CS274–Spring 2013 — A Delaunay Triangulation using Incremental Insertion

Kevin Lindkvist

May 2, 2013

This program was written and tested using Python 2.7.3, which can be found at `http://www.python.org/getit/releases/2.7.3/`, but should work using later versions of Python as well. I didn't use any special libraries, but I did stumble upon an implementation of the basic incremental version of the algorithm in $C++$ (without the conflict list) by Dani Lischinski that can be found at `http://www.karlchenofhell.org/cppswp/lischinski.pdf`. In many places I sacrificed elegance for performance, which in Python means a lot of inline code instead of method calls. The times in the below table are measured without garbage collection (except for the 1,000,000 point set), this gives around a 10 second improvement for the 100,000 set. To see the timing data, use `[-v]`. I would also recommend at least 16GB of memory in order to run the largest point set, otherwise there will be thrashing, and it wont be pretty.

```
python delaunay.py [option] inputfile.node
```

`-v`: Will run the program in verbose mode, outputting progress etc.

`-c`: Will run the program using a conflict list for point location

`-nele`: Will not output .ele file

`-edge`: Will output .edge file

`-p`: Will profile using ttimeu100000.node

`-nfs`: Need for speed (no garbage collection)

`-mp`: Will output GC information

Timing data:

Using conflict list	$\|S\|$	Time (s)
No	10000	6.4
No	100000	162.6
No	1000000	N/A
Yes	10000	3.1
Yes	100000	35.8
Yes	1000000	454

Is there an ordering of points that would lead to a significant decrease in runtime?

Yes, if the points were inserted in an order such that there would be a high number of conflicts (inserting each vertex creates many vertex moves) it would significantly increase the runtime of the algorithm. See `ordered.out` for an example of such an input, if inserted in random order it runs in < 2 seconds, if not it takes orders of magnitude longer.