ECM3428 Coursework Assessment

690019495

November 2021

Abstract

This report explores the Quickhull algorithm, its main principles and how it works. This report looks at the time complexity in the best case and worse case scenarios. The limitations of the algorithm will also be explored. Furthermore, the applications of the Quickhull algorithm will also be investigated.

Word count: 1299

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

1 Introduction: Quickhull Algorithm

The definition of the convex hull of a set of planar points is the smallest convex set containing all the points [1]. The calculation of the convex set is to find the convex polygon that encloses all the points on the plane [2].

Algorithms for finding the convex hull can be viewed as a data-parallel problem [3]. For each iteration, the operations done per point is the same [3]. Each point goes through some test to see whether or not it should be included, and those points which are left are regrouped and tested again until the final set of convex hull points are determined [3].

1.1 The main principals of the algorithm

- Input: An array of n points p (consisting of planer points).
- Output: An array of points ch out which contains the points in p in that make up the convex hull

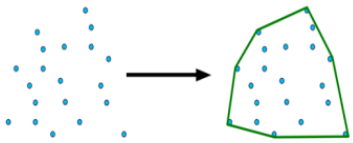

Figure 1: Illustration of Convex Hull in 2D [4]

The convex set encloses the every point within a set, as seen in Figure 1. The input is an array of points specified by their x and y coordinates. The output is a convex hull of this set of points in ascending order of x coordinates [1,3,4]. Quickhull shares many similarities with Quicksort from which its name derives, however it is more complex because of the dimensionality of the input data [3,4]. Quickhull also discards points from the input data it has deemed cannot be on the hull [3].

Quickhull is a type of divide and conquer algorithms, which shares the similar step of partitioning data into smaller subsets and operating on the subsets via recursive calls [1,3,6]. During each recursive call, Quickhull cuts down the workload as the algorithm goes deeper into the recursion by removing the unnecessary points that are not included in the hull.

2 Pseudo code

The Quickhull algorithm takes an input set of points in d dimensions. Let us consider the 2D case which this report uses. Quickhull begins by finding both the minimum and maximum (extrema) points along the x dimension and adding them to the convex hull.

The input set of points is divided into two subsets: the set of points below the line joining the extrema points and the set of points above the line. The extrema points of the input will always be vertices of the output convex line.

For each subset, the distance of each point to the dividing line is computed. The farthest point thus found is added to the convex hull. Then a triangle is constructed with the farthest point and the dividing line. All points in the subset which lies in the interior of this triangle are discarded. This step of division is carried on recursively with each subset until no points remain in any subset.

If we consider an extreme point p that is not assigned to an outside set and hence never added to the convex hull. Since p is an extreme point, it must have been outside at least one facet of the initial simplex [5]. By assumption, there is a point q with p in its visible outside set but not in its new outside set [5]. So p is above a visible facet and below all new facets for q [5]. This implies that p is inside the convex hull and hence not an extreme point [5].

The result will be a list of points the enclose the the whole dimensional set of points.

Algorithm 1 2 Dimensional Quickhull - list of planer points d

```
Find extrema points (maximum and minimum points along the x dimension) .
Add extrema points to the convex hull.
Discard extrema points from the set d
for points in d do
    if point above the joint extrema points then
        Add point to the set above the line d(A)
    else
        Add point to the set below the line d(B)
    end if
end for
for Each set d(N) do
    for Every point in d(N) do
        Pick the point with max distance from the polygon hull set.
        Add this point to the convex hull set.
        Form a new polygon simplex with the new convex hull set.
    end for
    for each point in d(N) do
        if point is inside the polygon then
            Discard point from the set.
        end if
    end for
end for
```

3 Complexity

Worst case complexity: for 2-dimensional space is considered to be $O(n \log(r))$, where n is the number of input points and r is the number of processed points.
In the case where every point is in the convex hull, then the worst case complexity is $0(n^2)$
Best case time complexity: for 2-dimensional space is considered to be $O(n \log(n))$
Space complexity: $O(n)$

4 Limitations

The worst case $0(n^2)$ can be constructed it by placing points on the border of a circle using the following rule (in polar coordinates): Pi = (r, π / 2i). With this set of points the furthest element will always be one of the points on the circle. In each step of the recursion the algorithm can only eliminate one point. The height of the recursion is therefore $0(n)$ resulting in an overall efficiency of $0(n^2)$. This is a major limitation, however improvements to the Quickhull algorithm is presented by Hoang *et al* [6], using orienting vectors for preprocessing the search space.

Given two points p,q, a set S of points lying to the left of the line pq, let r be one of the farthest points from the line pq among the points of S. Let tr$\rightarrow$ be an orienting vector of (p,q,r). The following assertions hold:

(i) If s $\in$ S and Orient(p,r,s)>0, then Orient(tr$\rightarrow$,s)>0.

(ii) If s $\in$ S and Orient(r,q,s)>0, then Orient(tr$\rightarrow$,s)<0

In Figure 2, the initial search space is reduced by removing the unnecessary points from the set, represented by the red coloured points. Therefore it is more computationally efficient.

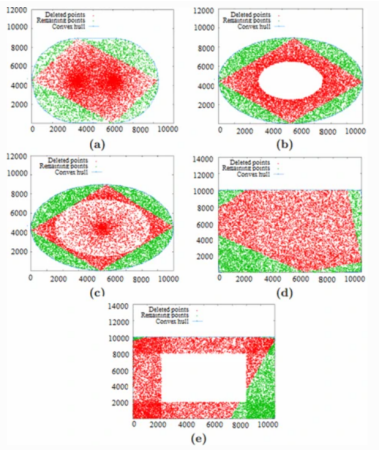

Figure 2: Preprocessing separation, where the red section is initially removed from the search space [6]

5 Applications

Applications of the Quickhull have been used used in collision avoidance for path planning. Dynamic vehicle detection is of great significance for the safety of autonomous vehicles and the formulation of subsequent driving strategies shown by Liu *et al* [7]. If the convex hull of a car avoids collision with other convex hull obstacles then will the car [7]. It is more efficient to approximate the area of an object using a convex hull than mapping the detailed specifications of the obstacle [7]. It is very useful for accurate computer vision technologies in approximating 3D models.

Dynamic objects are detected from preprocessed results, which constitute the regions of interest (ROI) [7]. The ROI extraction process includes virtual-scan mapping, virtual-scan difference, and dynamic-object detection [7].

The remaining point-clouds above the ground are projected to the virtual-scan map [7]. The map is a polar grid representation of the point-clouds [7]. As shown in Figure 3, the polar grid map is divided into grids with the same center angles and radial lengths [7]. The state of each grid is determined by the number and position of point-clouds [7]. In each sector divided by the same center angle, the grid closest to the sensor in the place where the object is located is defined as the occupied state, which are shown with the red color [7]. The states of the grids between the occupied grid and the farthest grid where the object is located are defined as the occluded states, which are shown in yellow [7]. This is an estimated convex hull of the occupied area. This technique is used in most object point estimation in 3D path finding with numerous obstacles.

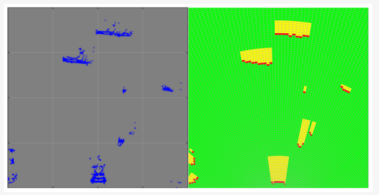

Figure 3: Preprocessing separation, where the red section is initially removed from the search space [7]

Quickhull has also been used for support structures in layered manufacturing, classification of molecules by their biological activity, vibration control and geographic information systems [5].

6 Video

Video demonstration: `https://youtu.be/B7is-4iQm2w`

References

[1] Wijeweera, K.R. and Kodituwakku, S.R., 2021. An efficient planar incremental convex hull algorithm to find the edges of the boundary polygon of the convex hull of a set of points. Ceylon Journal of Science, 50(3), pp.261-268.

[2] Zhang, J., Mei, G., Xu, N. and Zhao, K., 2015. A novel implementation of quickhull algorithm on the gpu. arXiv preprint arXiv:1501.04706.

[3] Tzeng, S. and Owens, J.D., 2012. Finding convex hulls using Quickhull on the GPU. arXiv preprint arXiv:1201.2936.

[4] Braga, R., Valdetaro, A., Nunes, G., Raposo, A. and Feijó, B., Evaluating the visibility algorithm of point-based graphics for real-time applications.

[5] Barber, C.B., Dobkin, D.P. and Huhdanpaa, H., 1996. The quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software (TOMS), 22(4), pp.469-483.

[6] Hoang, N.D. and Linh, N.K., 2015. Quicker than quickhull. Vietnam Journal of Mathematics, 43(1), pp.57-70.

[7] Liu K, Wang J. Fast Dynamic Vehicle Detection in Road Scenarios Based on Pose Estimation with Convex-Hull Model. Sensors. 2019; 19(14):3136. https://doi.org/10.3390/s19143136