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

Design Project 1

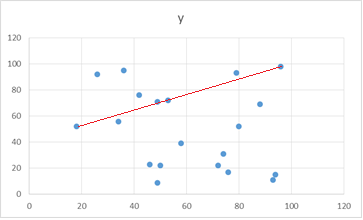
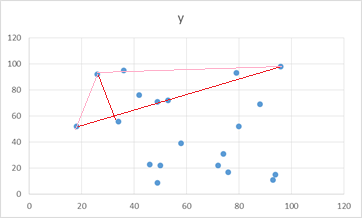
 We chose to implement the Quick Hull algorithm as our solution to the Convex Hull problem. This design gets its name from the Quicksort algorithm, with which it shares some similarity. Its normal complexity is but in the worst case can be as bad as . Since is a divide and conquer type algorithm, the number of points we have to look at each iteration 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. A temporary line is then created connecting the two points (see Figure 1). We then iterate through all the points above this line and find the one that is furthest away, adding it to the list of hull points (see Figure 2). A triangle is formed between the two extreme points and the farthest point. We then iterate over each point and compare the sign of the cross product between two of the legs of the triangle (AB and AC) with the sign of the cross product between one leg of the triangle (AB) and a line connecting the same leg of the triangle and the current point (AP). If the signs of these two cross products differ, the current point is on the side of the triangle with the line AB. The list of points is divided based on which side of the triangle each point is on, and any points within the triangles are discarded. The process then repeats recursively by finding the farthest point from each line and forming new triangles. The final list of points forming the convex hull is given when there are no points remaining outside the triangles.

P

B

C

A

The problem was not too difficult, as the basic algorithm was well documented. The main difficulties were in fleshing out the implementation details, such as determining which cross products to compute and how to compare the signs. Also, our implementation initially did not sort the list of points as they were computed, so we did need to sort the list afterwards. We did this by computing the angle of the different points in relation to the first point and using the angles to form a counter-clockwise sorted list.

After running several test cases ranging in size from 50 to 10 million points, we found the performance to be quite impressive. We were able to compute the convex hull for 10 million points in under 7 seconds. A smaller 1,000,000 point set is computed in about 0.74 seconds. Code profiling indicates that most of the time spent in the code is the parsing of the input file. (One profiler indicated that for the 7 second run, 6.13 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.