Technical Report of the Quickhull Algorithm

Oliver Wootton

December 11, 2022

Abstract

The convex hull is a fundamental construct in computational geometry, it offers a simple way to approximate a point set's shape. The Quickhull algorithm expands a divide-and-conquer approach to efficiently output the convex hull of a set of vertices.

I certify that all material in this report which is not my own work has been identified.

Signature: O.Wootton

1 Introduction

The Quickhull algorithm is the solution to the convex hull problem. The convex hull is the set of vertices that form a shape containing all other vertices within it, as seen in figure 1. Determining the convex hull is a fundamental geometric problem that has many applications across a range of industries.

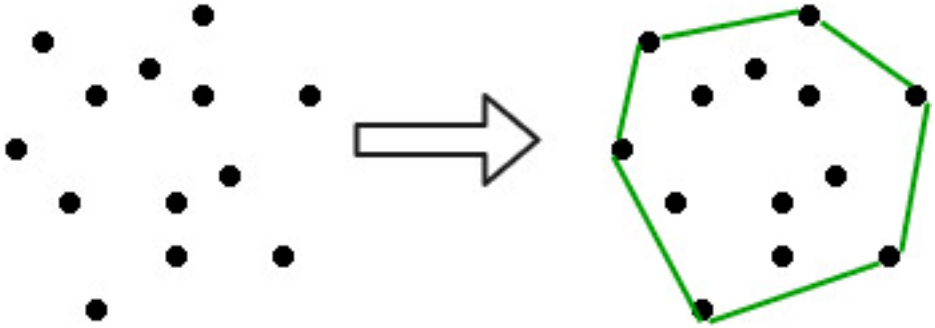

Figure 1: A set of vertices and its convex hull

2 Principals of the algorithm

The Quickhull algorithm uses a divide-and-conquer approach. First, a line is drawn between the points furthest to the left and right on a 2-dimensional plane, these points are part of the convex hull, and the line partitions the graph into two subsets. Then lines are drawn from the two points on the line to the point furthest from the line, this point is also apart of the convex hull. The points inside the shape already formed cannot be part of the convex hull, therefore can be ignored. This process is repeated until all nodes are either a part of the convex hull or contained within the convex hull. The aim of this algorithm, like the Quicksort algorithm, is to eliminate as much of the problem as quickly as possible[1].

3 Pseudo code of the algorithm

Algorithm 1 Quickhull

Require: A set S of n points
Ensure: *S has more than 2 points*

function QuickHull(S) *ConvexHull = ()* A $\leftarrow$ *left most point*
$B \leftarrow$ *right most point*
$ConvexHull+ = A$
$ConvexHull+ = B$
$S1 \leftarrow$ *points left the line AB*
$S2 \leftarrow$ *points right the line AB*
$FindHull(S1, A, B)$
$FindHull(S2, A, B)$
end function
function FindHull(Sk, P, Q) {Find points on convex hull from the set Sk of points that are on the correct side of the line from P to Q}
if Sk is empty **then**
    return
end if
for all $points$ in Sk **do**
    $C \leftarrow$ *Furthest point from segment PQ*
end for
$ConvexHull+ = C$
$S1 \leftarrow$ *points left the line PC*
$S2 \leftarrow$ *points right the line CQ*
$FindHull(S1, P, C)$
$FindHull(S2, C, Q)$
return $ConvexHull$
end function=0

The function FindHull as described in the pseudo-code above is illustrated in figure 2.

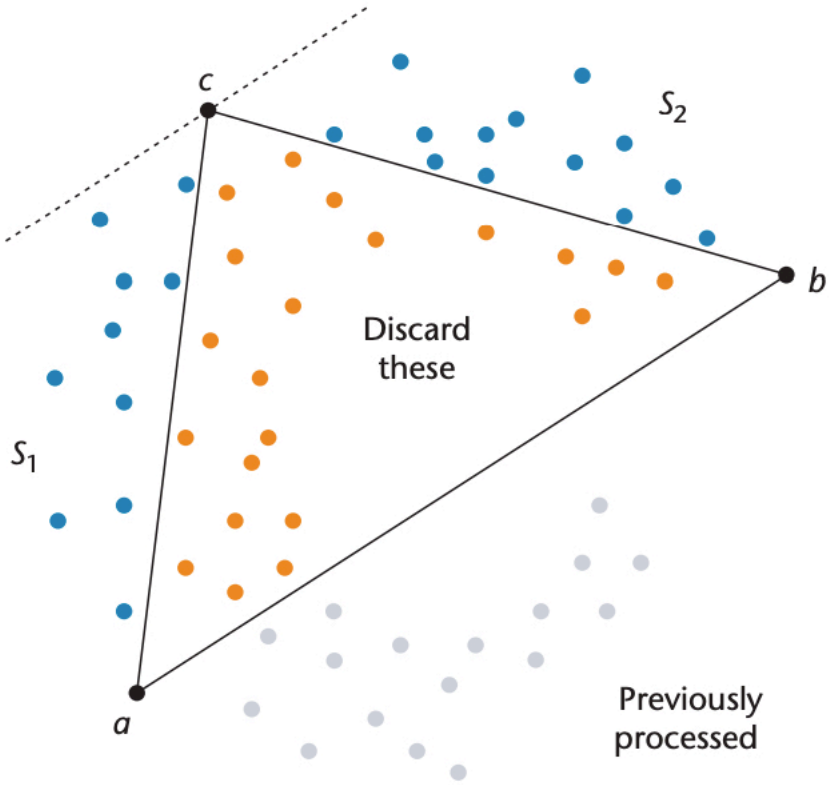

Figure 2: Quickhull's recursive step

4 Complexity (time and space) analysis of the algorithm

The time complexity of the Quickhull algorithm is $O(n^2)$ at worst and O(n(log(n)))[2] at best. The worst case occurs when all the vertices are a part of the convex hull. This requires O(n) calls to the FindHull function (line 8 of the pseudo-code), which requires O(n) computation[3] hence the worst case is $O(n^2)$. The best case occurs when each partition divides the nodes evenly and the number of vertices of the convex hull is small when compared to the number of vertices contained within.

As the algorithm uses an array to store the vertices for the convex hull. The algorithm uses a 2-dimensional array to store each coordinate which is why it cannot use a different data structure such as a binary search tree. The array has an access time complexity of O(1) and a search complexity of O(n) which is ideal for its use case in this algorithm. The space complexity of the algorithm is O(n) as it uses an array and only requires n extra space has been taken to compute the convex hull[3].

When comparing the time complexity of this algorithm to others that solve the convex hull problem, they all have similar if not the same time complexity. The only algorithm to perform slightly better is the gift wrapping (Jarvis march) algorithm with a time complexity of O(nh) where h is the number of points in the hull. This algorithm is faster than the rest but in practice when the number of points in the set is large, this algorithm tends to perform worse than others such as the Quickhull algorithm.

5 Limitations or constraints of the algorithm and the advances that could overcome them

The main limitation of the algorithm rises with the worst-case scenario. Using the example when the points are on the border of a circle. The algorithm's time complexity is dependent on the elimination of points; however, no points will be eliminated in the case of a circle as none of the points would lie within the convex hull. Unfortunately, this seems to be a limitation of the algorithm that has no solution.

Another limitation of the algorithm is that it only outputs the points of the convex hull but does not order them for a graphical output. Therefore, to overcome this limitation, within the Plot_Graphs.py file there is a function to order the convex hull vertices for a polygon so that the graph output show the shape as intended.

6 Applications of this algorithm in diverse areas for problem-solving (e.g. how the algorithm changed or is changing the world)

One application of this algorithm is in sports analysis. When put into the context of football, analysing the convex hull for each team when can gather statistics on territory control, positioning, and movement. This can also be useful when analysing the offside rule to show the positioning of the defence in comparison to the movement of the opposition, allowing teams to use this data to improve the line of their defence invoking the offside rule more often.
Computer graphics is another example of the use case for this algorithm. Simulations involving collisions have concepts called collision meshes, these are used to denote that collisions take place. These are used to simplify the way that collisions are handled. The animation of walking up a ramp is easier to animate than that of walking upstairs, so a collision mesh is used to make an object interact with stairs as it would with a ramp. Using the Quickhull algorithm is a quick and efficient way to mount the collision mesh of complicated shapes.
Collision avoidance is another application specifically used in the automation of driving[4]. If the convex hull of a car avoids collision with obstacles, then so does the car. Since the computation of paths that avoid collision is much easier with a convex car, then it is often used to plan paths.

References

[1] E. Mucke, "Computing prescriptions: Quickhull: Computing convex hulls quickly," *Computing in Science & Engineering*, vol. 11, no. 5, pp. 54–57, 2009.

[2] L. K. Nguyen, C. Song, J. Ryu, P. T. An, N.-D. Hoang, and D.-S. Kim, "Quickhulldisk: A faster convex hull algorithm for disks." *Applied Mathematics and Computation*, vol. 363, 2019. [Online]. Available: https://uoelibrary.idm.oclc.org/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0096300319306186&site=eds-live&scope=site

[3] J. S. Greenfield, "A proof for a quickhull algorithm," 1990.

[4] P. Sharma, "Quick hull algorithm to find convex hull," https://iq.opengenus.org/quick-hull-convex-hull/.