CS271 Computer Graphics II: homework1

Yang Hui
2020233290
yanghui1@shanghaitech.edu.cn

March 6, 2022

1 HW1 part1

1.1 problem1

Assuming the intersection C of convex set A and convex set B has m points. c_1,c_2, ... ,c_m in E^d.
Because A is a convex set and $c_i, c_j \in A$, $(1-t)c_i + tc_j \in A$, $(t \in R)$
Because B is a convex set and $c_i, c_j \in A$, $(1-t)c_i + tc_j \in B$, $(t \in R)$
hence $(1-t)c_i + tc_j \in C$, C is a convex set.

1.2 problem2

Data structure: use a dictionary to store the division information. Line segment l as key and the plane associated with the line segment as value. Taking Figure 1 as an example, the corresponding data structure is as follows, $\{l_1$:$[f_1, f_2]$, l_2:$[f_2, f_3]$, l_3:$[f_2, f_4]$, l_4:$[f_2, f_5]\}$. Details of line segments and planes such as included vertices can be stored in other arrays, l_i, f_i is the index of the array.

Algorithm: For a given line segment p_1p_2, if it intersects l_i,then the polygons with l as an edge intersects p_1p_2. So the algorithm is equivalent to finding all line segments in the plane that intersect p_1p_2. Connect p and the two endpoints of l_i to form two vectors, then make p_1p_2 cross-multiply these two endpoints respectively. If the signs of the two results are opposite, then the two endpoints of l_i are two sides of p_1p_2. If we verify that the two endpoints of l are two sides of p_1p_2,and the two endpoints of p_1p_2 are two sides of l at the same time, it means that l and p_1p_2 intersect.

Time Complexity: Suppose the plane is divided by n line segments, and the algorithm has only one loop, hence the complexity is O(n).

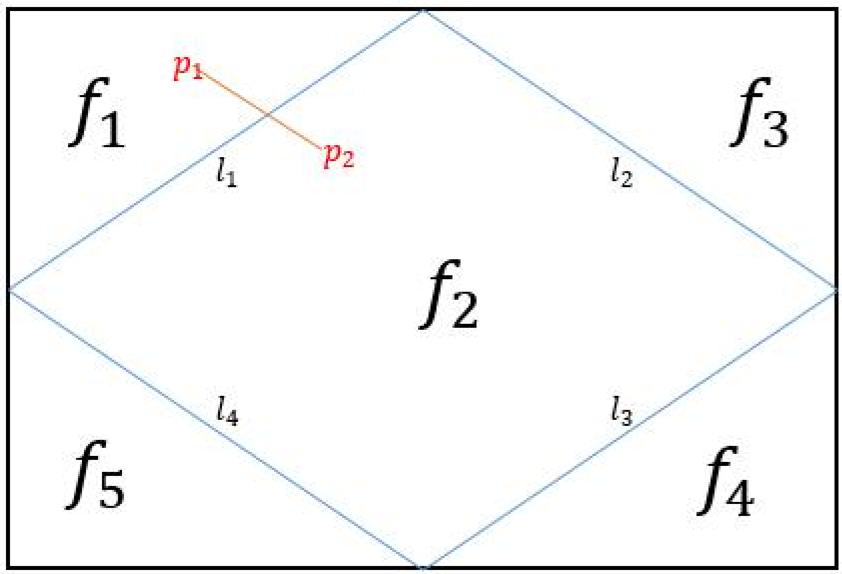

Figure 1: problem 2

Algorithm 1 polygons

Require: lineMap $< l_i, [f_1, \ldots, f_j] >$, p_1, p_2

```
function GETPOLYGONS(lineMap, p1, p2)
    resultPolygons ← ∅
    for < line, polygons > ∈ lineMap do
        v1, v2 ← line
        if dot(p1p2 × p1v1, p1p2 × p1v2) <= 0 and dot(l1l2 × l1p1, l1l2 × l1p2) <= 0 then
            resultPolygons.append(polygons)
        end if
    end for
    return resultPolygons
end function
```

2 HW1 part2

2.1 3D convex hull

I used the Incremental algorithm to build the convex hull by C++.
Data structure: there are 3 mainly data structures in my algorithm.

1. Vertex: which is used to store the initial points set. It has three properties(x,y,z) to save the 3d coordinates of a point.

2. Plane: which is used to store the convex hull. All valid planes form a convex hull. It has three properties. The one is used to store all vertex indices on the plane. The another is used to store the number of vertex on the plane. The last one is used to indicate whether the plane is valid or not.

3. Line: which is mainly used when dealing with three points collinear. It uses the lines's two vertex indices as its properties.

Algorithm: The algorithm implementation details are as follows.

Algorithm 2 Incremental algorithm

Require: Points[N]
Ensure: $N >= 4$

```
function ConvexHull(Points, N)
    planes[0..3] ← InitializeConvexHull(Points)
    for v ∈ Points[4..N] do
        edges[0..N][0..N] ← 0
        for p ∈ planes do
            n ← NormalVector(p)
            vector ← v − p.points[0]
            if dot(n, vector) > 0 then
                edges[p] + +
            end if
            if dot(n, vector) == 0 then
                p ← Coplanar(p, v)
            end if
        end for
        for p ∈ planes do
            for edge ∈ p do
                v1, v2 ← edge
                if edges[v1][v2] == 1 and edges[v2][v1]! = 1 then
                    planes ← AddPlane(v1, v2, v)
                end if
                if edges[v1][v2]! = 0 then
                    p.flag = false
                end if
            end for
        end for
    end for
    return planes
end function
```

Time complexity: traverse the points(3th line) takes $O(N)$,and initialize edge(4th line) takes $O(N^2)$. After initializing the convex hull, a point will add at most two new planes (that is, add 3 planes, delete the original one on the convex hull). Hence traverse the planes takes $4(N-4)+2+2\times2+2\times3+\cdots+2\times(N-5) = O(N^2)$. So the algorithm takes $O(N^3)$

Result:

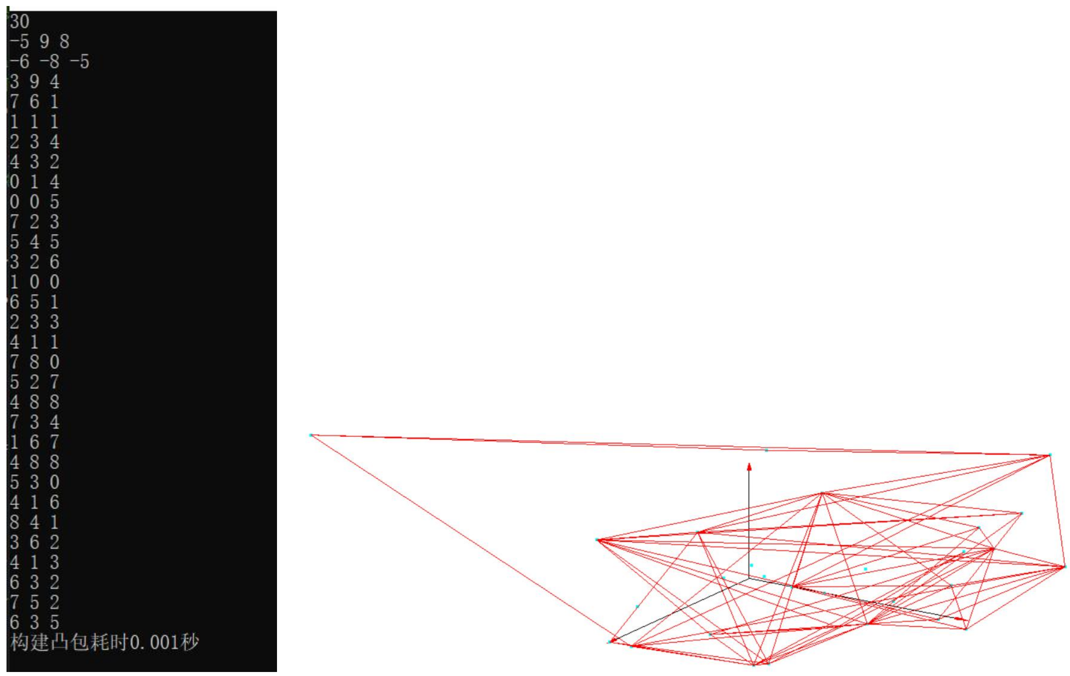

Figure 2: convex hull: time complexity result1

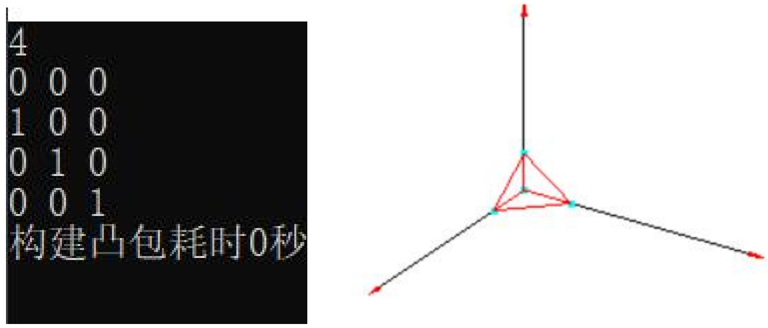

Figure 3: convex hull: time complexity result2

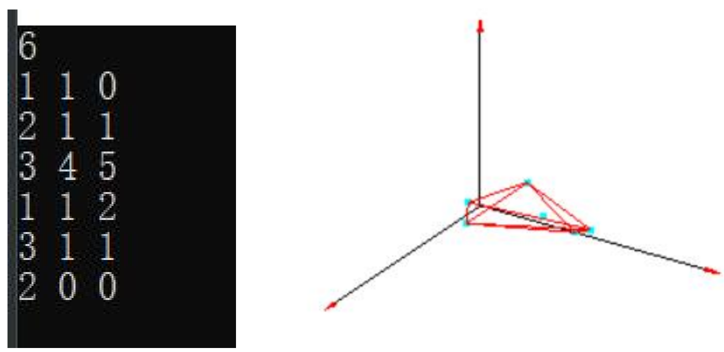

Figure 4: convex hull: result1

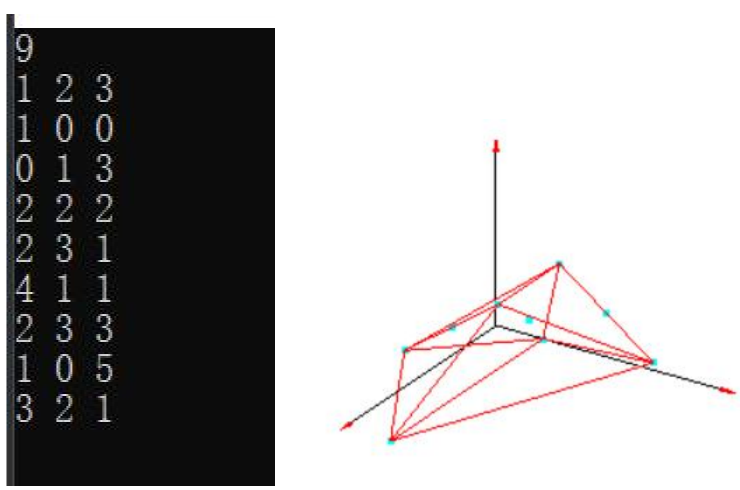

Figure 5: convex hull: result2

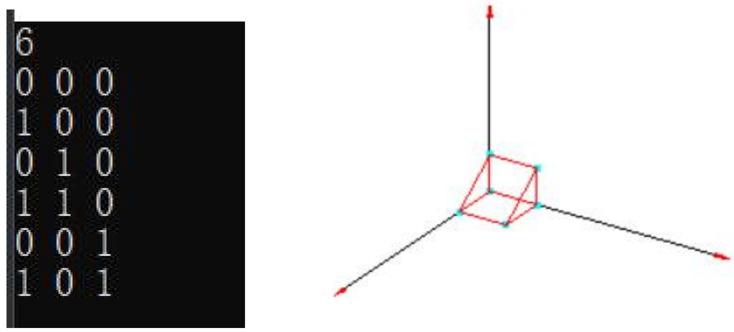

Figure 6: convex hull: result3

2.2 Collision detection

Algorithm: First obtain the convex hull of one point set, and then traverse the points in the other point set to determine whether the point is in the convex hull or on the convex hull surface. (flag==true indicates that the point collides with the convex hull)

Time Complexity: the time complexity of ConvexHull() is $O(N_1^3)$, traverse the points2 takes $O(N_2)$, traverse the planes of points1 takes $O(M_1)$(Assuming M1 is the total number of planes in the first convex hull.) Hence the time complexity of algorithm is $O(N_1^3) + O(N_2 M_1)$

Algorithm 3 Collision Detection

Require: Points1[N1],Points2[2N]

Ensure: $N1 >= 4, N2 >= 4$

```
function isCollision(Points1, N1, Points2, N2)
    planes1 ← ConvexHull(Points1, N1)
    flag ← true
    for v ∈ Points2[0..N2] do
        flag ← true
        for p ∈ planes1 do
            n ← NormalVector(p)
            vector ← v − p.points[0]
            if dot(n, vector) > 0 then
                flag ← false
                break
            end if
            if dot(n, vector) == 0 then
                for edge_(i,i+1), edge_(i+1,i+2) ∈ p do
                    angle1 = getAngle(edge_(i,i+1), edge_(i+1,i+2))
                    angle2 = getAngle(edge_(i,i+1), edge_(i+1,v))
                    if angle2 < angle1 then
                        flag ← false
                        break
                    end if
                    if angle2 == angle1 then
                        if length(edge_(i+1,i+2)) >= length(edge_(i+1,v)) then
                            flag ← false
                            break
                        end if
                    end if
                end for
            end if
        end for
        if flag == true then
            break
        end if
    end for
    return flag
end function
```

Result:

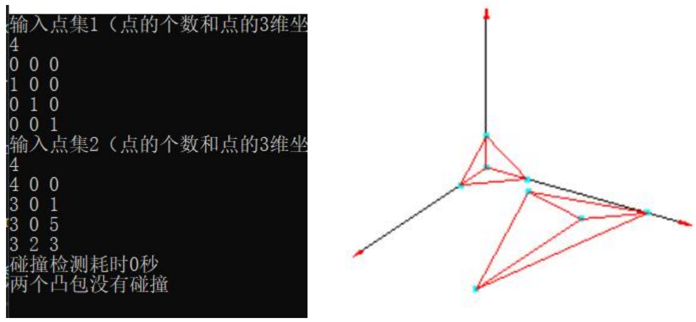

Figure 7: collision detection: result1

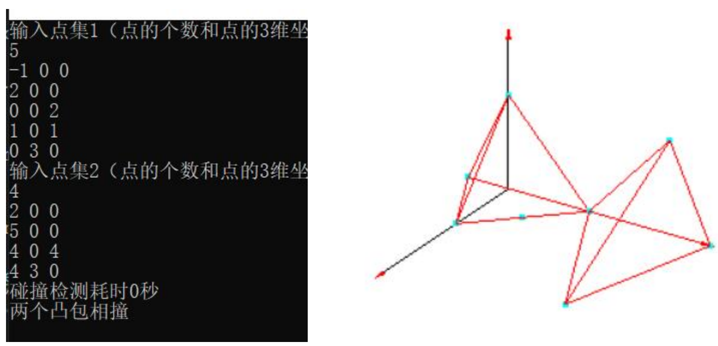

Figure 8: collision detection: result2

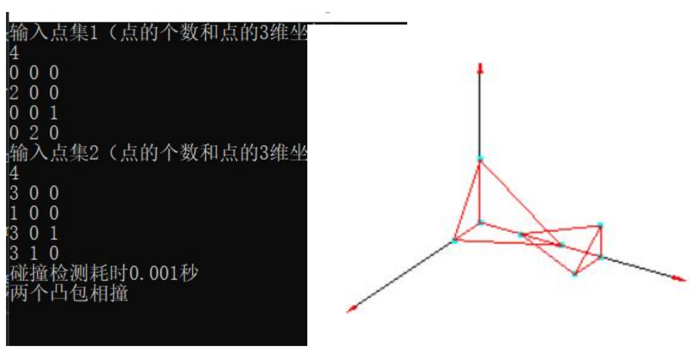

Figure 9: collision detection: result3