Brian Pinson
U91813366
Project 2
10/30/2017

I. Algorithms

Algorithm: Constructor
Input: text file of coordinates in counter-clockwise order
Output: Diagonals needed to triangulate simple polygon

```
1. build circular doubly-linked list of vertices from input file
2. set highest vertex as first node in doubly linked list
3. vertexTypeFinder();
4. build circular doubly-linked list of edges from input file
5. Make_Monotone();
6. print list of diagonals
7. end;
```

Algorithm: vertexTypeFinder()
Input: highest vertex from list
Output: type set for all vertices in list

```
1. for( all vertices v in the linked list of vertices)
2.      if( v is higher than v.next and v is higher than v.prev)
3.           if( a horizontal line of v passes through an even number of edges)
4.                v = start;
5.           else
6.                v = split;
7.      else if( v is lower than v.next and v is lower than v.prev)
8.           if( a horizontal line of v passes through an even number of edges)
9.                v = end;
10.          else
11.               v = merge;
12.     else
13.          v = regular;
14. end;
```

Algorithm: Make_Monotone()
Input: text file of coordinates in counter-clockwise order
Output: Diagonals needed to triangulate simple polygon

```
1. build priority queue Q of vertices listed from highest to lowest
2. build empty edge BST *T
3. while ( !Q.empty())
4.      vertex v = Q.top();
5.      Q.pop();
6.      if(v vertex type is start)
7.           Handle_Start(v);
8.      if(v vertex type is end)
9.           Handle_End(v);
10.     if(v vertex type is split)
11.          Handle_Split(v);
12.     if(v vertex type is merge)
13.          Handle_Merge(v);
```

14. if(v vertex type is regular)
15. Handle_Regular(v);
16. end;

Algorithm: Handle_Start(vertex v_i)
Input: vertex v_i whose type is start
Output:
1. insert edge e_i of vertex v_i into T
2. e_i->helper = v_i
3. end;

Algorithm: Handle_End(vertex v_i)
Input: vertex v_i whose type is end
Output:
1. find e_{i-1}->helper
2. if(e_{-1}->helper == merge)
3. insert (v_i, e_{i-1}->helper) into diagonal list
4. Delete e_{i-1} from T
5. end;

Algorithm: Handle_Split(vertex v_i)
Input: vertex v_i whose type is split
Output:
1. search T and find edge e_j left of v_i
2. insert (v_i, e_j→helper) into diagonal list
3. e_j->helper = v_i;
4. insert e_i into T
5. e_i->helper = v_i
6. end;

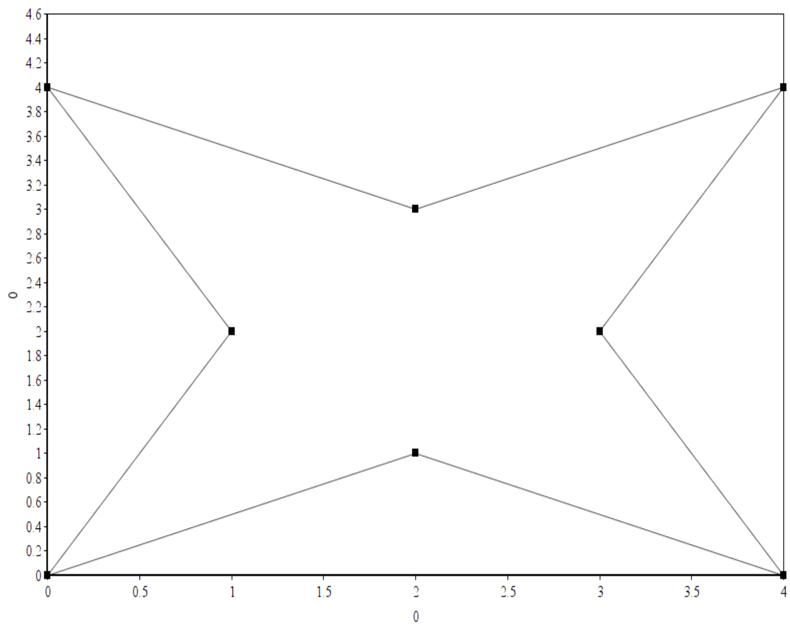
Algorithm: Handle_Merge(vertex v_i)
Input: vertex v_i whose type is merge
Output:
1. if (e_{i-1}->helper == merge)
2. insert (v_i, e_{i-1}->helper) into diagonal list
3. Delete e_{i-1} from T
4. search T and find edge e_j left of v_i
5. if (e_j->helper == merge)
6. insert (v_i, $e_{j\text{->}}$helper) into diagonal list
7. e_j->helper = vi;
8. end

Algorithm: Handle_Regular(vertex vi)
Input: vertex v_i whose type is regular
Output:
1. if (a horizontal line of v_i passes through an odd number of edges to the right)
2. if(e_{i-1}->helper == merge)
3. insert (v_i, e_{i-1}->helper) into diagonal list
4. Delete e_{i-1} from T
5. insert e_i into T
6. e_i->helper = v_i
7. else

```
        search T and find edge e_j left of v_i
        if(e_j->helper == merge)
                insert (v_i,  e_j->helper) into diagonal list
                if(e_j->helper = v_i)
end;
```

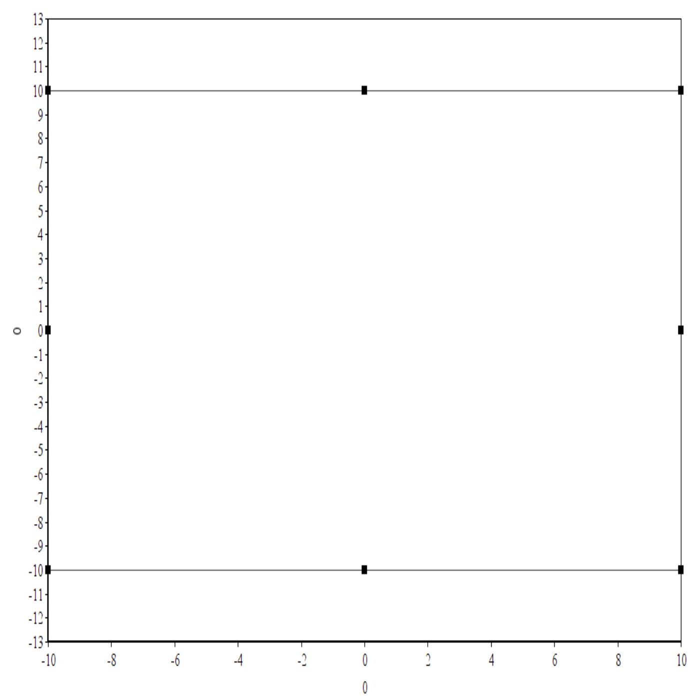
II. Test Case Examples

Test Case 1: Shuriken.txt
8
0 0
2 1
4 0
3 2
4 4
2 3
0 4
1 2
0 0



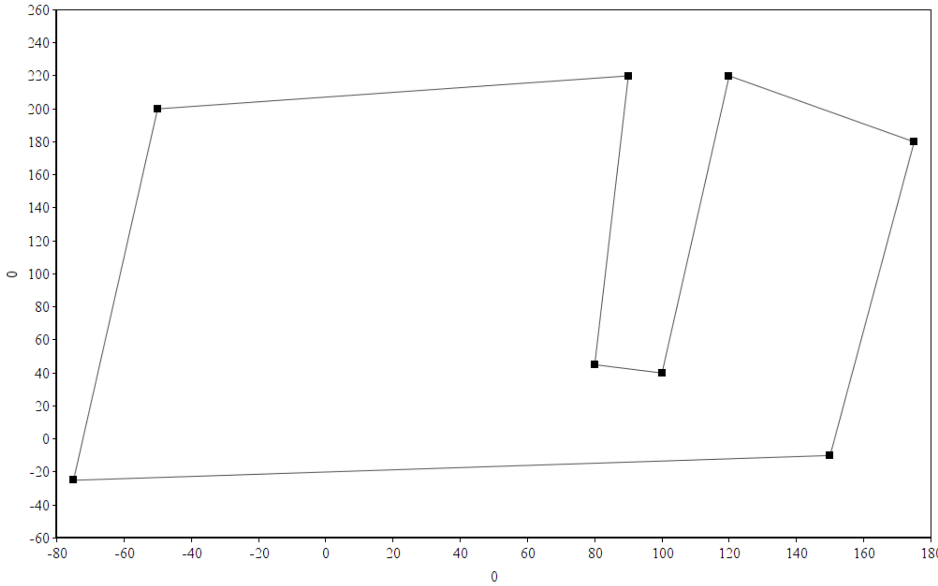
It’s a good way to test all five possible types of vertices: start, end, split, merge, and regular.

Test Case 2: Square.txt
8
-10 -10
0 -10
10 -10
10 0
10 10
0 10
-10 10
-10 0
-10 -10



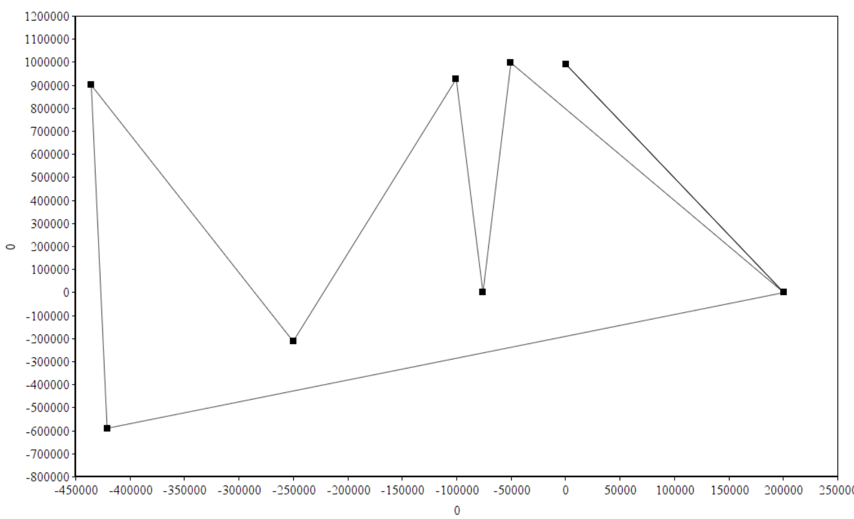
Square.txt was designed to test the algorithms ability to handle two consecutive edges that were parallel. It would be necessary to delete the center point (v_{i+1}) and link v_i to v_{i+2} and v_{i+2} to v_i.

Test Case 3: Chipped.txt
8
-50 200
-75 -25
150 -10
175 180
120 220
100 40
80 45
90 220
-50 200



Chipped.txt is designed to test if the program is capable of distinguishing if two points make a line that are inside or outside the polygon. It is necessary for when deciding if a point is the helper for an edge.

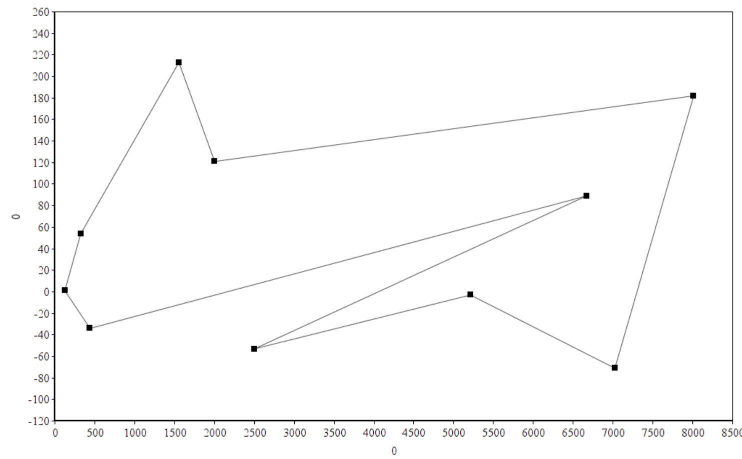
Test Case 4: Extreme.txt
9
-435678 900546
-421125 -589234
200456 -24
55 992435
200455 -5
-50234 997544
-75894 -25
-100456 927532
-250231 -211899
-435678 900546



Extreme.txt takes random high value numbers to test the programs ability to handle computation with extreme numbers. The far right most point appears to be connected to a line, that line is actually two edges forming a triangle so thin it cannot be accurately graphed. This is to test how well the program can handle possible floating point errors.

Test Case 5: Amorphous.txt

10
324 54
124 1
441 -34
6667 89
2502 -53
5212 -3
7023 -71
8012 182
2003 121
1555 213
324 54



Amorphous.txt is designed to test all parts of the program from its ability to distinguish if a potential helper is inside the polygon, it’s ability to identify various types of vertices, and diagonals at various angles.

III. Results

The program measured testing convex polygons. Polygons can be elaborate in shape with the vertexes: start, end, split, merge, and regular needing different functions. Start vertexes finishes in log(n) time as it must insert an edge into a tree. End vertexes need to calculating an edges helper requires a more complex algorithm to computer all adjacent vertices and finding the lowest one. In the graph’s shape implies n log (n) time. The cost is below quadratic in complexity.