Geometric Algorithms for Convex Hull Problem: A Survey

Shaishav Harshadbhai Shah
UFID: 1136 - 3317 - Email: shah.sh@ufl.edu
Department of Computer & Information Science & Engineering, University of Florida

ABSTRACT
Computation of convex hull is a fundamental geometric problem that has applications across a range of industries from image processing, image editing softwares, computer graphics simulations, robot motion planning, collision detection algorithms, data analytics etc. There are many algorithms proposed for computing convex hull in 2 dimensions as well as multi dimensional space. This article provides summary of few of these algorithms and a note on few applications of convex hull.

INTRODUCTION
The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.[1] We can visualize convex hull by imagining that the set of vertices as nails sticking out of the plane, and we have a rubber-band elastic enough that we can really stretch big and tight enough that it can shrink to almost the smallest possible shape. Now, when we stretch this elastic rubber-band around the nails and let it go, it will stop at the outermost nails and the shape formed by those nails is called convex hull.
In this article we will try to study two optimal geometric algorithms to solve convex hull problem namely Quickhull and Chanś algorithm. We will also look at some of the applications of convex hull. For simplicity, the algorithms are presented here by considering the points on a plane as itś input, but it can also be applied to line segments and other complex objects.

ALGORITHMS
For all the algorithms noted here, consider that P is a set of n vertices available on a $2d$ plane for which convex hull has to be computed.

1. Quickhull
Quickhull algorithm for finding convex hull uses divide and conquer approach. Quickhull starts by drawing a line between two vertices which are farthest to the left and right on a 2d plane; this two vertices must be on a convex hull as they are extreme vertices and the line drawn partitions P into two subsets, one below the line and the one above it.1
In the next step two more lines are drawn from the two end vertices of the line to the vertex farthest from the line.2
This step also partitions P into more subsets. The vertices inside the shape formed by this step cannot be part of the convex hull, and hence they can be ignored. The second step is repeated until all vertices are either part of the convex hull or contained inside of it.[2]
The average running time of Quickhull algorithm is O($n * log(n)$), while the worst case if O(n^2). The worst case occurs when all vertices in P are part of the convex hull, and the best case occurs when each partition routine divides the vertices evenly and the number of vertices on the convex hull is small as compared to the number of vertices contained within the convex hull.
2. Chanś algorithm (Shattering)
Chan's algorithm is an optimal output sensitive algorithm. The basic idea is to use an observation that some vertices of the input set will never be on convex hull, so discard them to speed up Jarvisś March. It works by using combination of Grahamś Scan and Jarvisś March algorithms. So letś take a brief on how these two algorithms works first.

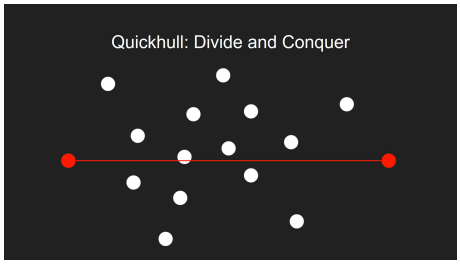

Figure 1. Quickhull: Divide and Conquer - Step 1

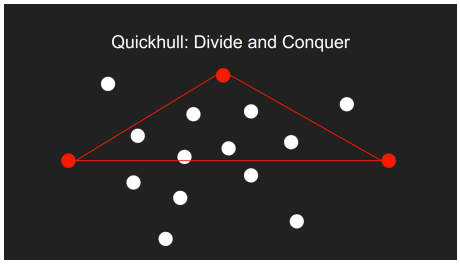

Figure 2. Quickhull: Divide and Conquer - Step 2

2.1. Jarviśś March (Gift wrapping)

The core idea of Jarviśś algorithm is to focus on identifying convex hull edges as opposed to the vertices. It starts by picking up the left-most vertex u i.e the vertex with smallest x co-ordinate value on the plane; this vertex must be on a convex hull. It then finds v by taking the right-most vertex with respect to u. After finding u and v, it performs a series of edge selection steps where in each step, given an edge uv of the convex hull, the next edge vw is selected such that it maximizes the angle uvw. At each edge selection step there are n vertices scanned and this is done h times - where h is the number of vertices on convex hull, so it is output sensitive algorithm, and hence it runs faster for smaller output. [1, 4]

Running time: O($n * m$) - where m denotes the number of vertices on the convex hull.

2.2. Grahamś Scan

Grahamś scan algorithm uses sorting in order to solve the convex hull problem. Like Jarviśś March, it starts by picking up the left-most vertex u i.e the vertex with smallest x co-ordinate value on the plane; this vertex must be on a convex hull. It then sorts all the other vertices on the plane in counterclockwise order around u. After sorting, the basic idea of the algorithm is to perform a single scan around the sorted vertices, and eliminate the ones that are not extreme. The elimination process works by repeatedly selecting 3 consecutive vertices from the sorted list starting from the first vertex let's say v_i, $v_{i_{+_1}}$ and $v_{i_{+_2}}$ and performing the orientation test. We will eliminate $v_{i_{+_1}}$ if it is making a right turn and select $v_{i-1}v_iv_{i+2}$ else we will select $v_{i+1}v_{i+2}v_{i+3}$ for the next iteration. This process is repeated until we hit u again. At the end, what we are left with is the convex hull vertices in the counterclockwise sorted order. [1, 5, 6]

Running time: O($n * log(n)$)

Now, letś discuss how Chanś algorithm uses these two algorithms to build a better running time algorithm.

Chanś algorithm works by computing the value of m(the number of vertices on the convex hull) in advance; letś not focus on the implementation of this compute step for now. Letś see how knowing the value of m in advance makes a difference. [3]

Once we have discovered m, in the first phase we will divide P into n/m arbitrary subsets of each of size m. It then computes convex hull for each subset by following Grahamś Scan algorithm. The running time for this part can be given as O(n/m * ($m * log(m)$)) = O($n * log(m)$).3

At the end of first phase, we will have n/m mini convex hulls using which we will compute the global convex hull. In the second phase, Jarviśś March is executed on pre-computed mini convex hulls.4 Starting with $u = l$, where l is the left-most vertex on the plane, we successively find the convex hull vertex that follows u in counterclockwise order until we return back to l again. The successor of u must lie on a right tangent line between u and one of the other subhulls - a line from u through a vertex of the subhull, such that the subhull lies completely on the right side of the line from uś point of view. We can find the right tangent line between u and any subhull in O($log(m)$) time using a variant of binary search. Since there are n/m subhulls, finding the successor of u takes O(n/m * $log(m)$) time altogether. Since there are m convex hull edges, the running time of this algorithm can be computed as O(n/m * ($m * log(m)$)) = O($n * log(m)$).

The worst case running time of this algorithm O($n * log(m)$) is better than other convex hull algorithms, but it is heav-

ily dependent upon the knowledge of m. So letś discuss the process of calculating m.
Chanś trick is to guess the correct value of m; letś denote it by m*. The goal is to make m* very close to m, or equal. The guessing process starts by taking a parameter t and setting $t = 1$ and m* = 2^{2^t} initially; t denotes iteration here. Chan then executes his algorithm using this m* and keeps a counter to calculate the number of output points in the hull; letś denote it by h. If by the end of an iteration, we found h to be too small, we will perform the next iteration by incrementing t by 1 and re-calculating m*. This process is repeated until m* $< m^2$. The last iteration has lower bound of O($n * log(m*)$) = O($n * log(m^2)$) = O($n * log(m)$).[7]
The total running time of Chanś algorithm is the summation of time taken by all the iterations.

$$O(\sum_{t=1}^{ceil(log(log(m)))} n * 2^t) = O(n * 2^{ceil(log(log(m)))+1}) = O(n * log(m))$$

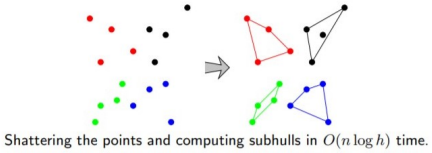

Figure 3

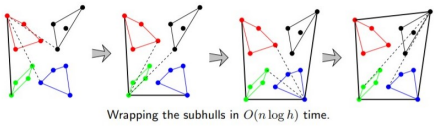

Figure 4

APPLICATIONS

Data Analysis: Consider a basketball game where the players are spread across the court. By collecting the data and analyzing the convex hull for each team we can gather statistics and perform analysis considering territory control, positioning and movement.
Robot motion planning: Consider a robot which has to move from point A to point B by following shortest path and there are many obstacles between this two points. In order to get from A to B, the shortest path will either be the straight line from A to B (if the obstacle doesn't intersect it) or one of the two polygonal chains of the convex hull.
Computer graphics: Consider the simulations in Computer graphics. Simulations that involves collisions have concepts called collision meshes which are used to denote that collisions take place. By using convex hull, the way collisions are handled can be simplified. For example, walking up a ramp is easier to animate than walking up stairs so we can use a collision mesh to make an object interact with stairs as they would with the ramp using the computed convex hull.

REFERENCES

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.Introduction to Algorithms, Third Edition. The MIT Press, 3rd edition, 2009
2. C. Bradford Barber, David P. Dobkin, and David P. Dobkin, Hannu Huhdanpaa. 1996. The quickhull algorithm for convex hulls. ACM Trans. Math. Softw. 22, 4 (December 1996), 469-483. DOI: https://doi.org/10.1145/235815.235821
3. Chan, T.M. Discrete Comput Geom (1996) 16: 361. https://doi.org/10.1007/BF02712873
4. Jarvis, R. A. (1973). "On the identification of the convex hull of a finite set of points in the plane". *Inform. Process. Lett.*, 2: 18-21, 1973.
5. R.L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132-133, 1972.
6. Suneeta Ramaswami, "Convex Hulls: Complexity and Applications (a Survey)", . December 1993.
7. Convex Hull: https://www.cise.ufl.edu/class/cot5520sp18/Convexhull.pdf