**Computation Geometry Report**

**Implementation of the Convex Hull**

**Ziyi Hu**

**100981750**

1.Convex Hull in 2d

1.1 Brute Force

Given a set of points P, test each line segment to see if it makes up an edge of the convex hull.

This is a very simple algorithm.

In Java, create the random points and use Ellipse2D to represent the 2d points

for(int i = 0; i< 100;i++){

int xPoints[];

int yPoints[];

xPoints = new int[1000];

yPoints = new int[1000];

Random ran = new Random();

xPoints[i] = ran.nextInt(500)+100;

yPoints[i] = ran.nextInt(500)+100;

// ranx = (Double) ranx;

Point p1 = new Point();

Ellipse2D p2 = new Ellipse2D.Double(xPoints[i],yPoints[i],5,5);

g2.fill(p2);

pointList.add(p2);

}

Test every segment and if all the other points are on the other side of this segment, add it to the List.

List<MyVector>myVectors = new ArrayList<MyVector>();

for(int j=0;j<i;j++){

for(int k = 0;k<i;k++){

MyVector myvector = null;

boolean valid = true;

for(int m = 0;m<i;m++){

Ellipse2D A =(Ellipse2D) pointList.get(j);

Ellipse2D B =(Ellipse2D) pointList.get(k);

Ellipse2D C =(Ellipse2D) pointList.get(m);

Point startPoint = new Point((int)A.getX(),(int)A.getY());

Point endPoint = new Point((int)B.getX(),(int)B.getY());

Point cPoint = new Point((int)C.getX(),(int)C.getY());

myvector = new MyVector(startPoint, endPoint);

if(myvector.lieOnMyLeft(cPoint)){

valid = false;

}

}

if(valid) myVectors.add(myvector);

}

}

Paint the segment

List<MyVector> myVectors = convexHull.getTheConvexHull1(convexHull.pointPanel1.getPointList());

for (int i = 0; i < myVectors.size(); i++){

k = 1;

Graphics g = pointPanel1.getGraphics();

// contentPanel.remove(pointPanel1);;

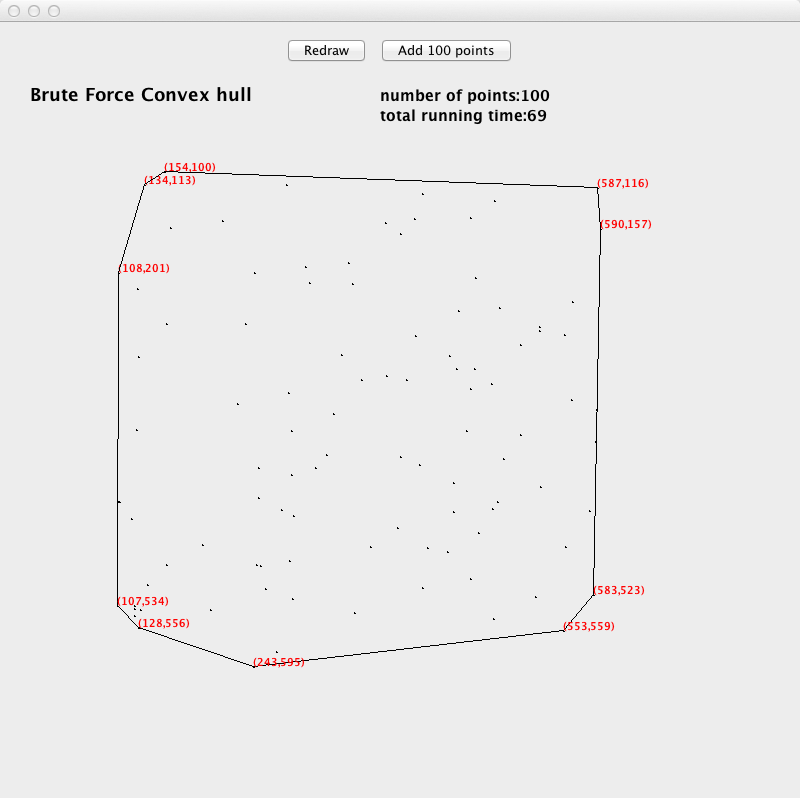
// g.drawString("HI", 10, 10);

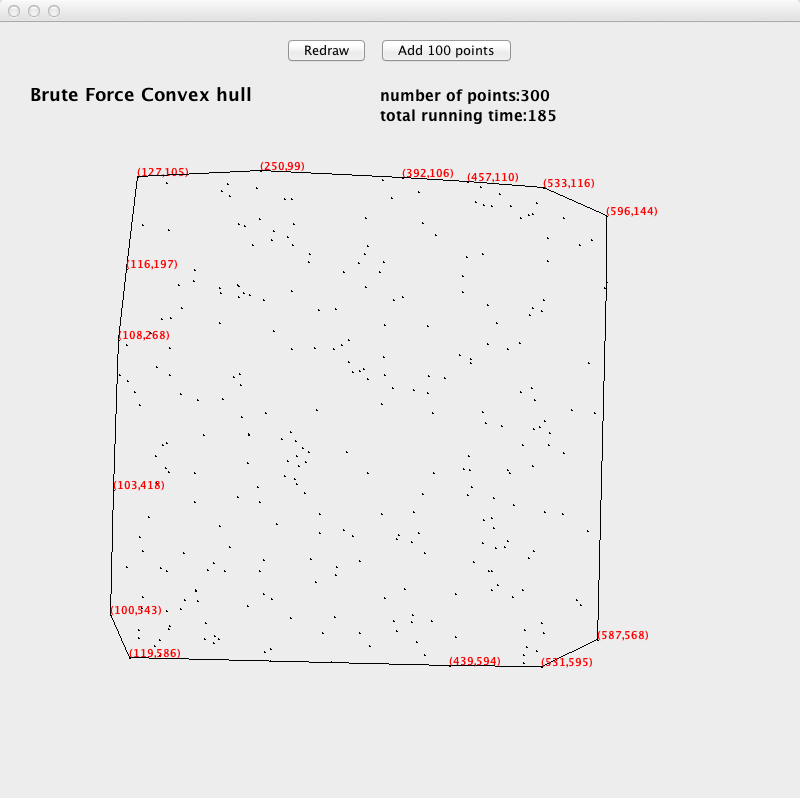
g.setColor(Color.BLACK);

g.drawLine(myVectors.get(i).startPoint.x, myVectors.get(i).startPoint.y, myVectors.get(i).endPoint.x, myVectors.get(i).endPoint.y);

}

The Result





1.2 Quick Hull

Given a set of points P, need to find the farthest two points and every time, add one point that is farthest from the current convex hull segments.

There are several functions in the program.

Public int right(int a, int b, int p); to check whether point p is right of line ab using three points location formula.

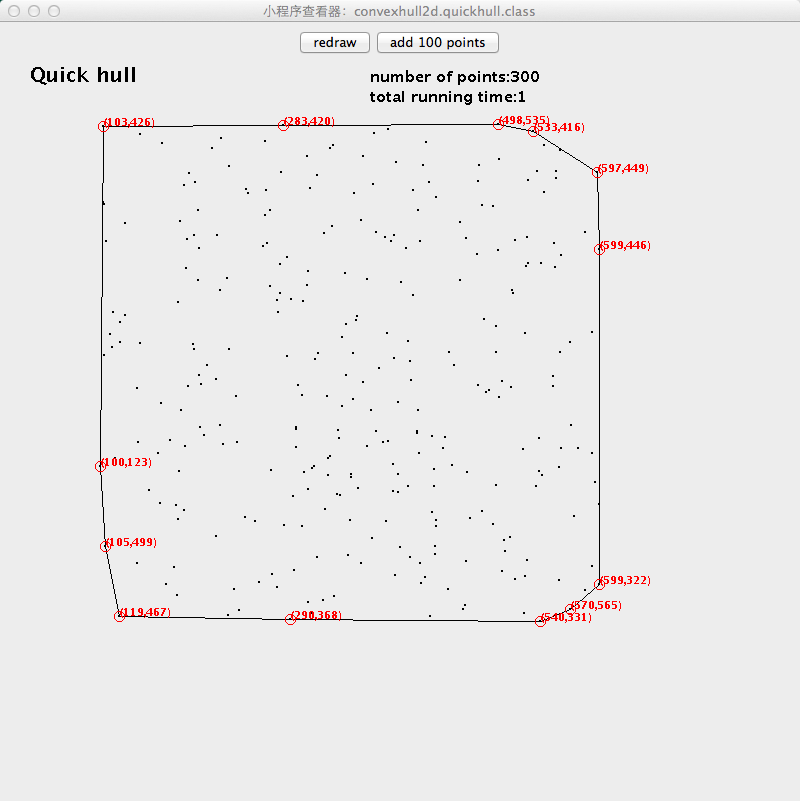
Public int distance(int a, int b, int p); because we need to find the farthest point, so we need to calculate the distance from p to segment ab

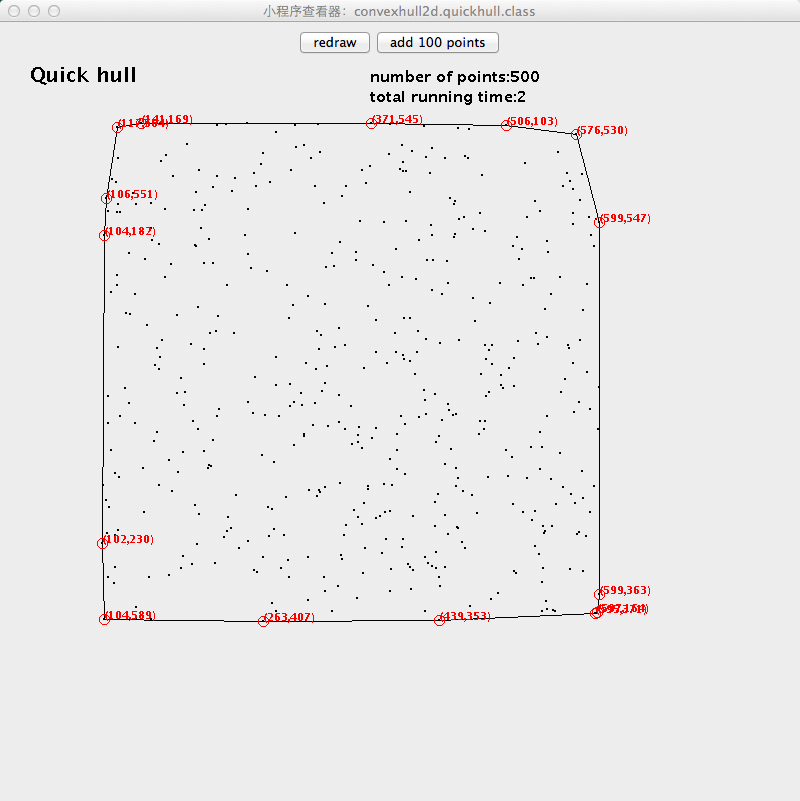
Public int farthestpoint(int a, int b, ArrayList<Integar>al); from all the points assigned to ab segment, find the farthest point, return maxP is the index of that point.

Public void quickhull(int a, int b, ArrayList<Integar>al); import two points a,b and all the other points assigned to the segment al. use the function farthestpoint(int a, int b, ArrayList<Integar>al) to get the farthest from the segment ab(assume that is c) reassign all the points P to ac or bc by using the function Public int right(int a, int b, int p). if right(a,c,p)>0, reassign the point p to ac(using the arraylist al1), if right(c,b,p)>0, reassign the point p to bc(using the arrarylist al2) do the quickhull(a,c,al1) and quickhull(c, b, al2), add point c to the convex hull(c is linked to a and b).

Public void quickconvexhull(int k); get the left most and right most points, calculate the points behind or above the segment ab, get arraylist al1 above the ab and arraylist al2 behind the ab, using quickhull(a,b,al1) and quickhull(b,a,al2) to get the convexhull.

The result:





1.3 Gramham convex hull

Given a set of points P. Find the point with the lowest y-coordinate. Sort points by polar angle with this point get the points[N+1] array. Set points[0] to be a sentinel point that will stop the algorithm(points[0] = points[N]). Three points are a counter-clockwise turn if ccw>0. The original three points are points[1], points[2] and points[3]. Do the ccw, if ccw>0, push points[3]. Then do ccw points[2], points[3] and points[4], if ccw>0, push points[4], else pop points[3], swap points[3] and points[4].Do ccw points[1],points[2] and points[3]. Every time, if ccw points[i], points[i+1], points[j]<0, pop points[i+1], swap points[i+1] and points[j], else push points[j](that is swap points[i] and points[j]).

There are several functions in the program.

Public double angle(int o,int a); This function is to calculate the angle between two points o and a.

Public long distance(int a,int b); Because there may have the points with the same angle to the original point, so need the distance to sort the points. The distance is smaller will be in the front of the list.

Public int ccw(int p1,int p2,int p3); This function is to determine whether the three points p1, p2, p3 is clock-wise. If ccw>0, there are clock-wise.

Public void swap(int[] stack,int a,int b); Swap points a and b, then points[a]=points[b], points[b]=points[a].

Public void grahamScan(int k); First find the lowest point, then calculate the angles and distance from the original point(lowest point). Sort the points by angle and distance. Create a stack to store the convex hull. If stack[0]=points[N] which will end the loop. The three start points are stack[1]=lowest point, stack[2]=sorted list point[0], stack[3]=sorted list point[1]. The code behind can get the convex hull.

for (int i = 3; i <= pNum\*k; i++){

while (ccw(stack[M-1], stack[M], stack[i]) <= 0){

M--;

}

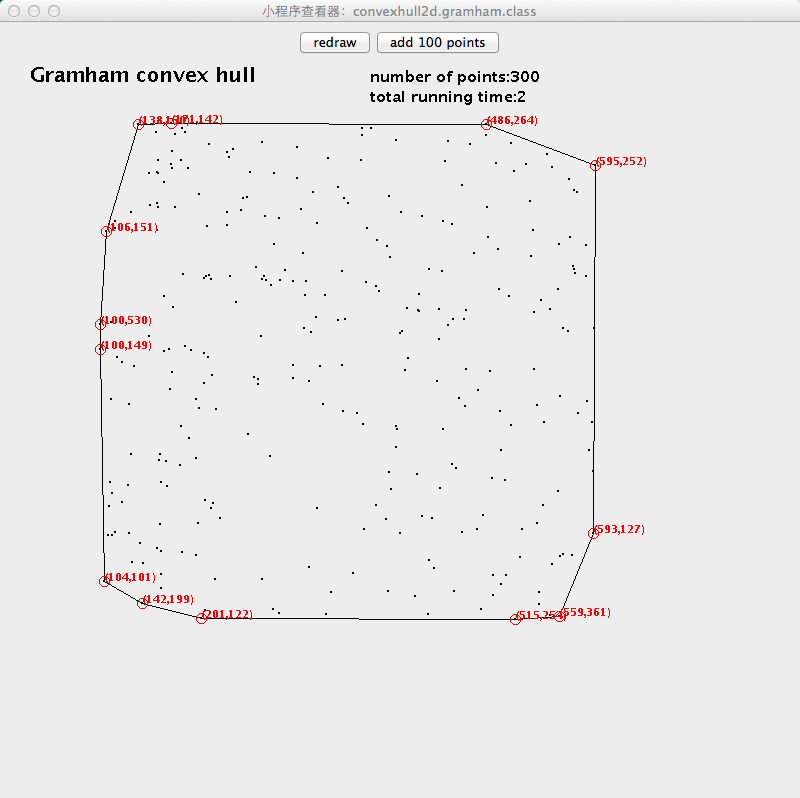
M++;

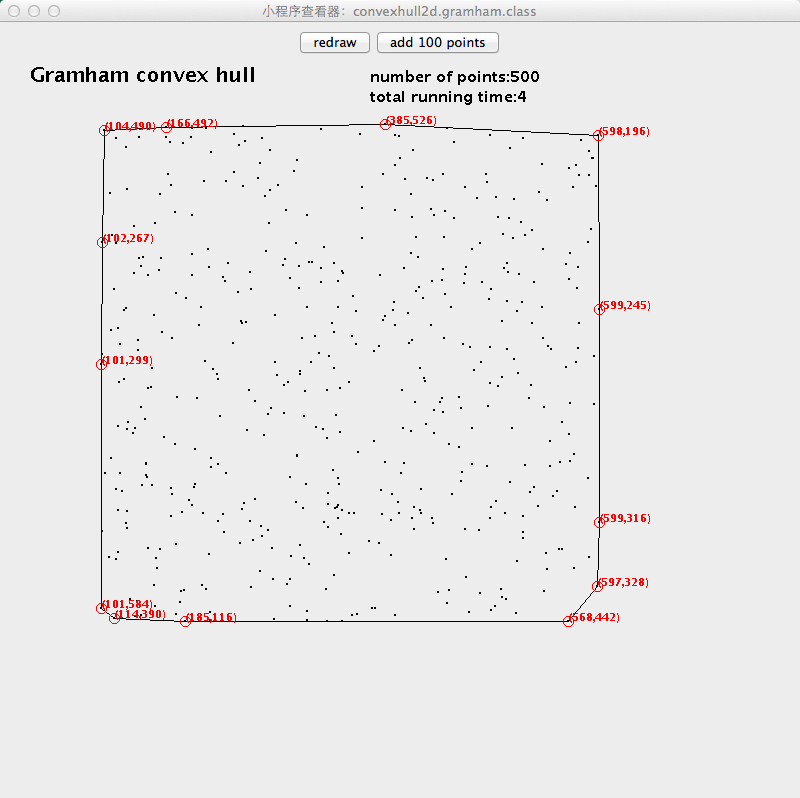
swap(stack, i, M);

}

Every time the ccw<0, it go back and pop the point M, point[i]=point[M],point[M]=point[i]. Because next time i will increase 1 and never go back to i, so point[M] has been pop. Next time it will do point[M-1],point[i],point[i+1]. If the ccw>0, point[i]=point[M+1],point[M]=point[i]. The point i has been pushed in the stack. Next time it will do point[M],point[i],point[i+1].

The result:





2.Convex Hull in 3d

2.1 Brute Force

Given a set of points P test each face to see if it makes up an edge of the convex hull.

Convex hull 3d in java need to download the j3d.jar and use the configure build path to import the jar to the project.

This is a very simple algorithm. However, when enter more than two hundred points, the algorithm runs very slow

public void bruteforce1(int k){

for(int i = 0 ; i < getVertexCount() ; i++){

for(int j = 0 ; j < getVertexCount() ; j++){

for(int m = 0; m < getVertexCount() ; m++){

if(i!=j && i!=m && j!=m){

Vertex v1 = getVertex(i);

Vertex v2 = getVertex(j);

Vertex v3 = getVertex(m);

Facet f1 = new Facet(v1,v2,v3);

boolean face = true;

for(int n = 0;n < getVertexCount() ; n++){

Vertex d = getVertex(n);

if(f1.behind(d) && n!=i && n!=j && n!=m){

face = false;

}

}

if(face){

Edge e = new Edge(v2, v3, f1);

Facet ff1 = new Facet(v1, e.getDest(), e.getSource());

ff1.setData(new ConflictList(true));

addFacet(ff1);

created.add(ff1);

ff1.setFilled(true);

}

}

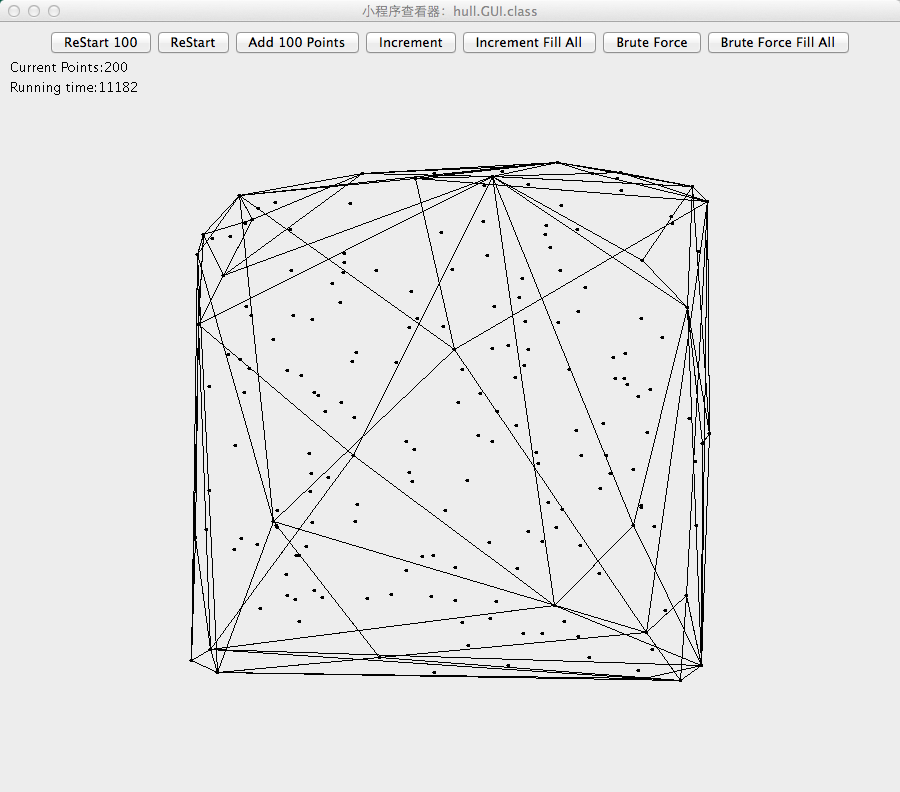
}

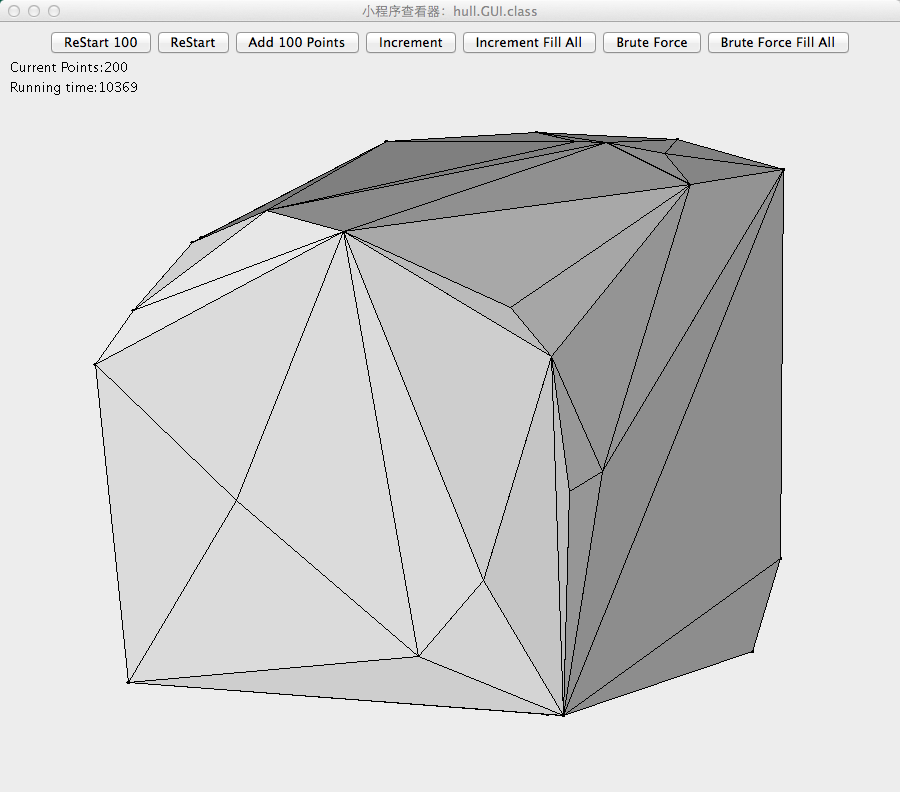
}

}

}

The result:





2.2Increment convex hull

Given a set of points P. First get four random points and these points are initial points. In this project, I used the point[0],point[1],point[2],point[3] and build four faces which are face(0,1,2) face(0,2,3) face(0,1,3) face(1,2,3) and for all the other points, if a point is above one face, add it to the conflict list for the face. Every time get a point from the conflict list. If the point is already inside the convex hull, try the next point and delete this point from the conflict list.

From this point, flag visible faces and find horizon edges. Create new faces to connect the point to the horizon edges. Hide the just-processed vertex and remove all previously visible faces.

There are several functions in the program.

The very important thing in this program is to find the visible faces to a vertex and for a face which vertices that can see it.

public class ConfictList{ } This class is used to link face to vertices and vertex to faces. If you want to draw a point or a face, you need to use the

public ConflictList(boolean facet); This function will disjoint the point or the face(if facet is true, it will clear all the linked vertices and will be used for the later processes)

public void add(GraphArc arc); This function will be used add vertex and face to the lists.

public void getVertices(List list); This function fill a list of vertices from the double-linked faces.

public void getFacets(List list); This function fill a list of faces from the double-linked faces.

We need to assign points to the faces. public void addConfilct(Facet f, Vertex v); This function add an arc to the conflict graph connecting the given facet and vertex. Then it add the face and vertex to the lists by using ConflictList.add(); method.

After we draw the new face, we also need removeConflicts(f); and removeFacet(f); to remove the previously visible faces

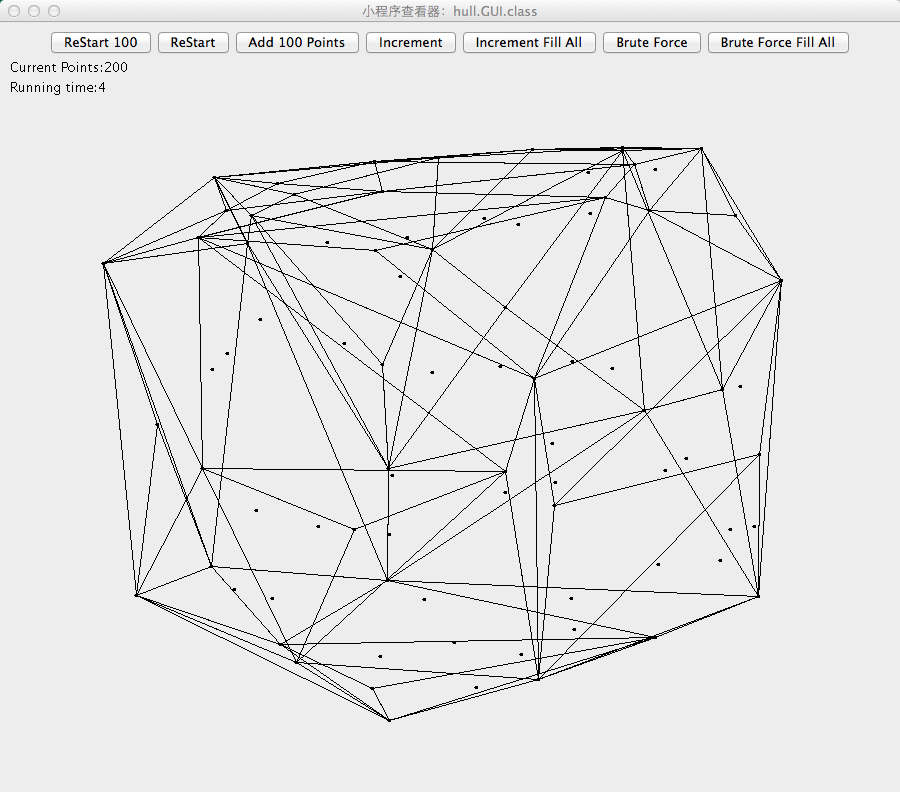
At the beginning, we get a point, test if it is already inside the convex hull, if that is true, we try the next point. Flag visible faces by using setMarked(true); method. Using the facet(i).getHorizonEdge(); to get the index of the edge. Using the edge.findHorizon(); to set the horizontal edges list and the number of horizontal edges.

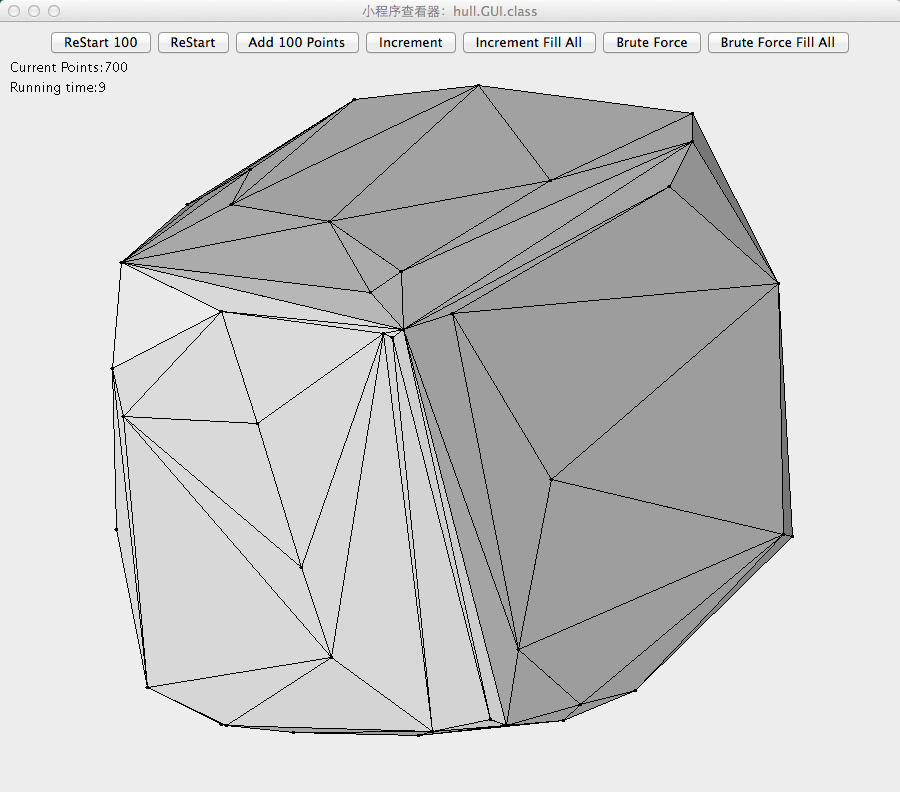
Next step is to create new faces to connect the eye point to the horizontal edges (create new faces and add them to the hull) reassign the points to the new faces by using addConflicts(f,face1,face2);

Remove all the previously visible faces and assigned points of those faces.

Keep doing that until no point is outside the convex hull.

The result:





Conclusion: Both in 2d and 3d, the running time for algorithm brute-force is very slow. In 3d, if the number of point set is more than 300, it takes so much time to get the result and if number gets bigger and bigger it may even not possible to get the result in one year. However, the other algorithms run quite fast and the time will influenced by the computer rather than the algorithms.