Delaunay Triangulation

Author: Caio Valente (https://github.com/valentecaio/geometry)
Date: 2023-11-12

Introduction

Delaunay triangulation is a computational geometry algorithm that partitions a set of points into non-overlapping triangles, revealing the underlying structure of the data. This method is widely used in various fields for tasks such as mesh generation, terrain modeling, and Voronoi diagram creation. This report explores the intricacies of Delaunay triangulation, highlighting its practical applications and significance in computational geometry.

Dependencies

We will use the Python library MatPlotLib for plotting results and a Lua script (matplotlua) for parsing Lua data to matplotlib. As a consequence, we need python and matplotlib to run it. We also need the lua-cjson Lua package for parsing json data. You will also need Imagemagick for generating animations (optional)

```
$ pip install matplotlib             # install python deps
$ sudo apt install lua-cjson         # install lua deps
$ sudo apt install imagemagick-6.q16 # install imagemagick (optional)
```

Project Structure

```
.
├── mesh/
│   ├── datasets/       -- Input files containing list of points randomly distributed.
│   │   ├── nuvem1.txt
│   │   └── nuvem2.txt
│   ├── figures/        -- Temporary dir for images and animations.
│   │   ├── nuvem1.mp4  -- Step-by-step animation of nuvem1.txt input file.
│   │   └── nuvem2.mp4
│   ├── delaunay.lua    -- Contains the developed Bowyer-Watson algorithm.
│   └── main.lua        -- Script to test the developed algorithm.
│
├── matplotlua/         -- Plot library. Used to plot input and output meshes.
└── utils/              -- Utils library. Useful methods reused by other modules.
```

How to execute tests

We just need to execute the main.lua file using the Lua interpreter:

```
$ cd delaunay/ && lua main.lua <dataset> # dataset may be a file from datasets/ dir
```

The output should be a visual plot of the generated mesh.

Algorithm: Bowyer-Watson Incremental

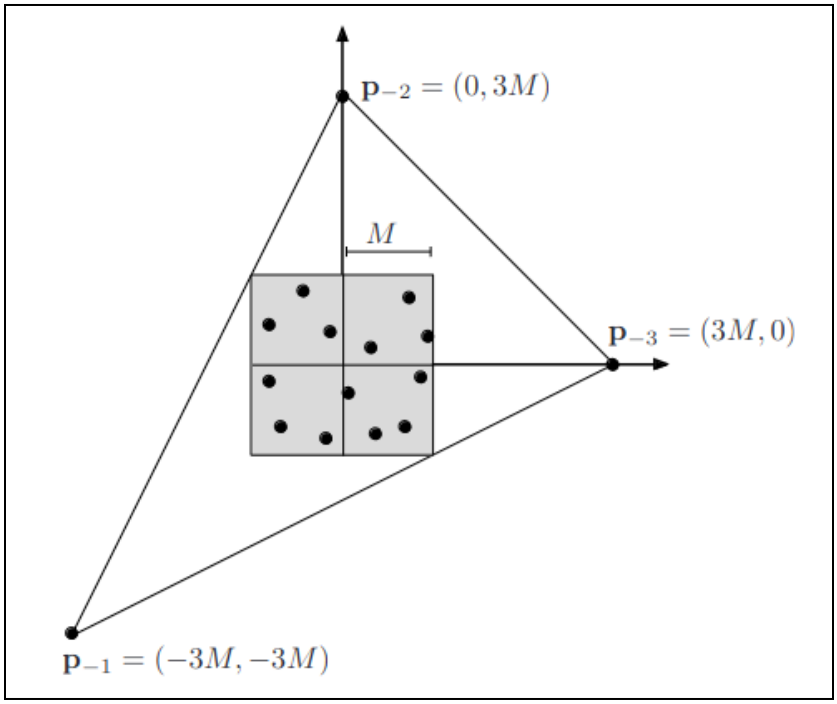
The Bowyer-Watson Incremental algorithm is a method for constructing a Delaunay triangulation incrementally by inserting points one at a time. The process begins with a super-triangle encompassing the point set. As each point is added, the algorithm dynamically adjusts the triangulation, ensuring the Delaunay property - no point lies within the circumcircle of any triangle. When a point is inserted, triangles are identified that are no longer valid and are removed, forming new triangles in their place. This incremental approach is efficient and straightforward, offering a clear strategy for generating a Delaunay triangulation while maintaining the integrity of the geometric structure.

This algorithm, when well implemented, has a complexity of O(n log n). This complexity arises from the sequential insertion of points and the subsequent adjustments made to the triangulation structure. The algorithm's efficiency stems from its ability to adapt the triangulation dynamically, avoiding the need to reconstruct the entire structure after each insertion. To achieve such efficiency, we should use tree-based data structures to maintain the triangulation.

The implementation presented here is not the most efficient possible, as it does not use any topological data structure to represent the mesh. In doing so, our algorithm is expected to run in O(n*n).

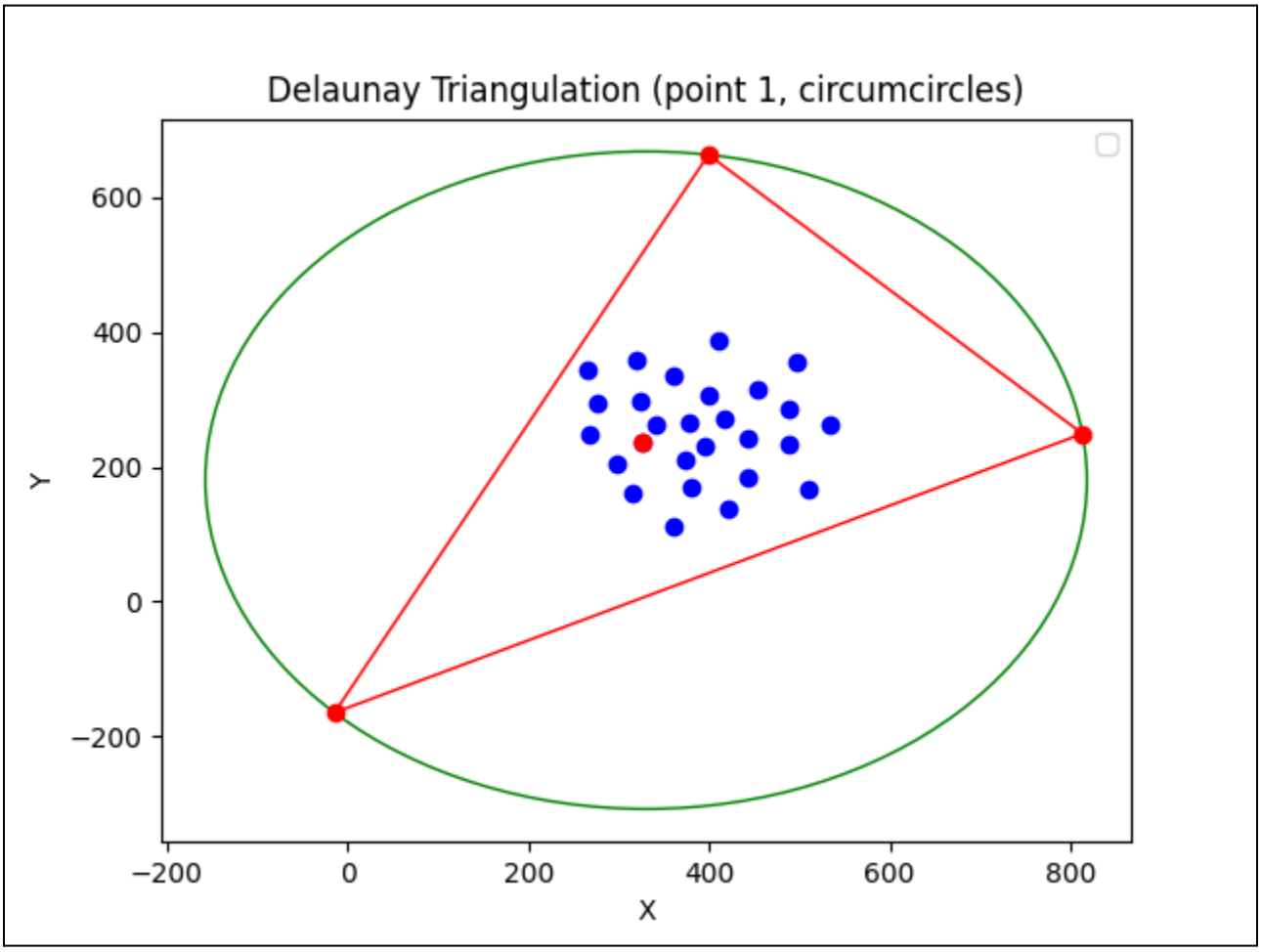
The first step (step zero) of our algorithm consists in finding the "Supra Triangle", which may be any triangle that contains all the points of the cloud.

We can achieve so by calculating a rectangle that contains all the points, which is as easy as finding the minimum and maximum values for X and Y. After we have such a rectangle, we can calculate the M factor that is half of the biggest size of this rectangle. After we find M, we can form a triangle whose 3 vertices are the ones from the image below.



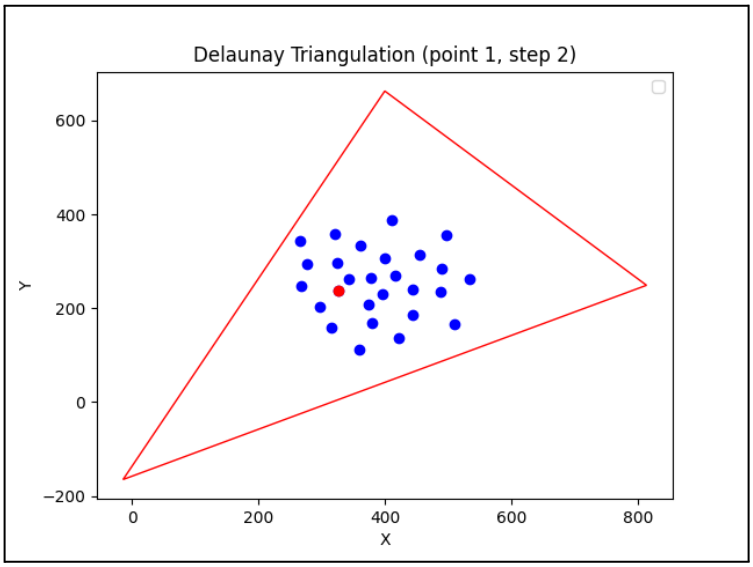
The Supra Triangle

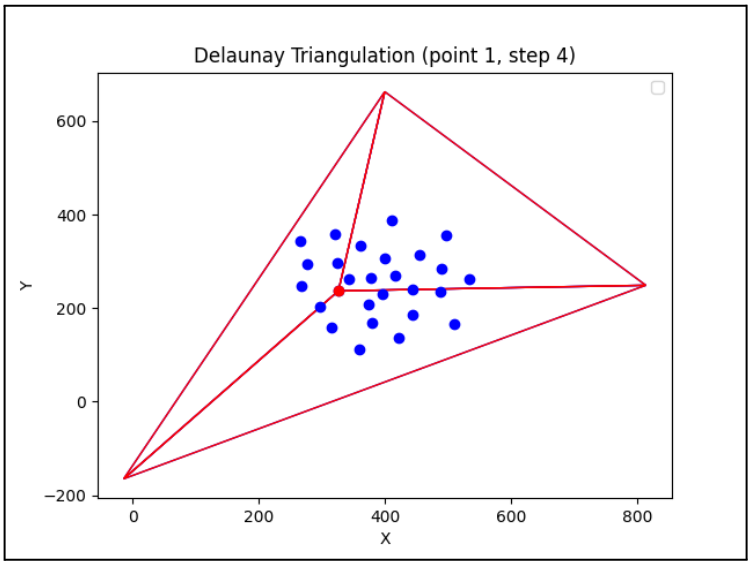
The supra triangle is a valid Delaunay mesh for the 3 points that form the triangle itself. From now on, the algorithm will insert new points in this mesh and modify the mesh locally so it maintains the Delaunay property, which means modifying the neighbors triangles so that the new point is not inside the circumcircle of any of them.



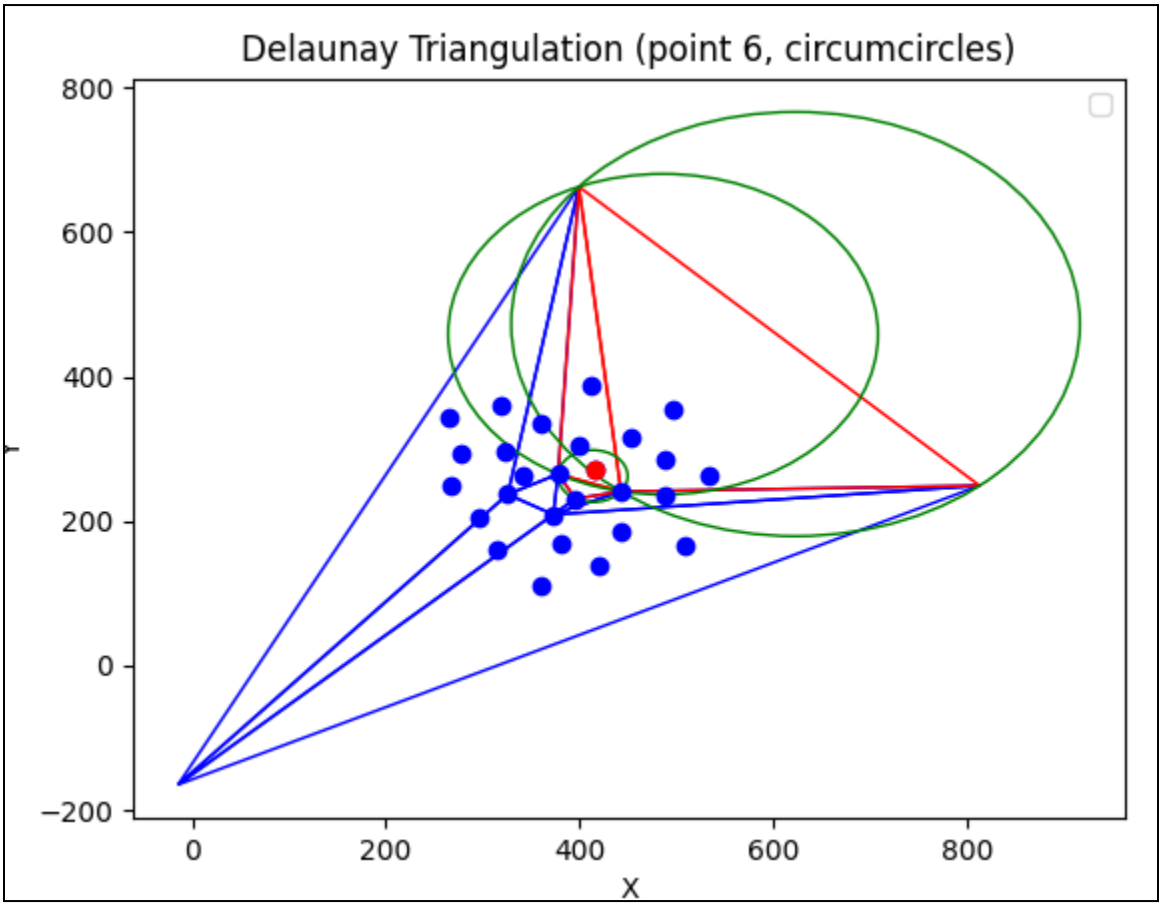
Supra Triangle (in red) and its circumcircle (in green).

After the supra triangle is found, the algorithm starts its main loop of points insertion. It will iterate on the input cloud, that is randomly ordered. For each point, it inserts the point in the Delaunay mesh (step 1) and checks which triangles of our current mesh should be changed so that the mesh can accommodate this new point without breaking the Delaunay property. These triangles are called "bad triangles" (step 2), and they may be removed and replaced by new triangles (steps 3 and 4). In the first run, the point will always be inside our only triangle, which is the supra triangle itself. The triangle will then be replaced by three new triangles:

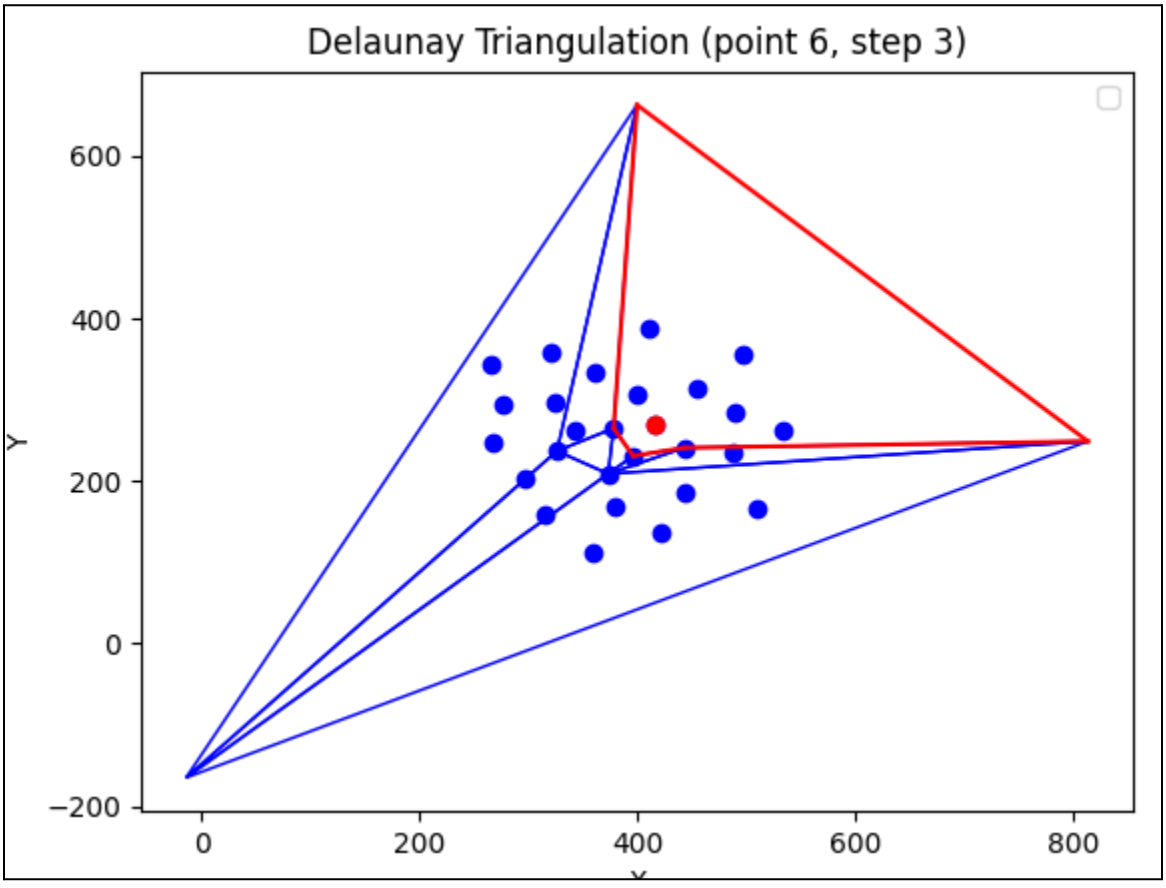




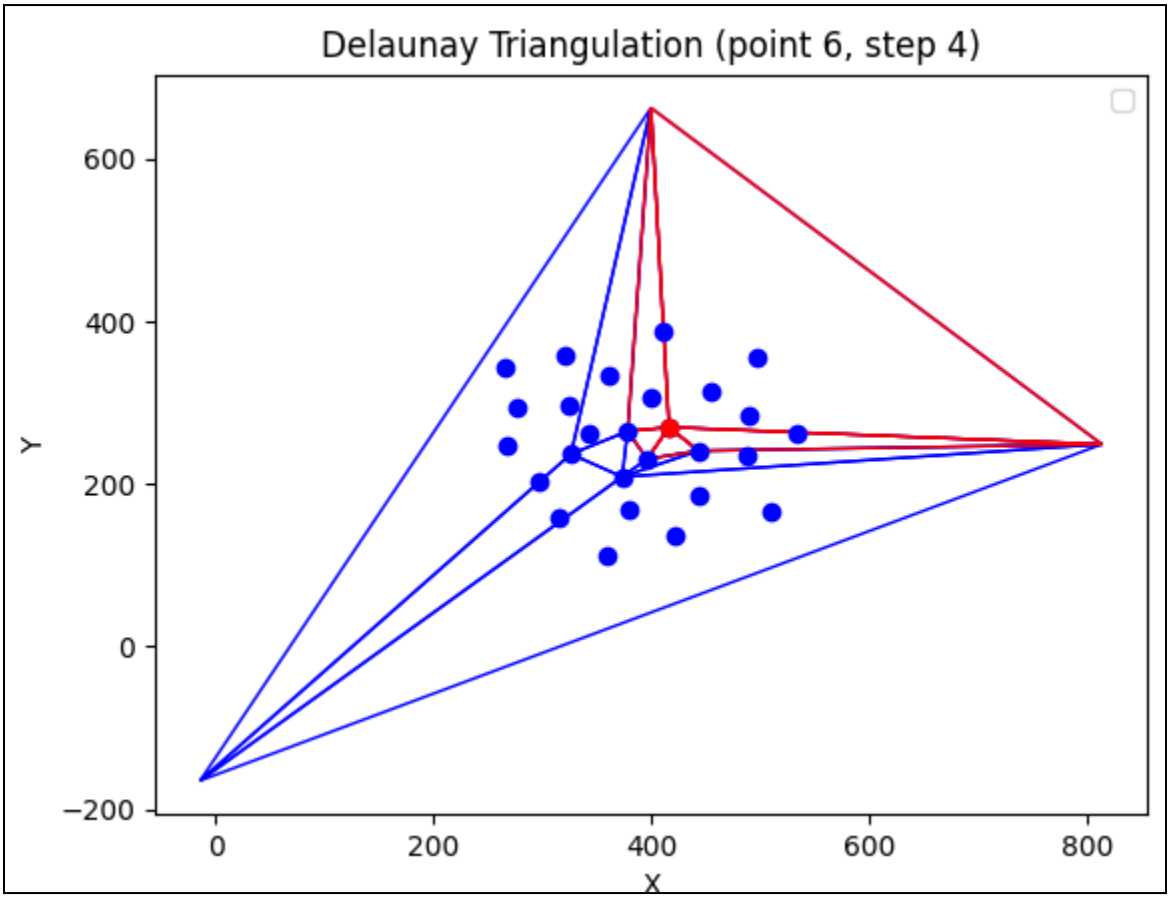
Some points will interfere in more than one triangle, such as the example below:



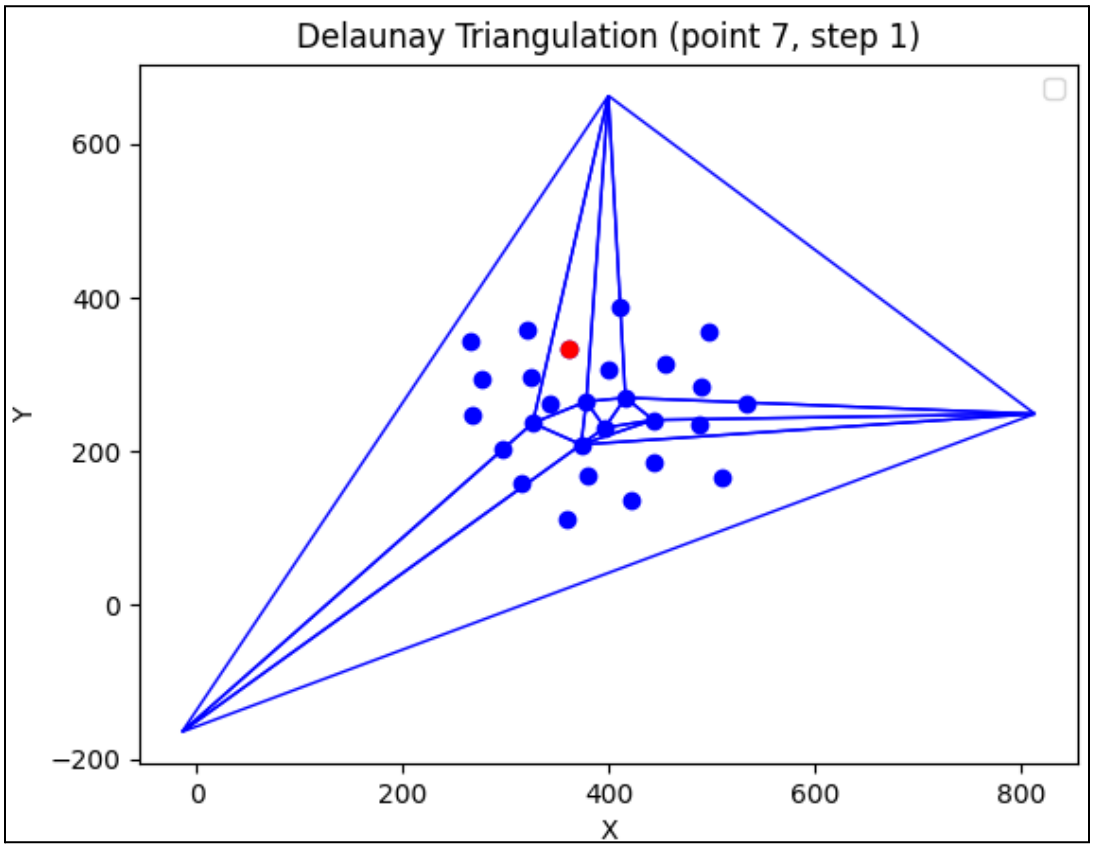
The new inserted point (in red) is inside the circumcircles (green) of three triangles (red). The algorithm is going to remove these three triangles, forming an empty hole in our mesh. This is the step 3:



The polygonal hole is, then, re-triangulated (step 4):



After the triangulation of the hole is done, the point is inserted and the mesh is a valid Delaunay mesh. Now, we can proceed to the next point.



After all points are inserted, we will have a Delaunay Mesh that includes all the points from the cloud and the three initial points from our supra triangle. Now, the last step is to remove these 3 points from the mesh: