Homework 1 : Delaunay Triangulations

In this first assignment, we'll ask you to calculate the Delaunay triangulation of set $P = (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1})$ of n points $\mathbf{x}_i = (x_i, y_i)$ of the plane.

The deadline for your assignment is Halloween 2024.

1 Input/Output

The input for your code will be a file structured as follows. The first line of the file will be an integer n corresponding to the number of points to be triangulated, and the n following lines will list the x_i and y_i coordinates of the n points to be triangulated. The python script `genpts.py` can be used to generate a file of this type with an arbitrary number of points.

Your program `del2d.py` must be able to run non-interactively : we must be able to test it as follows :

```
python del.py -i pts.dat -o triangles.dat
```

The output file `triangles.dat` will be structured in a similar way to the input file. The first line of `triangles.dat` will contain an integer n_t designating the number of triangles in the Delaunay triangulation $DT(P)$ and the n_t following lines will each contain three integers t_{j1}, t_{j2}, t_{j3}, $j = 0, \ldots, n_t - 1$ indexed from 0 to $n_t - 1$ and designating the three vertices of the j triangle.

2 The Algorithm

Here we describe the various steps in the algorithm used to calculate this triangulation.

2.1 Datastructure

You can (and you should) use the python class `TriangularMesh` which has been sent to you and is available on the course website. This `HalfEdge` data structure is ideal for programming Delaunay triangulation.

2.2 Initialization

You'll use an iterative algorithm to calculate $DT(P)$. You need to initialize the algorithm with a simple triangulation consisting of two triangles that completely enclose the point cloud. You must therefore start by calculating the axis oriented bounding box of P i.e. calculate two points $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$ that are such that $x_{\min} < x_i$, $\forall i$, $y_{\min} < y_i$, $\forall i$ etc.

You'll use an iterative algorithm to calculate $DT(P)$. You need to initialize the algorithm with a simple triangulation consisting of two triangles that completely enclose the point cloud. You must therefore start by calculating the axis oriented bounding box of P i.e. calculate two points $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$ that are such that $x_{\min} < x_i$, $\forall i$, $y_{\min} < y_i$, $\forall i$ etc. Four *infinite* points $\mathbf{x}_n = (x_{\min}, y_{\min})$, $\mathbf{x}_{n+1} = (x_{\max}, y_{\min})$, $\mathbf{x}_{n+2} = (x_{\max}, y_{\max})$ and $\mathbf{x}_{n+3} = (x_{\min}, y_{\max})$ are used to define an initial triangulation made of two triangles. $(\mathbf{x}_n, \mathbf{x}_{n+1}, \mathbf{x}_{n+2})$ and $(\mathbf{x}_n, \mathbf{x}_{n+2}, \mathbf{x}_{n+3})$.

2.3 Incremental construction of $DT(P)$ using Bowyer-Watson

There are several ways of obtaining $DT(P)$. You can use Bowyer Watson's algorithm as described in the course. Let DT_i be the Delaunay triangulation comprising the first i points of the list P. We now want to add $\mathbf{x}_{i+1}$ to form DT_{i+1}. To do this, you need to caculate the Delaunay cavity C_{i+1}, which contains all the triangles of DT_i that violate the Delaunay criterion, i.e. all the triangles whose circumscribed circle contains $\mathbf{x}_{i+1}$.

This cavity is simply connected and star-shaped, and the new point $\mathbf{x}_{i+1}$ is part of the noyay of C_{i+1}. It is therefore very simple to triangulate C_{i+1} by connecting $\mathbf{x}_{i+1}$ to all the edges forming the border of C_{i+1}. This operation adds exactly two triangles to DT_i.

At first, you may not care about the efficiency of your algorithm : iterate on each of the triangles of DT_i and use the robust predicate `incircle` which is available in the python package `geompreds`.

3 To infinity and beyond

If your program delivers a correct result – even if it's slow, we'll test your implementation up to 1000 points – your assignment will be successful and will allow you to have a maximal grade of 14/20 . To improve your score, we suggest several avenues of improvement.

3.1 Removing infinite points (max +3 points)

The triangulation your program produces at this stage has $n+4$ points. The Delaunay triangulation $DT(P)$ is supposed to contain only the n points of P and not the four extra points we used to initialize the algorithm. We want $DT(P)$ to cover exactly the convex hull of P and eliminating all the triangles in your triangulation that contain one of the 4 *infinite* points $\mathbf{x}_{n+j}$, $j = 0, 1, 2, 3$ does not guarantee that $DT(P)$ will be obtained. You are therefore asked to supply exactly $DT(P)$.

3.2 An implemetation in $\mathcal{O}(n \log n)$ (max +6 points)

In the course, we saw that sorting the set P using space-filling curves allows to reduce the complexity of the algorithm. Space filling curves are 1-dimensional traversals of n-dimensional space where each point in a discrete space is visited once. These family of curves have been heavily researched in the past for their locality preserving properties and are especially useful for dimensionality reduction. We've seen how to sort points along a Hilbert curve. There are python bindings for this sorting, in dimension 3, but which will obviously work if you set the 3rd coordinate of the points to zero.

```
from hilbertsort import HilbertSort3D
```

Once you've sorted your points, you can "walk" into DT_i from one of the triangles in the cavity C_i and quickly find the triangle in DT_i that contains $\mathbf{x}_{i+1}$.

3.3 Go Interactive (max +5 points)

Delaunay triangulations are beautiful graphic objects and it's natural to want to draw them. As mentioned above, it's not necessary to draw triangulations to obtain a pass note. Nevertheless, we leave you free to embellish your code by using `matplotlib`, which is an excellent tool for drawing triangulations.

It would also be possible to add a point with a mouse click, move a point, draw the voronoi diagram on the top of the triangulation, draw the circumcircles... Here, the sky is the limit and we're calling on your imagination.