Computing the Delaunay Triangulation For Point Location

Jamie Mendek
COMP 5008

April 1, 2015

Theorem

Let P be a set of points in the plane.

1. *Three points $p_i, p_j, p_k \in P$ are vertices of the same face of the Delaunay graph of P if and only if the circle through p_i, p_j, p_k contains no point of P in its interior.*
2. *Two points $p_i, p_j \in P$ form an edge of the Delaunay graph of P if and only if there is a closed disc C that contains p_i and p_j on its boundary and does not contain any other point of P.*

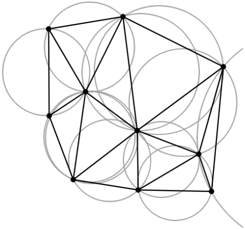

Figure 1: A delaunay triangulation. None of the circles passing through the faces of this graph have a point of P in the interior

Lemma

Let edge $\overline{p_i p_j}$ be incident to triangles $p_i p_k p_j$ and $p_r p_i p_j$ and let C be the circle through $p_r p_i p_j$. The edge $\overline{p_i p_j}$ is illegal if and only if the point p_k lies in the interior of C. Furthermore, if the points p_i, p_j, p_k, p_r form a convex quadrilateral and do not lie on a common circle, then exactly one of $\overline{p_i p_j}$ and $\overline{p_k p_r}$ is an illegal edge.

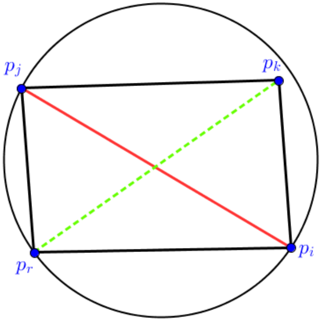

Figure 2: Edge $\overline{p_i p_j}$ is illegal

Setting up an Algorithm

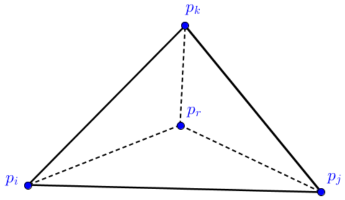

Figure 3: A point p_r inside a triangle creates three new faces in the triangulation. Additionally we must check if edges $\overline{p_k p_i}$, $\overline{p_i p_j}$, and $\overline{p_j p_k}$ are illegal.

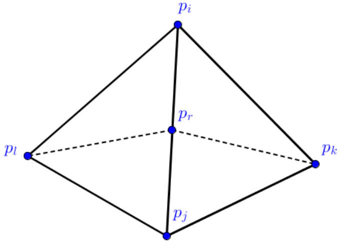

Figure 4: A point p_r on the boundary of two triangles (collinear with an edge) creates 4 new faces. In this case, we must check if $\overline{p_i p_l}$, $\overline{p_l p_j}$, $\overline{p_j p_k}$ and $\overline{p_k p_i}$ are illegal

Delaunay Triangulation Algorithm I

procedure $\textsc{DelaunayTriangulation}(P)$
  Let p_0 be the lexographically highest point of p
  Let p_{-1} and p_{-2} be two points such that P is contained in the triangle $p_0 p_{-1} p_{-2}$
  Initialize T as the triangulation consisting of the single triangle $p_0 p_{-1} p_{-2}$
  Compute a random permutation $p_1, p_2, ..., p_n$ of P
  for $r \Leftarrow 1$ **to** n **do** /* *Insert* p_r *into* T */
    Find a triangle $p_i p_j p_k \in T$ containing p_r
    if p_r lies in the interior triangle of $p_i p_j p_k$ **then**
      Add edges from p_r to the three vertices of $p_i p_j p_k$, splitting the face into three triangles
      $\textsc{LegalizeEdge}(p_r, \overline{p_i p_j}, T)$

Delaunay Triangulation Algorithm II

LegalizeEdge($p_r, \overline{p_j p_k}, T$)
LegalizeEdge($p_r, \overline{p_k p_i}, T$)
else /* p_r *lies on an edge of* $p_i p_j p_k$, *say edge* $\overline{p_i p_j}$ */
Add edges from p_r to p_k and to the third vertex p_l of the other triangle that is incident to $\overline{p_i p_j}$, thereby splitting the two triangles incident to $\overline{p_i p_j}$ into four triangles.
LegalizeEdge($p_r, \overline{p_i p_l}, T$)
LegalizeEdge($p_r, \overline{p_l p_j}, T$)
LegalizeEdge($p_r, \overline{p_j p_k}, T$)
LegalizeEdge($p_r, \overline{p_k p_i}, T$)
Discard p_{-1} and p_{-2} and all their incident edges from T.
return T

procedure $\textsc{LegalizeEdge}(p_r, \overline{p_i p_j}, T)$

*/*The point being inserted is p_r, and $\overline{p_i p_j}$ is the edge of T that may need to be flipped */*

    if $\overline{p_i p_j}$ is illegal **then**

        Let $p_i p_j p_k$ be the triangle adjacent to $p_r p_i p_j$ along $\overline{p_i p_j}$

*/*Flip $\overline{p_i p_j}$ */*

        Replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$

        $\textsc{LegalizeEdge}(p_r, \overline{p_i p_k}, T)$

        $\textsc{LegalizeEdge}(p_r, \overline{p_k p_j}, T)$

Choosing p_{-1}, p_{-2}

Let l_{-1} be a horizontal line lying below the entire set P, and let l_{-2} be a horizontal line lying above P. Conceptually, choose p_{-1} to lie on the line l_{-1} sufficiently far to the right that p_{-1} lies outside every circle defined by three non-collinear points of P, and such that the clockwise ordering of the points of P around p_{-1} is identical to their (lexicographic) ordering. Next, we choose p_{-2} to lie on the line l_{-2} sufficiently far to the left that p_{-2} lies outside every circle defined by three non-collinear points of $P \cup \{p_{-1}\}$, and such that the counterclockwise ordering of the points of $P \cup \{p_{-1}\}$ around p_{-2} is identical to their (lexicographic) ordering.

How to handle p_{-1}, p_{-2} when checking edge legality

Let $\overline{p_i p_j}$ be the edge to be tested

1. $\overline{p_i p_j}$ is an edge of the triangle $p_0 p_{-1} p_{-2}$. These edges are always legal.
2. The indices i, j, k, r are all non-negative. This is the normal case; none of the points involved in the test is treated symbolically. Hence, $\overline{p_i p_j}$ is illegal if and only if p_k lies inside the circle defined by pi , pj , and pr .
3. All other cases. In this case, $p_i p_j$ is legal if and only if $min(k, r) < min(i, j)$.

For my own implementation purposes, I represent my triangulation T as a Doubly Connected Edge List. (in counter clockwise order, each so-called half edge of T stores a pointer to ...

1. The next edge
2. The previous edge
3. The twin edge of the adjacent face
4. The leaf node of the delaunay triangle history tree D to which the edge is adjacent.

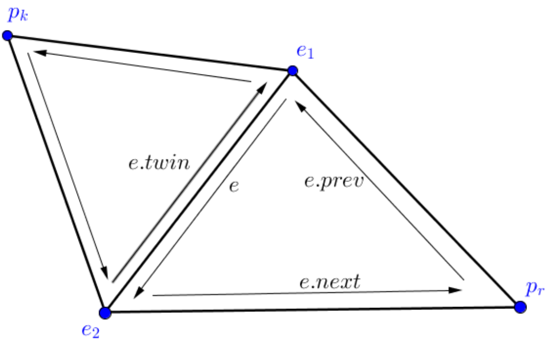

Figure 5: A DCEL of two faces of a triangulation T. If we want to check if e is illegal after inserting point p_r, we know $p_r = e.next.pTwo$. We draw a circle C through $e_1 e_2 p_r$ and check that $p_k = e.twin.next.pTwo$ is inside the C.

Example: Finding the face(s) that contains a point

```
public TriangleNode findPointRecurse(Point2D q,
    TriangleNode currentNode){

  if (currentNode.isEmpty()){
    HalfEdge e = currentNode.getTriangle().isCollinear(q);
    if (e!=null){//the query point q lies on an edge of the graph
      currentNode.addPointCollinear(q, e);
    }
    else{
    currentNode.addPointInside(q);
    }
    return currentNode;
  }
  else{
    for (TriangleNode child : currentNode.getChildren()){
      if (child.getTriangle().isInside(q)){
      return findPointRecurse(q, child);
      }
    }
  }
return null;
}
```

Example: DCEL Operations for adding a point from inside a face I

```
public void addPointInside(Point2D q){
    System.out.println("points␣be␣all␣inside␣"+ q.getX()+","+q.get
    TriangleFace f = triangle;
    copy();
    HalfEdge aq = new HalfEdge(f.getPoint(0), q);
    HalfEdge cq = new HalfEdge(f.getPoint(2), q);
    HalfEdge qc = new HalfEdge(q, f.getPoint(2));
    HalfEdge qa = new HalfEdge(q, f.getPoint(0));
    HalfEdge bq = new HalfEdge(f.getPoint(1), q);
    HalfEdge qb = new HalfEdge(q, f.getPoint(1));
    aq.setPrev(f.getEdge(2));
    aq.setTwin(qa);
    aq.setNext(qc);
    qa.setNext(f.getEdge(0));
    qa.setPrev(bq);
```

Example: DCEL Operations for adding a point from inside a face II

```
qa.setTwin(aq);
bq.setTwin(qb);
bq.setNext(qa);
bq.setPrev(f.getEdge(0));
qb.setTwin(bq);
qb.setNext(f.getEdge(1));
qb.setPrev(cq);
cq.setTwin(qc);
cq.setNext(qb);
cq.setPrev(f.getEdge(1));
qc.setTwin(cq);
qc.setNext(f.getEdge(2));
qc.setPrev(aq);
f.getEdge(0).setNext(bq);
f.getEdge(0).setPrev(qa);
f.getEdge(1).setNext(cq);
```

Example: DCEL Operations for adding a point from inside a face III

```
f.getEdge(1).setPrev(qb);
f.getEdge(2).setNext(aq);
f.getEdge(2).setPrev(qc);
TriangleFace f1 = new TriangleFace(f.getEdge(0), bq, qa);
TriangleFace f2 = new TriangleFace(f.getEdge(1), cq, qb);
TriangleFace f3 = new TriangleFace(f.getEdge(2), aq, qc);
addChild(f1);
addChild(f2);
addChild(f3);
f.getEdge(0).setFace(children.get(0));
f.getEdge(1).setFace(children.get(1));
f.getEdge(2).setFace(children.get(2));
aq.setFace(children.get(2));
qa.setFace(children.get(0));
bq.setFace(children.get(0));
qb.setFace(children.get(1));
```

Example: DCEL Operations for adding a point from inside a face IV

```
        cq.setFace(children.get(1));
        qc.setFace(children.get(2));
        legalizeEdge(qa.getNext());
        legalizeEdge(qb.getNext());
        legalizeEdge(qc.getNext());
    }
```

Effects of Point Insertion on the D Structure

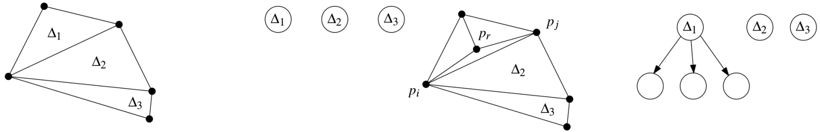

Figure 6: Moving from left to right, part 1 of the effects of the insertion of point p_r on the triangulation T and the 'delaunay history tree' structure D. Notice that p_r splits $\triangle_1$ into 3 sub triangles, and initializes three children under its corresponding node in D. See the impact of a flip operation on $\overline{p_i p_j}$ in the next slide

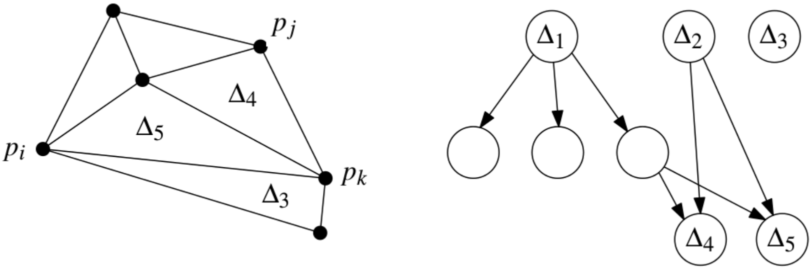

Figure 7: Part 2 of the p_r insertion: $\overline{p_i p_j}$ is swapped for $\overline{p_r p_k}$, creating faces $\triangle_4$ and $\triangle_5$ for the triangulation. These faces intersect a single child of $\triangle_1$ and $\triangle_2$, and are added to D accordingly as their children.

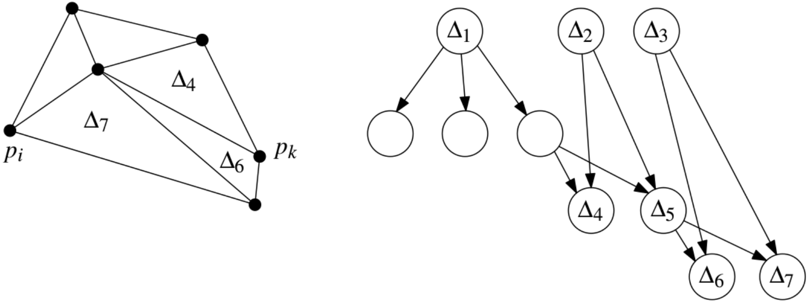

Figure 8: Part 3 of the p_r insertion: $\overline{p_i p_k}$ is swapped for $\overline{p_r p_l}$, creating faces $\triangle_6$ and $\triangle_7$ for the triangulation. These faces intersect a single child of $\triangle_3$ and $\triangle_5$, and are added to D accordingly as their children. In the end, the final leaves of D represent the legal triangulation of T.