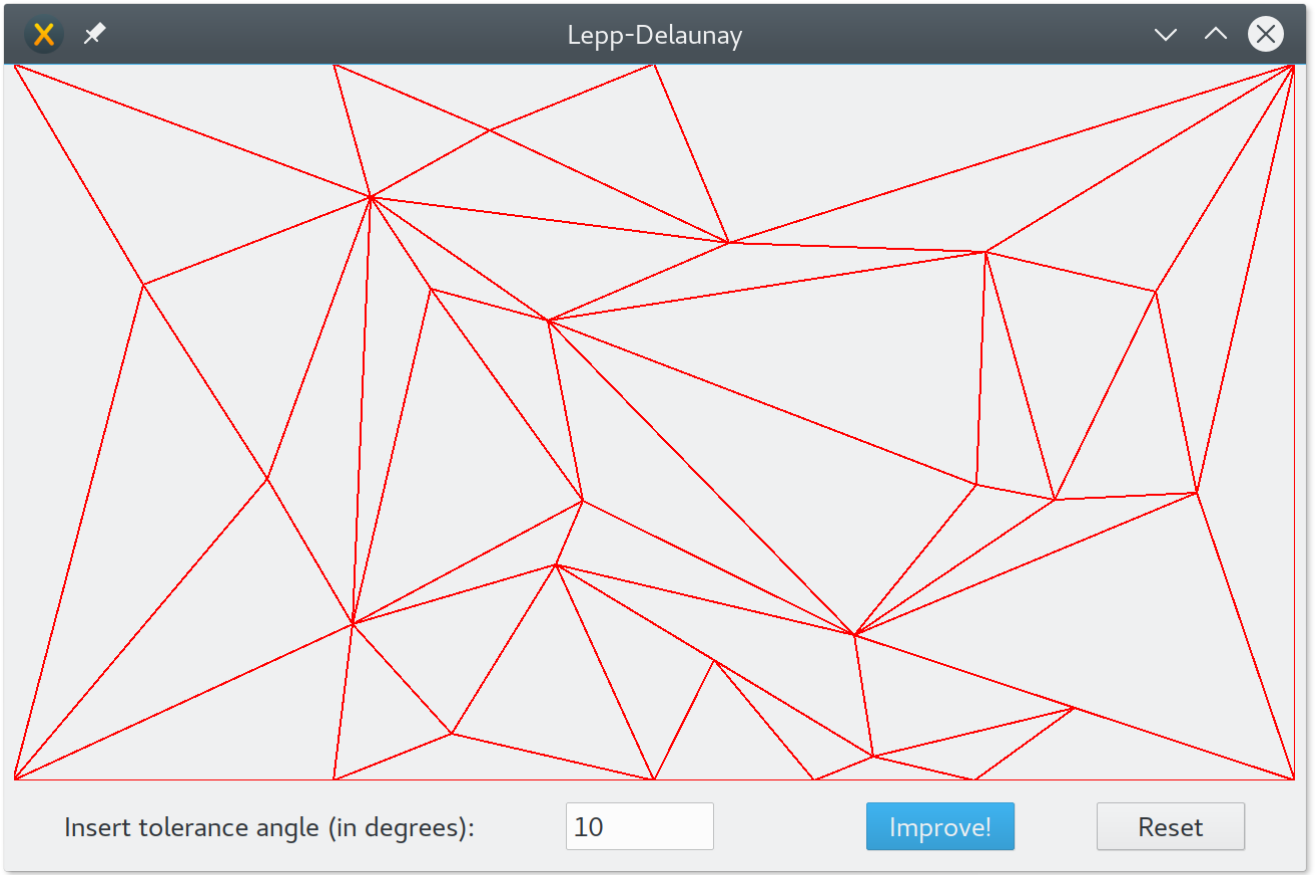
Resultados: Test 3

- Triangulación original



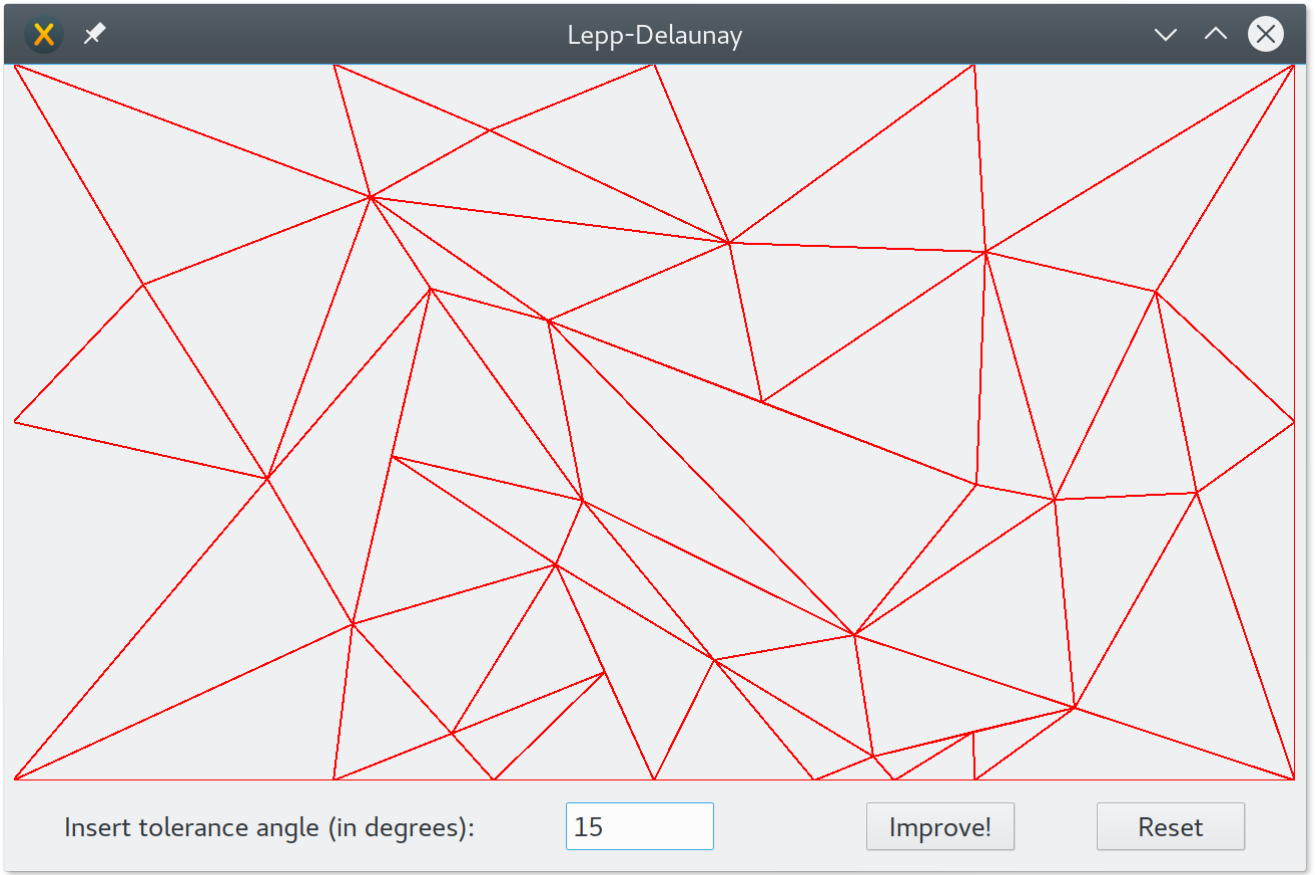
Resultados: Test 3

- Refinamiento con $\sphericalangle_{tol} = 10°$



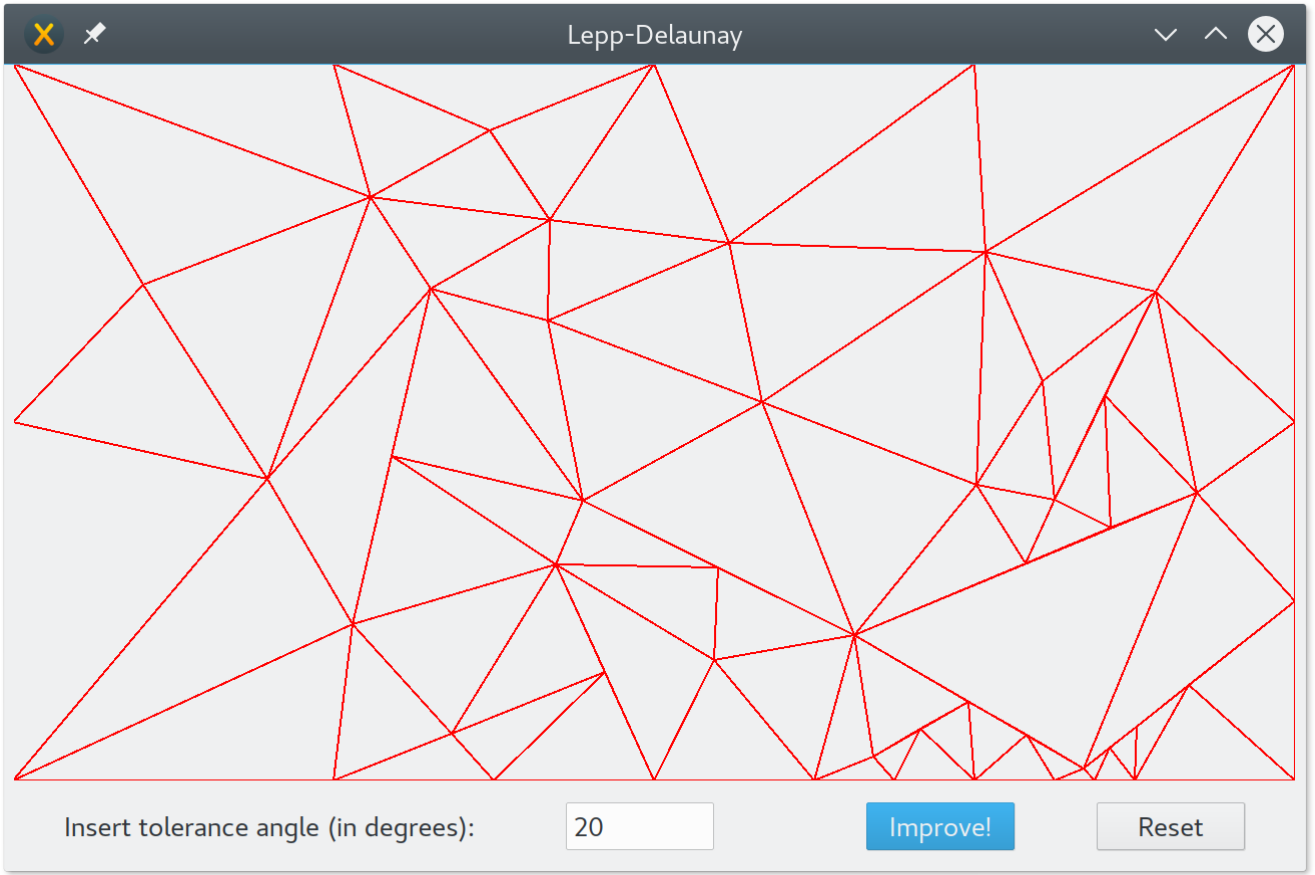
Resultados: Test 3

- Refinamiento con $\sphericalangle_{tol} = 15°$



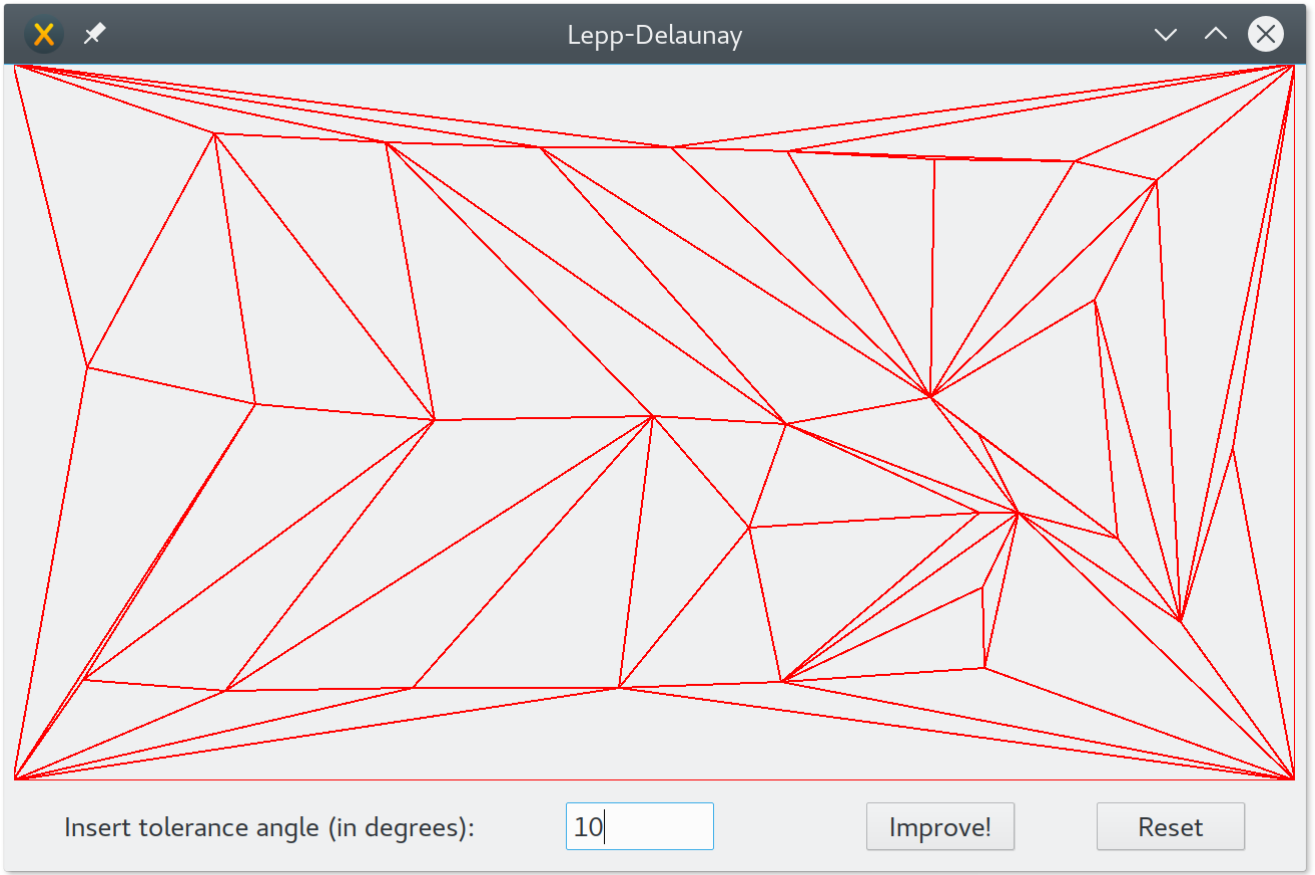
Resultados: Test 3

- Refinamiento con $O_{tol} = 20°$



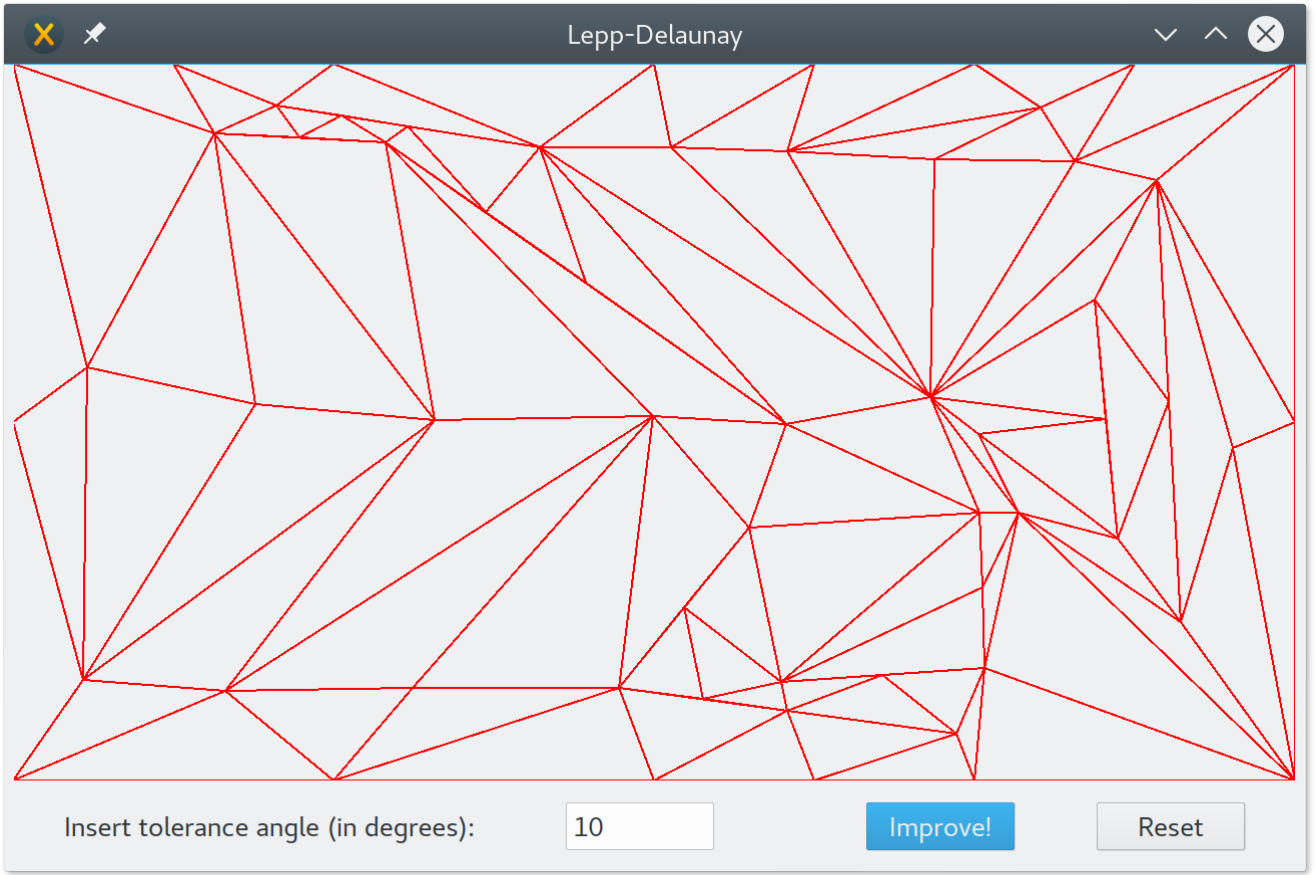
Resultados: Test 4

- Triangulación original



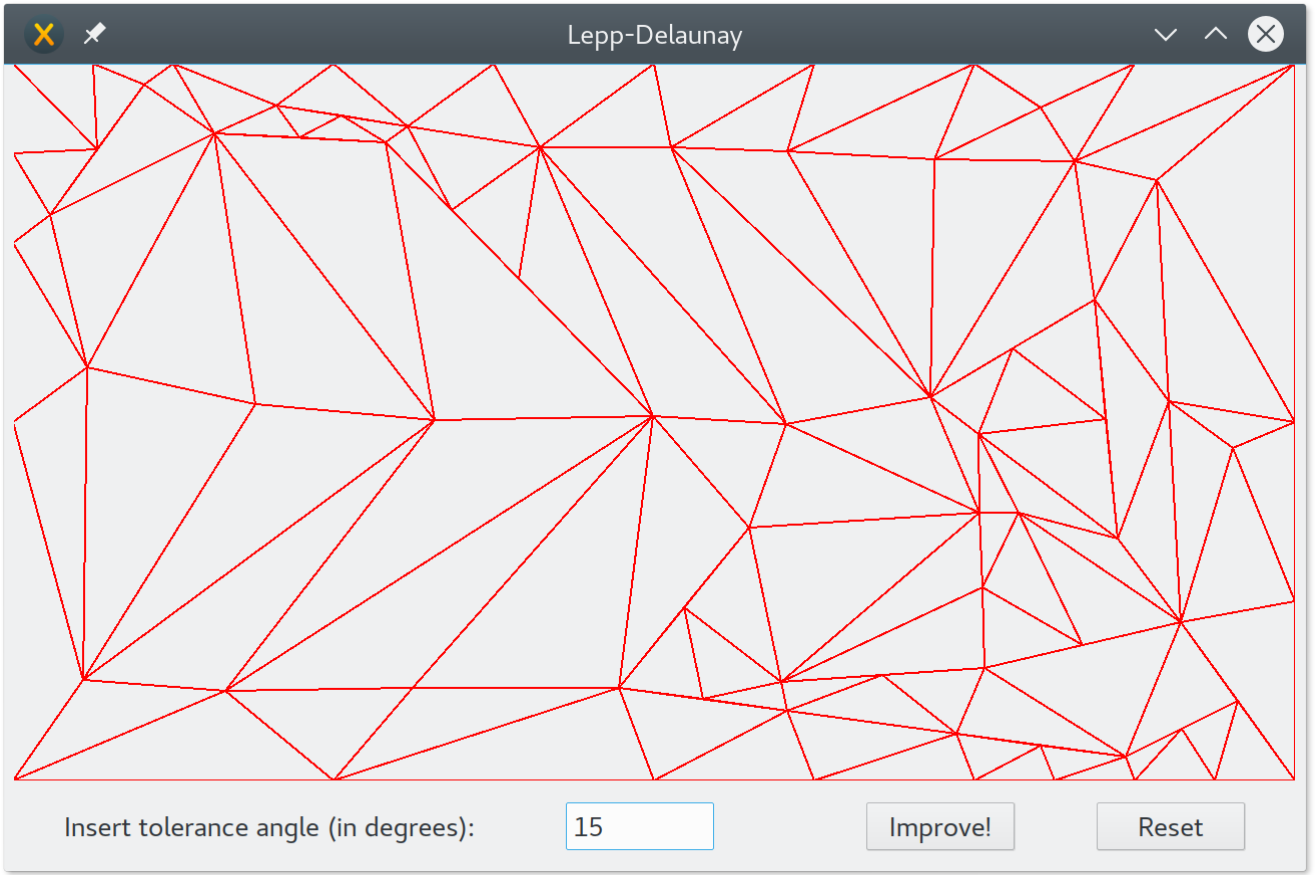
Resultados: Test 4

- Refinamiento con $\sphericalangle_{tol} = 10°$



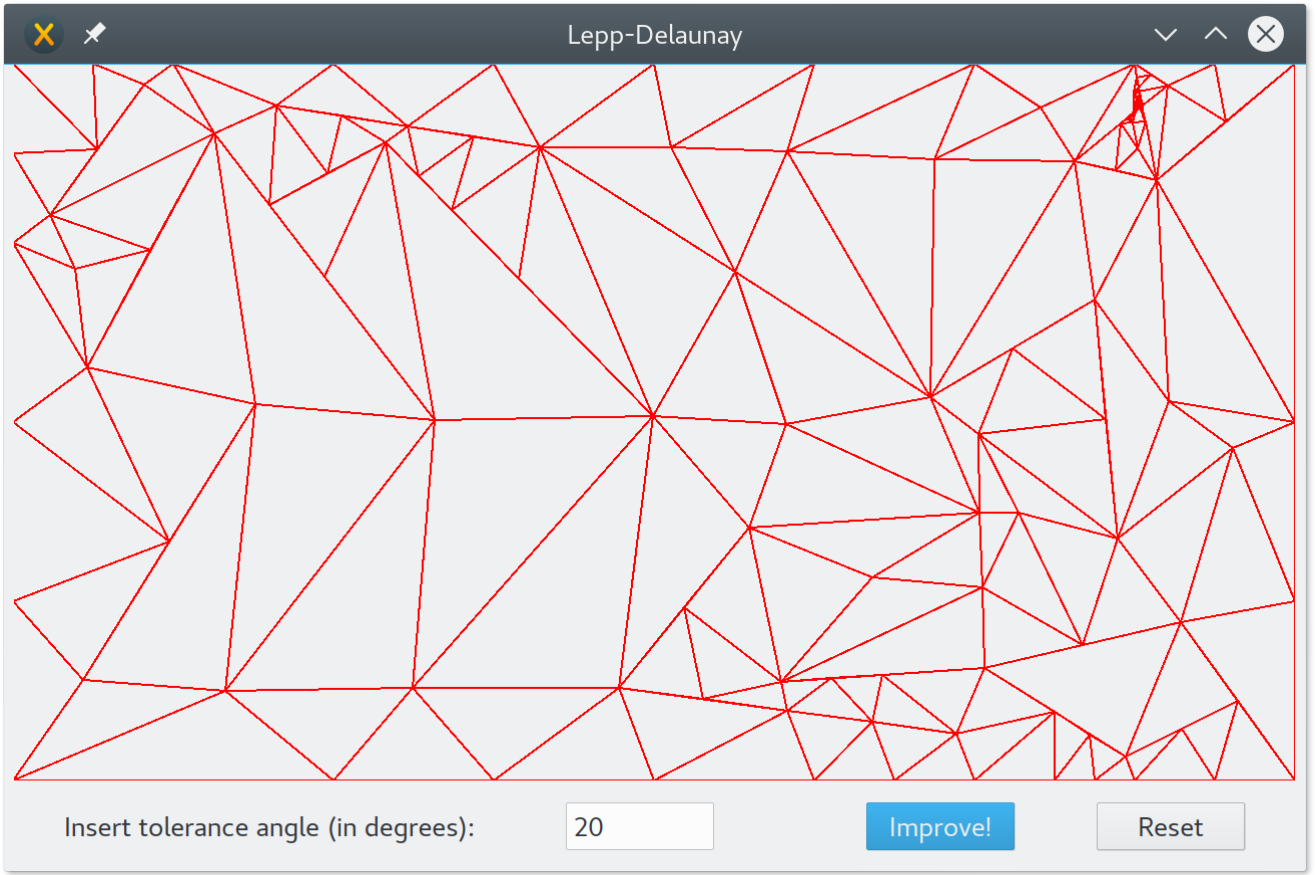
Resultados: Test 4

- Refinamiento con $\sphericalangle_{tol} = 15°$



Resultados: Test 4

- Refinamiento con $\theta_{tol} = 20°$



Dificultades

- Hubo que pelear mucho con la precisión de los cálculos realizados, por lo que hubo que asumir vértices de coordenadas *double*.
- La detección de “vecino de borde“ v/s “vecino terminal“ fue más compleja de lo esperado (tiraba falsos positivos), por lo que el algoritmo de búsqueda de terminales tuvo que modificarse y agregar una idea de estilo *read-ahead* para evitar que se cayera.

Conclusiones

- El refinamiento de triangulaciones usando el método Lepp es muy eficiente para mejorar las triangulaciones.
- El largo de la lista Lepp estuvo en su mayoría entre los 2 y 3 elementos, por lo que se comprueba empíricamente que converge a 2.