Ruppert's Delaunay Refinement Algorithm

Tong, Shen. *Wei, Cao.† Yifei, Jin ‡

June 14, 2014

Abstract

Delaunay triangulation maximizes the smallest angle among all possible triangulations of a given input and hence is a powerful discretization tool. However, Delaunay triangulation can have arbitrarily small angles depending on the input configuration. Thus, Delaunay refinement algorithms which iteratively insert additional points were developed to remedy the problem.

Delaunay refinement method is arguably the most popular due to its theoretical guarantee and performance in practice. The first step of a Delaunay refinement algorithm is the construction of a constrained or conforming Delaunay triangulation of the input domain. This initial Delaunay triangulation is likely to have bad elements. Delaunay refinement algorithm then iteratively adds new points to the domain to improve the quality of the mesh and to ensure that the mesh conforms to the boundary of the input domain. The points inserted by the Delaunay refinement are *Steiner points*. A sequential Delaunay refinement algorithm typically adds one new vertex at each iteration. Each new vertex is chosen from a set of candidates – the circumcenters of bad triangles and the mid-points of input segments. Chew [2] showed that Delaunay refinement can be used to produce quality-guaranteed in two dimensions. Ruppert extended the technique for computing not only quality-guaranteed but also size-optimal triangulations.

In our experiment report, we implement the Ruppert algorithm. And give a friendly interactive application.

*Email: 385989829@qq.com
†Email: fatboy_cw@163.com
‡Email: bluewould@yeah.net

1 Algorithm

1.1 Background

In two dimensions, the input domain Ω is usually represented as *planar straight line graph* (PSLG) – proper planar drawing in which each edge is mapped to a straight line segment between its two endpoint [3]. The segments express the boundaries of Ω and the endpoints are the vertices of Ω.

Definition 1 (diametral circle)**.** *The diametral circle of a segment is the circle whose diameter is the segment. Equivalently, the diametral circle of a segment is also the smallest circle that encloses the segment.*

Definition 2 (encroach)**.** *A segment is said to be encroached if a vertex lies strictly inside its diametral circle, or if the segment does not appear as an edge of the triangulation.*

Definition 3 (Constrained Delaunay triangulation)**.** *constrained Delaunay triangulation is a generalization of the Delaunay triangulation that forces certain required segments into the triangulation.* [1]

Definition 4 (Radius-edge Ratio)**.** *Radius-edge ration of a triangle is the ratio of its circumradius to the length of its shortest side.*

Definition 5. *skinny triangle*

Skinny trinangle, alternately named bad triangle, is a triangle whose radius-edge ratio is larger than a specified constant β.[2]

Note, in the refinement triangulation, we demand all triangles' angle larger than some constant α. It is easy to prove, in two dimensions, an upper bound β on the radius-edge ration implies a lower bound $\alpha = \arcsin(1/2\beta)$ on the smallest angle and vice versa.

1.2 Ruppert's Algorithm

The pseudocode Ruppert's algorithm is listed in 1. As an example , the figure 1 give a explanation.

[1] Note: Because a Delaunay triangulation is almost always unique, often a constrained Delaunay triangulation contains edges that do not satisfy the Delaunay condition. Thus a constrained Delaunay triangulation often is not a Delaunay triangulation itself.

[2] in this experiment report, we define $\beta = \sqrt{2}$

The algorithm input an PSLG graph. At first step, we do a Delaunay triangulation of the input vertices. But there are some segments of subsegments are not strongly Delaunay, just like the figure 1(c), which means these segments do not satisfies *empty circle property*[3]. And some vertices lie strictly inside a diametral circle of some segments, just like figure 1(d). In order to include the segments, second step, we add its mid-points. Because there are some segment locates outside the interval region, at third step, we just delete them. Then at fourth step, we process the skinny triangle just like the figure 1(e). we add the circumcenter. If the circumcenter is not cause any encroach, we accept it and re-triangulate the graph. If not we delete it and add the encroach segment. [4]. Through iterations, we could get final refinement Delaunay triangulation.

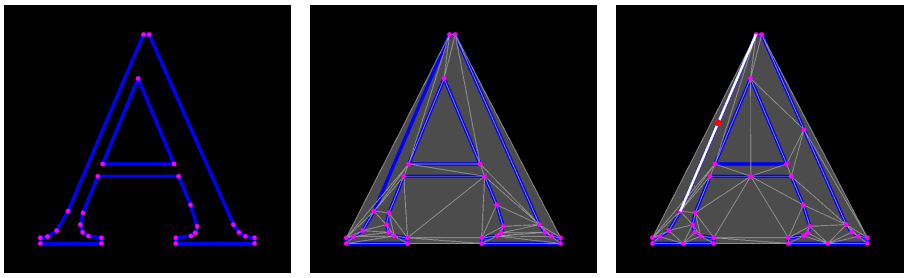
Algorithm 1: Ruppert's refinement triangulation algorithm

input : points, segment, threshold.
output: refinement triangulation algorithm.

```
1  T: = DelaunayTriangulation (points) ;
2  Q: = the set of encroached segments and skinny triangles ;
3  T, Q: = AddMidOfEncroach (T,Q) ;
4  T: = DelExternalSegment (T) ;
5  while Q is not empty: do
6      if Q contains a segment s: then
7          insert the midpoint of s into T ;
8      else Q contains poor quality triangle t:
9          if the circumcenter of t encroaches a segments s: then
10             add s to Q ;
11         else
12             insert the circumcenter of t into T ;
13         end if
14     end if
15     update Q ;
16 end while
17 return T ;
```

[3]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 .

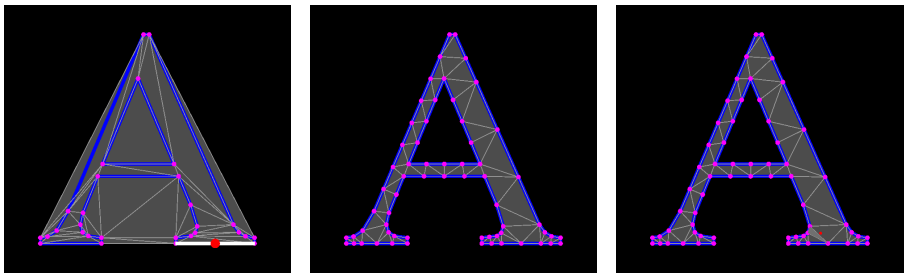
[4]Here note that: the circumcenter is deleted, in fact the segment is just encroach a virtual point. The reason is guarantee the following circumcenter locate in the internal region which we prove later



(a) A sample input PSLG

(b) Delaunay triangulation of the input vertices, note that some input segment are missing

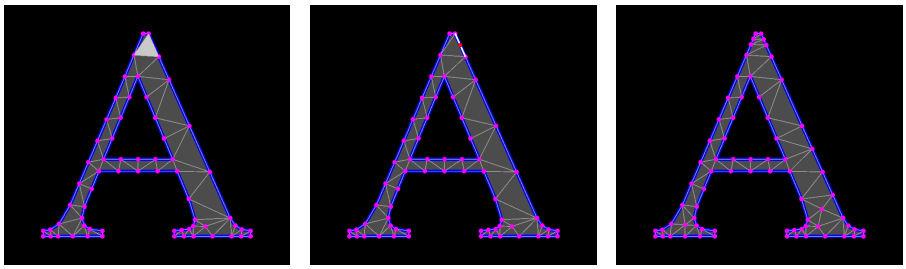
(c) Vertex insertion restores the missing segment



(d) Encroached subsegment is split

(e) Delete the outer segments

(f) Shinny triangle's circumcenter is inserted



(g) This circumcenter encroaches upon a segment, and is rejected

(h) Although the circumcenter was rejected, the segment it encroached upon is still marked for bisection

(i) Final mesh

Figure 1: The flow of Ruppert's Algorithm

1.3 Algorithm Analysis

1.3.1 Circumcenter lies in boundaries

The algorithm is not difficult, but their some question remained to answer. First is whether v always locate in boundaries or not. If not, the algorithm won't convergence. Fortunately, we can prove all circumcenter of triangles lies in boundaries.

Theorem 6. *Let T be a segment-bounded Delaunay triangulation (hence, any edge of T that belongs to only one triangle is a subsegment). Suppose that T has no encroached subsegments. Let v be the circumcenter of some triangle t of T. Then v lies in T. [4]*

Proof. Suppose for the sake of contradiction that v lies outside T. Let c be the centroid of t. c clearly lies inside T. Because the triangulation is segment-bounded, the line segment cv must cross some subsegment s, as Figure 2 illustrates. Because cv is entirely contained in the interior of the circumcircle of t, the circumcircle must contain a portion of s ; but the Delaunay property requires that the circumcircle be empty, so the circumcircle cannot contain the endpoints of s. Say that a point is inside s if it is on the same side of s as c, and outside s if it is on the same side of s as v . Because the center v of the circumcircle of t is outsides, the portion of the circumcircle that lies strictly insides (the bold arc in the illustration) is entirely enclosed by the diametral circle of s. The vertices of t lie uponts circumcircle and are (not strictly) insides. Up to two of the vertices oftmay be the endpoints of s, but at least one vertex oftmust lie strictly inside the diametral circle of s. But T has no encroached subsegments by assumption; the result follows by contradiction. □

1.3.2 Convergence

The claim that Ruppert's algorithm produces nicely graded meshes is based on the fact that the spacing of vertices at any loxation in the mesh is within a constant factor of the sparsest possible spacing. To formalize the idea of "sparsest possible spacing", Ruppert introduces a function called the *local feature size*

Definition 7 (local feature size)**.** *local feature size of each point $x \in \mathcal{R}^2$, denoted by $lfs_\Omega(x)$, is the radius of the smallest disk centered at x that touch two different features which is the vertices and boundary segments of Ω.*

Figure 3 illustrates the notion by giving examples of such disks for a variety of points.

It is not difficult to prove the lemma below:

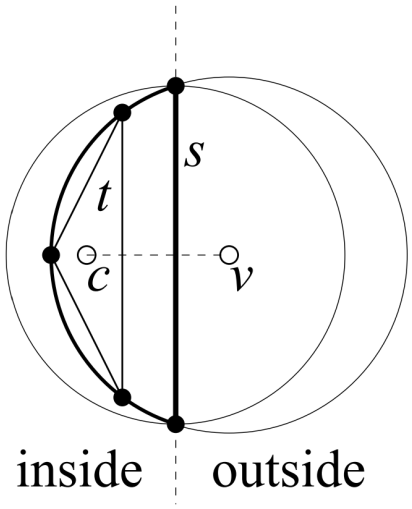


Figure 2: : If the circumcenter v of a triangle t lies outside the triangulation, then some subsegment s is encroached.

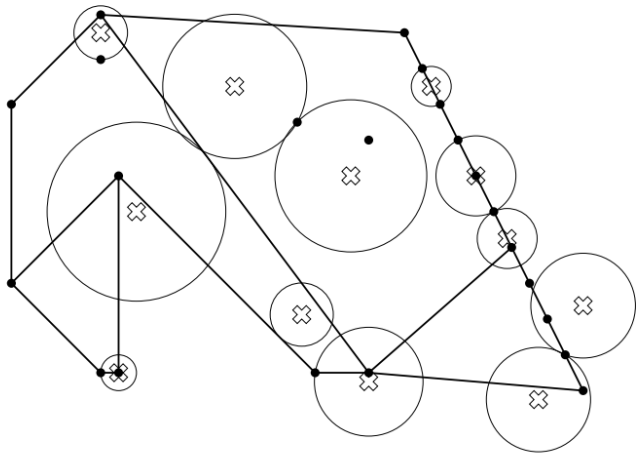


Figure 3: the radius of each disk illustrated is the local feature size of the point at its center

Theorem 8. *For any PSLG X, and any two points u and v in the plane,*

$$lfs(v) \leq lfs(u) + |uv|$$

Proof. Since $lfs(u)$ represent a disk centered at u and intersects two non-incident

features of X. So the disk whose center is at v and radius is $lsf(u)+|uv|$ also intersect at least two features since it contains disk $lfs(u)$. Hence $lfs(v) \leq lfs(u) + |uv|$. □

Ruppert prove that radius-edge ratio larger than $\sqrt{2}$ and any two incident segments in the input PSLG are separated by an angle of 60° or greater, that the algorithm produces meshes that are nicely graded and size-optimal.

Theorem 9. *Let $lfs_{\min}$ be the shortest distance between two non-incident entities (vertices or segments) of the input PSLG*

suppose that any two incident segments are separated by an angle of at least 60°, and a triangle is considered to be skinny if its circumradius -to-shortest edge ratio is larger than B, where $B \geq \sqrt{2}$ Ruppert's algorithm will terminate, with no triangulation edge shorter than lfs_{min}, where lfs_{min} is the $\min_u lfs(u)$, where u is chosen from among the input vertex.

The proof of the theorem is not difficult but tedious. You can refer reference [4]. Since no edge shorter than lfs_{min}, and the region is limited. Therefore the algorithm must terminate.

1.3.3 Size-Optimality

Theorem 9 guarantees that no edge of the final mesh is smaller than $lfs_{\min}$, but it is not satisfying spatially graded mesh.

What following is a proof that each edge of the output mesh has length proportional to the local feature sizes of its endpoints.

Here omit the proof detail, just give the result.

Theorem 10. *For any vertex v of the output mesh, the distance to its nearest neighbor w is at least $\frac{lfs(v)}{D_s+1}$, where D_s is a constant larger than 1.*

1.4 Implement trick

The algorithm has several implement trick, we list some important below:

- At first we count the encroach segment. The segment is not strongly Delaunay , i.e. the segment is not used by Delaunay triangulation. There must be a vertex on or inside its diametral circle. This observation is important because it unifies the theoretical treatment of missing subsegments and encroached subsegments that are not missing.

- A subsegment may be tested for encroachment by inspecting only those vertices that appear directly opposite the subsegment in a triangle. Consider Figure 4. Both of the vertices (v and w) opposite the segment s lie outside the diametral circle of s. Because the mesh is constrained Delaunay, each triangle's circumcircle is empty (on its side of s), and therefore the diametral circle of s is empty.

  The Delaunay refinement process may cause other subsegments to become encroached. According to the observation above, we don not check all segment each time but just test each of the edges that appear opposite the vertex in some triangle is enough.

- Each time we insert a new circumcenter, we need know the point locate in which triangle. So we need a point a location algorithm to remain the information. But in this algorithm, we already know the Delaunay triangulation, so there exist a simple point location algorithm. Just long the segment between circumcenter and orthocenter. Search all the triangle along the segment, we could find the circumcenter location.

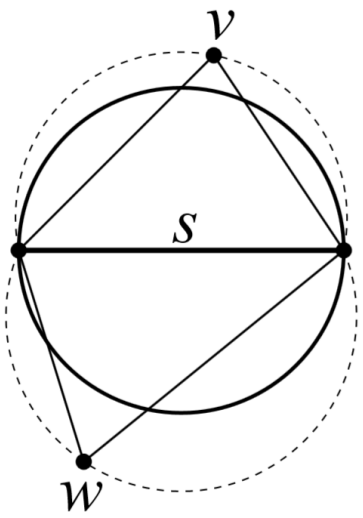


Figure 4: if the apices(here, v and w) of the triangles that contain a subsegment s are outside the diametral circle of s, then no vertex lies in the diametral circle of s, because the triangles are Delaunay

2 Code Structure

The code of the this project is written by *Python 2.7* based on *PyOpenGL* and *PyQt5*. Only the initial delaunay triangulation is implemented with package *triangle* provided y CMU. The code structure of main algorithm part is associated in following way:

- cgalgo.py
  - Sgn(x):Check the sign of a float number, the eps is setted by $1e12$.
  - TurnLeft(v):return the 90 degree left turn vector of input vector.
  - Cross(a,b, c):return the multiplication cross of vector $\vec{ba}$ and $\vec{ca}$.
  - GetDistance(u,v):return the Euclidean distance of point u and v.
  - GetIntersection(lu, lv):return the intersection of line lu and lv.
  - GetBisector(u, v):return bisector of line uv
  - GetCircleCenter(a, b, c):return circumcenter of triangle abc.
  - GetSegIntersection(a, b, c, d):return the intersection of segment ab and cd
  - InTriangle(p, a, b, c):check whether point p is inside of triangle abc.
  - InCircle(p, a, b, x):checht whether the point x is inside of circumcircle of triangle pab.
- Ruper.py
  - InitializeDelaunay: return the initial delaunay triangulation implemented with third part package *triangle*.
  - TrianglePointLocation: find the circumcenter of a skinny triangle by flipping triangle sequentially.
  - IsEncroached:check whether a segment is a encroached segment.
  - IsSkinny:check whether a triangle is a skinny triangle.
  - UpdateDelaunay: Do swap test and maintain some necessary information when the triangulation changed.
  - RecoverTriangles: recover the triangulation if a circumcenter is failed to insert into triangulation.

- SplitSegment:split a segment by inserting a midpoint of corresponding segment.
- InsertCircleCenter:insert a circumcenter of a skinny triangle.
- EliminateSegment:Main loop of segments processing.
- EliminateAngle:Main loop of triangles processing.
- RemoveOutSide:Remove the triangles outside the planar graph.

3 Software instruction

This part you can see our homepage http://caow13.github.io/Rupers_Algorithm/.

4 Experiment

We list some comparison about origin Delaunay triangulation and Ruppert's refinement algorithm. Just like figures 5 and 6

5 Conclusion and Future work

The Ruppert's algorithm is an fundamental work about refinement Delaunay triangulation. But the algorithm time complexity is difficult to analysis. We already know there are some algorithm can achieve $O(m + n\log(n))$ where m is the extra points we add. The further work you can see reference [1]

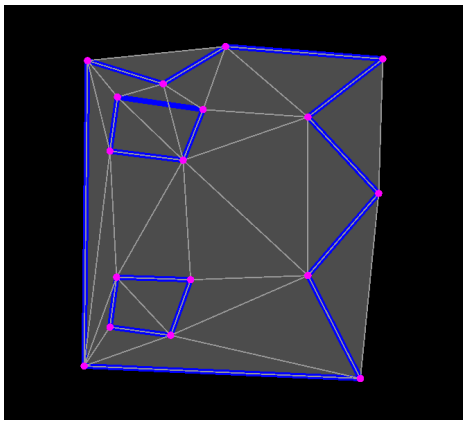
References

[1] Har-Peled, Sariel, and Alper Üngör. "A time-optimal Delaunay refinement algorithm in two dimensions." Proceedings of the twenty-first annual symposium on Computational geometry. ACM, 2005.

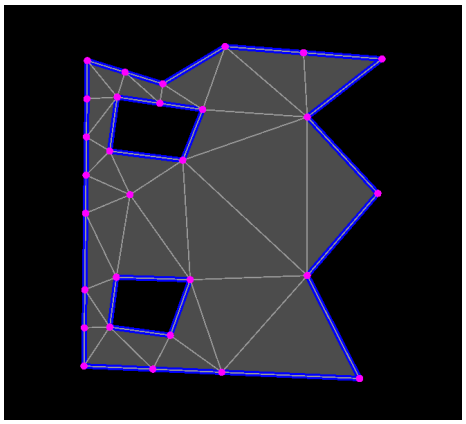
[2] L.P.Chew. Constrained Delaunay triangular meshes. Technical Report TR-89-983, Dept. Comput. Sci. Cornell Univ., Ithaca, NY, Apr. 1898

[3] J. Ruppert/ A new and simple algorithm for quality 2-dimensional mesh generation. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithm, pages 83-92, 1993*

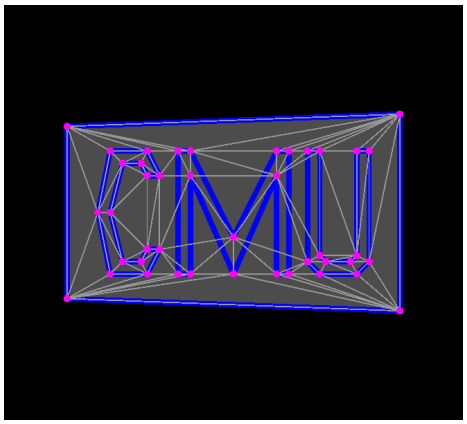
[4] Lecture Notes on Delaunay Mesh Generation, Jonathan Richard Shewchuk, September 20, 1999



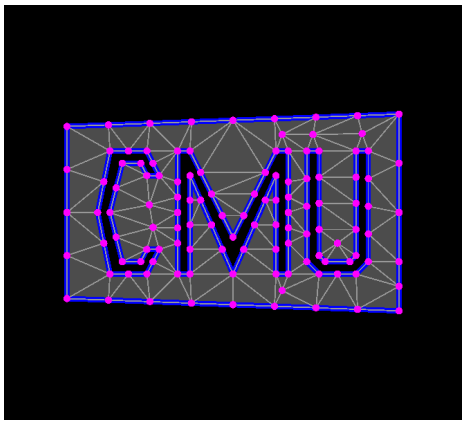
(a) Holes- Delaunay



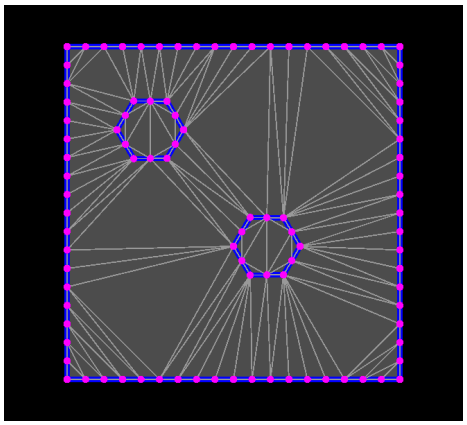
(b) Holes- Ruppert



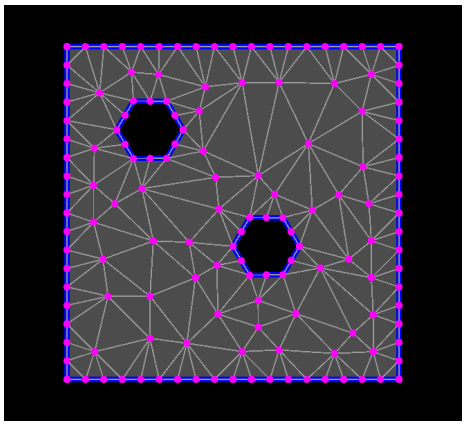
(c) cmu- Delaunay



(d) cmu- Ruppert

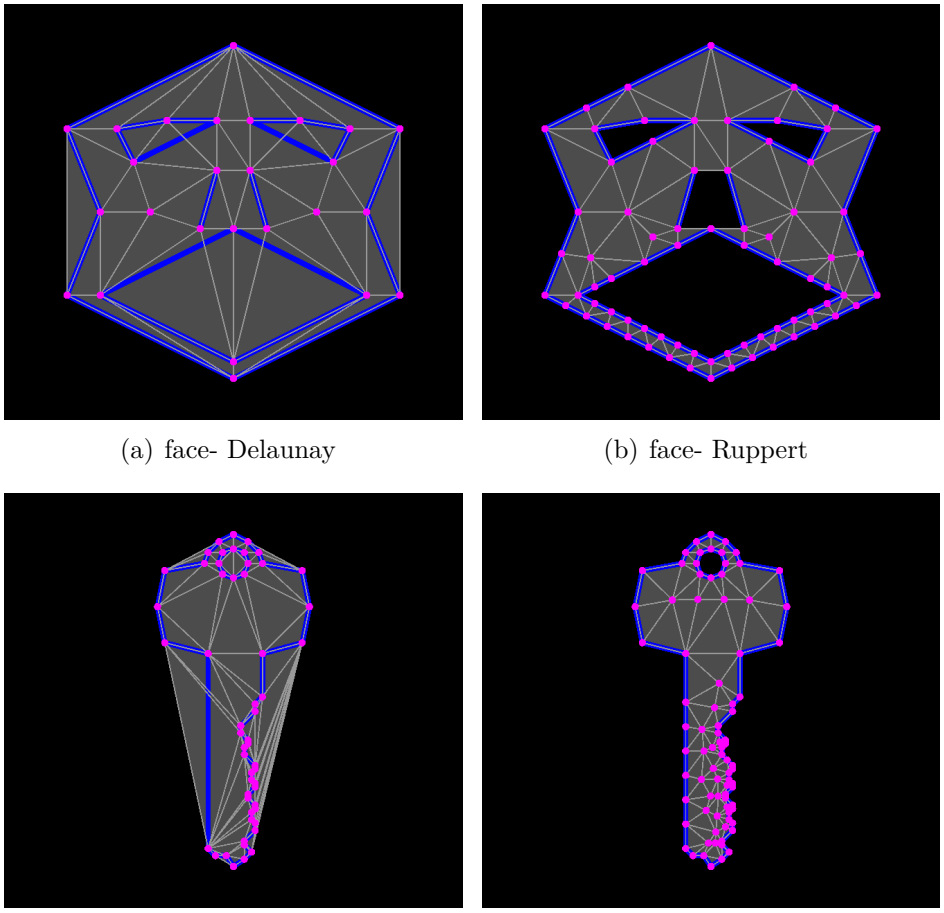


(e) double_hex- Delaunay



(f) double_hex- Ruppert

Figure 5: experiment results(1)



(a) face- Delaunay

(b) face- Ruppert

(c) key- Delaunay

(d) key- Ruppert

Figure 6: experiment results(2)