Comparison of algorithms for solving convex hull

Ajinkya Shinde

CS 584/ 684 – Algorithm Design and Analysis

Spring Term, 2019

Introduction

The purpose of this project is to compare two efficient algorithms for solving the convex hull problem. For comparison, we evaluate the algorithms on a variety of input types like random scatter, circle. The algorithms which we evaluate for comparison are Jarvis March and Graham Scan. We perform different tests on the variety of inputs mentioned above and then evaluate the efficiency of the two algorithms. Below is the overview of convex hull problem

Convex Hull Overview

Convex Hull of a set Q of points in the euclidean space is the smallest convex polygon P that contains Q such that each point in Q is either on the boundary of polygon P or in its interior. In this project, the set Q of points lies within a finite set. Solving convex hull is a type of computational geometry problem. Convex hull problem can also be visualized as given the points in Q ,we can think each point as being a nail sticking out from a board. The convex hull is then the shape formed by a tight rubber band that surrounds all the nails. Below is the convex hull for set of points

Figure 1: Convex hull of a finite set of points

Motivation behind project

Creating art is one of the hobbies that I pursue. Being a computer science major, one of the areas that excites me is the algorithmic art generation. Since, a lot of computational geometry problems like Voronoi diagrams, fractals, convex hull involve using algorithms to create art, it is one of the reasons I decided to pursue this project.

Algorithms Overview

Graham Scan Algorithm - This algorithm solves the convex hull problem by maintaining a stack S of candidate points. It is named after Ronald Graham, who published the algorithm in 1972. Briefly, the algorithm pushes each point of the input set Q of points onto the stack at one time and pops the points that is not a possible vertex of convex hull. Below is the pseudo-code for graham scan algorithm

```
GRAHAM-SCAN(Q)

(1)   let p0 be the point in Q with the minimum y-coordinate,
        or the leftmost such point in case of a tie

(2)   let  p1, p2, . . . , pm  be the remaining points in Q,
        sorted by polar angle in counterclockwise order around p0
        (if more than point has the same angle, remove all but
        the one that is farthest from p0)

(3)       let S be an empty stack

(4)       PUSH(p0,S)
(5)       PUSH(p1,S)
(6)       PUSH(p2,S)

(7)       for i = 3 to m
(8)           while the angle formed by points NEXT-TO-TOP(S),  TOP(S),
                  and pi makes a nonleft turn
(9)               POP(S)
(10)          PUSH(pi,S)
(11)    return S
```

Figure 2: Graham Scan Psuedo-Code

Explanation of Pseudo-code

The above procedure GRAHAM-SCAN(Q) takes an input set Q of points. Line (1) first finds point with minimum y co-ordinate and in case of ties choose the point with minimum x co-ordinate or the leftmost point which is labelled as p0 . The point p0 is the vertex of convex hull since it has the lowest x and y co-ordinate. Line (2) then sorts the remaining set of points p1,p2 till pm(where m is the number of points other than p0) in Q using polar angle with respect to p0. The polar angle is calculated using the cross product of two vectors. For sorting, any sorting-based algorithm with O(n.log n) is used. We sort the points from smallest to largest polar angle and label them correspondingly from p1, p2 till pm. Line(3)-(6) creates an empty stack S and pushes the points p0, p1 and p2 onto the stack. Lines (7)-(10) then performs the test on each candidate point pi to see if it a possible vertex on convex hull. For this, it calls the procedure NEXT-TO-TOP(S) and TOP(S) which returns the second element and first element from top of the stack respectively. Then, it takes the next candidate point pi and checks if pi is on the left side of line segment formed by points NEXT-TOP(S) and TOP(S). If it is on the left side, we simply push pi on to the stack and do the test for the next candidate point from Q. If it is not on the left side , then the point returned by TOP(S) lies in the interior. We then pop the elements from the stack until the point pi is on the left side of points with respect to TOP(S) and NEXT-TO-TOP(S) points and then push pi onto the stack. We perform this process till the point pi is the same as P0. And then finally return the stack S in Line (11).

Algorithmic Complexity of Graham Scan

The graham scan algorithm's complexity is O(n . log n). This complexity comes from the sorting done at Line (2) and it predominates all the cost of other steps in the graham scan algorithm. For sorting, any sorting-based algorithm like insertion sort, merge sort etc can be used. Although there is a nested loop in Line (7)-(10) and it seems that the algorithm's complexity should be O(n^2), the actual inner loop executes at most once for every outer iteration. Since, we pop the elements from the stack S only when addition of the candidate point pi causes a non-left turn, it is not always that case. Thus, the inner loop executes at most once and the overall complexity of Line (7) – (10) is O(n) which is dominated by sorting cost. The cost of other lines in the algorithm is almost cost and can be ignored. Hence, the overall complexity of graham scan comes to O(n. log n)

Jarvis March Algorithm

This algorithm also computes the convex hull of a set of points – Q by simulating the technique of package or gift wrapping around the convex hull vertices. Visually, we start by taping the end of paper to the lowest end point , in our case it is p0. Then, we pull the paper to the right to make it taut and pull it higher until it touches a point. This point should also be the vertex of the convex hull. We keep this process until we reach the endpoint from where we started i.e. p0. Below is the pseudo-code for the algorithm.

```
JARVIS-MARCH(Q)

(1)   Create a temp sequence hullV = [] to store the vertices of convex hull

(2)   let p0 be the point in Q with the minimum y-coordinate,
        or the leftmost such point in case of a tie. Append
        p0 to the temp sequence H

(3)   Initialize pointonHull to store the candidate vertex point and initialize
        it to p0

(4)        i = 0
(5)        while True
(6)      Store pointonHull at hullV[i]
(7)      Initialize endpoint to p0

(8)      for j = 1 to m
(9)          if endpoint == P0 or p[j] is on the left side from line pi to endpoint
(10)             endpoint = p[j]

(11)        pointonHull = p[j] i.e. is the next vertex of convex hull

(12)        i = i + 1

(13)  until endpoint == P0
(14)  return hullV[]
```

Figure 3: Jarvis-March Psuedo Code

Explanation of Pseudo-code

The above procedure JARVIS-MARCH(Q) takes an input set of points Q and returns the convex hull stored in temporary sequence/ array hullV[] which is created at Line (1) . Similarly, to graham scan in Line(2), we let p0 to be the minimum point which is leftmost among all the points. At Line(3), we initialize pointOnHull to store a candidate vertex and then using it we perform test on other points. At Line (3), we first set it to point p0, the test then checks for rest points p[j] if point[j] is the right-most point with respect to other points in set Q. If point p[j] is the leftmost point then it must be the vertex of convex hull. Thus, for every point p[j], we check it with all the points from p[j-1] till rest of points in Q. At Line(11), we then take the leftmost point p[j] and then find the next potential vertex of the convex hull in the next iteration. We also store the point p[j] at start-of next iteration in the hullV[] at index i+1 where i is the index from previous iteration.

Algorithmic Complexity of Jarvis March

We perform the check on point p[j] with respect to all other points in the set Q. Thus, if there are h number of hull vertices of the convex polygon, we check if there are rightmost with respect to all other points or in other words, all the points other than p[j] are on the left side. Since there are n vertices in the set Q, the time complexity of Jarvis March algorithm is O(n*h) where n is the number of points in the set Q and h is the number of hull vertices of convex polygon. In other words, for every ‘h’ operation, we perform O(n) operations resulting in O(n*h) time complexity