Convex Hull Assignment 1

Manoj Kumar (manoj18.iitd@gmail.com)

December 5, 2020

Problem Statement

The main purpose of this assignment is to implement an algorithm which computes the **convex hull** for a set of n points. in three-dimensional Euclidean space, i.e. (x, y, z).

Structure of Input file:

- first line contains total no. of points n in 3D space.
- next n lines contain coordinates of each point (in float) in this format:

```
x-coordinate<space>y-coordinate<space>z-coordinate
```

Sample input file:

```
5
0.0 0.0 1.0
1.0 0.0 0.0
0 0 0
0.0 2.0 0.0
-1.0 -1.5 -3.0
```

Structure of Output file:

The output file contains a set of points that specify the convex hyull for the input points.

- The first line specifies the number of k points in the convex hull.
- the points are specificed in the subsquent k lines in the same format as the input file. Clearly,

$$k \leq n < 10,000$$

Sample output file:

```
4
0.0 0.0 1.0
1.0 0.0 0.0
0.0 2.0 0.0
-1.0 -1.5 -3.0
```

Implementation Approach

I am using **quickhull** algorithm to implement the convex hull in 3D space. QuickHull is a method of computing the convex hull of a finite set of points in n-dimensional space[2].
It uses a divide and conquer approach similar to quicksort. The Quickhull algorithm approaches the correct solution for 3-dimension as it is based on the following key-idea:
when a convex hull H of a set of points S is known, then the convex Hull $H1$ of the set of points $S \cup P$, where P is a new point, is computed as follows:[1]

(i) Let $F_1, F_2, F_3, F_4, ..., F_n$ be the faces of convex hull H, for which point P is a front point(these faces are visible by the eye sight from point P).

(ii) Remove above mentioned faces and let $E_1, E_2, E_3, ..., E_m$, be the boundary edges. (An edge is a boundary edge if exactly one face of this edge is removed.)

(iii) Now Create new faces by joining these edges and the point P.

Algorithm:

Main steps of my implementations are as follows:

(1) Create a tetrahedron with largest volume. It will work as a initiator for the above approach.

- (i) Find all the extreme points of each axis(points with maximum or minimum values of x,y or z coordinates).
- (ii) Find the first edge by joining two points of maximum distance among the above extreme points (this is the edge with maximum length).
- (iii) Create a *face* by joining this edge with the farthest point from this edge.
- (iv) Now find a point with the maximum distance from the above created *face*.
- (v) Create all 4 *faces* of a *tetrahedron* by these 4 points and set the direction of their *normals* outwards.

(2) Now divide all the points to these 4 faces of the tetrahedron in such a way that a point P is a front point of a face f if and only if face f is visible in the eye sight of the point P.

- (i) We do this by iterating all the points and checking for each point if distance between the face f and the point is positive.

(3) initialize an array *convexFaces*, which contains the faces of the convex hull for a finite set of points. (initialize it with 4 faces of the tetrahedron.)

(4) Do the following steps until no point remains in the list of front points of any face.

- (i) Pick a face f with non zero size of its front points.

(ii) Find the farthest point P from f among its front points.

(iii) Apply DFS to find all the faces which are visible in the eye sight of the point P, store all such faces in set *tobeRemovedFaces*.

(iv) Also store all the boundary edges in the set *boundaryedges* while applying DFS.

(v) Store the front points of all the faces of *tobeRemovedFaces* in array *allfrontPoints*.

(vi) Now create new faces by joining the edges of *boundaryedges* with the point P and add these faces in *convexFaces*.

(vii) Remove all the faces of *tobeRemovedFaces* from *convexFaces*.

(5) Now the remaining faces of *convexFaces* are the part of the convex hull.

(6) Extract all the different points from these faces.

Time Complexity[1]

For 3-dimensional spaces, An execution of Quickhull is *balanced* if:

- the average number of new faces for the jth processed point is $3f_j/j$. where f_j is the maximum number of faces of j vertices $f_j = O(j^{\lceil 2 \rceil}/\lfloor 2 \rceil !)$.[3]
- the average number of partitioned points for the jth processed point is $3(n-j)/j$.

Now let n be the number of input points and r be the number of processed points. If the *balance condition* holds, the worst-case complexity of Quickhull is $O(n \log r)$.[1]

In normal case, average time complexity of the implementation is $O(n \log n)$ and in worst case, $O(n^2)$.

Commands to Execute Program

The program is implemented in C++.
To make an executable file, run 'make' or use this command :

```
g++ main.cpp functions.cpp -o exec
```

To compile the executable:

```
./exec CONVEX.IN CONVEX.OUT
```

or simply, use :

```
make
make run
```

References

[1] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. "The Quickhull Algorithm for Convex Hulls". In: *ACM Trans. Math. Softw.* 22.4 (Dec. 1996), pp. 469–483. ISSN: 0098-3500. DOI: 10.1145/235815.235821. URL: https://doi.org/10.1145/235815.235821.

[2] Wikipedia the free encyclopedia. *Quickhull*. URL: https://en.wikipedia.org/wiki/Quickhull.

[3] V. Klee. "Convex polytopes and linear programming." In: *Proceedings of the IBM Scientific Computing Symposium: Combinatorial Problems.* (1966), pp. 123–158.