Convex Hull, Voronoi Diagram and Delaunay Triangulation

Na Lei[1]

[1]DUT-RU International School of Information Sciences and Technology
Dalian University of Technology

2017-09-26

Halfedge Data Structure

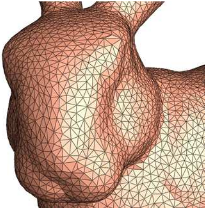
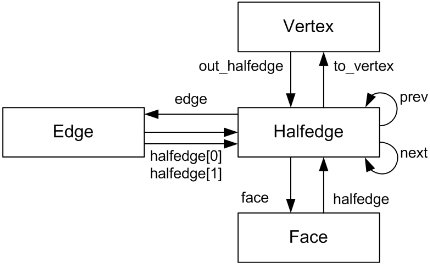
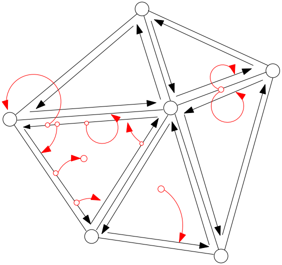


Figure: Half edge for the bunny mesh

Halfedge Data Structure





Course Contents

Course Contents

1. Orientation
2. Signed Area
3. Convex Hull
4. Voronoi Diagram
5. Delaunay Triangulation

Course Contents

1. Orientation
2. Signed Area
3. Convex Hull
4. Voronoi Diagram
5. Delaunay Triangulation

Course Contents

1. Orientation
2. Signed Area
3. Convex Hull
4. Voronoi Diagram
5. Delaunay Triangulation

Course Contents

Course Contents

1. Orientation
2. Signed Area
3. Convex Hull
4. Voronoi Diagram
5. Delaunay Triangulation

Orientation

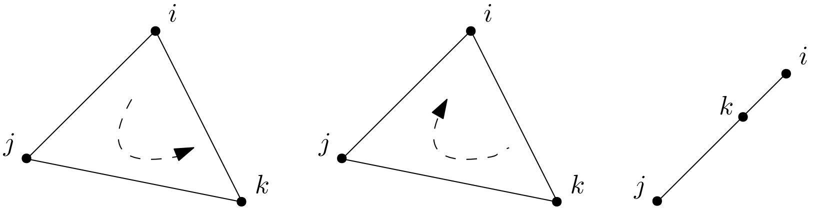


Figure: Positive, Negative & Zero orientation.

Given three ordered points $[i, j, k]$ in the plane, we say they have **positive orientation** if they define a counterclockwise oriented triangle, **negative orientation** if they define a clockwise oriented triangle, and **zero orientation** if they are collinear.

Orientation

How to determine the orientation of a triangle(2-simplex) $[i,j,k]$ in a plane(2D space)?

$$Orient(i,j,k) = det\left(j-i, k-i\right) = det\begin{pmatrix} x_j - x_i & x_k - x_i \\ y_j - y_i & y_k - y_i \end{pmatrix}$$

The triangle $[i,j,k]$ has positive orientation(CCW) if $Orient(i,j,k) > 0$, negative orientation (CW) if $Orient(i,j,k) < 0$, and zero orientation (vertices i,j,k are collinear) if $Orient(i,j,k) = 0$.

Orientation

How to determine the orientation of a tetrahedron(3-simplex) $[i,j,k,l]$ in a 3D space?

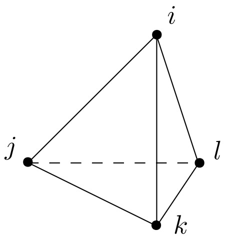


Figure: A tetrahedron.

$$Orient(i,j,k,l) = det\left(j-i, k-i, l-i\right) = det\begin{pmatrix} x_j - x_i & x_k - x_i & x_l - x_i \\ y_j - y_i & y_k - y_i & y_l - y_i \\ z_j - z_i & z_k - z_i & z_l - z_i \end{pmatrix}$$

Orientation

Why can not we determine the orientation of a triangle in a 3D space?

[1] https://en.wikipedia.org/wiki/Simplex

Signed Area

In general[1], the signed volume of an n-simplex in n-dimensional space with vertices $(v_0, v_1, \ldots, v_n)$ is

$$\frac{1}{n!} det\left(v_1 - v_0, v_2 - v_0, \ldots, v_n - v_0\right)$$

For instance:

- $SignedArea(i,j,k) = \frac{1}{2!} det\left(j - i, k - i\right) = \frac{1}{2} det \begin{pmatrix} x_j - x_i & x_k - x_i \\ y_j - y_i & y_k - y_i \end{pmatrix}$
- $SignedVolume(i,j,k,l) = \frac{1}{3!} det\left(j - i, k - i, l - i\right) =$
  $\frac{1}{6} det \begin{pmatrix} x_j - x_i & x_k - x_i & x_l - x_i \\ y_j - y_i & y_k - y_i & y_l - y_i \\ z_j - z_i & z_k - z_i & z_l - z_i \end{pmatrix}$

[1] https://en.wikipedia.org/wiki/Simplex

Area

Above, we talk about the area of a triangle in 2D space, but what about in 3D space?

$$Area(A,B,C) = \frac{1}{2}\left\|\vec{AB} \wedge \vec{AC}\right\| = \frac{1}{2}\left\|\begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_B - x_A & y_B - y_A & z_B - z_A \\ x_C - x_A & y_C - y_A & z_C - z_A \end{vmatrix}\right\| > 0$$

Convex Hull

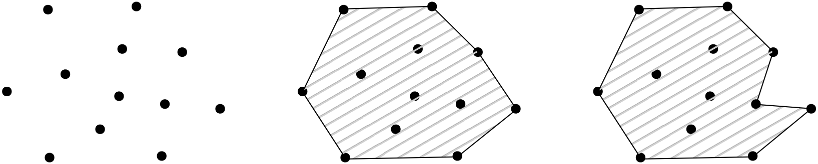


Figure: A point set, its convex hull and another concave polygon.

A set S is **convex** if given any points $p, q \in S$ any convex combination of p and q is in S, or equivalently, the line segment $\bar{pq} \subseteq S$.

The **convex hull** of set S is the intersection of all convex sets that contains S, or more intuitively, the smallest convex set that contains S.

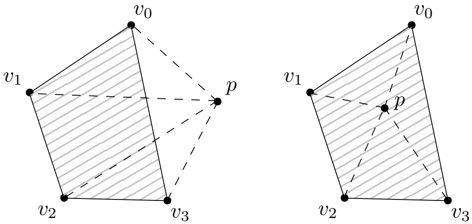
Convex Hull

Why is convex hull important?

1. It is one of the simplest shape approximations for a set of points.
2. Many algorithms compute the convex hull as an initial stage:
   - Find the smallest rectangle which can enclose a set of points.
   - Shape matching.
   - Optimal Mass Transport
3. . . .

Convex Hull

Convex hull of planar points



Initialize all points' state to *UNVISITED*, and an empty convex hull H;
Mark three outmost unvisited points v_0, v_1, v_2 with state *VISITED*, and put them in H with CCW;
while($\#$*UNVISITED* points > 0)
  Pick one unvisited point P, and mark it with state *VISITED*;
  for each directed edge AB in H do
    if $Orient(A, B, P)$ is CW then
      remove AB from H;
    end if
  end for
  connect P to the boundary of H, and keep H with CCW;
end while;

Convex Hull

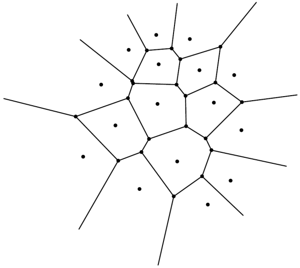
Above is the incremental convex hull algorithm which can be extended to n-dimension easily, but hard to implement in high dimension($n > 3$).

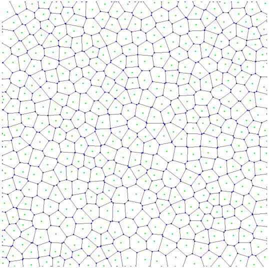
In 3D situation, we should consider tetrahedron instead of triangle mainly.

Also, there are many algorithms that can construct convex hull, and they have different features.[1]

[1] https://en.wikipedia.org/wiki/Convex_hull_algorithms

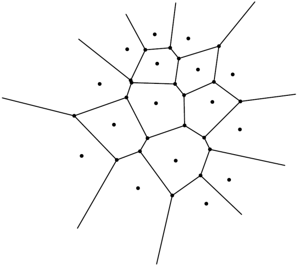
Voronoi Diagram

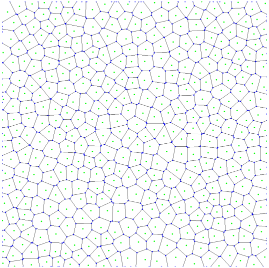




A **Voronoi Diagram** is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. That set of points (called seeds, sites, or generators) is specified beforehand, and for each seed there is a corresponding region consisting of all points closer to that seed than to any other. These regions are called **Voronoi cells**. The Voronoi diagram of a set of points is dual to its Delaunay triangulation.

Voronoi Diagram





Let $P = \{p_1, p_2, ..., p_n\}$ be a set of points in the plane, which we call **sites**. Define $\mathscr{V}(p_i)$, the **Voronoi cell** for p_i, to be the set of points q in the plane that are closer to p_i than to any other site. The Voronoi cell for p_i is then defined as:

$$\mathscr{V}(p_i) = \{q : |p_i q| < |p_j q|, \forall j \neq i\}$$

Voronoi Diagram

A series of properties:

- **Voronoi edges:** Every point on a Voronoi edge is the center of an empty circle through two neighboring sites p_i and p_j.
- **Voronoi vertices:** Each Voronoi vertex is the center of an empty circle through three neighboring sites p_i, p_j and p_k.

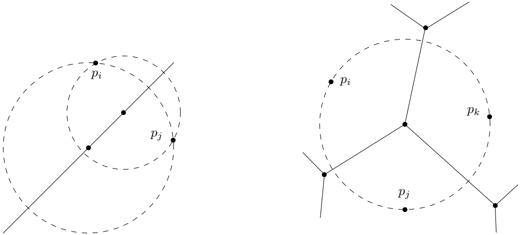


Figure: Voronoi edges and vertices of a Voronoi diagram

Voronoi Diagram

A series of properties:

- **Partition:** The Voronoi diagram of n sites divide the plane exactly into n cells.
- **Degree:** The vertices of the Voronoi diagram all have degree three (assume that there is no four sites are cocircular).

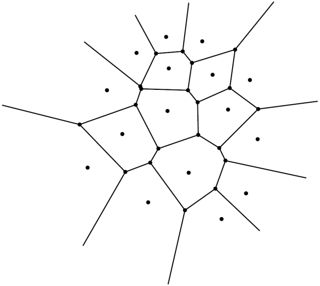


Figure: Partition and Degree properties.

Voronoi Diagram

A series of properties:

- **Convex hull:** A cell of the Voronoi diagram is unbounded if and only if the corresponding site lies on the convex hull.
- **Delaunay Triangulation:** The dual of Voronoi diagram of points set P is equivalent to the Delaunay triangulation of P.

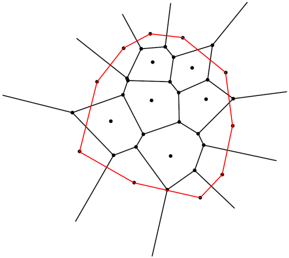


Figure: Induce the $ConvexHull(P)$ and $Delaunay(P)$ from the $VoronoiDiagram(P)$

Voronoi Diagram

There are several algorithms to construct Voronoi diagram directly[2] , also we can construct it by computing the dual of Delaunay triangulation of points P.

[2] https://en.wikipedia.org/wiki/Voronoi_diagram#Algorithms

Delaunay Triangulation

A series of properties:

- **Convex hull:** The boundary of the exterior face of the Delaunay triangulation is the boundary of the convex hull of the point set.
- **Circumcircle property:** The circumcircle of any triangle in the Delaunay triangulation is empty (contains no sites of P).
- **Empty circle property:** Two sites p_i and p_j are connected by an edge in the Delaunay triangulation, if and only if there is an empty circle passing through p_i and p_j.

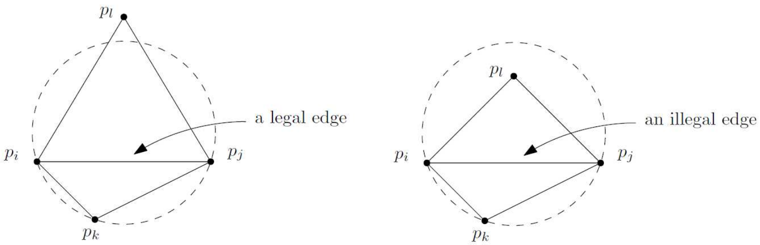


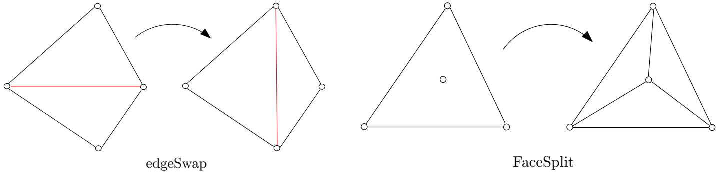
Figure: legal and illegal edges.

If p_l lies in the circumcircle for triangle $p_ip_jp_k$, then

$$inCircle(p_i,p_k,p_j,p_l) = det\begin{pmatrix} p_{ix} & p_{iy} & p_{ix}^2+p_{iy}^2 & 1 \\ p_{kx} & p_{ky} & p_{kx}^2+p_{ky}^2 & 1 \\ p_{jx} & p_{jy} & p_{jx}^2+p_{jy}^2 & 1 \\ p_{lx} & p_{ly} & p_{lx}^2+p_{ly}^2 & 1 \end{pmatrix} > 0$$

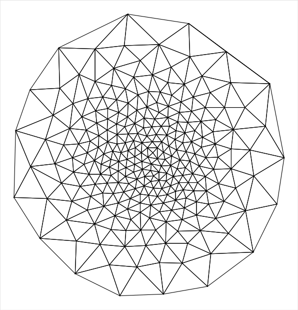
Half Edge Data Structure

Task One: Add edgeSwap, faceSplit methods.



Half Edge Data Structure

Task Two: Delaunay Triangulation.



Incremental Delaunay Triangulation

Incremental Delaunay Triangulation.

1. Construct an initial triangle, which is large.
2. Randomly generate a point p in the unit square.
3. InsertVertex(p).
4. Repeat step 2 and 3.

Incremental Delaunay Triangulation

InsertVertex(p)

1. pFace = LocatePoint(p)
2. FaceSplit(pFace)
3. Legalize three edges of the original pFace.

LocatePoint

Triangle Area

Given a triangle $[v_0, v_1, v_2]$ with $v_i = (x_i, y_i)$, the area is given by

$$S(v_0, v_1, v_2) = \frac{1}{2} \begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}$$

LocatePoint

Barycentric Coordinates

Given a triangle $[v_0, v_1, v_2]$ with $v_i = (x_i, y_i)$, p is a point on the plane, the barycentric coordinates

$$\alpha_k = \frac{S(p, v_{k+1}, v_{k+2})}{S(v_0, v_1, v_2)}$$

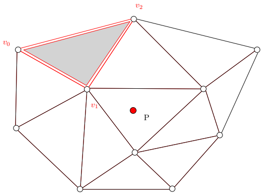
$(\alpha_0, \alpha_1, \alpha_2)$ are called the barycentric coordinates of p with respect to triangle $[v_0, v_1, v_2]$.

LocatePoint

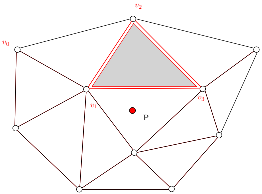
Face * LocatePoint(Point p)

1. Arbitrarily choose the initial face $[v_i, v_j, v_k]$
2. Compute the barycentric coordinates of p with respect to current face.
3. If $\alpha_i, \alpha_j, \alpha_k$ are non-negative, then return the current face.
4. Suppose α_i is negative, get the face adjacent to the current face sharing edge $[v_j, v_k]$, denote as $\tilde{F}$
5. If $\tilde{F}$ is empty, return NULL. The point is outside the whole range.
6. Set current face to be $\tilde{F}$, repeat through step 2.

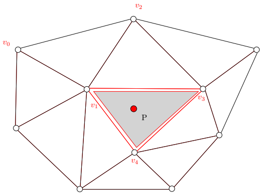
LocatePoint Example



current=$[v_0, v_1, v_2]$



current=$[v_2, v_1, v_3]$



current=$[v_3, v_1, v_4]$

Face Split

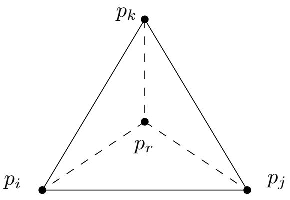


Figure: two situations while inserting sites.

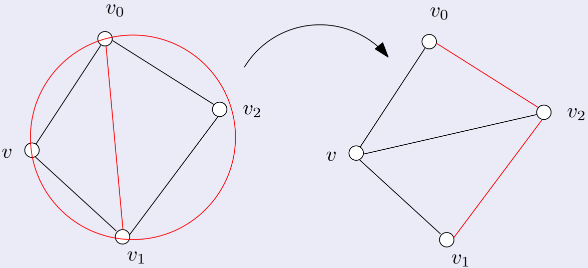
Legalize Edge

bool $LegalizeEdge(Vertex v, Edge e)$

1. Suppose $e = [v_0, v_1]$, the vertex against v is v_2, compute the circum circle c through v, v_0, v_1.
2. If v_2 is outside c, then return false.
3. $EdgeSwap(e)$
4. Recursive call $LegalizeEdge(v, [v_1, v_2])$;
5. Recursive call $LegalizeEdge(v, [v_0, v_2])$;
6. return true.

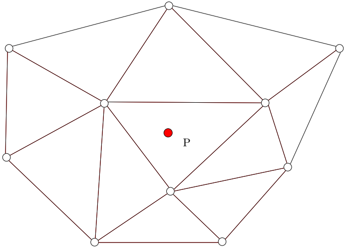
Legalize Edge

bool LegalizeEdge(Vertexv, Edgee)

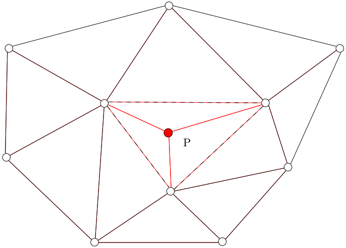


LegalizeEdge

Example InsertVertex(p)

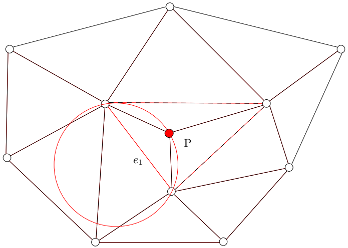


face = LocatePoint(p)

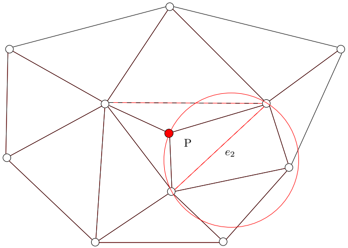


faceSplit(face)

Example InsertVertex(p)

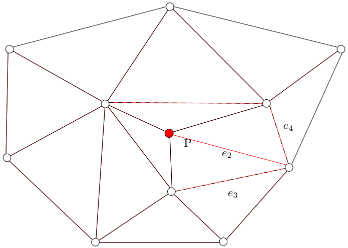


LegalizeEdge(e_1)

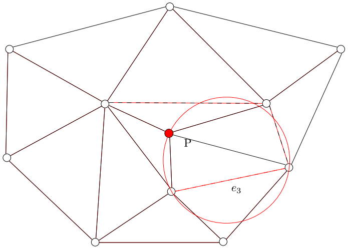


LegalizeEdge(e_2)

Example InsertVertex(p)

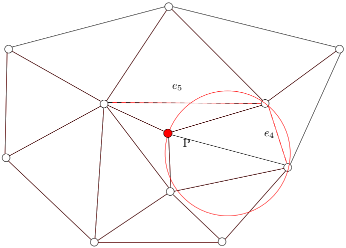


EdgeSwap(e_2)

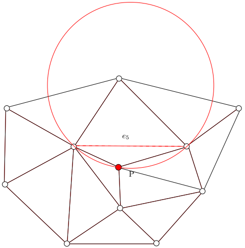


LegalizeEdge(e_3)

Example InsertVertex(p)

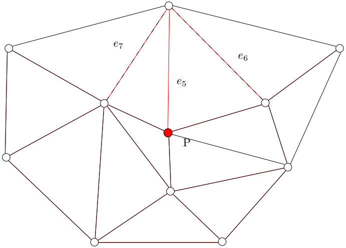


EdgeSwap(e_4)

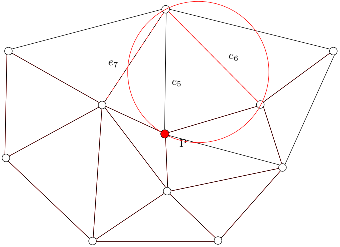


LegalizeEdge(e_5)

Example InsertVertex(p)

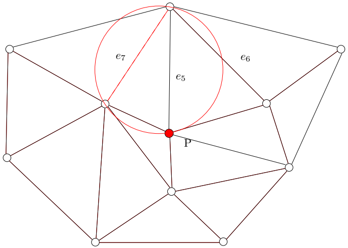


EdgeSwap(e_5)

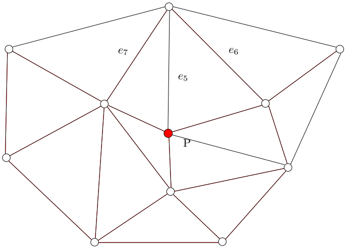


LegalizeEdge(e_6)

Example InsertVertex(p)



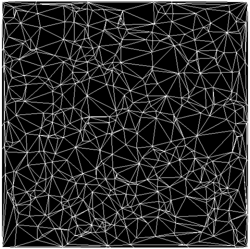
LegalizeEdge(e_7)



Result

References

- ”Incremental Delaunay Triangulation” by Dani Lischinski
- ”A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation” by Jim Ruppert
- ”QuadEdge Data Structure”, handout30, handout31, by Jim Stewart,



Delaunay Triangulation(Lifting map)

Given a finite point set P, the **lifting map** transforms the Delaunay triangulation of P into faces of a convex polyhedron in 3-dimension.

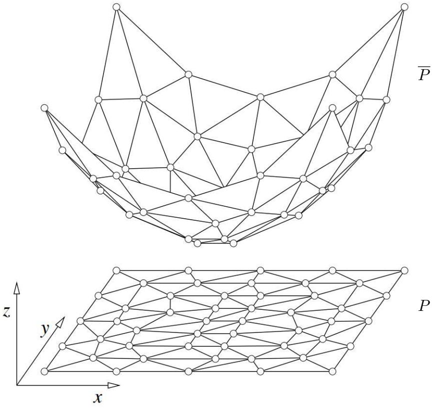


Figure: The lifting map

Delaunay Triangulation(Lifting map)

This relationship between the Delaunay triangulation and the corresponding convex hull has two merits:

1. It makes many properties of the Delaunay triangulation intuitive. For example, from the fact that every finite point set P has a convex hull, it follows that P has a Delaunay triangulaiton.
2. It brings to mesh generation the power of a huge literature on polytope theories and algorithms.
   For example, every convex hull algorithm is a Delaunay triangulation algorightm.

Delaunay Triangulation(Lifting map)

Algorithm 1 The lifting algorithm

Require: A 2D point set P

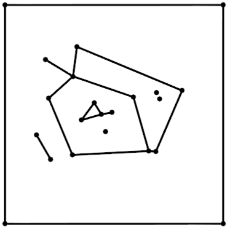
Ensure: A Delaunay triangulation $DT(P)$

Initialize a new 3D point set $\overline{P}$ by lifting $P = \{(x,y)\}$ to $\overline{P} = \{(x,y,x^2+y^2)\}$.
Compute the convex hull $CH(\overline{P})$ on the new point set $\overline{P}$.
for all face $\overline{f} \in CH(\overline{P})$ **do**
    if normal($\overline{f}$) is downward **then**
        project down $\overline{f}$ as f on the x-y plane
    end if
end for
return $DT(P) = \{f\}$

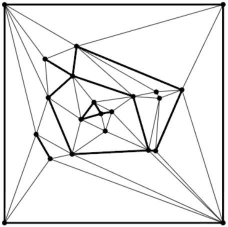
Rupper's Refinement

Rupper's refinement algorithm takes planar straightline graph (PSLG) as input, and output mesh with minimal angle greater than 20.7 degrees.

Rupper's Refinement

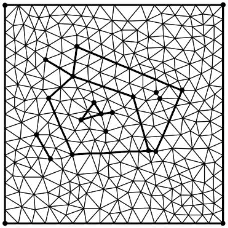


(a) Input PSLG and bounding box

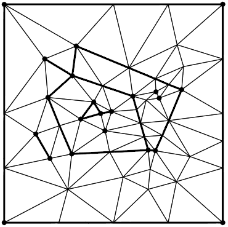


(b) Delaunay triangulation without Steiner points.

Rupper's Refinement



(a) Uniform mesh with min angle 22.5 degrees.



(b) Delaunay refinement with min angle 20 degrees.

Rupper's Refinement

Denote the vertex set of the input PSLG as V, the edges (segments) set as S. $DT(V)$ as the Delaunay Triangulation of V.

subroutine SplitTri(triangle t)

Add circumcenter of t to V, updating DT(V).

subroutine SplitSeg(segment s)

Add midpoint of s to V, updating DT(V).
Remove s from S, add its two halves s_1 and s_2 to S.

Rupper's Refinement

Algorithm DelaunayRefine

Input: planar straightline graph X;
desired minimum angle bound α;
Output: triangulation of X, with all angles $\geq \alpha$.

Initialize:

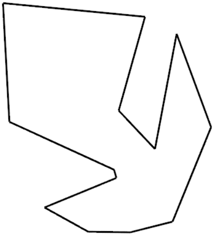
1. add a bounding square B to X:
    1. compute extremes of X: xmin, ymin, xmax, ymax
    2. let $span(X) := max(xmax - xmin, ymax - ymin)$
    3. let B be the square of side $3 \times span(X)$, centered on X
    4. add the four boundary segments of B to X.
2. let segment list S be the edges of X
3. let vertex list V be the vertices of X
4. compute initial Delaunay triangulation $DT(V)$.

Rupper's Refinement

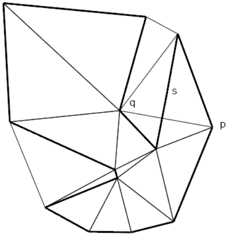
Algorithm DelaunayRefine

1. **Repeat**:
    1. **While** any segment *s* is encroached upon: SplitSeg(s);
    2. **let** *t* be any skinny triangle (min angle $< \alpha$)
    3. **let** *p* be *t*'s circumcenter
    4. **if** *p* encroaches upon any segment $s_1, s_2, \cdots, s_k$, **then**
        1. **for** i=1 to k: SplitSeg(s_i)
    5. **else**
        1. SplitTri(t)
2. **Until** no segment encroached upon, and no angle $< \alpha$
3. **Output** current Delaunay triangulation $DT(V)$.

Rupper's Refinement

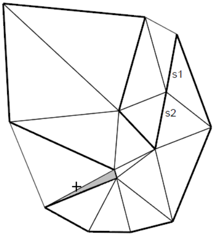


(a) Input polygon

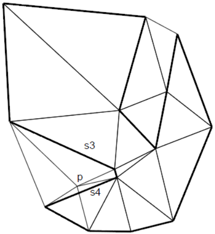


(b) Delaunay triangulation of input vertices. Segment s is not a Delaunay edge.

Rupper's Refinement

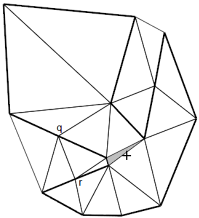


(c) Segment S is split at middle point into s_1 and s_2, shaded triangle has smallest angle, cross - circum center.

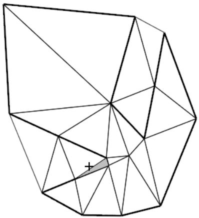


(d) If circle center p were added, it would encroach upon s_3 and s_4.

Rupper's Refinement

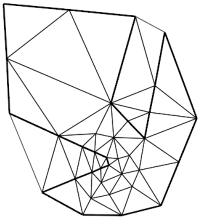


(e) 2 segments were split at q and r, Shaded triangle has minimum angle, will be split.

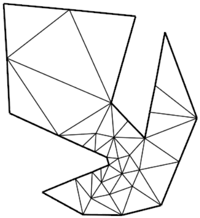


(f) New minimum angle 11.6 degree.

Rupper’s Refinement



(g) Final result with minmum angle 25 degree



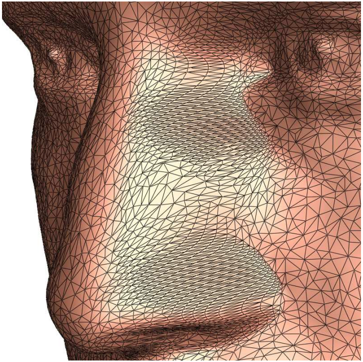
(h) External triangles removed

Examples for Rupper's Refinement

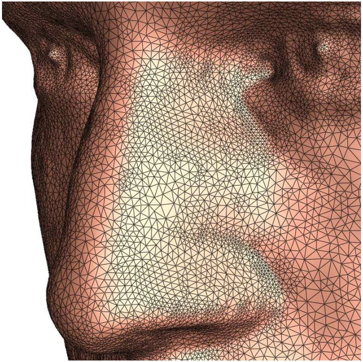
(a) Original mesh

(b)Conformal paramerization

Examples for Rupper's Refinement



(c) Original Triangulation



(d) Refinement result

Assignment

Assignment 2

Implement of incremental Convex Hull algorithm and Delaunay Triangulation algorithm based on Halfedge data structure.

(Send to shawnxpzheng@gmail.com, due on Oct 15.)