# CS274 Delaunay Triangulation Project Document

Weilun Sun
UC Berkeley
sunweilunjwilson@berkeley.edu

April 22, 2015

## 1 Instruction

Randomized insertion algorithm is implemented in this project. The code can be compiled by executing "make" command in the source file folder.(UNIX OS) There's a "USE_GL" variable defined at the top of "makefile" file. The variable is set to "false" by default. If you want to use my GL display program to visualize, just set this variable to "true" before you execute "make". Program argument options are listed below.

- "-i [path]" specifies the input file path. (Required)
  Example: "-i 633.node"
- "-o [path]" specifies the output file path. (Optional)
  Example: "-o out" generates "out.node" and "out.ele".
  If this argument does not exist, then no file would be generated.
- "-t" prints the execution time of the triangulation algorithm. (Optional)
- "-r[int]" randomizes the input before triangulation. (Optional)
  Where "[int]" can be empty or a positive integer.
  If "[int]" is empty, a seed based on current time is used for random numbers.
  Otherwise, the random seed is specified by "[int]".
  Examples: "-r" "-r13"(13 is a lucky number.)
- "-c" uses conflict list for point location. (Optional)(Default)
- "-w" uses walking for point location. (Optional)
- "-d" displays the triangulation result. (Optional)
  Note that "USE_GL" variable needs to be set to "true" before you make to use this option.

## 2 Timings

Computer Settings:

- OS: ubuntu 14.04 LTS (64-bit)
- Memory: 3.9G
- Processor: Intel Core i7 CPU 920 @ 2.67GHz 8

Note that only one core is used in the program. Reference command Lines used are listed below. For 1,000,000 nodes case, fast point location is 5x faster than slow point location.

- "Walk" ./Delaunay -i ttimeu10000.node -t -w
- "Conf" ./Delaunay -i ttimeu10000.node -t -c
- "Rand+Walk" ./Delaunay -i ttimeu10000.node -t -w -r13
- "Rand+Conf" ./Delaunay -i ttimeu10000.node -t -c -r13

| Input | Walk | Conf | Rand+Walk | Rand+Conf |
|---|---|---|---|---|
| `ttimeu10000.node` | 0.137 s | 0.162s | 0.140s | 0.152s |
| `ttimeu100000.node` | 4.783s | 2.550s | 4.356s | 2.612s |
| `ttimeu1000000.node` | 209.719s | 40.949s | 199.474s | 36.995s |

Conflict list is faster for point location especially when the data size is huge because it allows us to quickly locate where the new vertex to insert is. Although we need to update the conflict list after each insertion, the work gets easier as we insert more vertices. This is because the hole each new vertex make gets smaller and smaller as we insert more vertices. We have proved in class that the complexity is expected $O(nlogn)$. In contrast, the walking method needs very few steps in the begining but many steps in the end. The number of steps to take is roughly proportional to the number of vertices inserted. The complexity of the walking method is hard to estimate. But if we assume that we take $\sqrt{n}$ steps for each vertex on average, then the complexity would be $O(n^{1.5})$.

For the orderly point set, I created a 100x100 grid and tested with and without randomization using the fast point location. The difference is huge. The randomized version, which took about 0.15s, is about 90x faster than the unrandomized one, which took 13.7s.

## 3 Libraries

- GLM for vector and matrix calculation.
- vector, stack and unordered_set for mesh data structure.
- GLUT for visaulization.