## Outline

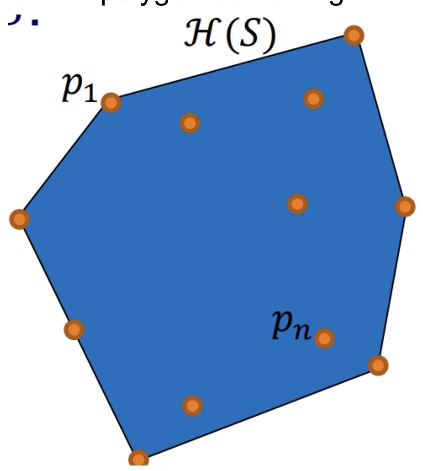
1. Definition ~30mins
2. Review of algorithms ~30mins

## 1 Definition

- Simple polygon: It is a polygon that does not intersect itself and has no holes. In this paper, it is defined as a closed polygonal chain of line segments such that the plane is partitioned by the polygon into three connected parts: bounded interior, unbounded exterior, and the polygon itself. (Carrie)

- Star-shaped polygon: Formally, a polygon P is star-shaped if there exists a point z such that for each point p of P the segment zp lies entirely within P.
  In this paper, a star-shaped polygon is defined as such: there is an internal point from which all other points inside the polygon are visible.
  Question: What is not a star-shaped polygon (Jack's funky weird polygon)?
Convex / Non-convex?
  Convex polygons are star-shaped
  Regular star polygons (a type of non-convex polygons, a type of decagons)
  (Caroline)
  https://www.twinkl.com/teaching-wiki/star-shape

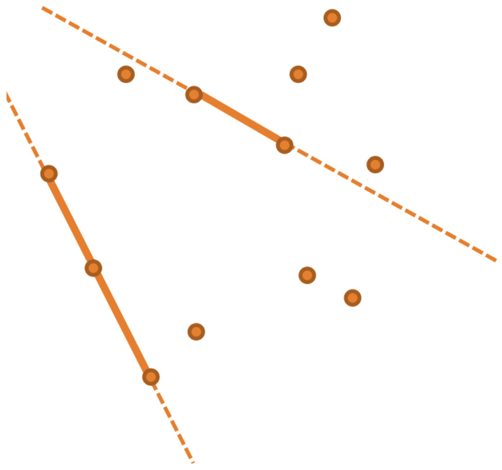
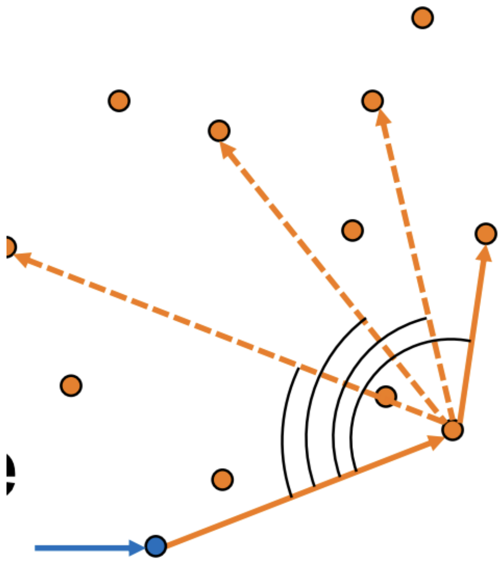
- Convex hull: (Carrie)
  The convex hull of a set of points $S \in R^d$, denoted $H(S)$ is the smallest convex polygon containing $S$



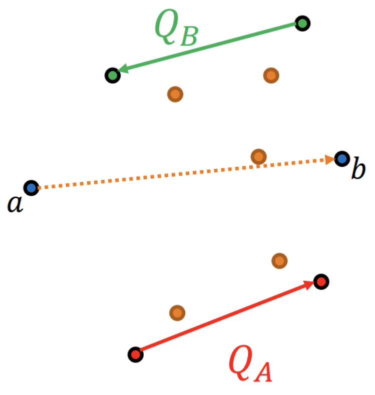
The extreme points of a set of points $S \in R^2$ are the points which are on the convex hull and have interior angle strictly less than $\pi$

Question: Given a set of points $S \in R^2$, how do we compute the convex hull efficiently?
- Do we want all points or just the extreme ones
- Do the output vertices need to be sorted?

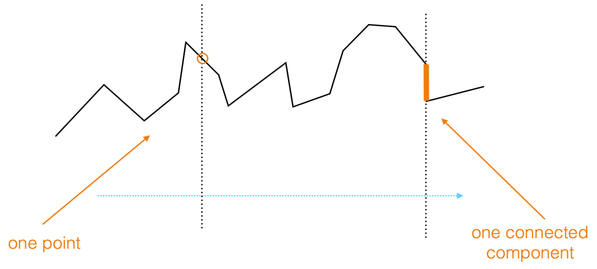
Some algorithms:

1. Naive Algorithm
    a. $O(n^3)$
    b. For each directed edge $e \in S \times S$, check if half space to the right of $e$ is clear of points and there are no points along the line of $e$
    c. 
    d. Then frow the hull from a hull vertex and searching for the next edge on the hull by trying all possible points $O(n^2h)$ h-- number of points on the hull
2. Gift wrapping
    a. $O(nh)$
    b. Finding the next edge on the hull by choosing the vertex that makes the largest angle to the hull vertex
    c. 
3. Quick Hull (ipad)
    a. In the worst case: O(n^2)
    b. In reality: O(n logn)

```
QuickHull( $S \subset \mathbb{R}^2$ )
    ◦ $(a, b) \leftarrow$ HorizontalExtrema( $S$ )
    ◦ $A \leftarrow$ RightOf( $S$ , $\overrightarrow{ab}$ )
    ◦ $B \leftarrow$ RightOf( $S$ , $\overrightarrow{ba}$ )
    ◦ $Q_A \leftarrow$ QuickHalfHull( $A$ , $\overrightarrow{ab}$ )
    ◦ $Q_B \leftarrow$ QuickHalfHull( $B$ , $\overrightarrow{ba}$ )
    ◦ return $\{a\} \cup Q_A \cup \{b\} \cup Q_B$
```



- Monotone polygon: A simple polygon *P* is called monotone with respect to a given direction of *d* if the boundary of *P* can be partitioned into two chains such that both of them are monotone with respect to *d*. A polygonal chain is L-monotone if any line perpendicular to line L intersects it in one point (one connected component). Can use 'horizontal/vertical' line test. Any non-monotone polygons? (Caroline)



This is a paper on how to generate x-monotone polygon
https://reader.elsevier.com/reader/sd/pii/0925772195000313?token=110125EACE6F80E86CE958E24C55528B87DA9120435CB63CCBC76F6C1A5765F25E44CF327C21976399449364E771ED08&originRegion=us-east-1&originCreation=20220801201311

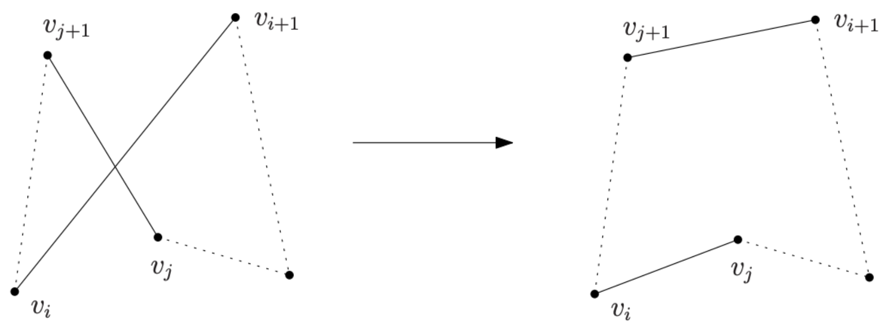
Fun game on convex/concave polygons
https://www.mathopenref.com/polygonconvex.html

- Minimum spanning tree

Aug 2

# 2 Chapter 2: Brief review of existing algorithms for generating polygonal shapes

Previous approaches to generating polygon:
1. Angular-sorting, time complexity O(nlogn)
2. Polygon of certain property–star-shaped, time complexity O(n^2)  -- starting from convex hull (Thomas et al.)
3. Given vertices→permutation connecting them
-exhaustive search, time complexity exponential # of nodes
To correct self-intersection, apply ’2-opt’ moves. Replaces a pair of intersecting edges by switching the ‘coordinates’



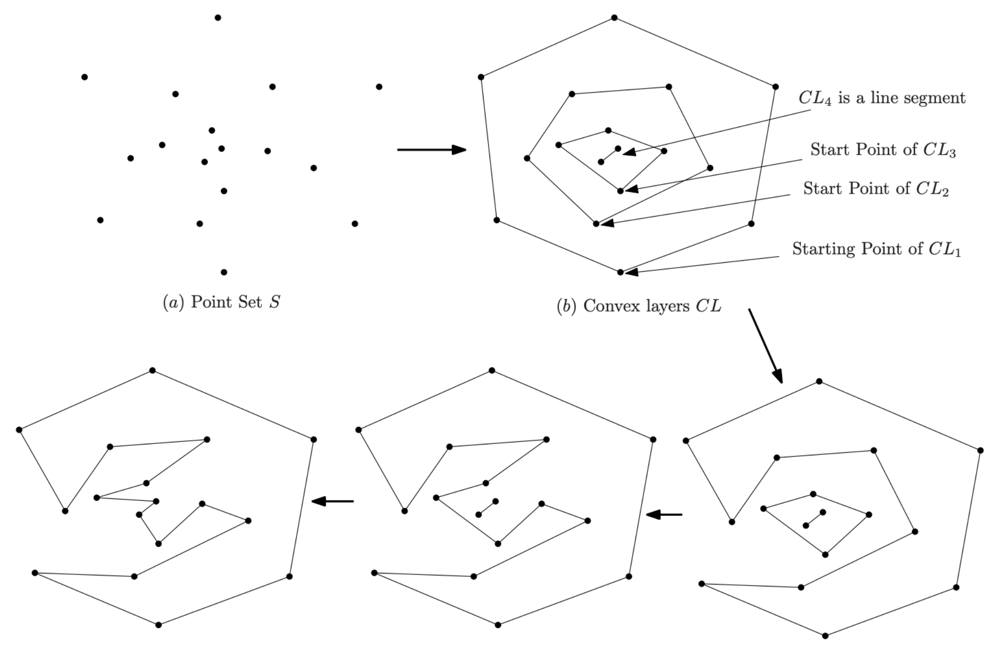
4. Random generation of convex polygons, unique way (convex hull)
(Their question is about making possible convex polygons out of the subset of dots (not using all of them) though. Also, we don’t need it to be necessarily a convex polygon)
A polygon P is generated randomly for a given set S of n points in the plane if the probability of generating P is 1/k, where k is the number of simple polygons that can be generated from S. No method is known yet to determine the value of k.

5. Generate polygons for a given set of points S in 2-d plane with less time complexity
- Find left-most point p (if two are the same, find the one with less y)
- Jarvis’s March algorithm to find convex hull

```
$m \leftarrow 1$;
**repeat**
    Find the lowest point(smallest y coordinate) from $S$;
    Let $i_0$ be its index, and set $i \leftarrow i_0$;
    Take an extra point $p_0(-\infty, y)$ at extreme left;
    Consider $e(p_0, p_i)$ as hull edge;
      /* Do not add this hull edge in convex layer $CL_m$ */
    **repeat**
          /* $j$ is an index of a point $p \in S$ */
        **for** *each* $j \neq i$ **do**
            Compute counterclockwise angle $\theta$ from previous hull edge;
            Let $k$ be the index of the point with the smallest $\theta$;
            Add $e(p_i, p_k)$ as a hull edge in $CL_m$ ;
            $i \leftarrow k$;
    **until** $i \neq i_0$ ;
    $S \leftarrow S - \{Vertices\ on\ CL_m\}$;
    $m \leftarrow (m+1)$;
**until** $S \neq \Phi$ ;
**return** $CL(S)$;
```

- 
- Change one edge of two layers respectively

```
$i \leftarrow 1$;
$P \leftarrow CL_i$;
while $(i \neq m)$ do
    Select an edge $e(p,q)$ randomly from $CL_i$;
    Find the tangents to the $CL_{i+1}$ from $p$ and $q$ of $CL_i$;
    Compute the set of edges $EV(e(p,q)) \in CL_{i+1}$ which are "end visible" with respect to $e(p,q) \in CL_i$;
    Randomly select an edge $e(u,v) \in EV(e(p,q))$;
      /* Delete the edges $e(p,q)$ and $e(u,v)$ from $CL_i$ and $CL_{i+1}$ respectively; */
    $CL_{i+1} \leftarrow CL_{i+1} - e(u,v)$;
    $CL_i \leftarrow CL_i - e(p,q)$;
      /* Connect the points $p$ with $u$ and $q$ with $v$ respectively; */
    $P \leftarrow \{P - e(p,q) + CL_{i+1} + e(p,u) + e(q,v)\}$;
    $i \leftarrow (i+1)$;
return $P$;
```



$(a)$ Point Set $S$ $(b)$ Convex layers $CL$

(e) Polygon after merging the $CL_3$ and $CL_4$ (d) After merging the $CL_2$ and $CL_3$ (c) After merging the $CL_1$ and $CL_2$

Open Problem: generating a random polygon from given vertices (not sure if it means going through all the vertices, or making a selection?)

Open Problem: how to determine the number of k

Aug 3

# 3 Chapter 3: Inward Denting and Polygon Generating Algorithms

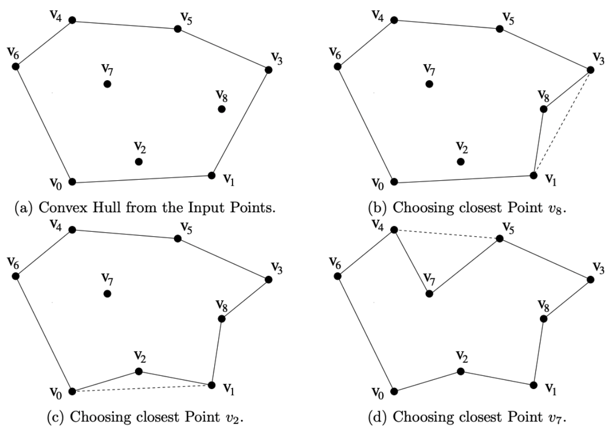
## 3.1 Inward denting Algorithm

- <u>Inward-denting:</u> It constructs a polygonal shape iteratively starting from the convex hull boundary of the polygon.

-Nodes very close to each other should also appear within a small ‘hop distance’ along the constructed polygonal boundary. ‘Hop distance’ between two nodes u and v is defined as the number of edge links between u and v in the polygonal path.’

-Breaking distance: If there are unconnected nodes inside the approximate boundary then each edge has a closest node. The distance of an edge to its closest node is called the breaking distance.

-Splitable edge: The boundary edge with the smallest breaking distance is called the splitable edge . In Figure 3.1a, edge (v1, v3) is the splitable edge.



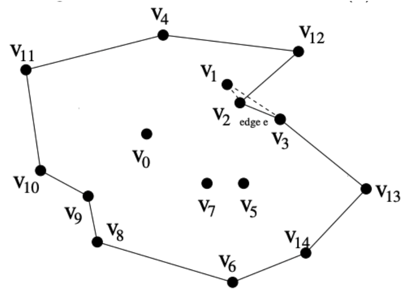
(a) Convex Hull from the Input Points. (b) Choosing closest Point $v_8$.
(c) Choosing closest Point $v_2$. (d) Choosing closest Point $v_7$.

Figure 3.1: Illustrating Denting.

However, there can be a problem……When intermediate polygons get complex (I wonder what exactly is complex lol) complicated, like having lots of nodes connecting/have lots of nodes, there can be self-intersection. (what if we add a constrain s.t. The line of the edge to the vertex inside needs to be greater than pi/2) Question for 小天才们

-Invalid (unsplittable edge): denting edge of e that results in a self intersecting polygon.

Mark invalid and search for the next splitable edge.

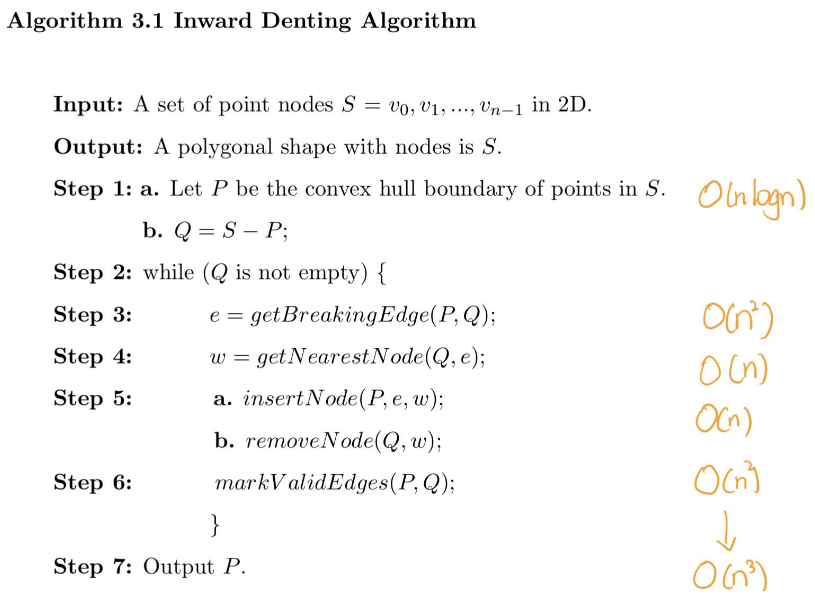


(e) Invalid Edge $e$.

Data storage: Boundary of polygon (P = n + e) as a list of nodes and a list of edges. Internal nodes Q as a simple list of data.
P = n + e
N -- list of nodes on the polygon (vertices) -- assumingly sorted
E -- list of edges -- start & end nodes should be able to access
Q -- internal nodes

Function 3.1 whether an edge of P containing the set of internal nodes Q is valid (check for intersection) -- each edge of e
Function 3.2 find nearest node to an edge -- repeated for every node in Q
Function 3.3 mark valid edges
Function 3.4 find breaking edge through an internal node (smallest total length) -- examine each valid edge of P

Steps:



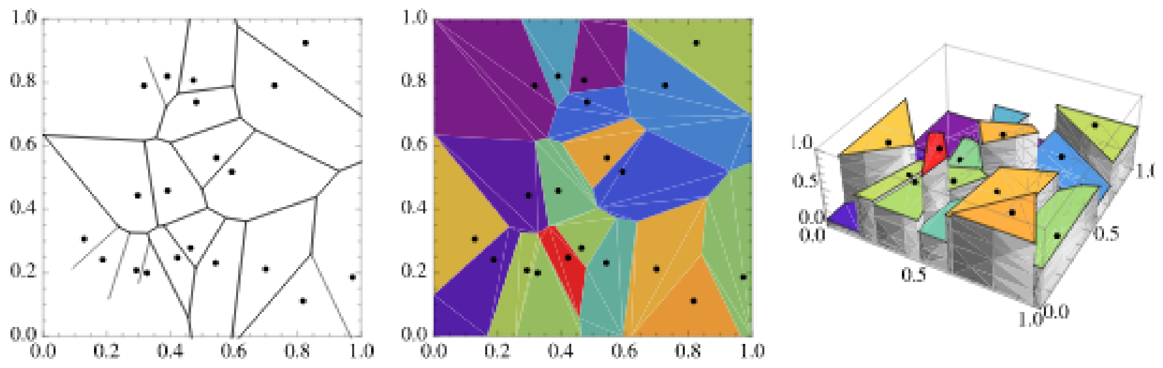
**Algorithm 3.1 Inward Denting Algorithm**

```
Input: A set of point nodes $S = v_0, v_1, ..., v_{n-1}$ in 2D.
Output: A polygonal shape with nodes is $S$.
Step 1: a. Let $P$ be the convex hull boundary of points in $S$.        $O(n \log n)$
        b. $Q = S - P$;
Step 2: while ($Q$ is not empty) {
Step 3:     $e = getBreakingEdge(P, Q)$;                                   $O(n^2)$
Step 4:     $w = getNearestNode(Q, e)$;                                    $O(n)$
Step 5:     a. $insertNode(P, e, w)$;                                      $O(n)$
            b. $removeNode(Q, w)$;
Step 6:     $markValidEdges(P, Q)$;                                        $O(n^2)$
        }                                                                  $\downarrow$
Step 7: Output $P$.                                                        $O(n^3)$
```

## 3.2 Voronoi based inward denting algorithm

- Reduce time complexity by finding faster way to compute nearest neighbor of each node
- <u>**Voronoi diagrams**</u>:

  The partitioning of a plane with n points into convex polygons such that each polygon contains exactly one generating point and **every point in a given polygon is closer to its generating point than to any other.**



- **How?**

1. Have an array of generating points.
2. Loop through every pixel on your canvas.
3. For every pixel look for the closest generating point to it.
4. Depending on what diagram you wish to get color the pixel. If you want a diagram separated with a border, check for the second to closest point, then check their difference and color with the border color if it's smaller than some value.

**Delaunay Triangulation**

**(https://www.baeldung.com/cs/voronoi-diagram)**

**Def: for a given set P of discrete points, construct DT(P) s.t. No point in P is inside the circumcircle of any triangle in DT(P)**

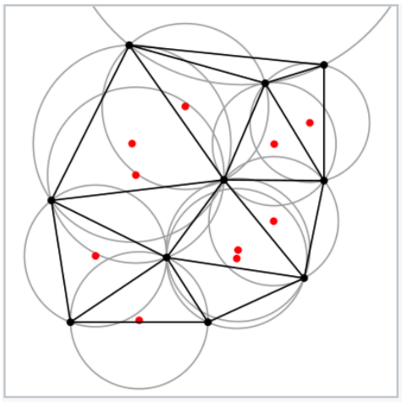
**Algorithm: Bowyer-Watson Algorithm**

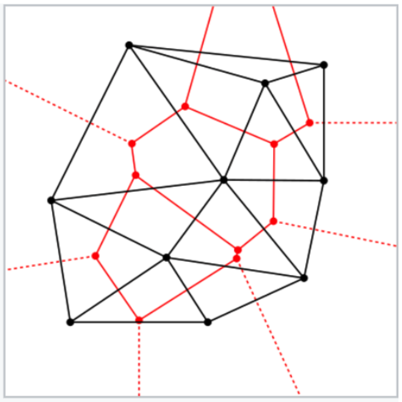
- **Create a super triangle that encloses all the points in P**

**For each point in P:**

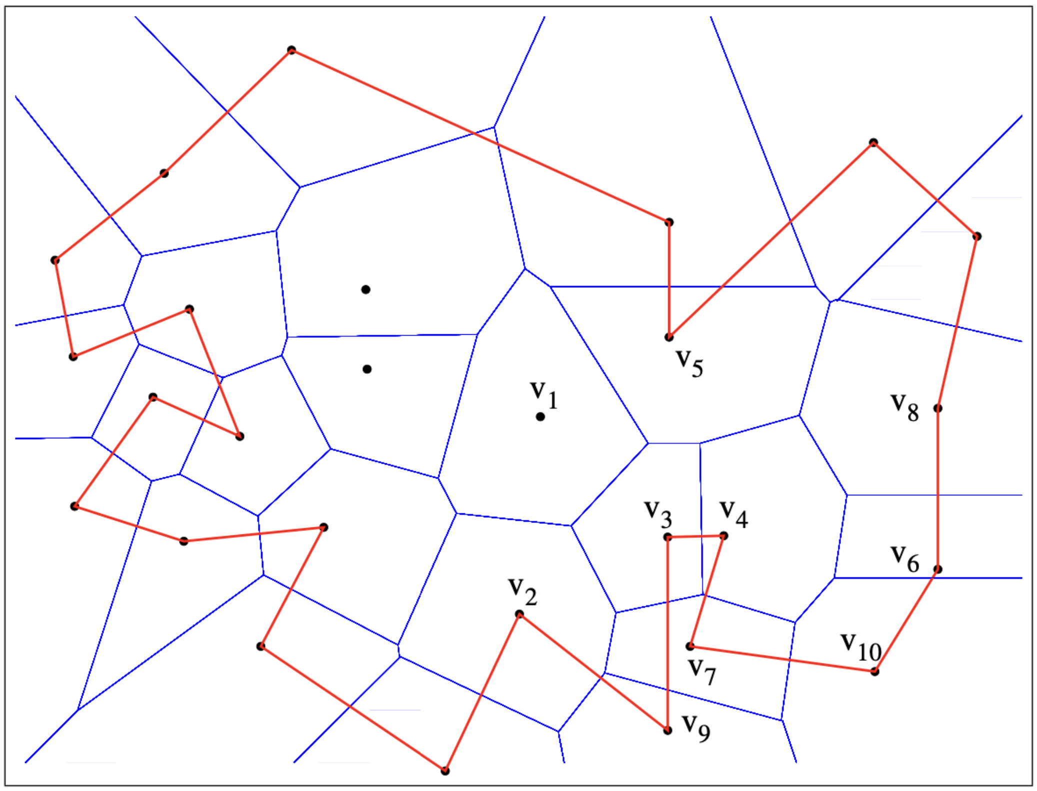
- **Create a <u>edge list</u> (empty at first)**
- **Create a <u>triangle list</u> (super triangle at first)**
    - **Check if any points is within any triangle**
    - **Add edges of triangle to edge list**
    - **Remove incorrect triangles**
- **Check each edge in edge list and remove all shared edges**
- **Form a triangle between point and each edge**
- **Add triangle to triangle list**

**Find each neighboring triangle's circumcenter and form an edge**


The Delaunay triangulation with all the circumcircles and their centers (in red).


Connecting the centers of the circumcircles produces the Voronoi diagram (in red).

Matlab has a built-in function.



- Voronoi diagram

- Find candidate internal nodes (~~polygonal~~ / internal nodes) -- v7 closest to <v2 , v9>
- If all Voronoi neighbor of a candidate polygonal node are polygonal nodes, we can examine all k − hop neighbors (say k = 3) of the candidate polygonal edge to the determine nearest internal node.