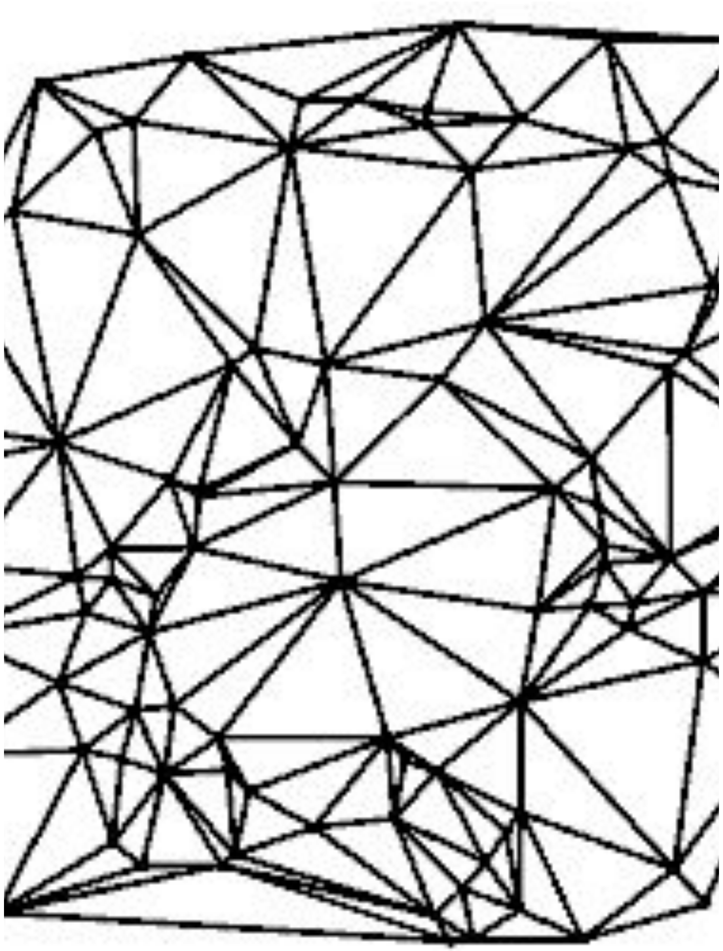
Delaunay Triangulation

Giovani de Almeida Valdrighi - FGV EMAp

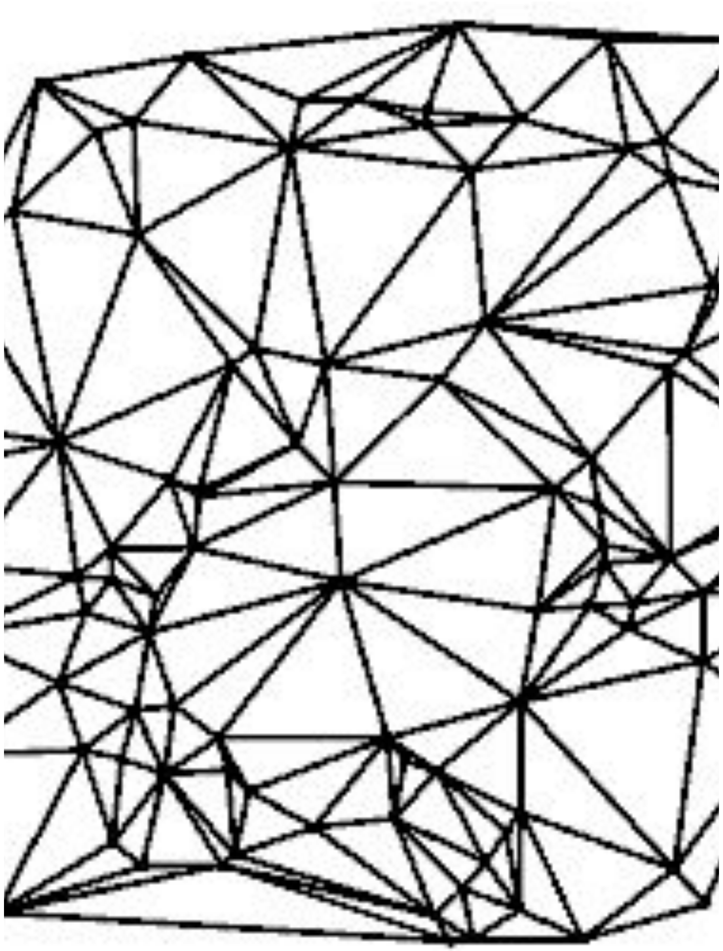
Motivation

- A terrain can be represented as a function from R^2 to R, for every position (x, y) is assigned an height.
- Usually, we don't have the height for every pair (x, y), just a sample of it. How can we approximate this terrain?



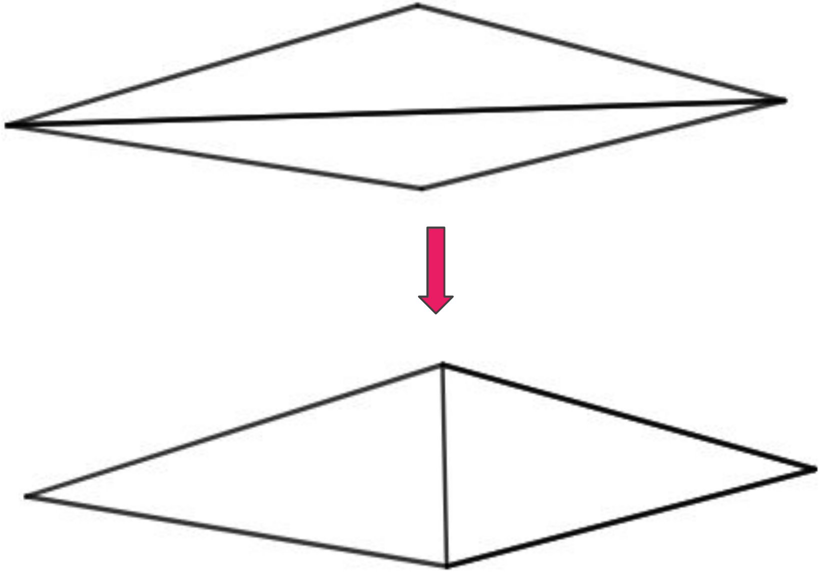
Motivation

- We first determine a **triangulation** of the sample of 2D points.
- Then we lift the points to their heights and connect the edges of the triangulation in 3D.



Motivation

- What is a good **triangulation**?
- A triangulation that contain small angles is bad.
- So we must look for the triangulation that have the biggest minimum angle.



Math Intuition

Maximizing minimal angle

- For a fixed number of points, there is a finite number of possible triangulations. So there is a triangulation that contain the biggest minimum angle. At least, a brute force algorithm exists.

Maximizing minimal angle

- With a circle defined by three points, the angle formed by a third point outside this circle is smaller than the angle formed by a point inside.

Maximizing minimal angle

- We can obtain an optimal triangulation removing all “illegal” edges, i.e., edges where the two adjacent triangles contain 4 triangles that are inside the same circle.
- We removing it by flipping edges.

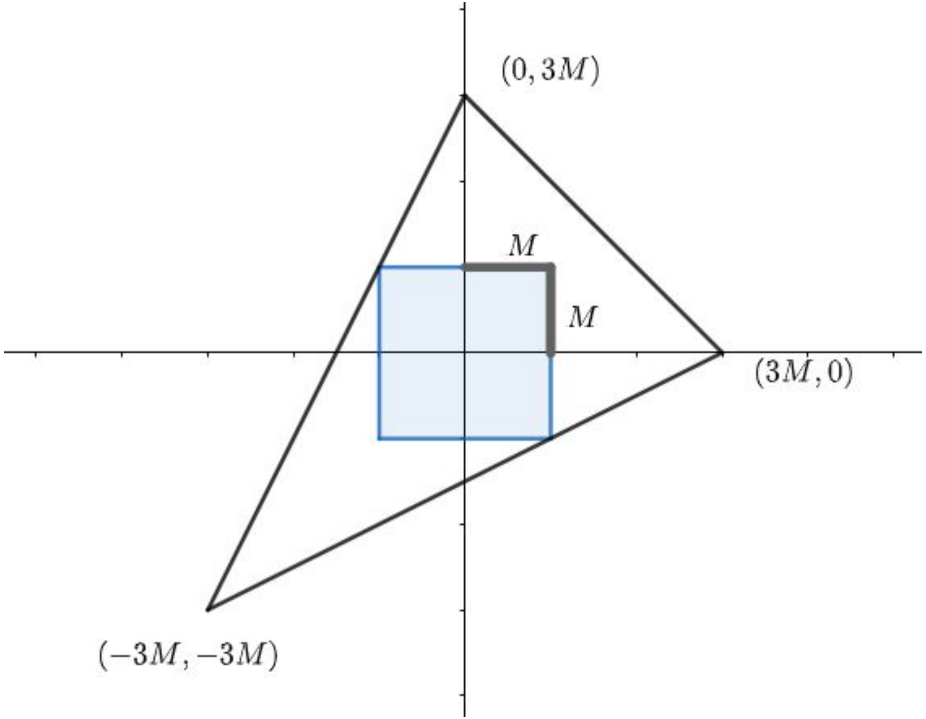
Algorithm

Overview

- Start with a triangle that contain all points.
- With the points in a random order, iterate:
  - Identify what triangle contains this point.
  - Add edges from the new point to the points of this triangle.
  - Check if the adjacents edges remain legal.
  - Fix illegal edges.

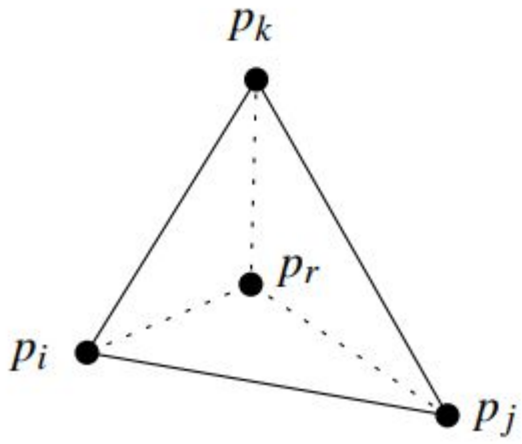
Initial triangle

- The position of points must be such that for every circle determined by three points, the vertices of this outer triangle are outside.
- If all points are inside the centered square if side 2M, the following triangle satisfies.



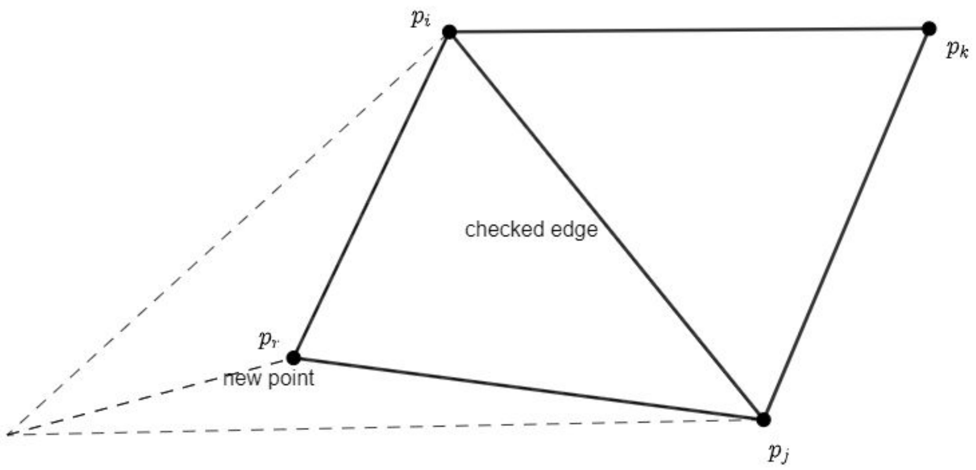
Iteration

- For each point, verify which of the current triangles contains this point.
- This new triangle will be updated with 3 new edges linking to its vertices.
- Then adjust edges if is necessary.



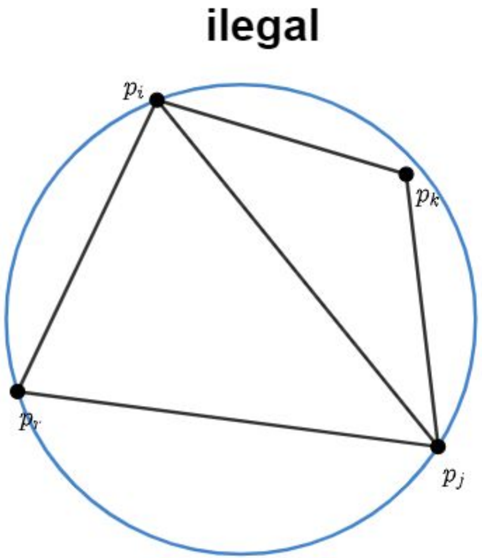
Adjust edges

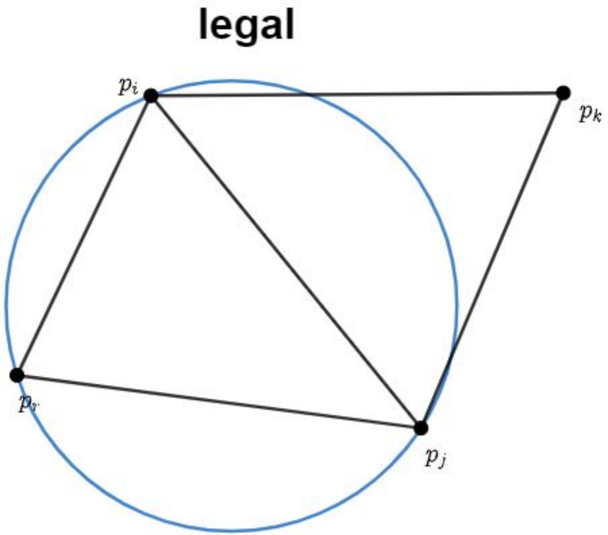
- The 3 edges of the updated triangle must be checked.
- For each edge, identify the triangle adjacent to the triangle formed by this edge and the new point.



Adjust edges

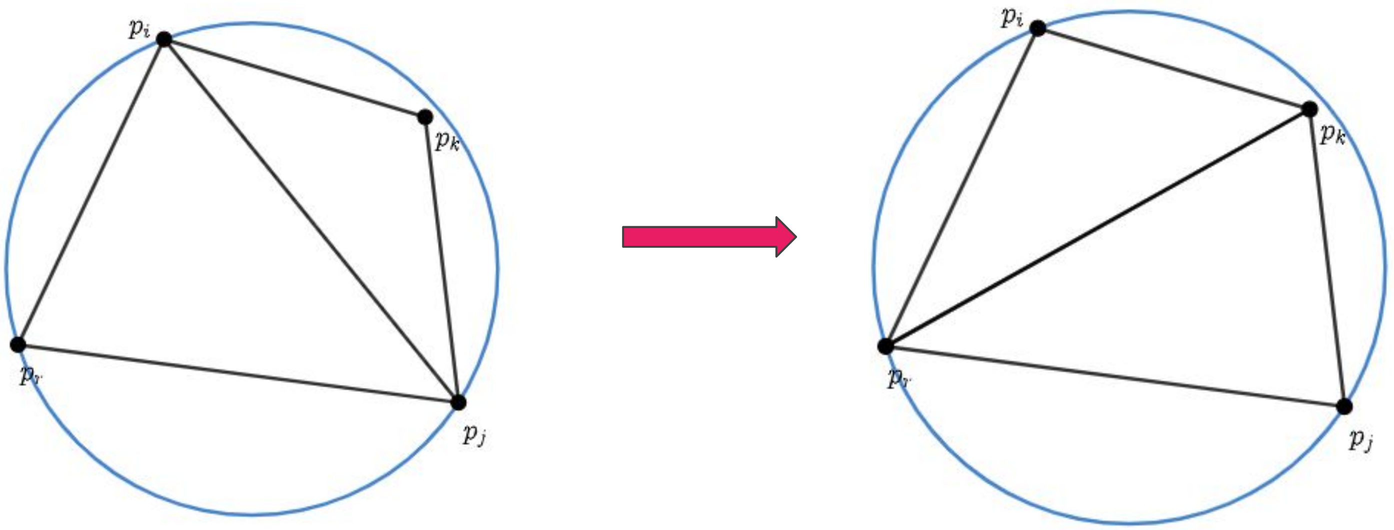
- Create a circle with three points and check if the fourth is inside or outside.





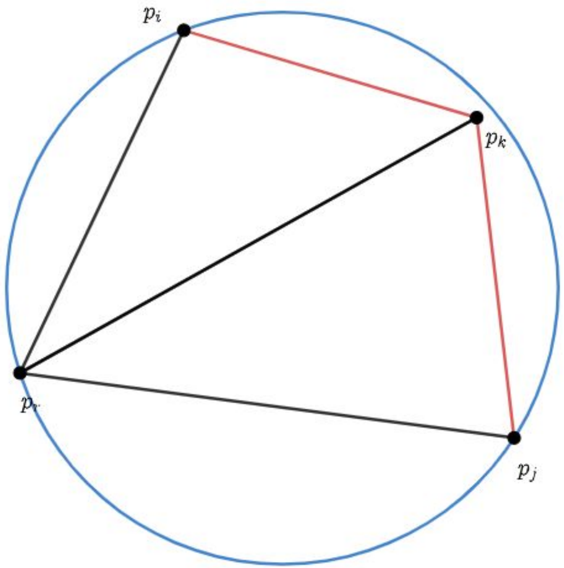
Adjust edges

- If the edge is legal, go to the new iteration, if it is illegal, flip this edge.



Adjust edges

- After flipped, these other two edges must be checked.



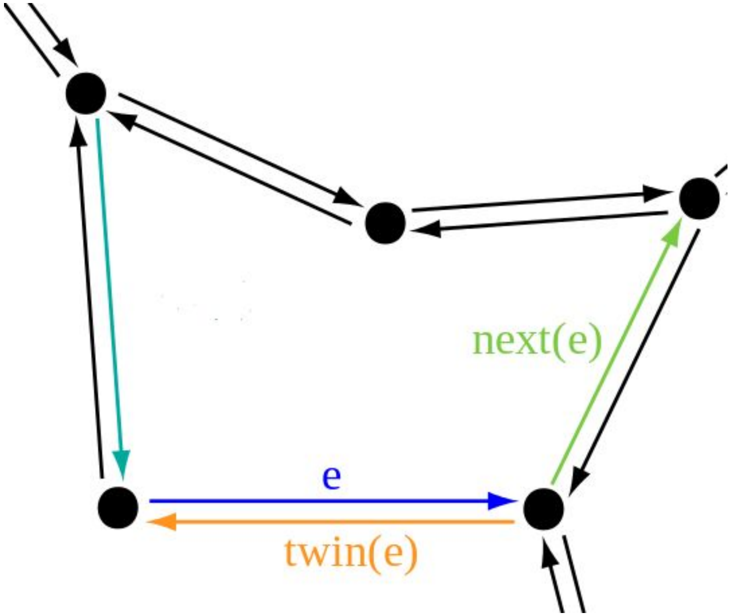
Implementation

Structures

- Doubly connected edge list (DCEL) - to keep track of the current triangulation
- Point Location graph - to find the triangle that contain each point in a sublinear time

DCEL

- Composed by vertex and half-edges.
    - Vertex contain a CGAL::Point, a pointer to an half-edge.
    - Half-edges contain points to a vertex, to a next and a twin half-edge and for a node in the poin location graph.



DCEL

- Functions:
  - Add vertex and set vertex color.
  - Create two half-edges between two vertices.
  - Link three half-edges in a triangle.
  - **Start the initial triangle** that contain all points.
  - **Add a vertex to a center of a triangle.**
  - **Check if an half-edge is legal.**
  - **Flip an half-edge.**
  - Save triangulation in a json format.

Point Location

- Composed by TriangleNodes.
    - TriangleNode contain a pointer to an half-edge, a vector of child, a CGAL::Triangle, and an array of three vertices. It contain a function that return if a point is inside the represented triangle.
- The leaves represent the current triangles of the triangulation, the knots represent triangles that have already been divided or flipped.

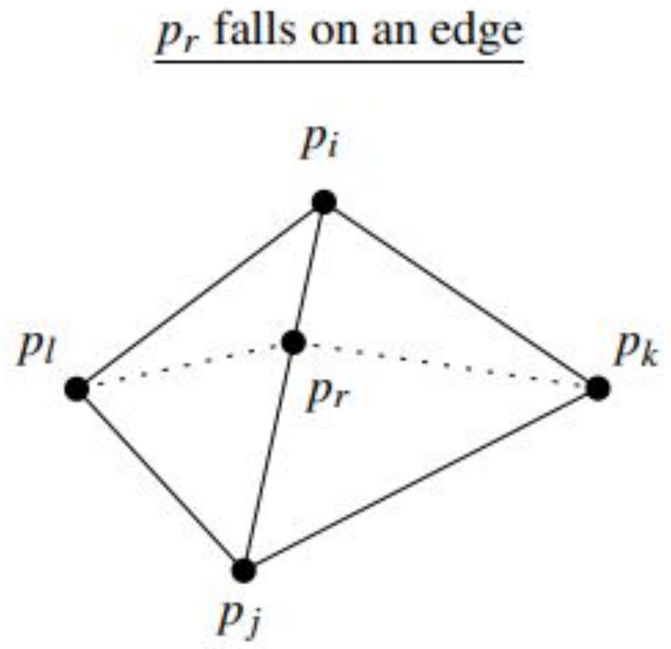
Point Location

- Functions:
  - Set root node.
  - **Search the triangle** that contain a point.
  - **Append child to TriangleNode**.

Detail

Special case

- There is a case where when searching for a point in the triangulation, the point isn't inside any triangle, it is inside a edge.
- It only occurs when the there is 3 collinear points in the sample.



Special case

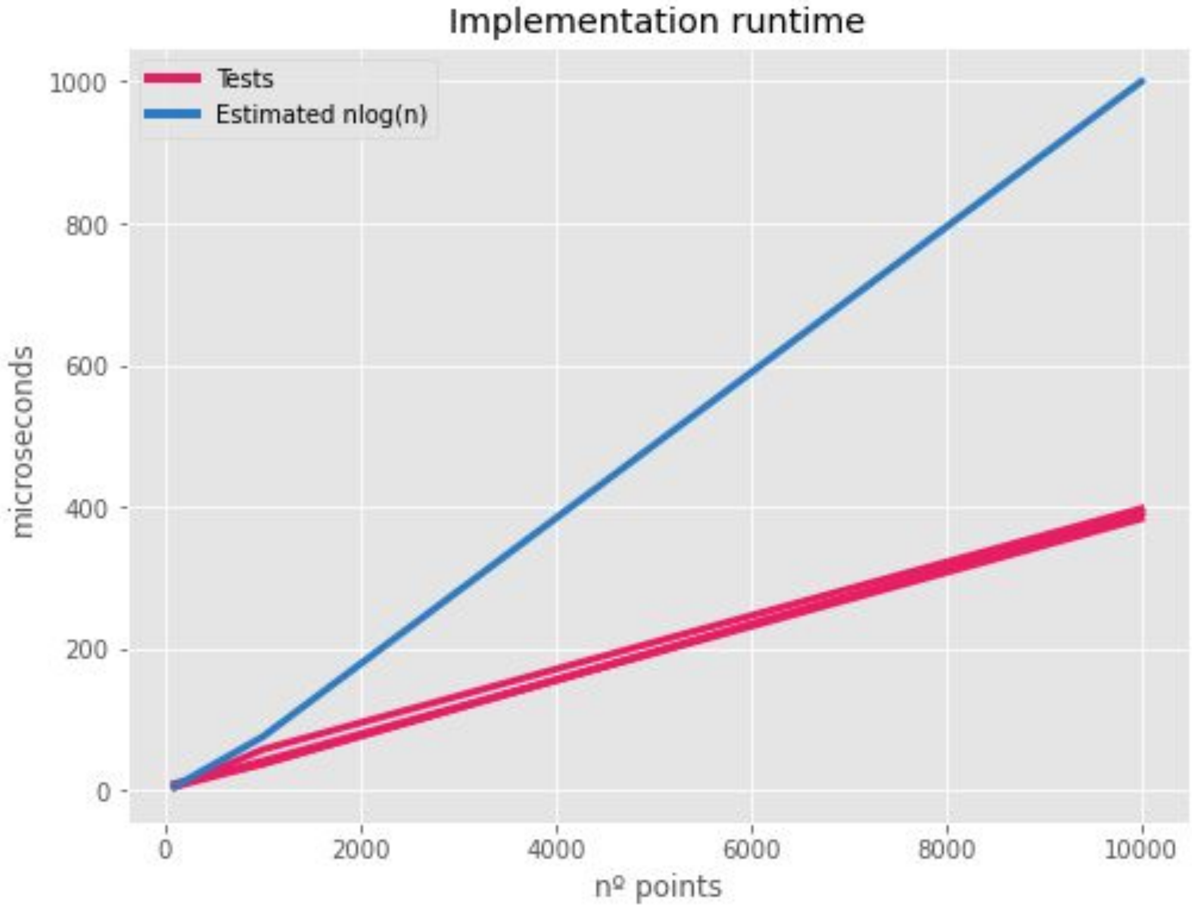
- The way my implementation deals with this, is that it adds a random noise to every point, so the probability of 3 collinear points is very small.

Complexity

Complexity

- The Delaunay Algorithm compute the triangulation of n points in expected time O(nlogn) and using memory O(n).
    - There is a iteration with n steps, and in each step, the expected time for the search of a point in the point location structure is O(logn).

Does my implementation runs in O(nlogn)?


Implementation runtime
Tests
Estimated nlog(n)
microseconds
nº points
0
200
400
600
800
1000
0
2000
4000
6000
8000
10000

Application

Artistic application

- To demonstrate the results of the triangulation, it was produced a visual application of the method.
- With an arbitrary image, it was selected random points (that can be from a edge detected or not), and computed a triangulation.
- With the result, it was produced an image where each triangle of the image is colored by the mean color of the vertices.

Cute example