CS 274 Computational Geometry
Final Project Report

Ankush Gupta
gupta@berkeley.edu

The task assigned was to implement an algorithm for computing delaunay triangulations of two dimensional points.

I implemented the *divide-and-conquer* algorithm from Stolfi and Guibas. The correctness of the algorithm was validated by visually comparing its output with that of `Triangle`[1]on the same data set.

Compiling/Running the Code

This code was tested on a machine running Ubuntu 12.04. The only external libraries it needs are the `boost` libraries.
The code can be compiled by making a `build` directory and then running cmake, followed by make, as following:

```
cd build
cmake ${PATH TO PROJECT DIRECTORY}
make delaunay
```

The code is executed as following (assuming we are in the build directory):

```
./bin/delaunay -i input_filename [-o output_filename] [-V or -A][-T]
```

For an explanation of the various flags, run `./bin/delaunay -h`.

Timing

The implementation works fast and correctly on small data-sets. It performs well for the `ttimeu10000.node` (10k points) (see the table below). However, it takes unusually long on the data-set `ttimeu100000.node` (100k points). As it took over 30 minutes on the 100k points data-set, the program was not run on the `ttimeu1000000.node` (1 million points) data-set.

Even though it takes that long, the output does seem visually correct. The postscript files for the output generated by the program on these large data-sets is available in the `data` directory.

[1]Jonathan Richard Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator

Following are the times it took on the large data sets; it does not include the file i/o time. These times were recorded when the program was executed on a machine with `Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz` CPU and 16GB of RAM.

	Alternating Cuts	Vertical Cuts
10k points	4.59 seconds	10.24 seconds
100k points	2336.24 seconds	N/A
1000k points	N/A	N/A

As my implementation took over 30 minutes on the 100k data-points input, I did not try it on the 1 million points data-set.

Sample output on the `spiral.node` data-set is shown below. More pictures are available in the `data` directory.

From the above timings, we can conclude that the alternating cuts algorithm performs almost twice as fast as the vertical cuts only algorithm. This speed up is due to the fact that if the points are uniformly distributed in the plane then by using alternating cuts, the base-cases would end up being in a square rather than a vertical slab; this means that the points are less likely to be collinear. This avoids expensive orientation test and extra-edges (which are eventually deleted), hence making the alternating cuts algorithm faster.

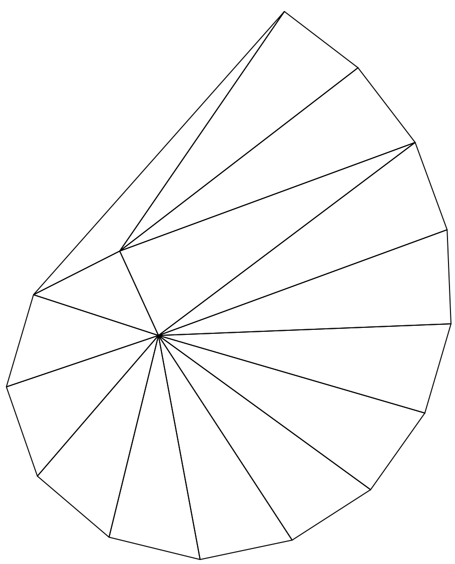

Figure 1: Output of the program on `spiral.node` data-set. Generated using `ShowMe`.

Alternating Vs. Vertical Cuts

If we have a data-set which has a large horizontal spread and very-small vertical spread, then it would take alternating cuts algorithm significantly longer than vertical cuts only algorithm. This is because then, it will encounter almost collinear points, which would mean that the geometric predicates need to calculate more bits to correctly evaluate the sign of the determinants.