CSC 254 Fall 2015
Assignment 6
Katarzyna Ruszowska - kruszows@u.rochester.edu
Mohan Liu - mliu26@u.rochester.edu
Due: Sunday, December 12, 2015

#Summary
In this assignment, we used multithreading techniques to parallelize the Dwyer's Delaunay triangulation algorithm and Kruskal's algorithm.

#Implementation Details
For Dwyer's algorithm, since it is inherently a divide and conquer algorithm, we created new threads before the division to handle each partition separately.

For Kruskal's algorithm, we calculated as much as we could ahead by using multiple threads. For the actual merging process, we merged the edges if the precalculation was correct; if not, we recalculated the edge.

#Extra Credit
we did extra credit 1 and 2

For extra credit 1, we extended the command line parsing tools to support "-c config.txt". Details of the format are specified in configuration file in the directory.

For extra credit 2, we limited the case of spawning two threads in the triangulate() function. We calculated how many points were left inside the subtree, which is based on the difference between the higher node and the lower node. The leaves of the tree control the effect on the runtime. For a tree with n nodes, there will be (n+1)/2 leaves; it is a linear relation. For a small number of nodes (i.e., in the test case with n = 50), it is not significant, but in a relatively big tree, it significantly reduces the time wasted managing the threads.

#Additional extra credit
Even though Professor Scott didn't mention thread recycling in the assignment, we implemented this feature in both algorithms. Essentially, we find dead threads and reinitialize them for later use.

#Speedup Analysis
There is a huge difference between the time used for full animation mode and pure IO mode. This first result is based on pure IO mode, in which the cost of managing threads is emphasized.

Our test is conducted with thread number in set {1,2,3,4,5,6,7,8,12,16,24,32,48} and seed parameter 1.
To get an accurate result, the numbers are averaged from 20 sets of data for each condition. See the devtest folder for the scripts.

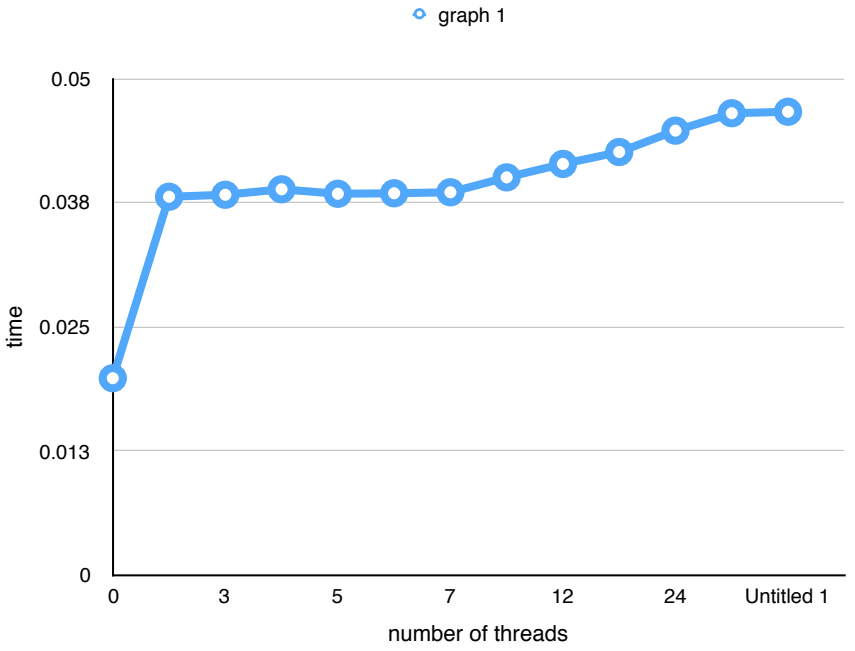
Threads	Triangulation	Kruskal's	Total
1	0.01985	0.01825	0.0381
2	0.0381	0.126	0.1641
3	0.0383	0.12625	0.16455
4	0.03885	0.093	0.13185
5	0.0384	0.09355	0.13195
6	0.03845	0.0861	0.12455
7	0.03855	0.0843	0.12285
8	0.04005	0.08295	0.123
12	0.0414	0.0819	0.1233

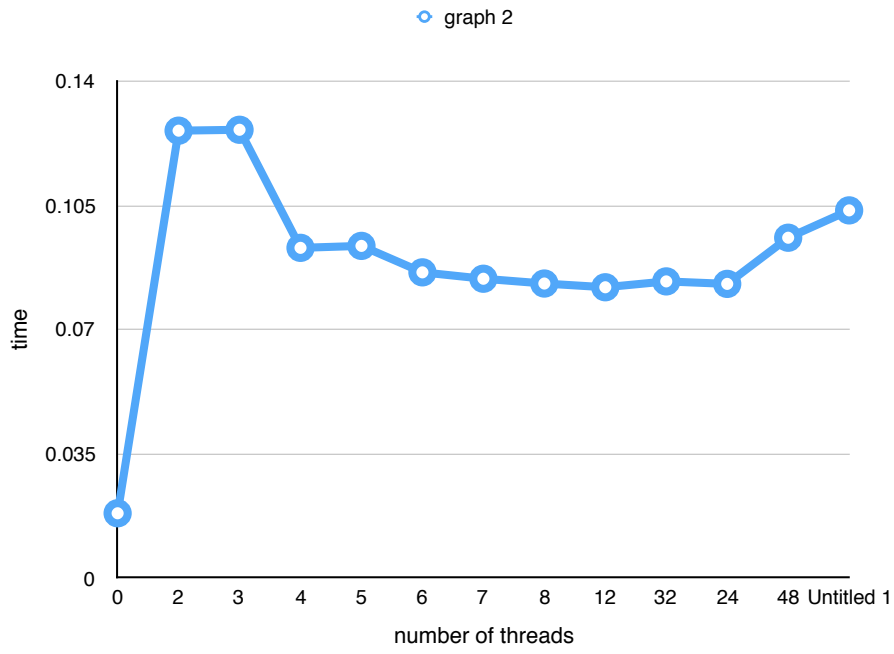
16	0.0426	0.08355	0.12615
24	0.04475	0.08285	0.1276
32	0.0465	0.09585	0.14235
48	0.04665	0.10355	0.1502

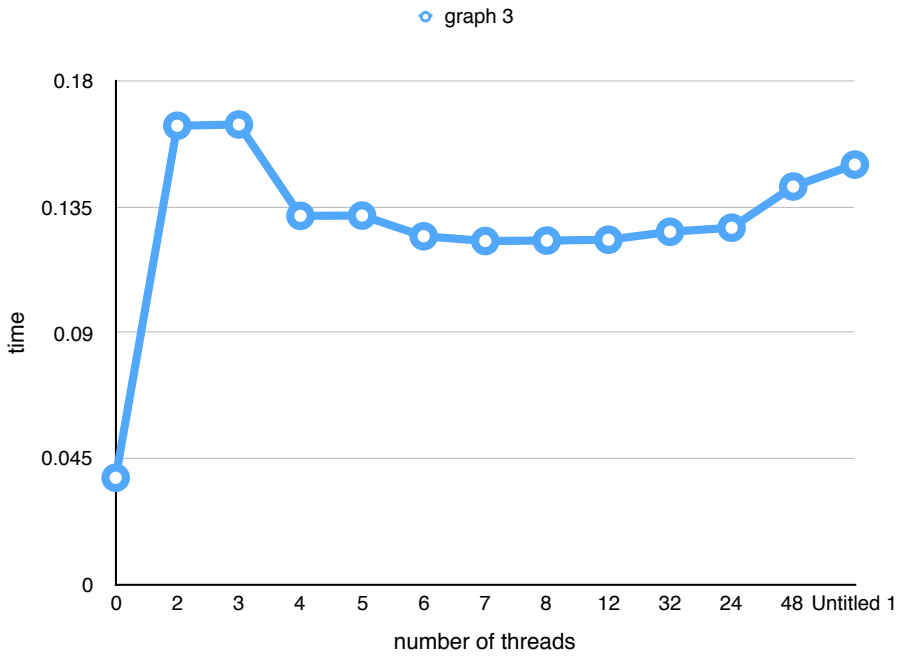
Graph1 shows that the more threads created, the more time spent on finishing the divide and conquer process.
Graph2 shows an increase in the performance for Kruskal's algorithm with a certain amount of threads < 24, but a decrease in performance with thread count > 24 due to the cost of managing the threads, which exceeds the advantage gained from multithreading.
Graph3 shows an total time.

Table 1

	tri	krus	total
1	**0.01985**	**0.01825**	**0.0381**
2	0.0381	0.126	0.1641
3	0.0383	0.12625	0.16455
4	0.03885	0.093	0.13185
5	0.0384	0.09355	0.13195
6	0.03845	0.0861	0.12455
7	0.03855	0.0843	0.12285
8	0.04005	0.08295	0.123
12	0.0414	0.0819	0.1233
16	0.0426	0.08355	0.12615
24	0.04475	0.08285	0.1276
32	0.0465	0.09585	0.14235
48	0.04665	0.10355	0.1502







1

The second result was observed with animation mode -a 2, which included the time spent on painting. We believe this result was better because in this case, the threads running in the background performed the triangulation and Kruskal's while the main threading took care of the painting. The overlapping time reduce the emphasis we saw in result one.

Here is an url where you can check out a demo with -s 1 -t 30 :
http://csug.rochester.edu/users/ugrads/mliu26/MST.html