Analysis of the Triangulation Algorithm:

The Triangulation Algorithm that has been used in this programming assignment consists of two primary steps. The first step is formation of convex layers and the second step is to triangulate the points using the boundaries of the convex layers formed.

Convex Layers Algorithm:

In-order to make the convex layers from the set of points, a regular convex-hull divide and conquer algorithm that was submitted in the last HW assignment is used. The convex layers algorithm first forms the convex boundary and returns the set of points. This set of points is subtracted from the initial set of points, that we are triangulating and the convex hull algorithm is called again. In this way we create multiple convex layers, CH(S1), CH(S2), CH(S3), CH(S4)…...CH(Sk) until we are only left with one/two points. The pseudo code is as follows:

```
ConvexLayers = []              #Store all the convex boundaries
S                                      #Set of Points
While len(S) > 1
        S1 = CH(S)
        ConvexLayers.append(S1)
        S – S1
End
```

Complexity:

The convex layers algorithm uses divide and conquer CH algorithm to make the layers. The Divide and Conquer method makes the hulls in “nlogn” time. The number of points to make a convex hull out of decreases for every iteration. Which means, that for next iteration it would take (n-k)log(n-k) time and this n decreases continuously as we progress towards the end. If we simply this to “k(nlogn)” time, this “k” does not scale with n and is much smaller than n. Therefore, even though one might think that this algorithm would be n^2logn, that is not the case.

Triangulation of Convex Layers Algorithm:

In order to triangulate the convex layers, identification of visible vertices to a vertex on another layer is necessary. In-order to do so, the rotating calipers algorithm is employed. The implementation can be visualized from the following picture.

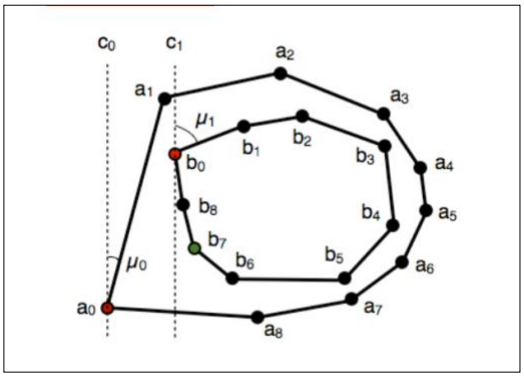


Image Source: Google Images

The implementation is as follows. Although any points can be chosen at random, this implementation choses the left most points A[0] and B[0]. Next, an edge is drawn between A[0], A[1] and another edge is drawn between B[0] and B[1]. We conduct a test to check the direction of the area as follows, Area([0,0], [B[0] – B[1]], [A[0] – A[1]]). This direction tells us which point to choose. This logic is written in the pseudo code below:

```
For all the convex layers
        A = Outer Layer
        B = Inner Layer
        While k <= Len(A) & m <=Len(B)  #This is to limit the traversal
        FindArea([0,0], [B[m] – B[m+1]], [A[k] – A[k+1]] )
        If area > 0
                        Triangle =  A[k], B[m],A[k+1]
                        k++
        If area < 0
                        Triangle = B[m],A[k],B[m+1]
                        M++
        Else
                        Triangle = B[m], B[m+1], A[k]
                        Triangle = A[k], B[m+1], A[k+1]
                        m++
                        k++
        If B consists of only 1 point
                        Draw triangles from that point to all the points on the outer hull
```

Complexity:

From initial instincts, complexity of the triangulation algorithm can be seen quickly. There is a while and for loop inside a main for loop, therefore the complexity has to be O(n^2). But thinking about this conceptually gives us more insight. As deduced previously, the number of convex layers is very less compared to the total number of points. Next, the Length of A and Length of B usually will be different with inner layer having lesser points than outer layer. BUT, in the worst case scenario, both of them can have the same number of points N. Which would make the total complexity of this step, kO(N).

Total Complexity:

Total Complexity = complexity of CH layers + complexity of CH layer trianguation.
=> Total Complexity = k(nlogn) + k(n) = k(nlogn) where k<<n.

Description of the Data Structure:

The data structure consists of 3 classes. The vertex class, the edge class and the face class. High level description of the classes is as follows:

```
Class Vertex:
        Stores the X,Y location
        Stores all the edge objects that share the vertex in an array
        Stores all the face objects  that share the vertex in an array
        //Methods
        Getter and Setter methods for setting and removing information

Class Edge:
        Stores the 2 Vertex Objects that define it
        Stores the Face Objects that Share it in an array
        //Methods
        Getter and Setter methods for setting and removing the information

Class Face:
                Stores the 3 Vertex Objects that define it
                //Methods
                Getter and Setter methods for setting and removing information
```

All these objects are stored in global arrays so that retrieval of these objects can be done in O(N). In order to make these objects, the following procedure is used:

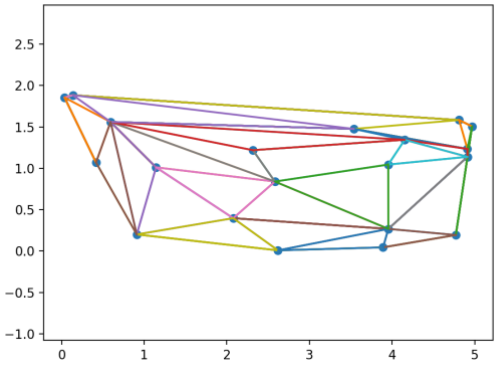
```
Triangle Faces = Triangulaate(Convex Layers)  // All the triangles with points are retrieved here

For i in Triangle Faces
        Create Vertex Objects and Edge Objects from the vertices of the face
        //Storing Face Data
        1. Add Face objects to the global face array. No need to check for duplicates as triangle points
                are unique
        //Storing Vertex Data
        1. Add these vertex objects to the global vertex objects array after checking for duplicates
        2. To these vertex objects add the edge Objects and Face Object[i]
        //Storing Edge Data
                1. Add the Edge Objects to the global Edge array after checking for duplicates
        2. To these Edge Objects, add face objects
```

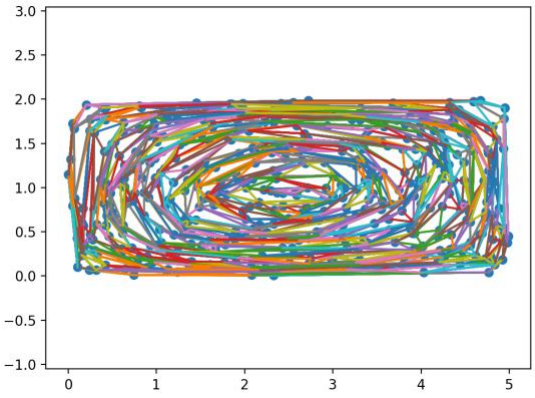
In order to check for duplicates for vertices and edges, two methods that check the X,Y of the vertices and Point A, Point B that define an edge respectively have been written. Because faces are defined by points, the points can be retrieved from a face In O(1). The getter method to get faces from edges directly returns the first and the second index of the faces array in edge class thus making the process O(1). Next, the edges and faces sharing a vertex can be returned in O(k) as the

arrays are returned and the arrays can be traversed to get the objects. Finally, all the faces, edges and vertices can be retrieved in O(n) as all the objects are stored in global arrays.

Testing:

Testing has been done using the input file generator provided for HW2. Different number of points were chosen at random. The number of points that were used include 20,40,30,300,200,50,60 etc. Below are some of the results of testing (Plots).





Testing: