Report for Computer GraphicII, HW1
3D convex hull algorithm and collision detection

Qi Jiang 2018533134

March 10, 2022

Acknowledgements: Deadline: 2022-03-10 23:59:59

You can choose C++ or Python, and no restrictions on programming framework. You can freely use frameworks such as openGL.

The **report** submits as a PDF file to gradscope, the programming part should package all the files include code, input files, executable file, readme.txt, and report. The **package** name is **your_student_name+student_id.zip**.

You will get Zero if the code not passing the plagiarism check.

1 Part 1 (20 points)

1. (5 points) Prove the intersection of two convex set is still a convex set. (Counter-proof) Assume the two convex set are A and B, take any two points $p, q \in A \cap B$. If $A \cap B$ is not a convex set, then there exist a point k on line pq that $k \notin A \cap B$. However, k must be in A and in B by the definition of the convex set A and B. Therefore, $A \cap B$ must be convex set.

2. (15 points) If a plane is divided into polygons by line segments, please design a data structure to store the division information so that for the given line passing two points p_1 and p_2 on the plane, it is efficient to find all the polygons intersected with the line. Please provide the main idea and pseudocode of the algorithm and give the complexity analysis.
   Main Idea: In a planar graph, each edge is shared by at most two polygons. Therefore, graph can be stored as a dictionary of edge-polygons pair. Then the main computation cost fell on finding all edges intersected with the line $p1p2$, and fetch the adjacent polygons will be in $O(1)$.
   Data Structure:
   Points. Each point is stored with its unique point number and Cartesian's coordinate in a list. For example, the first point with coordinate $(1.2, -0.2)$ sampled from the point set is stored as $[0, 1.2, -0.2]$.

   Polygon. A facet is represented by its vertex, which are stored in a linked-list in consistent clockwise order.

   Edges. Base on the fact that each edge is shared by exactly two visible facets, the edges is stored in a *python* dictionary, in which each edge is a key-value pair of which $key = (min(point_1[0], point_2[0]), max(point_1[0], point_2[0]))$ and $value = [Polygon_1, Polygon_2]$. So that each edge is stored uniquely with no duality due to the order of its two vertex.

   Planar Graph. Based on the data structure mentioned above, a planar graph could be stored with a set of points, polygons and edges.

   Algorithm: Given any line with two point p_1p_2, check for each edge if it crosses with the line, which could done by compute the dot product of the norm vector of the line and the vector from any point on the line to the vertex of the edge. The sign of the result should be the same if the edge crosses with the line. This operation will only $O(E)$. For each crossing edge, we fetch the value from the dictionary, which only cost O(1). The algorithm will cost $O(E) = O(n)$ in total.

2 Part 2 (80 points)

2.1 3D convex hull algorithm(55 points)

The **3d incremental algorithm**, which built a simple convex hull with four points and update the hull with all other points in an iterative approach, is implemented in this homework. The basic steps of this algorithm is stated as follow:

Algorithm 1: Incremental Convex Hull Algorithm

```
Result: CONVEX HULL hull
hull = InitConvexHull(P);
while P is not None do
    p = P.pop();
    if p is not in hull then
        hull.facets.visibility = UpdateFacetsVisibility(hull,p);
        key_edges = FindKeyEdges(hull);
        hull = UpdateConvexHull(hull,key_edges) end
    end
```

2.1.1 Input

The input point set is randomly generated by *numpy* API. The number of the input points could be assigned by *point_num* flag.

2.1.2 Initialization

The simplest convex hull contains at least four non-coplanar points in a 3-dimensional space. The first four points in the point list is used to built the initial convex hull in this homework. A coplanarity check is enforced at first to ensure no degradation occurs. Any degradation case where four or more points are coplanar is not implemented in this homework and will lead to an assertion error that quits the program.

A proper initialization could give a sharp acceleration to the algorithm, since the more points is contained inside the initial convex hull, the less following iteration requires performing an updating operation. An intuitive initialization approach is to select the first few points by **farthest point sampling** to build the initial convex hull. In this way, the initial convex hull will be constructed by points that are far away from each other so that more remaining points will locate inside the initial convex hull.

2.1.3 Geometric Tricks

There are a few notable geometric tricks in the incremental process.

Norm Vector. The norm vectors of the facets in the convex hull is crucial to the subsequent algorithm. For each facet of the convex hull, the norm vector could be pointing towards two different direction, namely, inner side and outer

side of the convex hull. The direction of all norm vectors must stay consistent during the iterative process to ensure the success of the algorithm. To achieve this, a center point is calculated and updated by taking the mean of the points coordinates. Then the direction of all the norm vector is assigned so that the dot product of the norm vector and the vector from any point on the facet to the center point stay positive. In this way, all norm vectors of the convex hull are pointing towards the inner side of the convex hull.

Point inside Hull. For each iteration of the incremental process, the sampled point is first checked that whether it is inside the current convex hull or not. This is implemented as to check for each facet whether the dot product of the norm vector and the vector from one point on the facet to the sampled point is positive. If all results are positive, then the sampled point is inside the current hull., otherwise not. The update operation is only performed on the points out of the current hull.

Tangent Facet. The key operation of the updating process is to find all the tangent facets and remove all those invisible facets. The key insight here is that *the dot product of the norm vector and the vector from any point on a* ***visible facet*** *to the sampled point is positive and the dot product of the norm vector and the vector from any point on an* ***invisible facet*** *to the sampled point is negative* and that *those edges shared by a visible facet and an invisible facet are the edges shared by the tangent facets and the current convex hull.* The edges shared by two invisible facets will be removed.

2.1.4 Data Structure

The data structure used in this homework to store the convex hull is stated as follow.

Points. Each point is stored with its unique point number and Cartesian's coordinate in a list. For example, the first point with coordinate $(1.2, 2.3, -0.5)$ sampled from the point set is stored as $[0, 1.2, 2.3, -0.5]$.

Facets. A facet is represented by its three vertex. Its visibility and norm vector is also stored.

Edges. Base on the fact that each edge is shared by exactly two visible facets, the edges is stored in a *python* dictionary, in which each edge is a key-value pair of which $key = (min(point1[0], point2[0])$ max(point1[0],point2[0])) and $value = [Facet1, Facet2]$. So that each edge is stored uniquely with no duality due to the order of its two vertex.

Convex Hull. With all data structure mentioned above, a convex hull is stored with all points, edges and facets of it in the above manner.

2.1.5 Time Complexity

During each iteration, an O(1) operation is done between each facet and the sampled point and all edges are traversed to find the tangent facets. n-4 iterations are executed in total. Therefore, the algorithm should be in $O(n^2)$. Practically, the run time of the program is stated as follow.

Points	10	100	1000	10000
Time(s)	0.0013	0.014	0.13	1.61

Table 1: Runtime of the program.

2.1.6 Visualization

A convex hull computed from 100 random points is first shown with its collision detection. Then the convex hull computed from 10, 1000, 10000 points are shown.

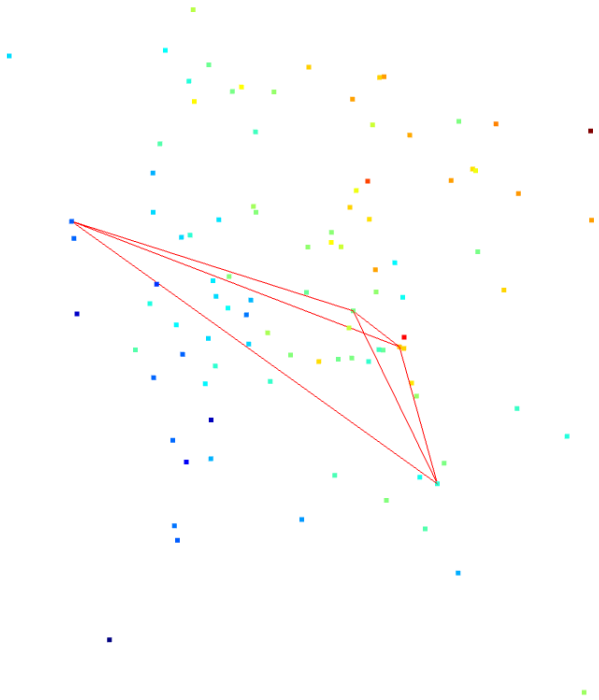

Figure 1: Initial convex hull with four points sampled from the point set of size 100.

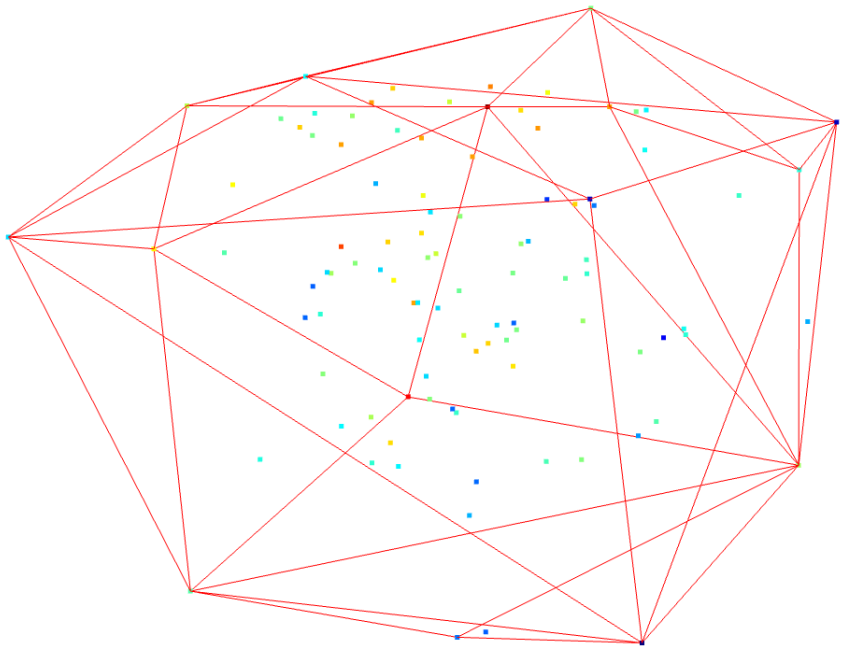

Figure 2: The convex hull computed from the above point set.

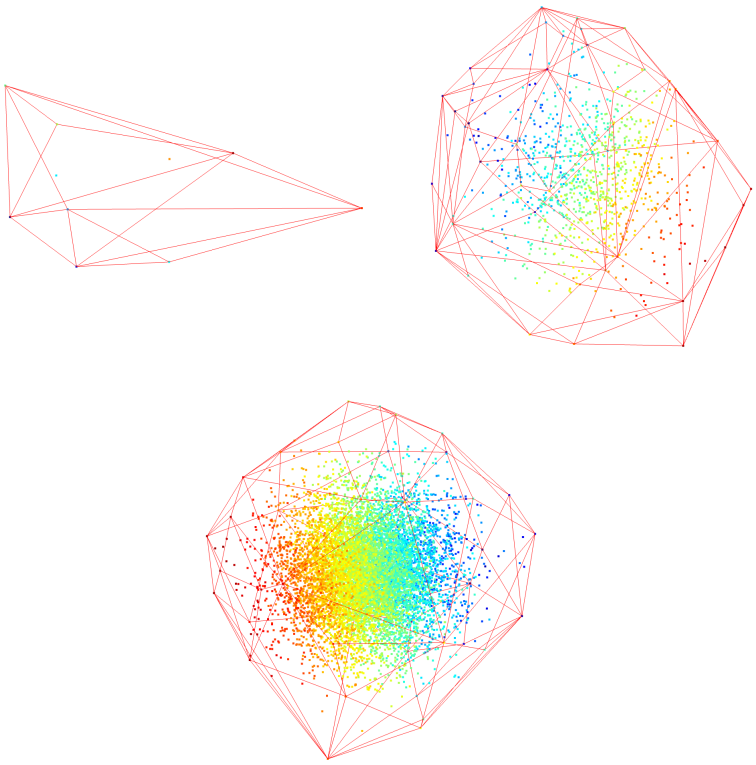

Figure 3: The convex hull computed from point sets of size 10,1000,10000.

2.2 Collision detection(25 points)

Collision detection is implemented with a simple bi-directional point-in-hull test. If no vertex on one convex hull is not contained by the other convex hull and vice versa, then the two convex hull are not collided. Otherwise collide. The time complexity is $O(n^2)$.

The collision detection experiments are conducted on two identical convex hull computed as stated in 2.1. The second convex hull is generated by an offset from the origin one. The experiment is conducted with the offset of 2 and 5.

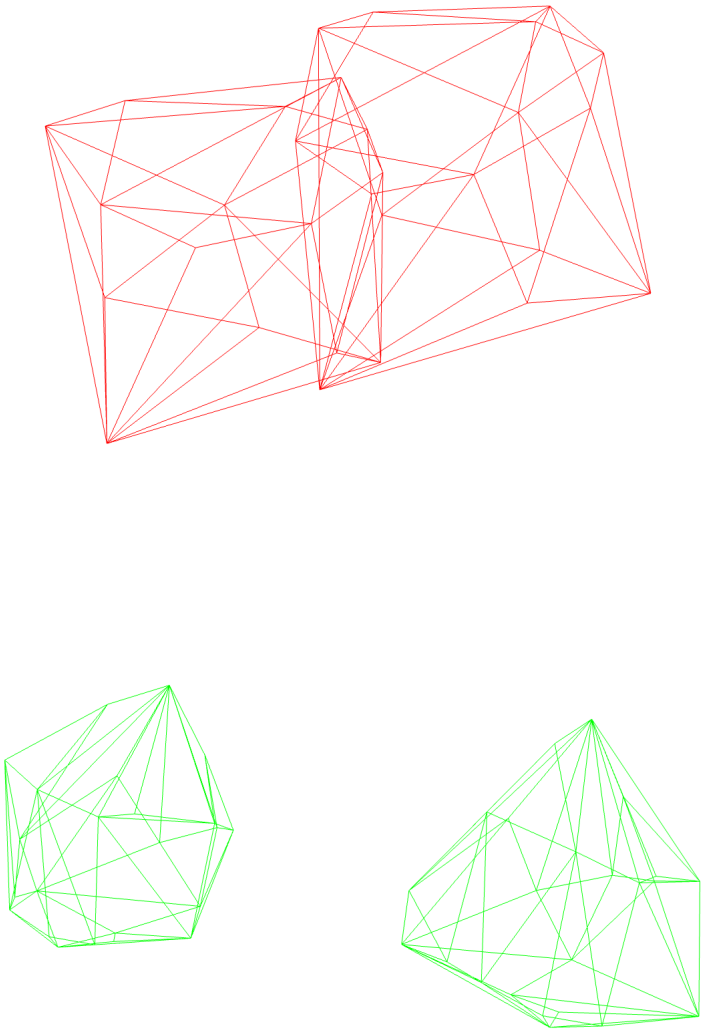

Figure 4: The collision detection experiments with offset 2(red, collide) and 5(green, not collide)