ECEN 521  
Oct 06 2015  
Matthew James  
Connor Smith  
Ricky Wyman’s Brother

Design Project 1

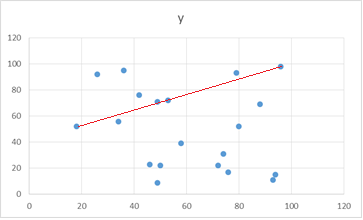
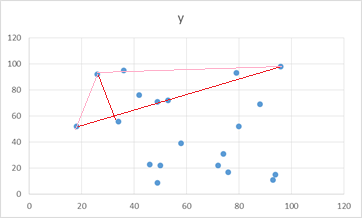
 We chose to implement the Quick Hull algorithm as our solution to the Convex Hull problem. This design gets its name from the Quicksort algorithm, with which it shares some similarity. Its normal complexity is but in the worst case can be as bad as . It is a divide and conquer type algorithm, with each iteration, the number of points we have to look through is greatly reduced.

Figure 2

Figure 1

Our implementation of this algorithm takes the points with the lowest x value and the highest x value (and adds them to our list of convex hull points--see Figure 1) and connects a line between them. It then iterates through all the points above this line and finds the one that is furthest away (and adds it to the list of points in our convex hull--see Figure 2). Using that third point it makes a triangle. With that triangle we go through each point and using the cross product of the different vectors forming the triangle we find out if the different points are within the triangle or outside of it. Using that data, we can make a new line and repeat the process. We continue this recursively, throwing away any points within the triangles until there are no points remaining outside the triangle. We then do the same thing for the bottom half.

The problem is not too difficult as the algorithm was well documented. The only difficulties were finding out implementation details such as computing cross products and comparing the sign of two different ones. Our implementation does not return the data in sorted form so we did need to sort it afterwards which we did by computing the angle of the different points in relation to the first point and then using that to make a counter clockwise sorted list.

Running several test cases we found the performance to be quite good. We were able to run a 10,000,000 point set in around 18 seconds. A smaller 1,000,000 point set is computed in a little under 2 seconds. Code profiling indicates that most of the time spent in the code is the parsing of the file. (One profiler indicated that for the 18 second run, 16 seconds were spent parsing the input file). Even with the time spent parsing the input, the algorithm does seem to live up to its name, and appears to execute in n\*log(n) time.