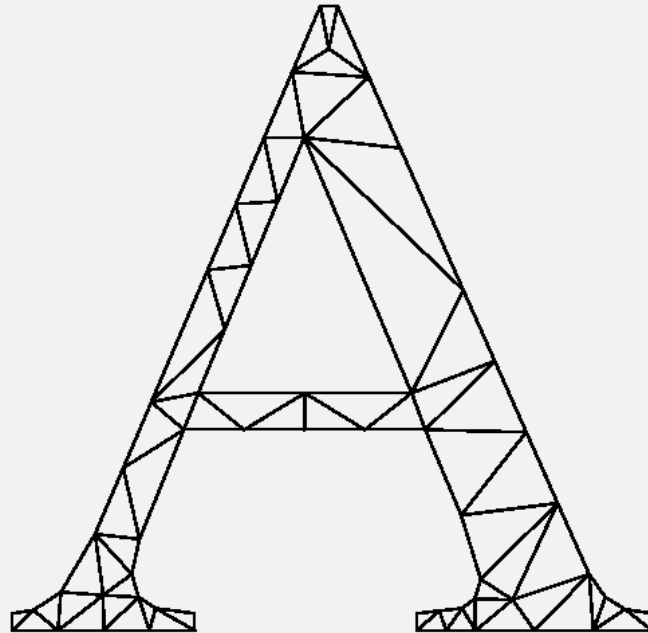


IFT 6113

A (VERY) SHORT INTRO TO COMPUTATIONAL GEOMETRY

tiny.cc/6113



Pic from <https://www.cs.cmu.edu/~quake/triangle.defs.html>

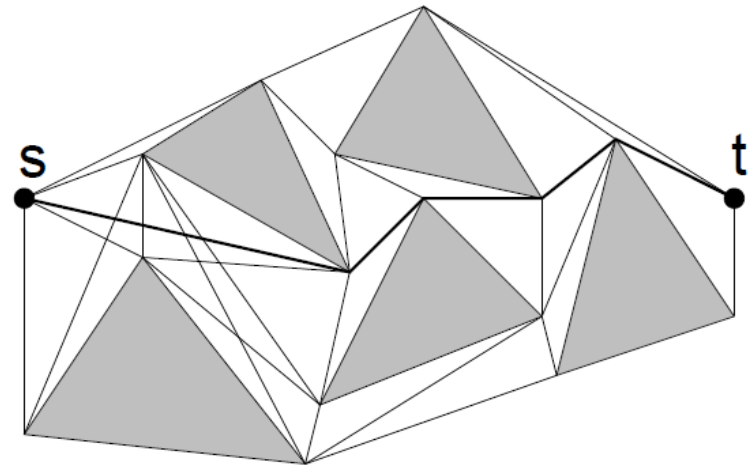
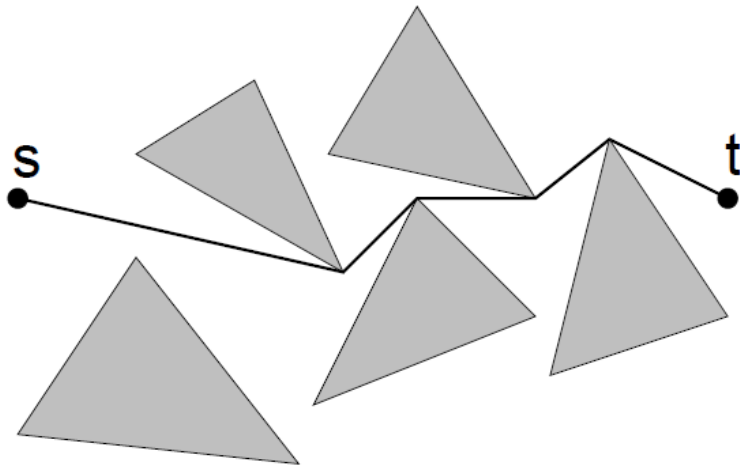
Mikhail Bessmeltsev

Today

- Intro
- Orientation and convex hulls
- Line segment intersection
- Polygons and triangulations
- Voronoi diagrams
- Delaunay triangulations

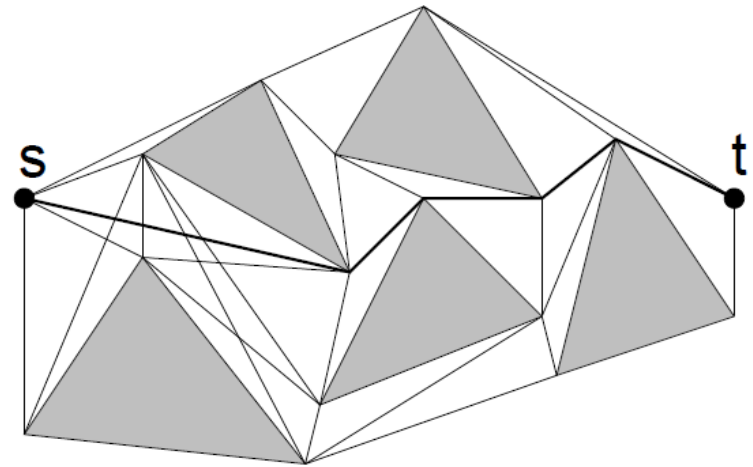
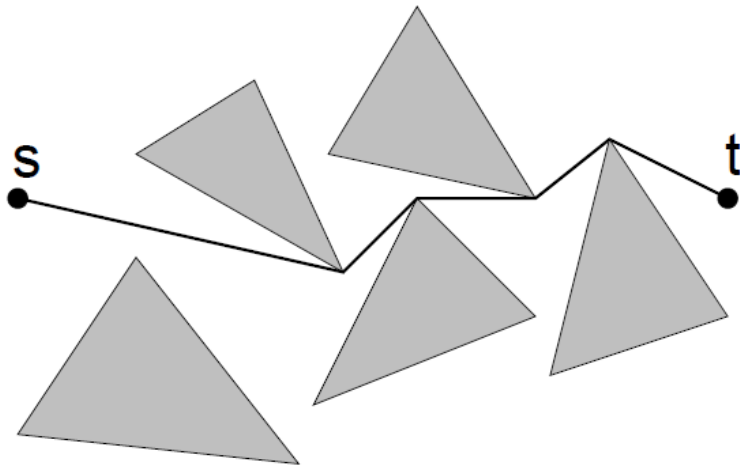
Focus: 2D algorithms

- Typical problem: shortest paths



Focus: 2D algorithms

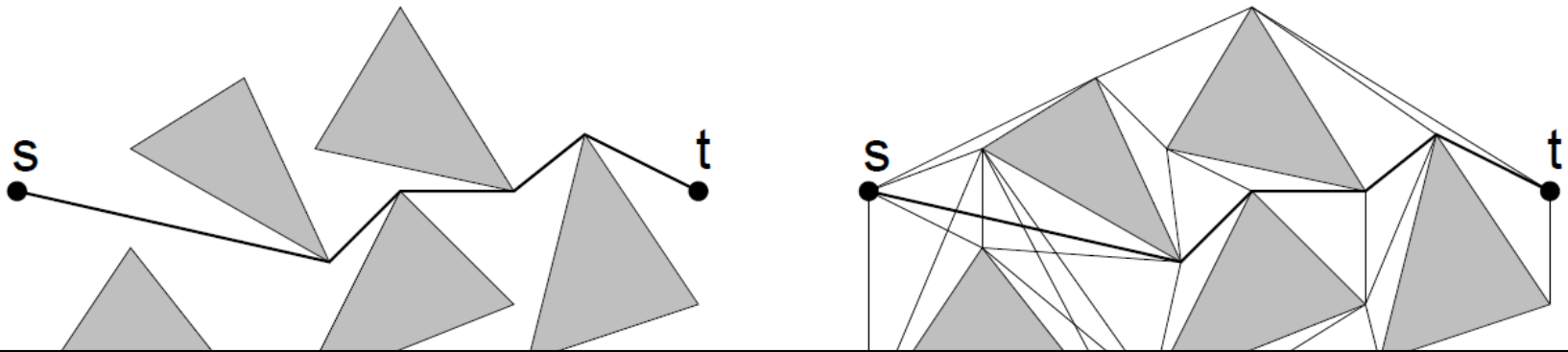
- Typical problem: shortest paths



→ graph → Dijkstra algorithm?

Focus: 2D algorithms

- Typical problem: shortest paths



Use geometric ideas!→
more efficient algorithms

graph → Dijkstra algorithm.

Complexity Analysis

- Big O notation
- Mostly worst case (*sometimes* average)
- Less attention to constants...

Main objects

Convex hulls

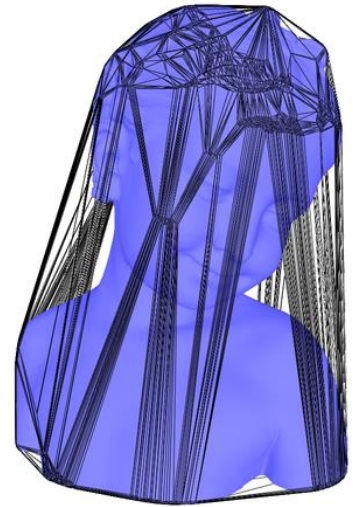
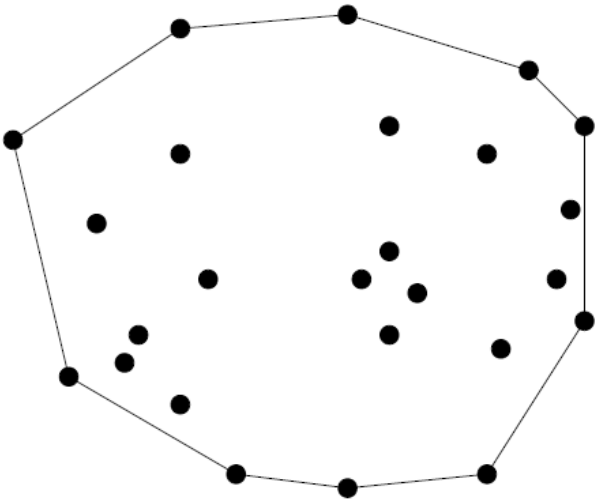
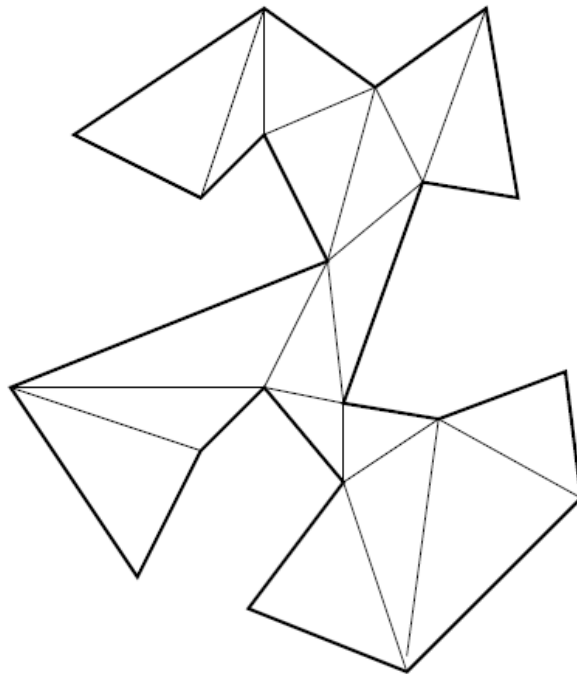


Image from CGAL manual

Main objects

Triangulations



Main objects

Line arrangements

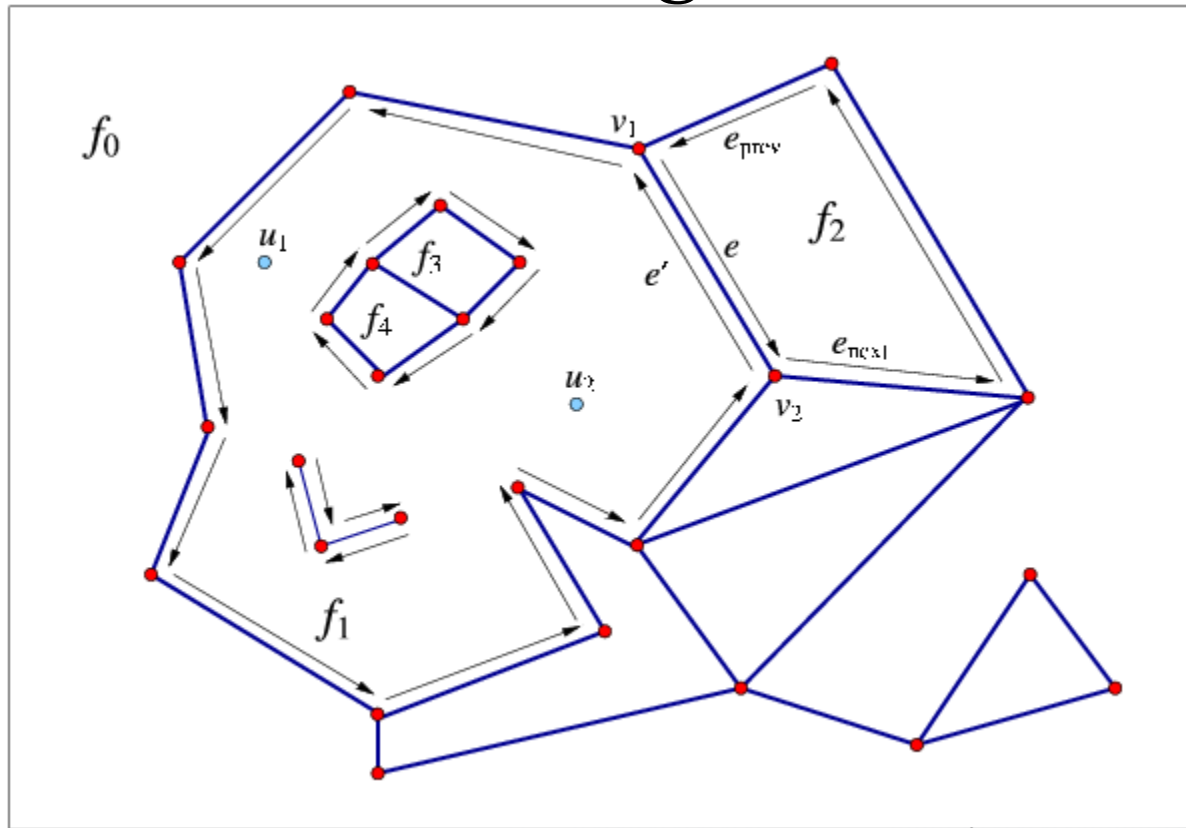
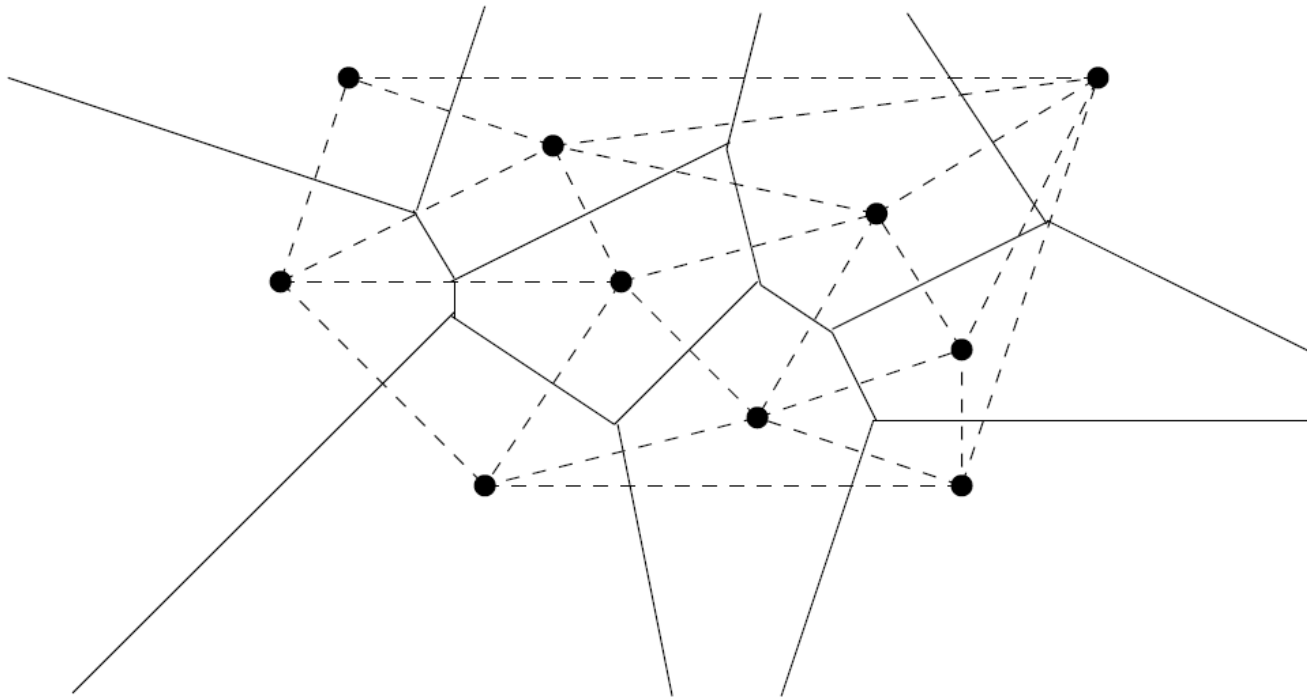


Image from CGAL manual

Main objects

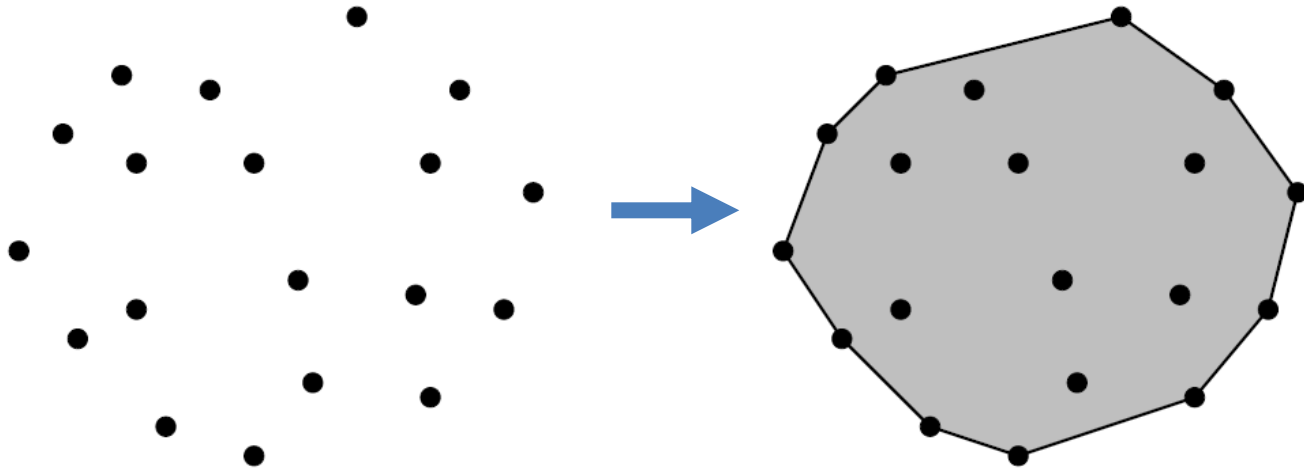
Delaunay triangulations and Voronoi diagrams



Main objects

Queries:

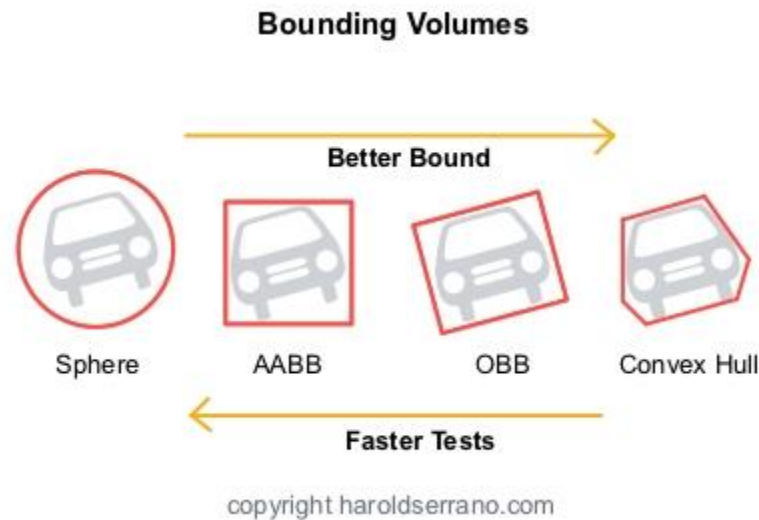
- Nearest neighbors?
- Range searches?



CONVEX HULLS

Why do we need convex hulls?

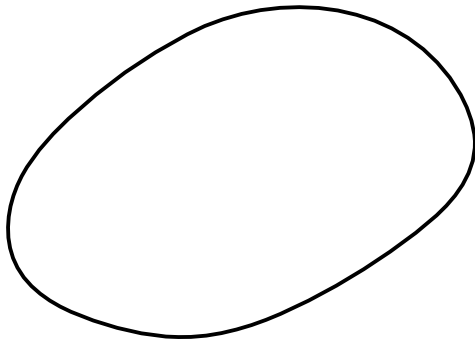
- Collision detection



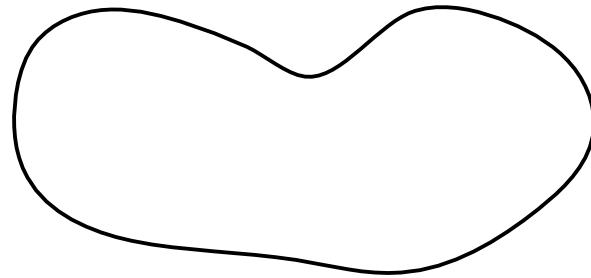
- Reconstructing convex objects from point clouds
- Farthest distance computation

Convex hulls

- A set S is convex, if
 - for any two points $p, q \in S$, the line segment $\overline{pq} \in S$



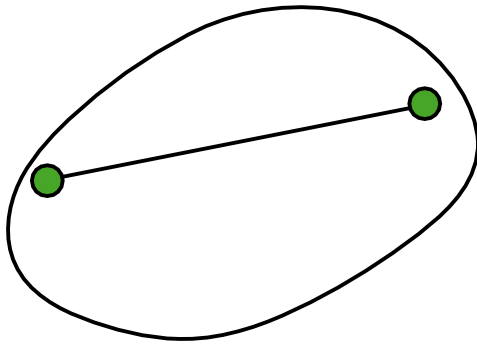
Convex



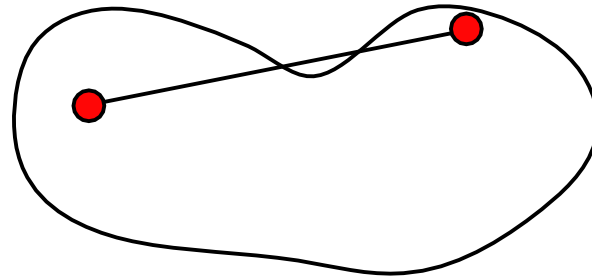
Not convex

Convex hulls

- A set S is convex, if
 - for any two points $p, q \in S$, the line segment $\overline{pq} \in S$



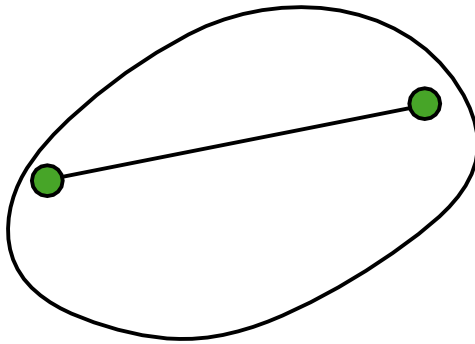
Convex



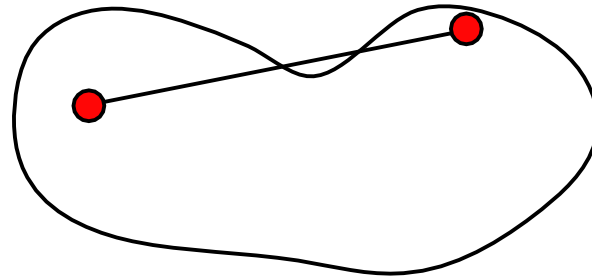
Not convex

Convex hulls

- A set S is convex, if
 - for any two points $p, q \in S$, the line segment $\overline{pq} \in S$



Convex



Not convex

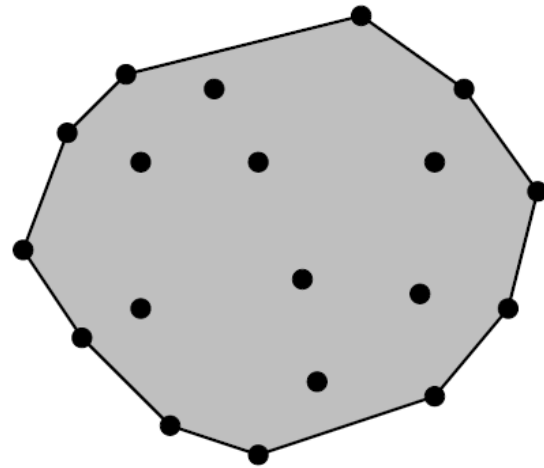
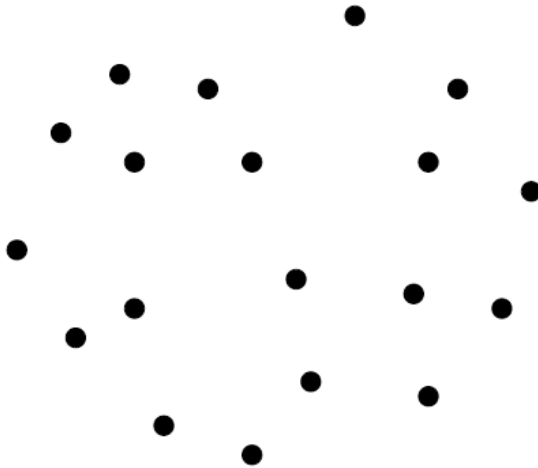
- *Convex hull* of a set of points S = the smallest convex set that contains S

CH: Representation

- A sequence of points!

Simplest algorithm

- Ideas?



Simplest algorithm

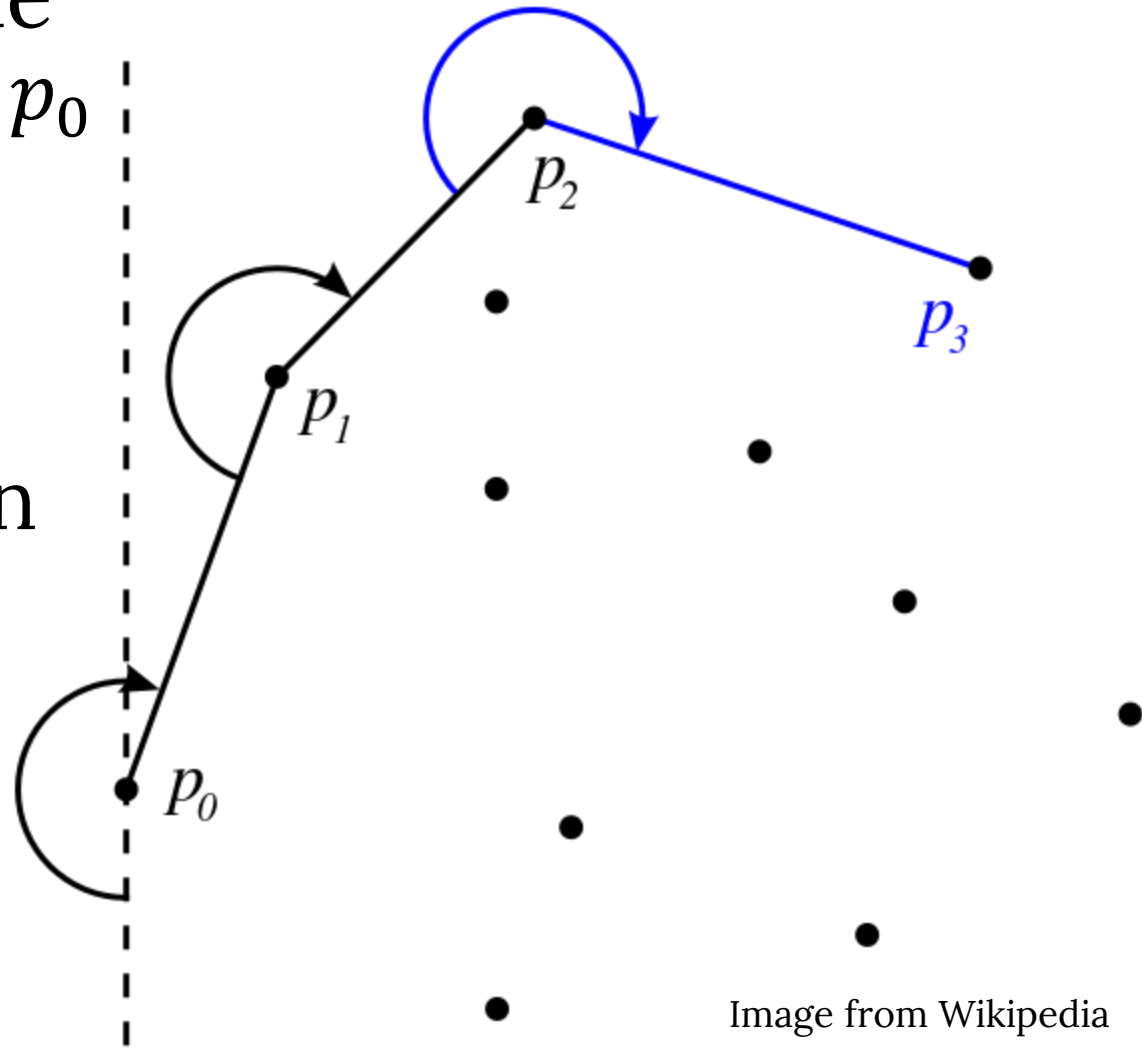
- Ideas?
- Try every possible $\overline{pq} \in S$, test if all the other points lie on one side
- Complexity?

Better: Gift Wrapping

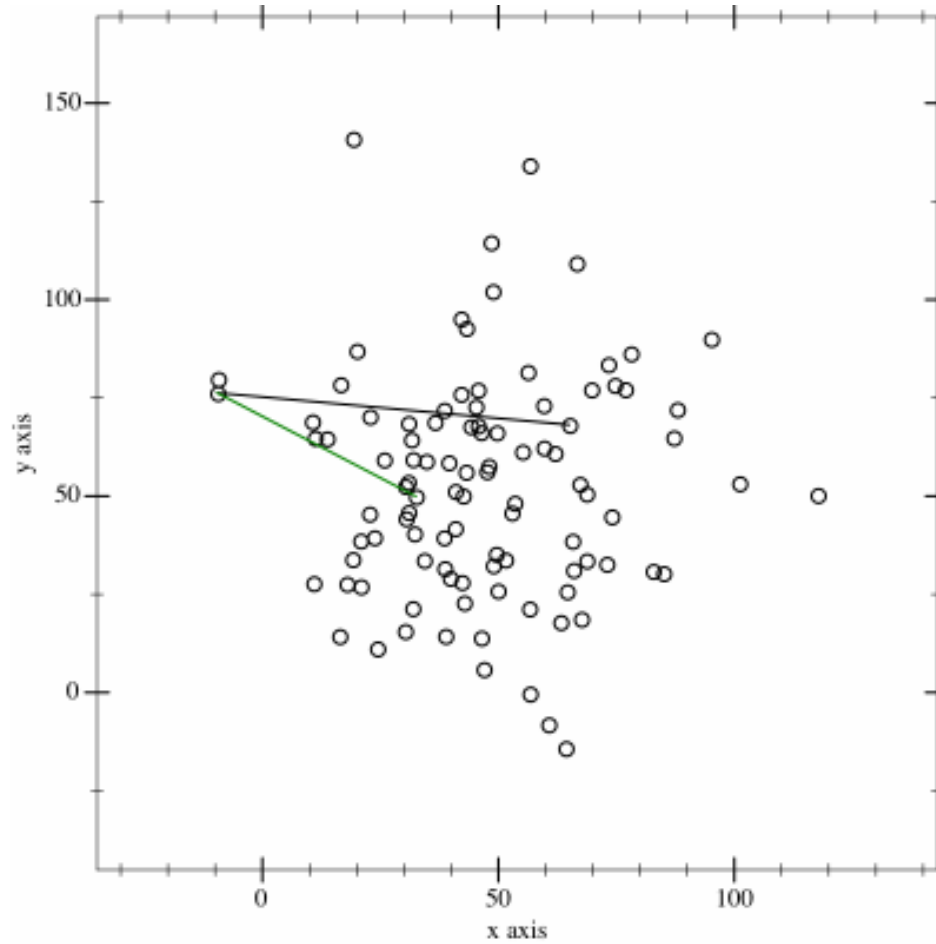
- Start with some extreme point p_0
 - (e.g. leftmost)
 - It belongs to CH
- Choose the next point so that $p_{i+1}p_i$ has all other points on the right
 - Sort by angle, choose minimum
 - $O(n)$
- Repeat

Gift Wrapping

- Start with some extreme point p_0
- Choose p_i , s.t. $p_{i+1}p_i$ has all other points on the right
- Repeat



Gift Wrapping



Output-sensitive complexity

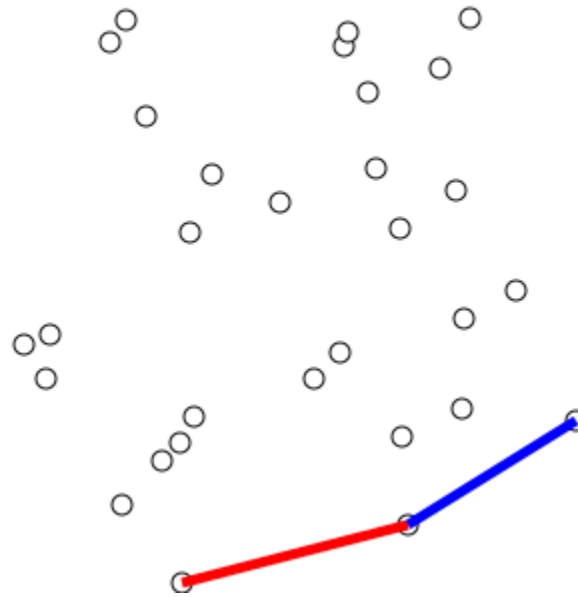
$$O(nh)$$

h - number of vertices on the CH

Graham's Scan

- Lower hull:
- Store CH vertices in a stack: (\dots, H_2, H_1)
- Sort points by angle
- Take new point p , check if the (p, H_1, H_2) is counterclockwise
 - Yes \Rightarrow push!
 - No \Rightarrow pop!

Graham's Scan



Graham's Scan

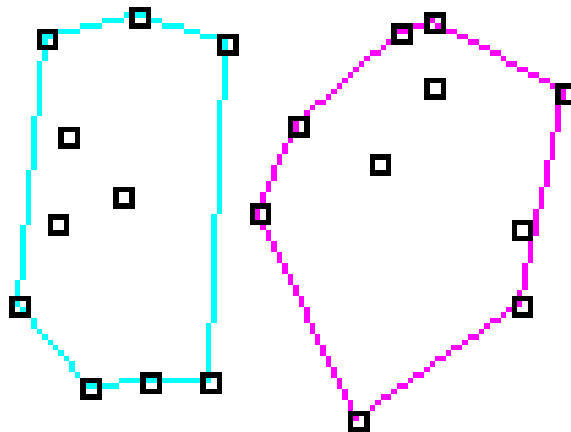
- Complexity?

Divide-and-Conquer CH

- Sort by x
- Split by the median point
- Recursively find CH for left and right parts
- Merge

Divide-and-Conquer CH

- Sort by x
- Split by the median point
- Recursively find CH for left and right parts
- Merge



Divide-and-Conquer CH

- Sort by x: $O(n \log n)$
- Split by the median point
- Recursively find CH for left and right parts
- Merge
 - Find common tangent line, $O(n)$

Today

- ✓ Intro
- ✓ Orientation and convex hulls
- **Line segment intersection**
- Polygons and triangulations
- Halfplane intersection and LP
- Voronoi diagrams
- Delaunay triangulations
- Point location

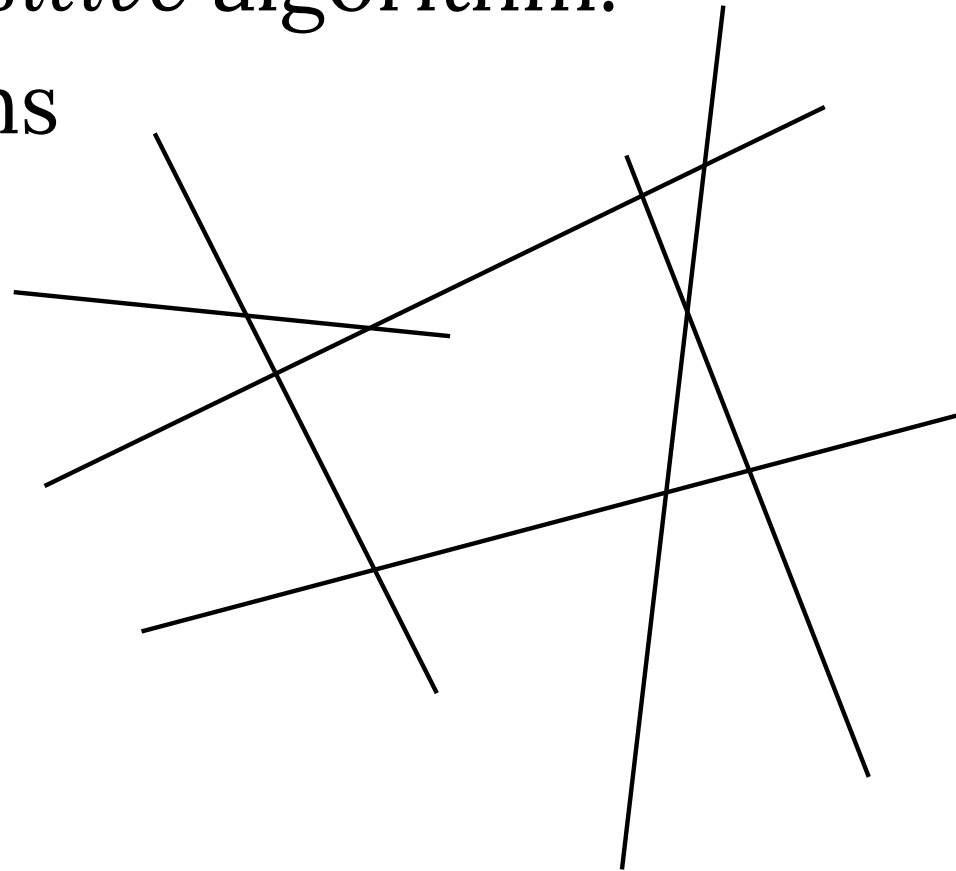
Credit: most of images and ideas from Dave Mount's lecture notes

<http://www.cs.umd.edu/~mount/754/Lects/754lects.pdf>

And David Kirkpatrick's lectures

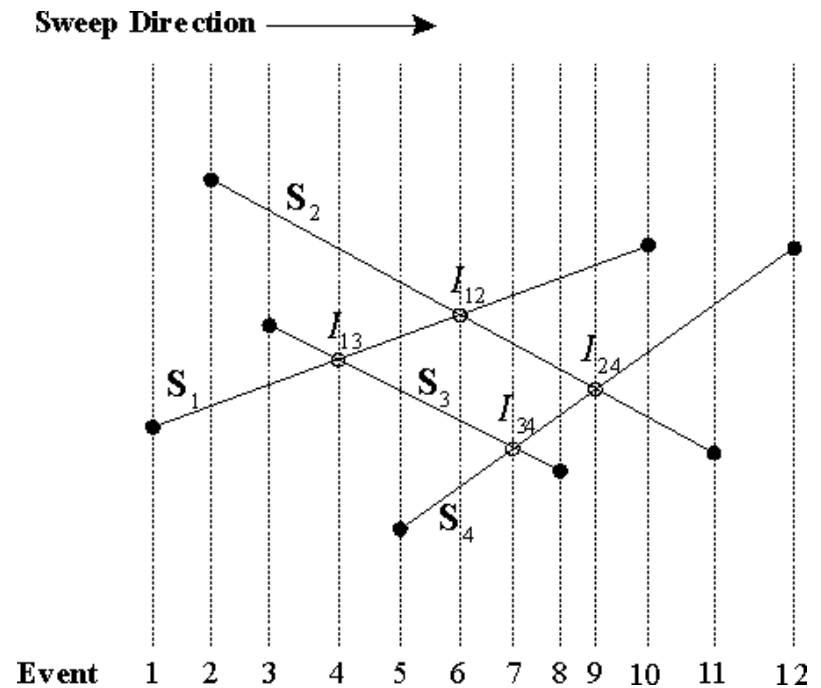
Line Segment Intersection

- Max # of intersections: $O(n^2)$
- Need an *output sensitive* algorithm!
- I - # of intersections



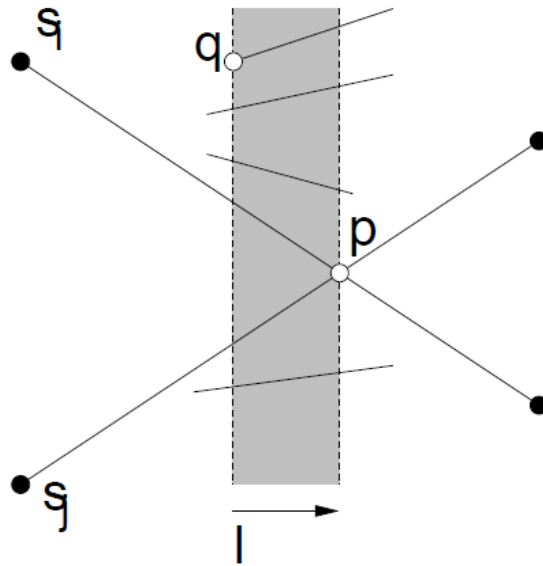
Plane Sweep

- Sweep virtual line
- Stop at events
 - segment started/finished
 - intersection



Plane Sweep

- Presort the endpoints
- How to detect next intersection?
 - If two segments are adjacent along the sweep line,
 - Check if they intersect to the right of the sweep line



Today

- ✓ Intro
- ✓ Orientation and convex hulls
- ✓ Line segment intersection
- **Polygons and triangulations**
- Voronoi diagrams
- Delaunay triangulations

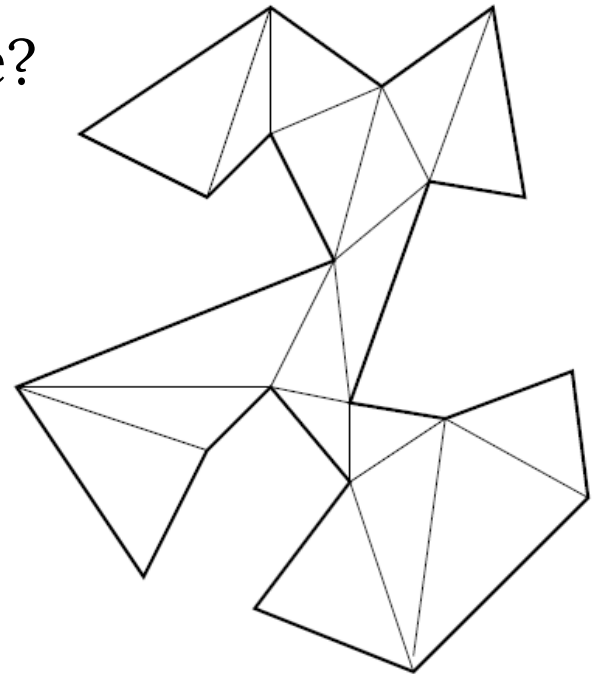
Credit: most of images and ideas from Dave Mount's lecture notes

<http://www.cs.umd.edu/~mount/754/Lects/754lects.pdf>

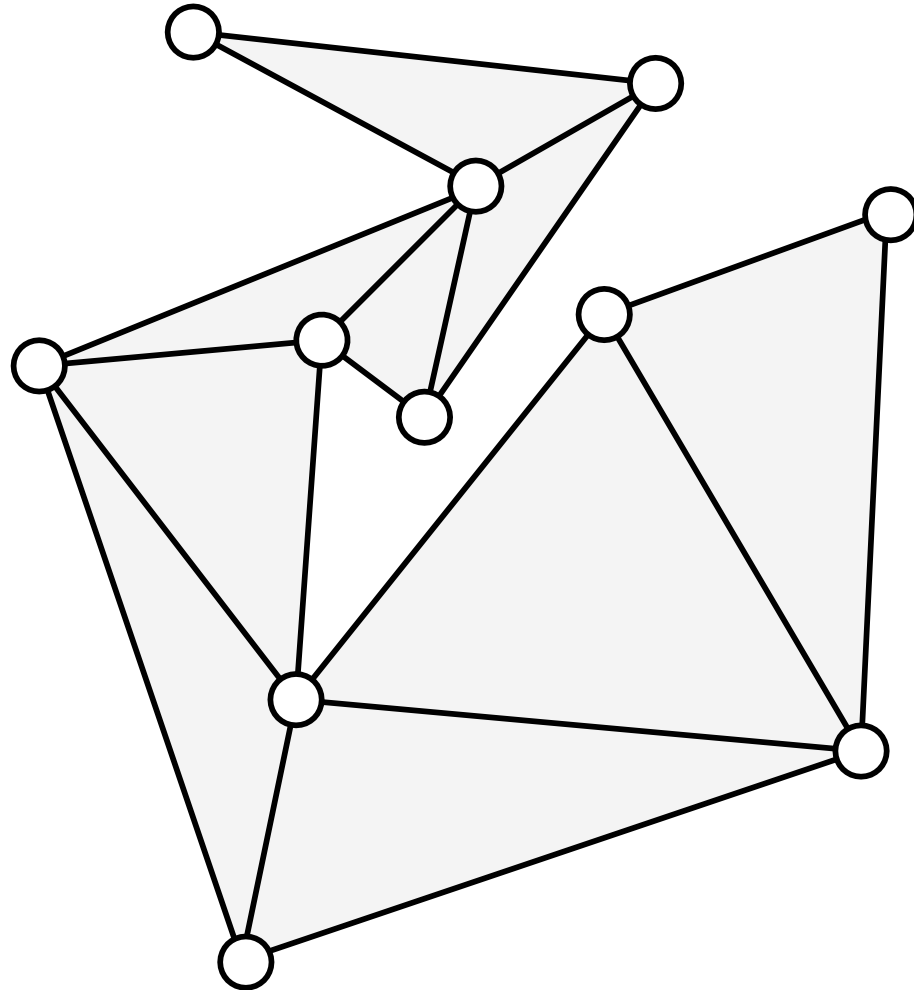
And David Kirkpatrick's lectures

Polygons and triangulations

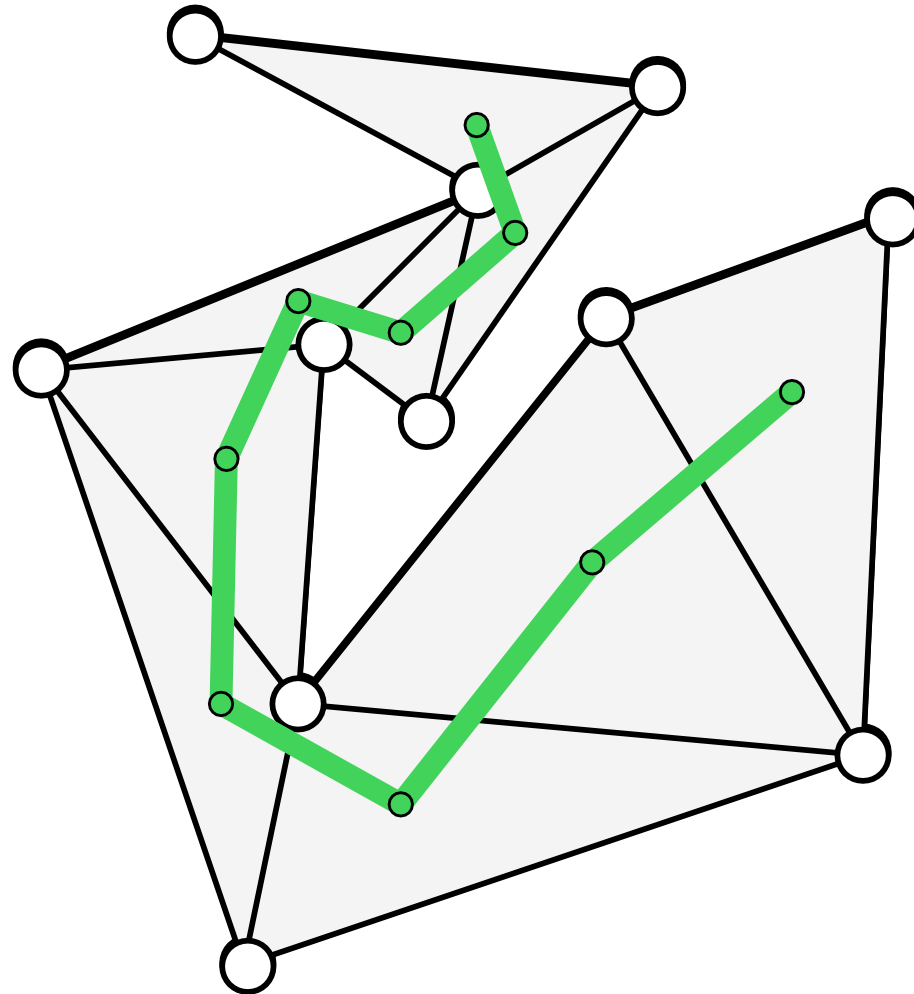
- Why?
 - Decomposition of complex shapes
 - How to compute area of a polygon?
 - Art gallery problem
 - How many cameras and where?



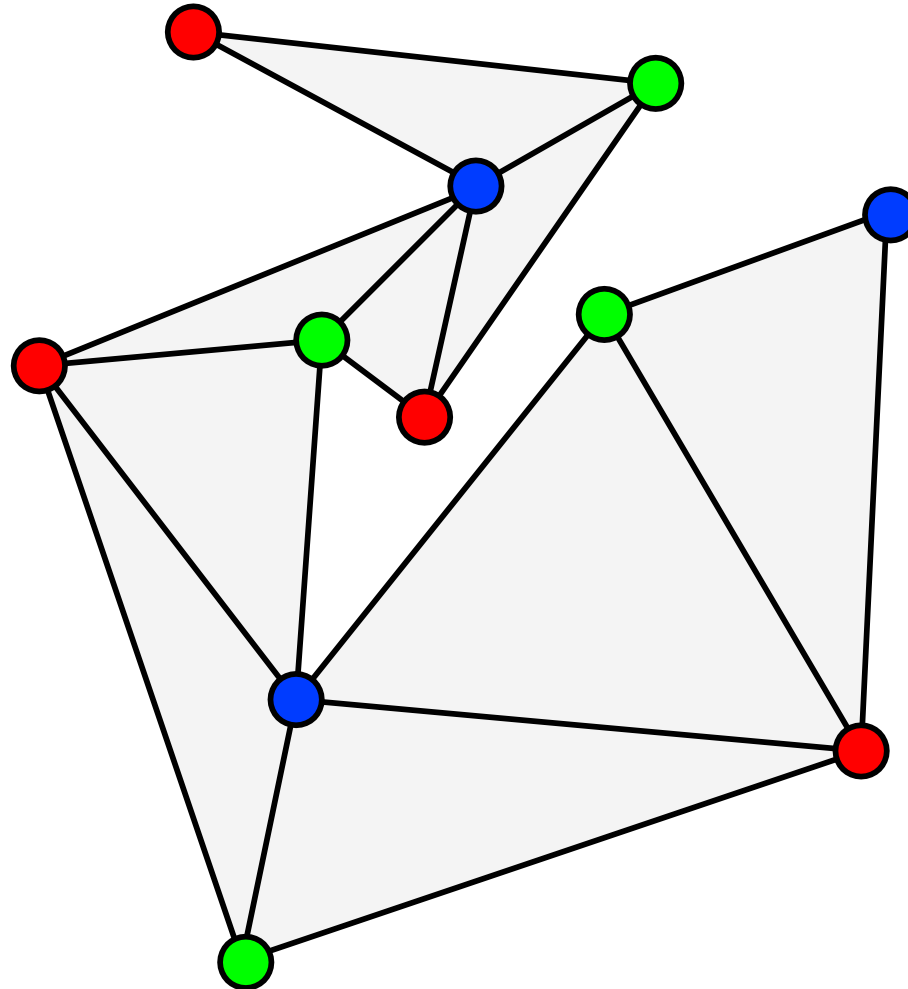
Art Gallery Problem



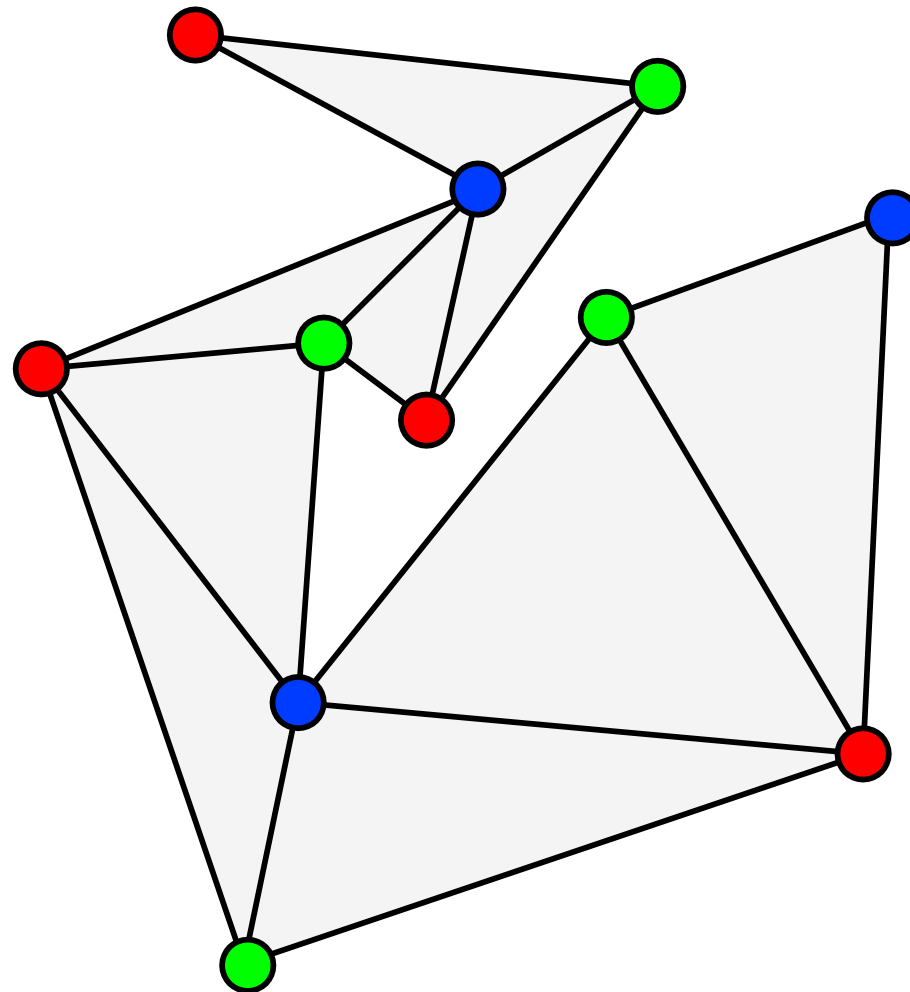
Art Gallery Problem



Art Gallery Problem



Art Gallery Problem

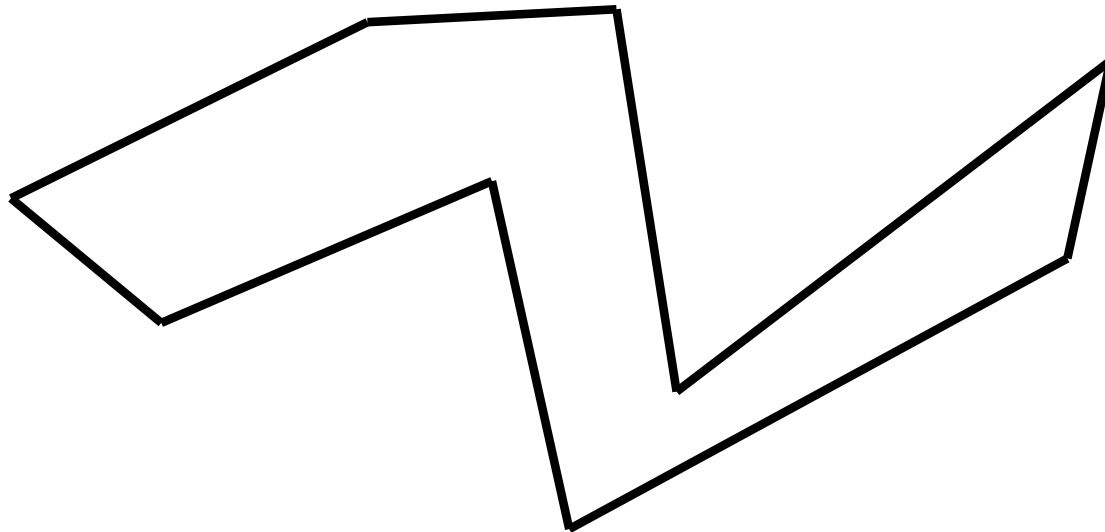


$\lfloor n/3 \rfloor$ cameras are
enough!

Is it optimal?

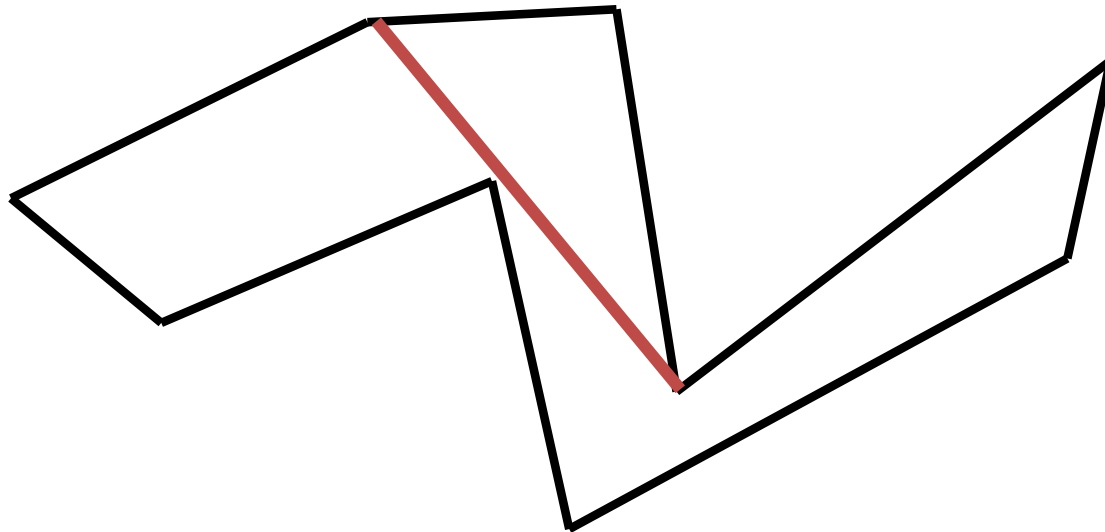
Triangulations

- How to find *a* triangulation?



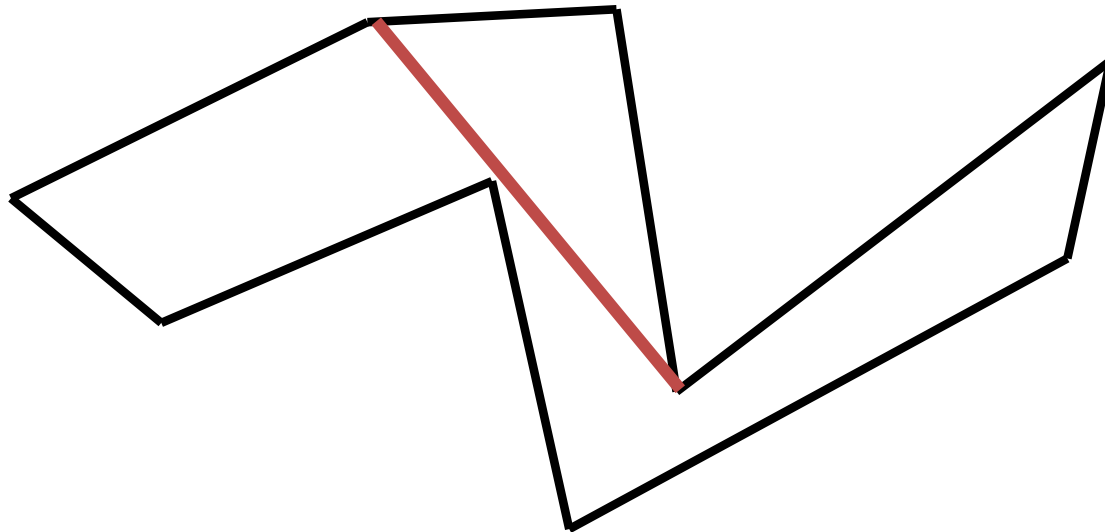
Triangulations

Diagonal: line segment connecting two vertices completely within the polygon



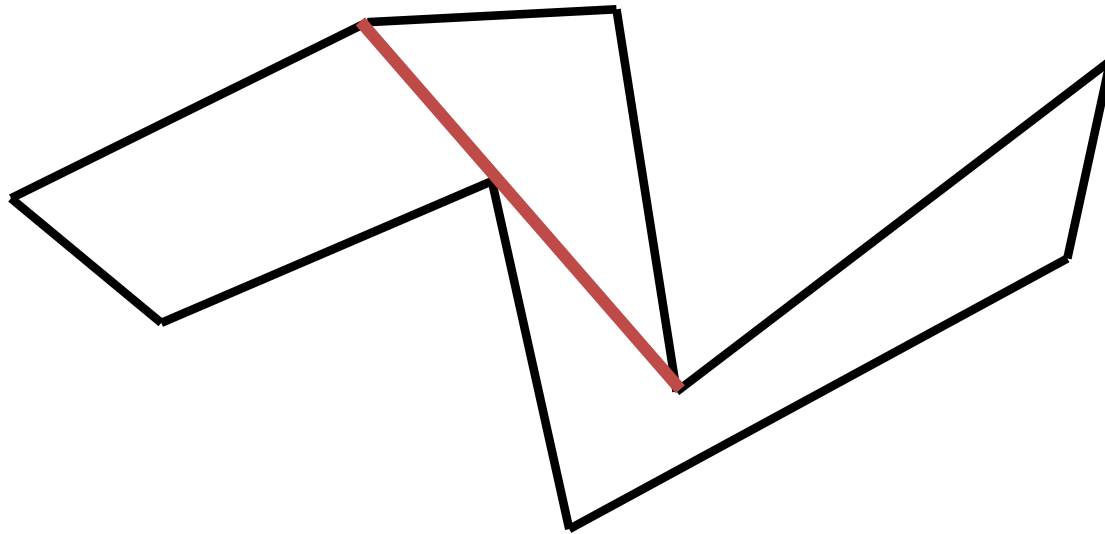
Triangulations

Diagonal: line segment connecting two vertices completely within the polygon

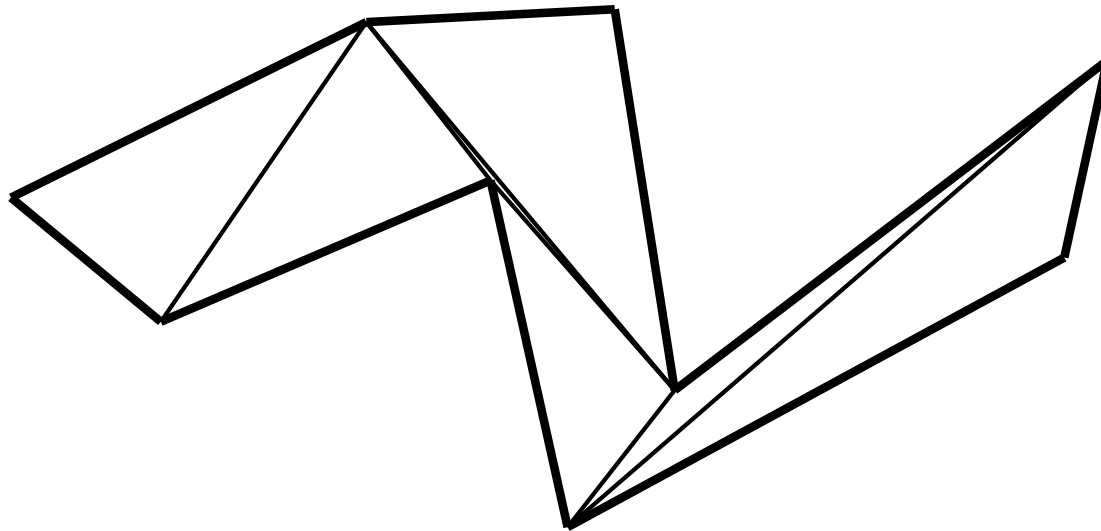


Every polygon has at least 1 diagonal!

Triangulations

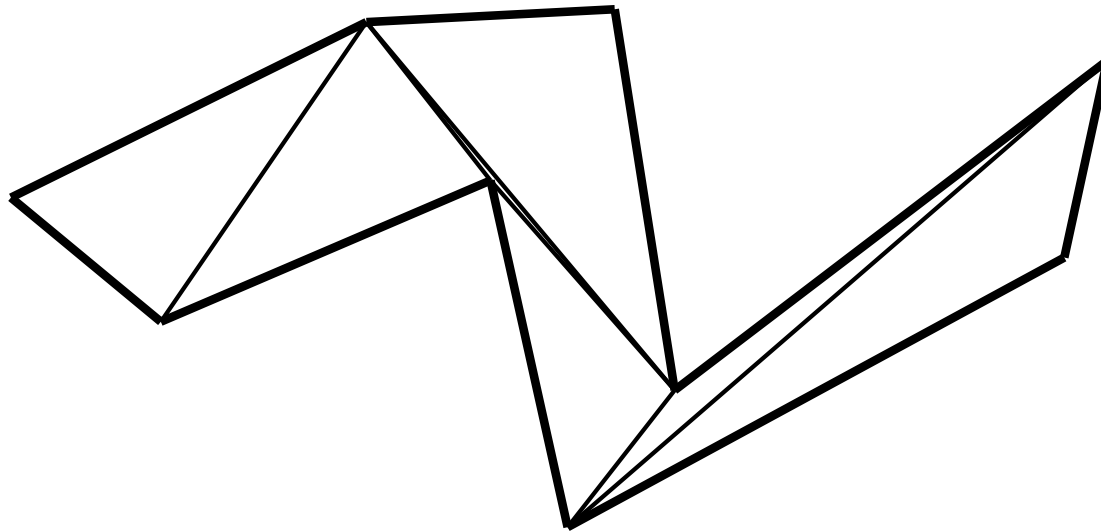


Triangulations



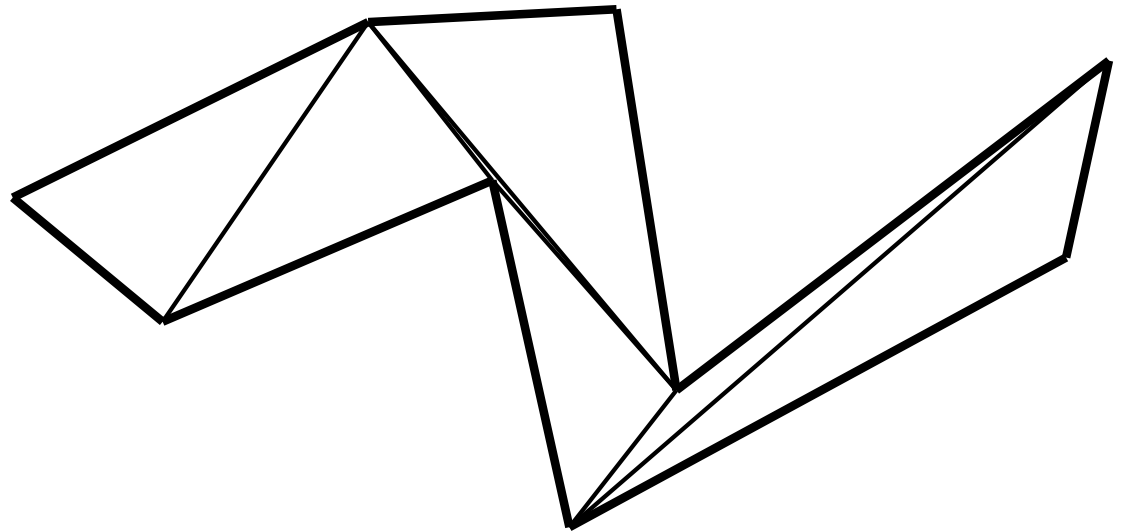
Triangulations

- Algorithm: add diagonal, repeat for the two new polygons



Triangulations

- Algorithm: add diagonal, repeat for the two new polygons
 - Bottleneck?
 - Complexity?



A better algorithm

- $O(n \log n)$
- Input: cyclic list of vertices
 - Presorted!

TODO list:

1. Learn how to triangulate monotone polygons
2. Subdivide arbitrary polygon into monotone ones
3. Profit!

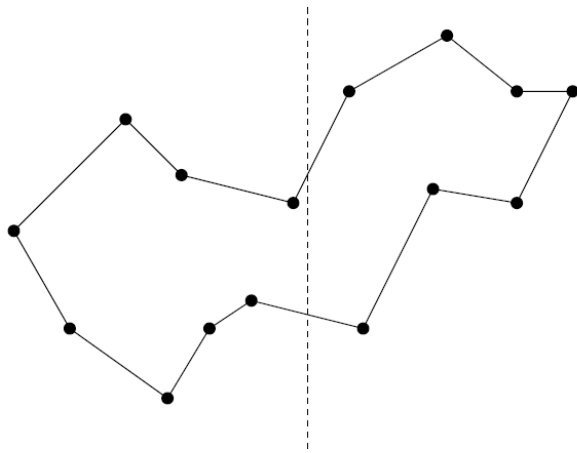
Monotone polygons

Polygonal chain is *x-monotone* \Leftrightarrow

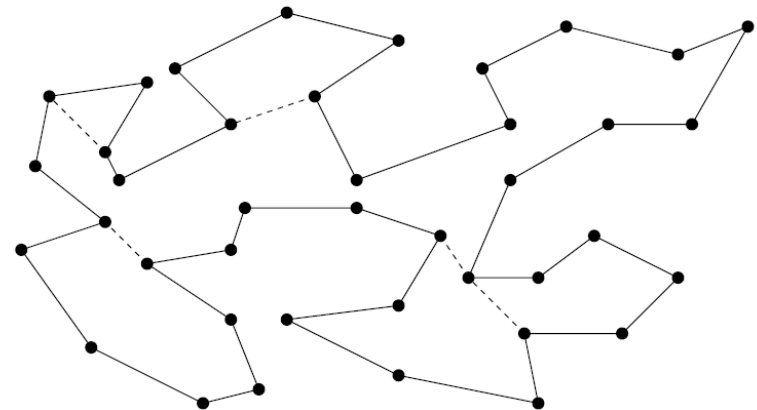
every vertical line intersects chain in
at most 1 point

Polygon is *x-monotone* \Leftrightarrow

can be split into two *x-monotone* chains



x-monotone polygon

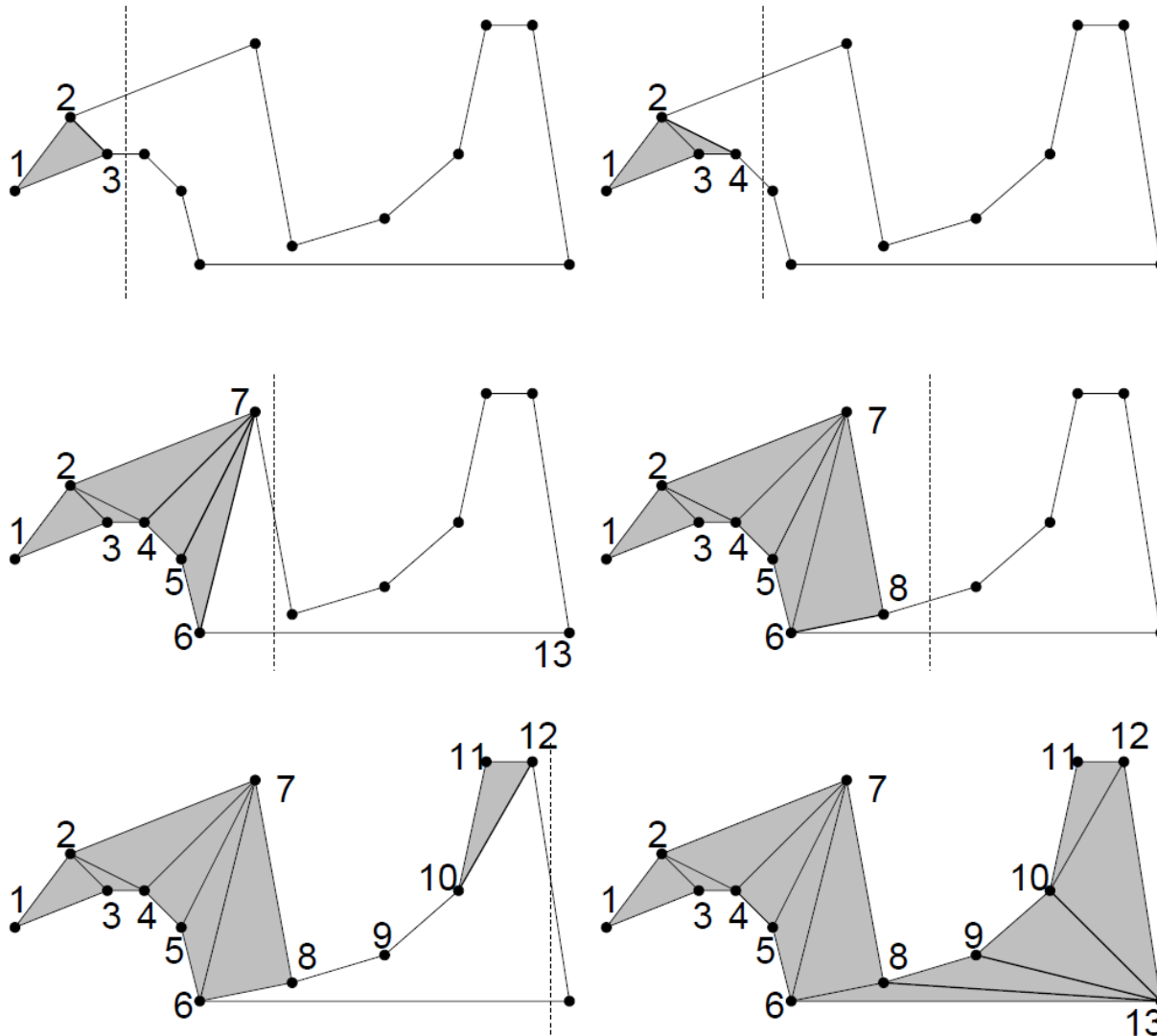


Monotone decomposition

Triangulating monotone polygons

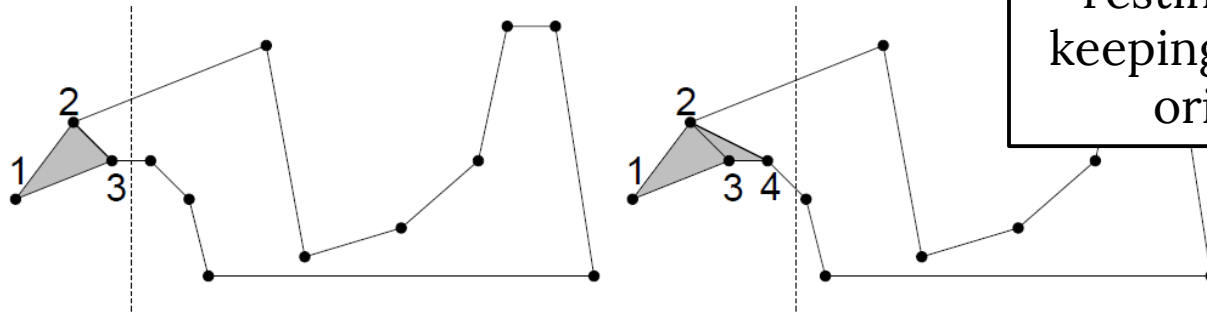
- Sort vertices left to right
 - Doesn't require actual sorting (why?)
 - Requires $O(n)$ time
- Line sweep
 - Triangulate everything to the left of the sweep line
 - Discard the triangulated part
 - Testing diagonals is now constant-time
 - Why?

Triangulating monotone polygons

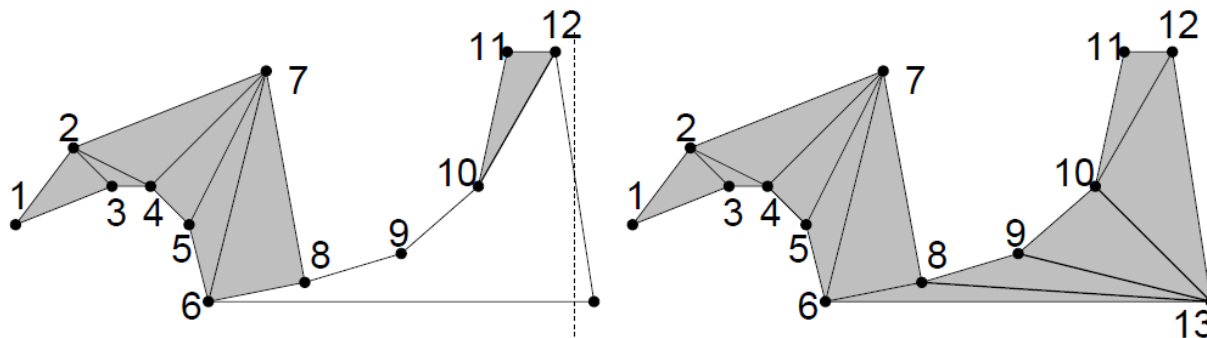
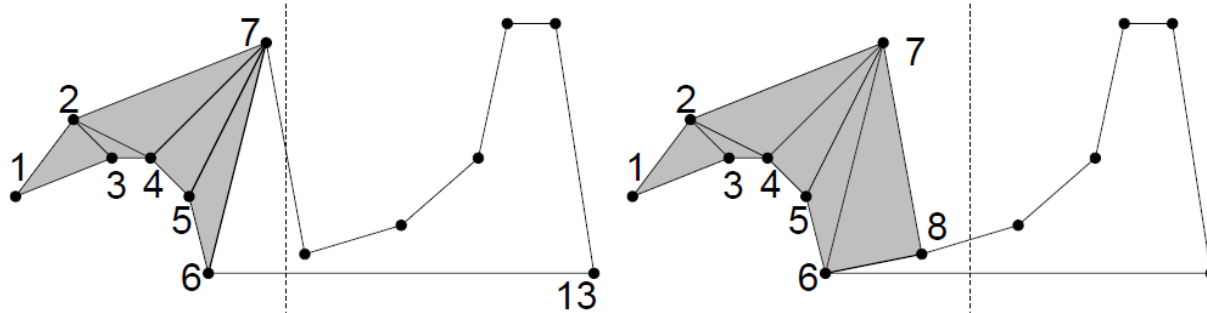


$O(n)$

Triangulating monotone polygons



Testing diagonals =
keeping track of edge
orientations



$O(n)$

Today

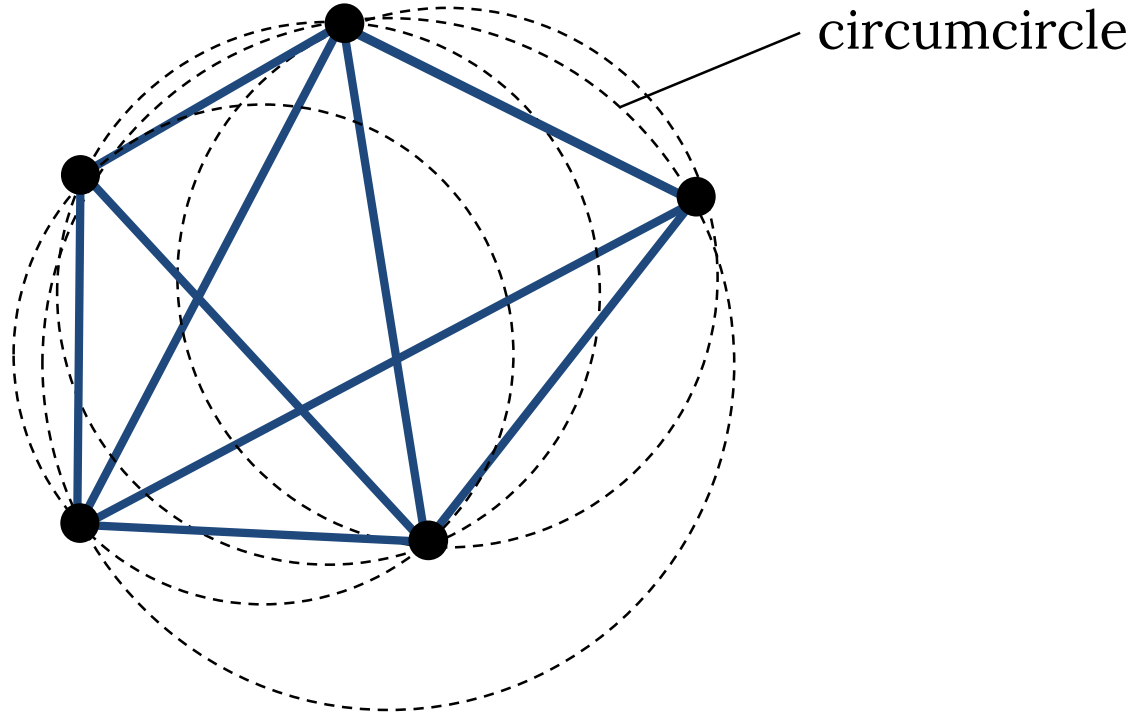
- ✓ Intro
- ✓ Orientation and convex hulls
- ✓ Line segment intersection
- ✓ Polygons and triangulations
- **Voronoi diagrams**
- **Delaunay triangulations**

Credit: most of images and ideas from Dave Mount's lecture notes

<http://www.cs.umd.edu/~mount/754/Lects/754lects.pdf>

And David Kirkpatrick's lectures

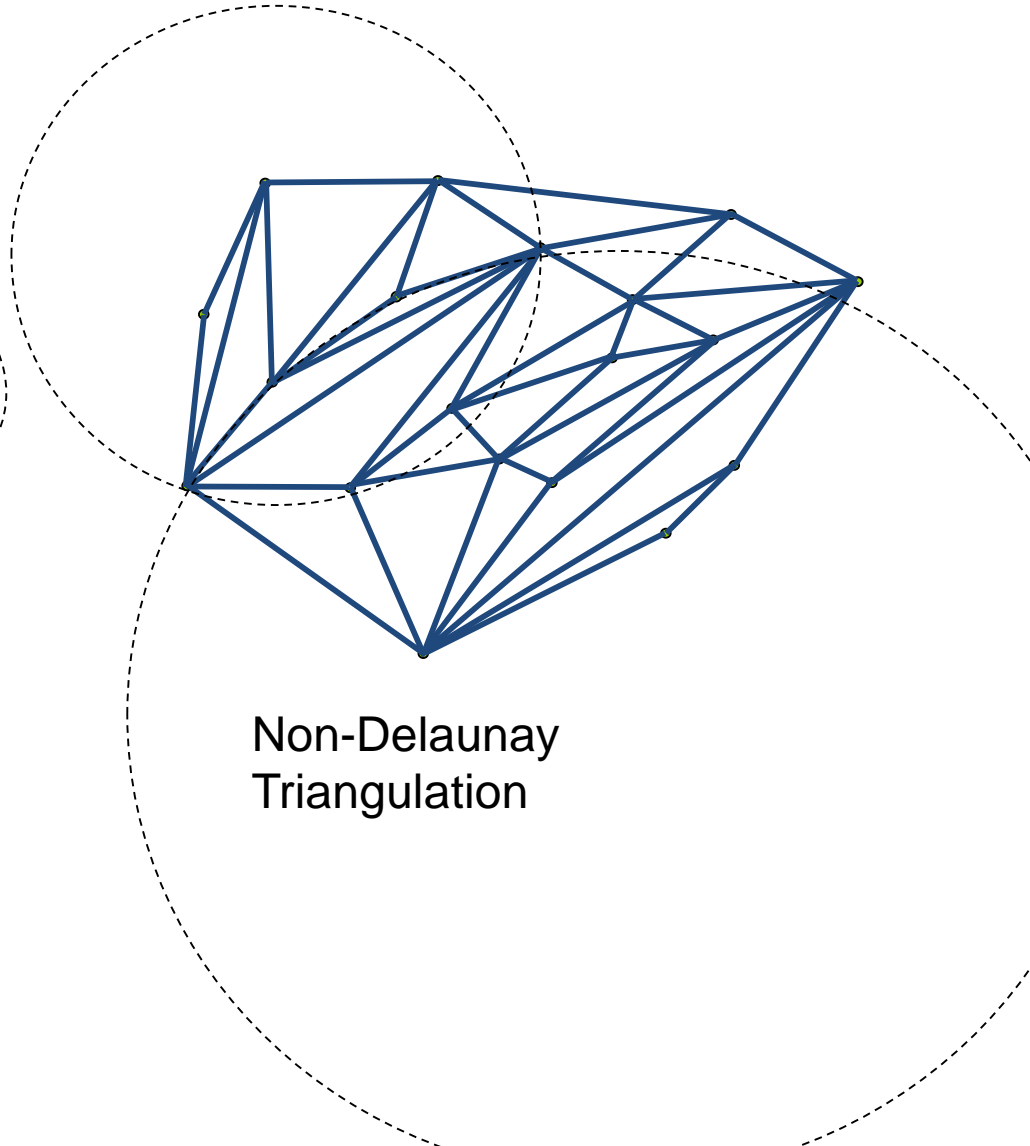
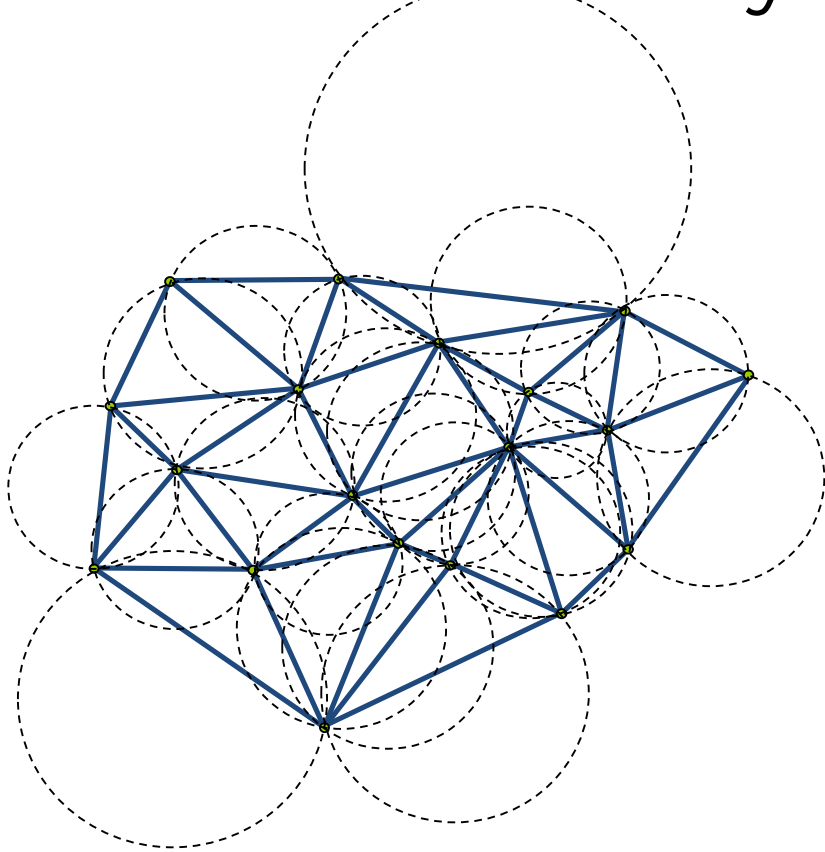
Triangulation Quality: Delaunay Criterion



Empty Circle Property:

No other vertex is contained within the circumcircle of any triangle

Delaunay Triangulation



Delaunay Triangulation

- Obeys empty-circle property
- Exists for any set of vertices
- Is **unique** (up to degenerate cases)
- Proven to provide best triangles in terms of quality for given vertex positions
- To test – enough to check pairs of triangles sharing common edge

Edge flipping

Start with any triangulation

1. Find a non-Delaunay edge
2. Flip edge
3. Repeat

Converges to Delaunay

Slow: $O(n^2)$

Vertex Insertion

Start with Delaunay mesh covering domain

- Typically 2 triangle bounding box

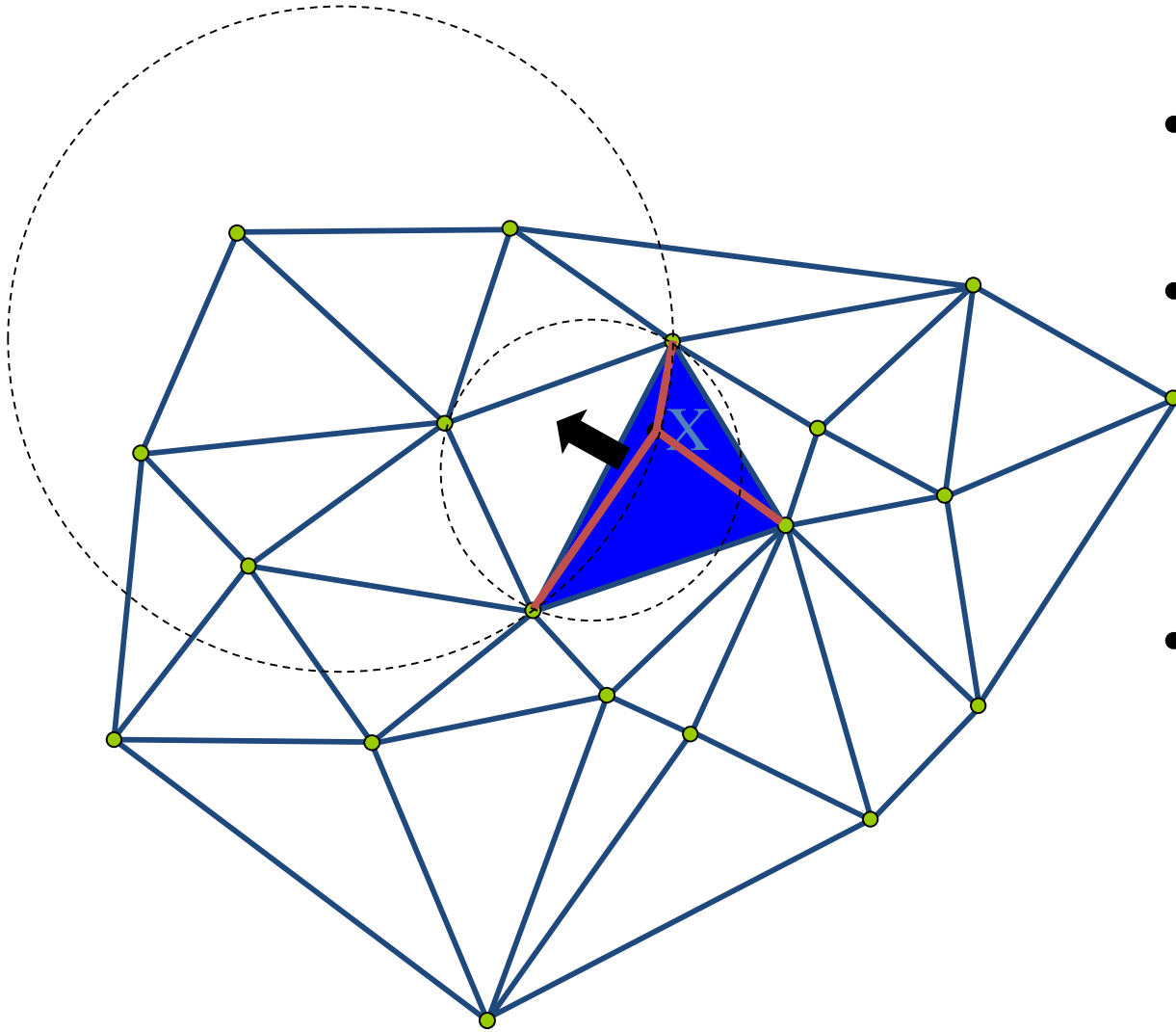
Insert one vertex at a time

- Add vertex to mesh
- Flip edges locally to maintain Delaunay property

Boundary recovery

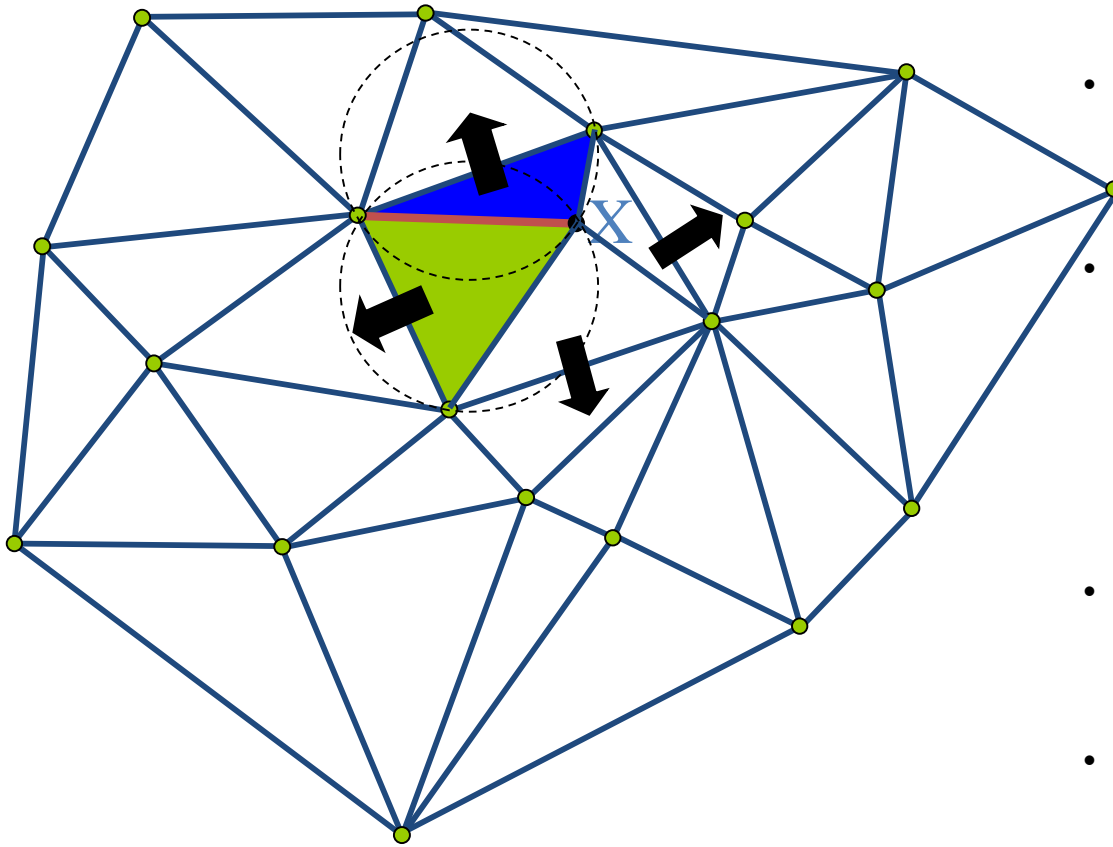
Expected time $O(n \log n)$

Vertex Insertion



- Locate triangle containing X
- Subdivide triangle
- Recursively check adjoining triangles to ensure empty-circle property

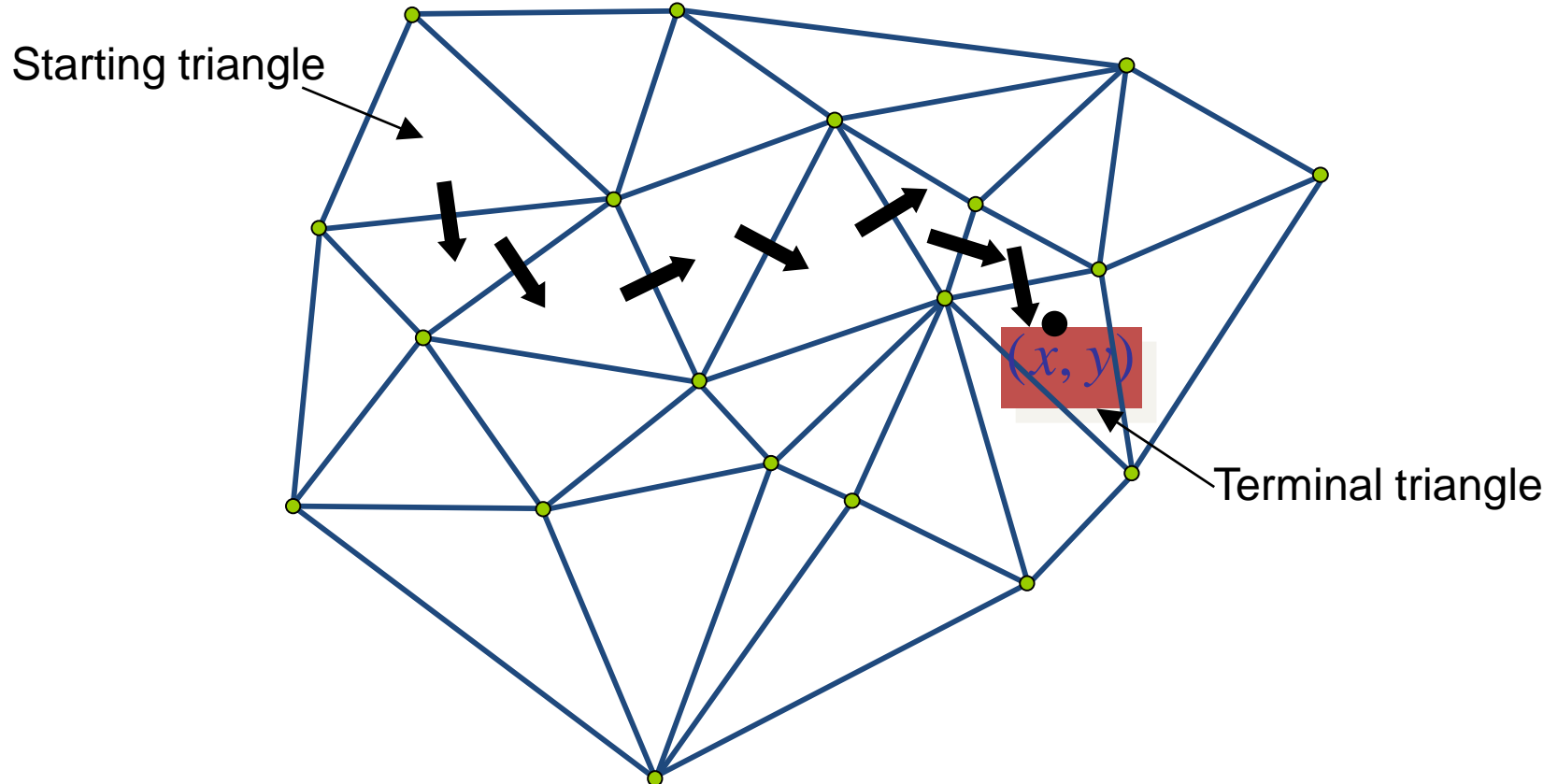
Vertex Insertion



- Locate triangle containing X
- Subdivide triangle
- Recursively check adjoining triangles to ensure empty-circle property
- Swap diagonal if needed
- Typically very small number of swaps

Find Triangle - Efficiency

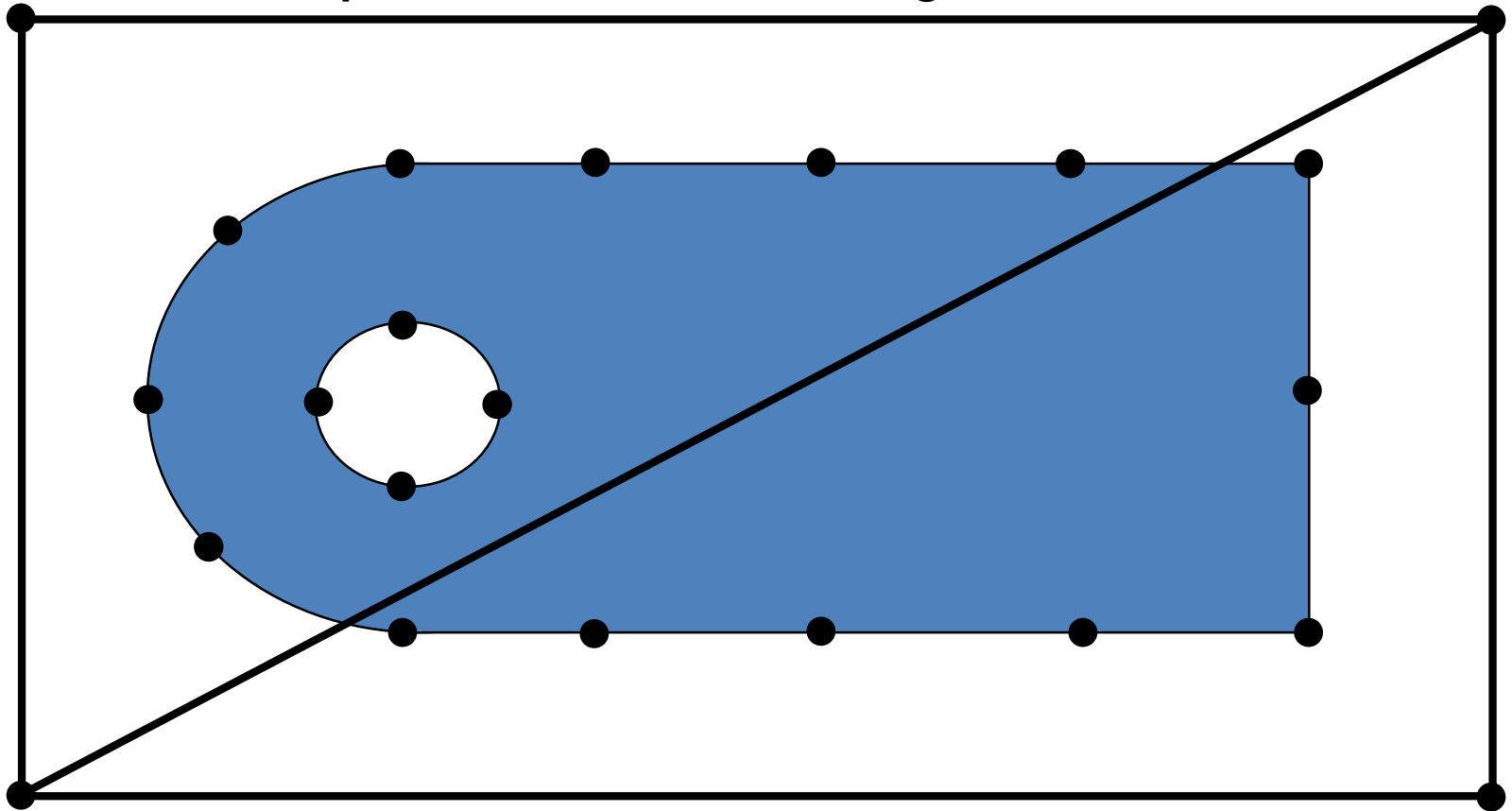
- Use Barycentric Coordinates
 - Test inside triangle
 - If outside - outside which edge?



Vertex Insertion

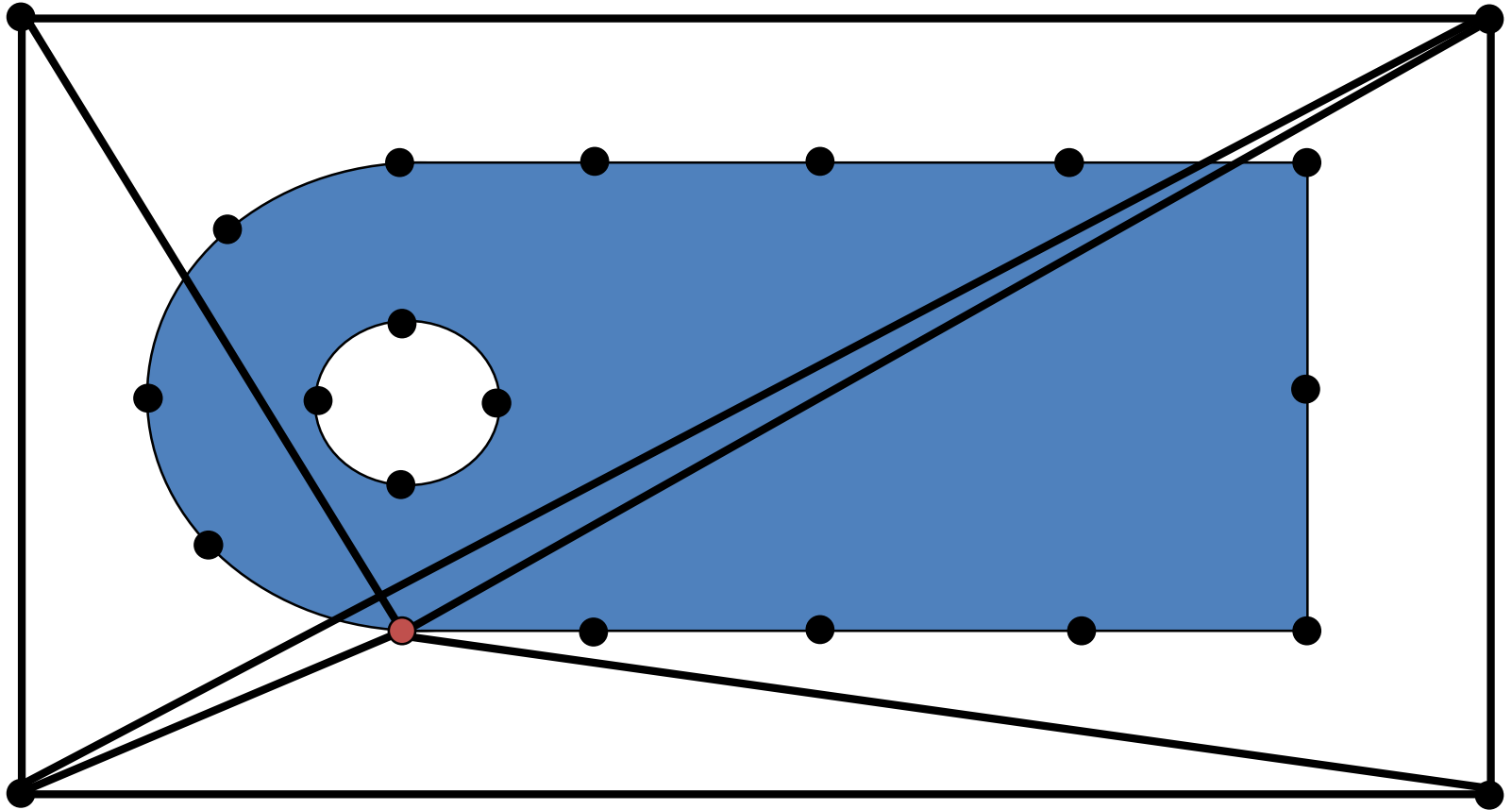
- Start with Delaunay mesh covering domain
 - Typically 2 triangle bounding box
- Insert one vertex at a time
 - Add vertex to mesh (locate triangle to split)
 - Flip edges (locally) to preserve Delaunay property
- Boundary recovery

Example: Boundary Insertion



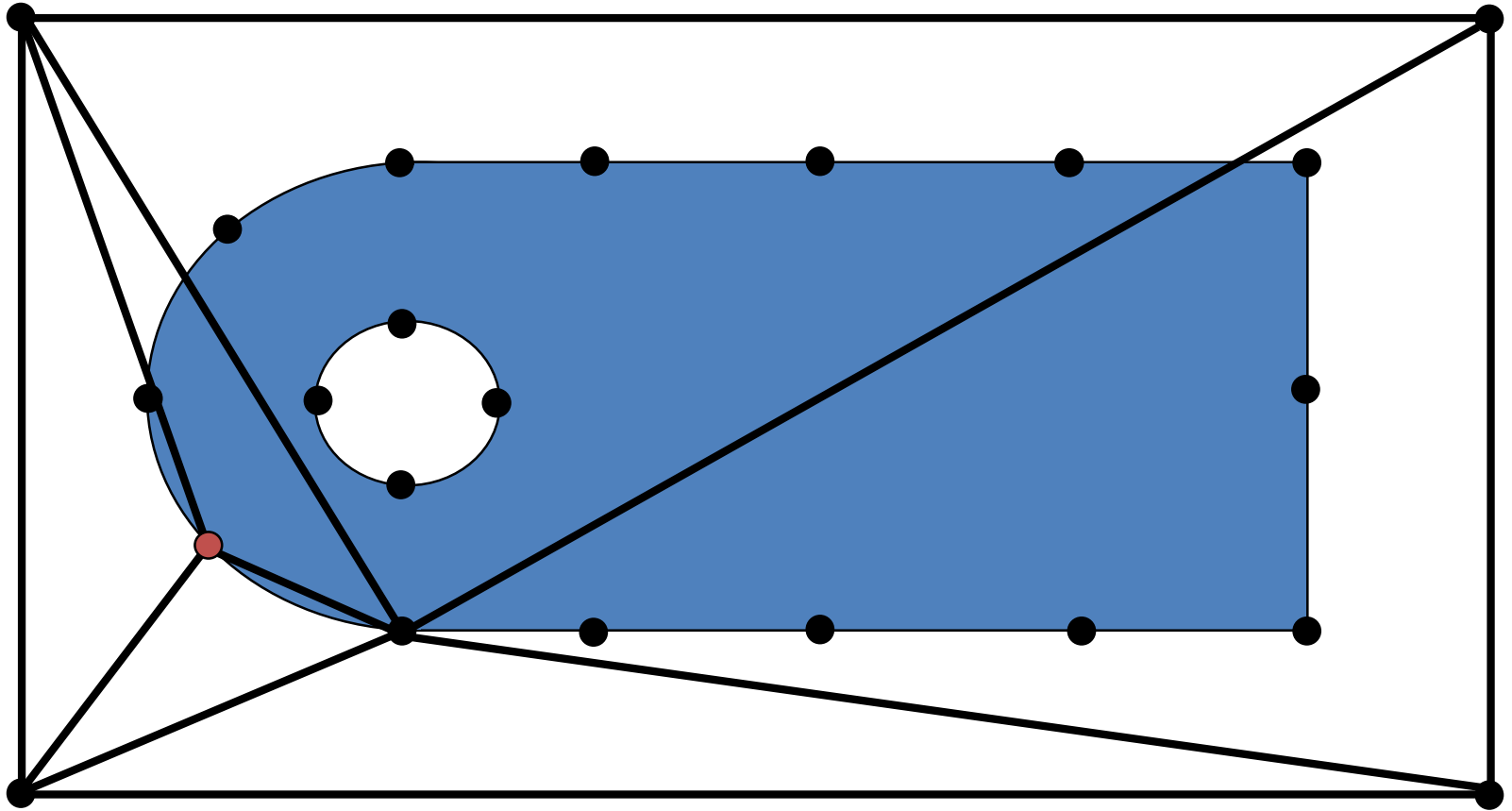
□ Create bounding triangles

Boundary Insertion



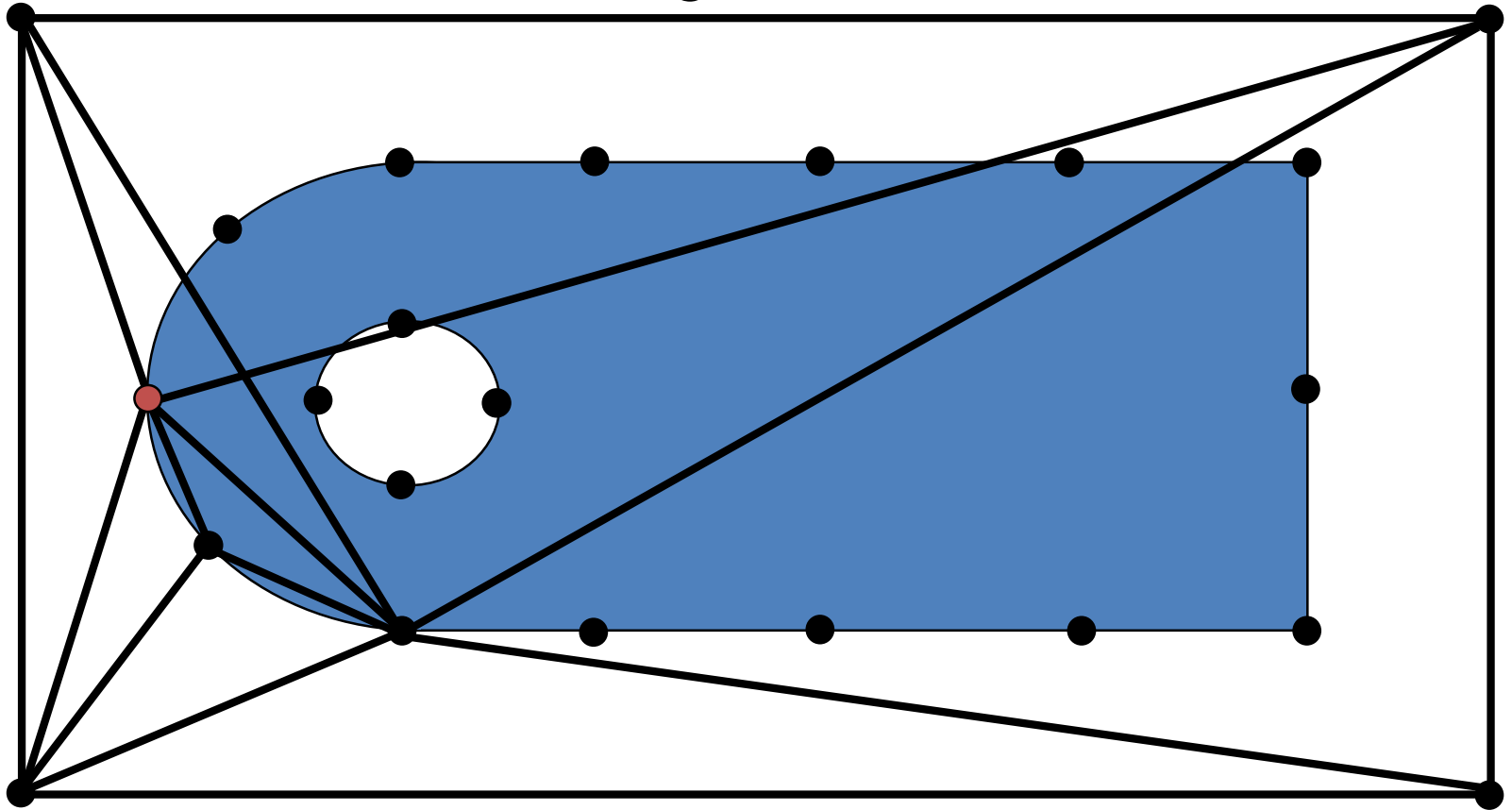
□ Insert vertices using Delaunay method

Boundary Insertion



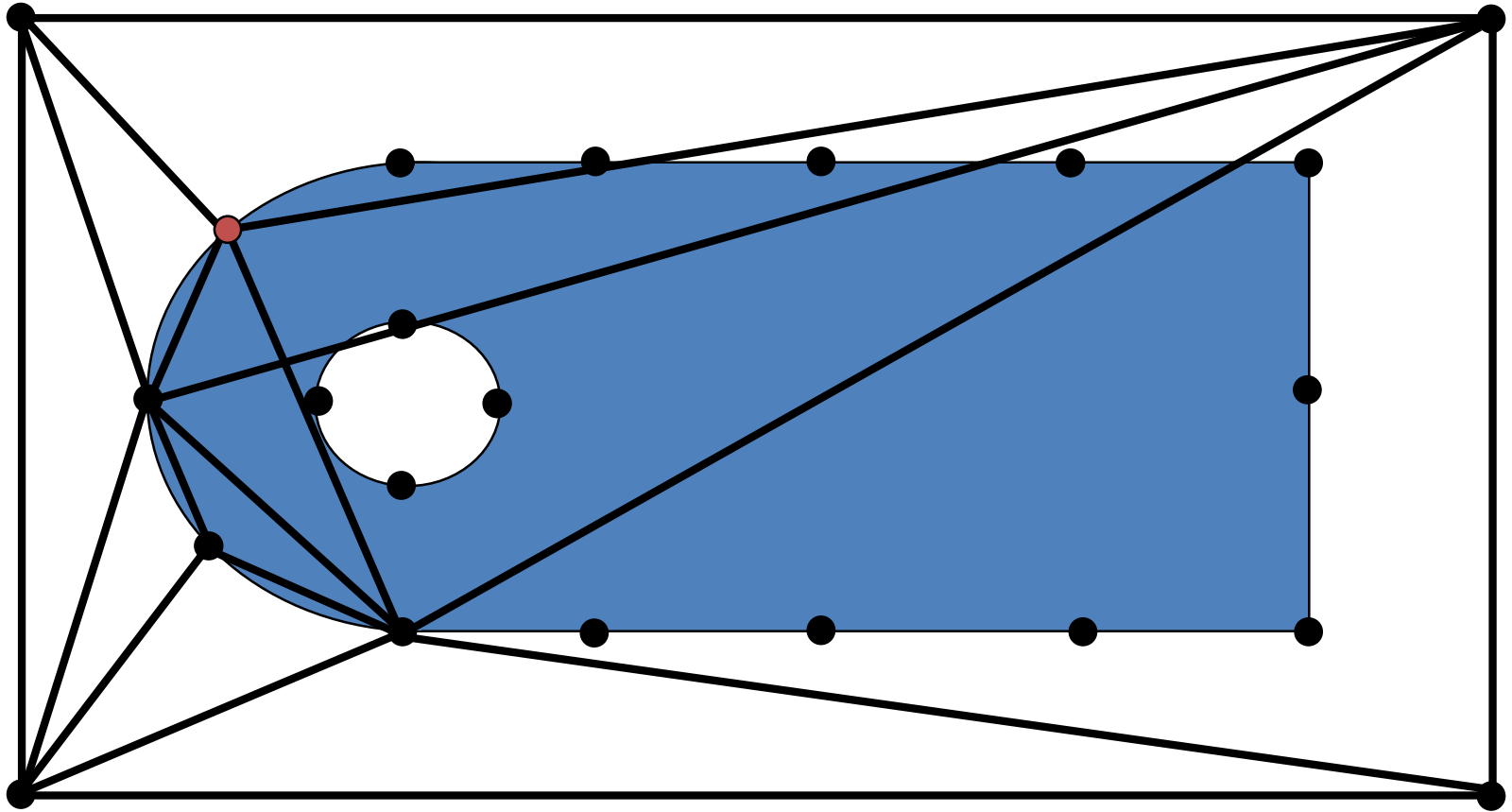
□ Insert vertices using Delaunay method

Boundary Insertion



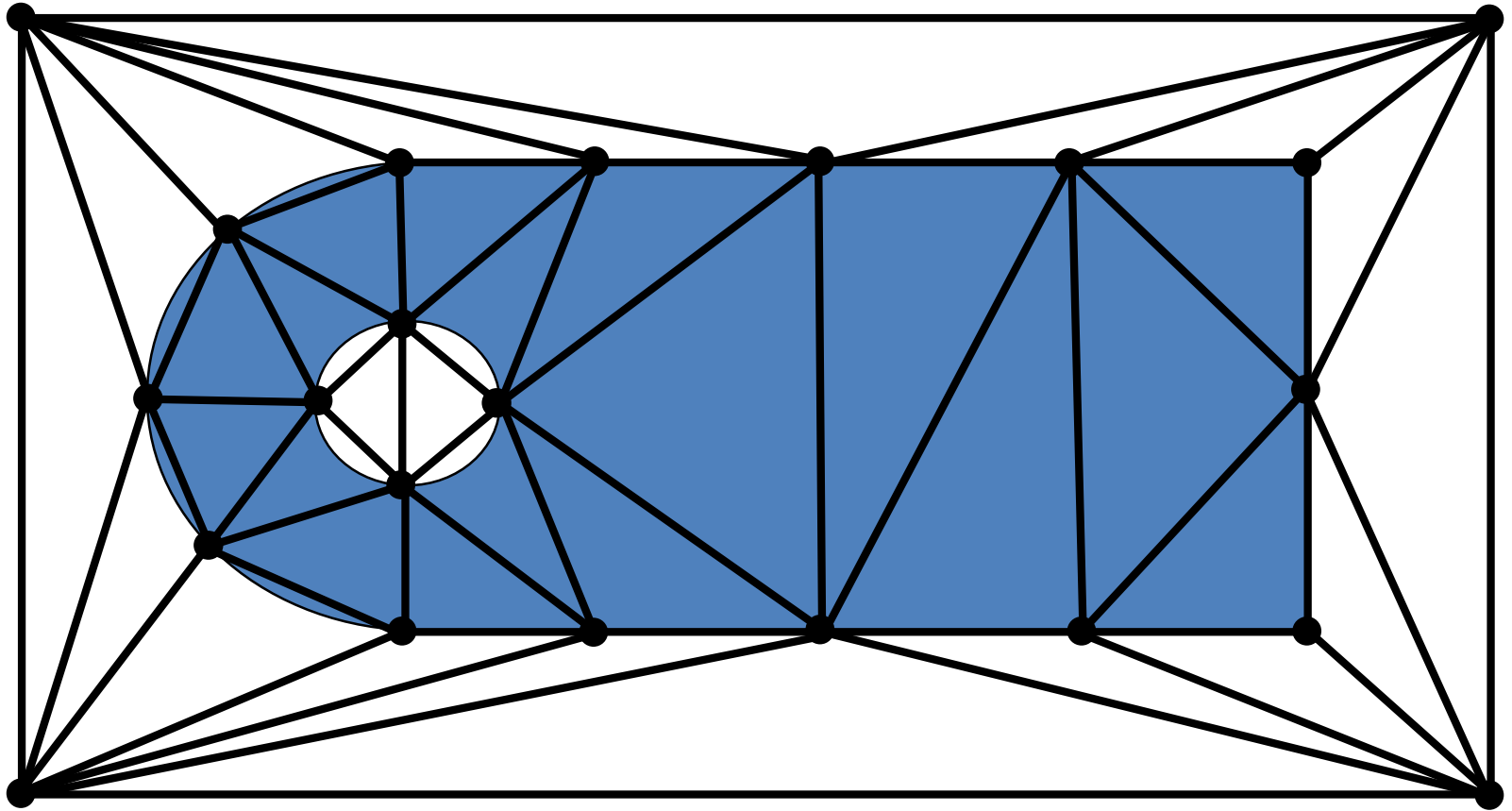
□ Insert vertices using Delaunay method

Boundary Insertion



□ Insert vertices using Delaunay method

Boundary Insertion

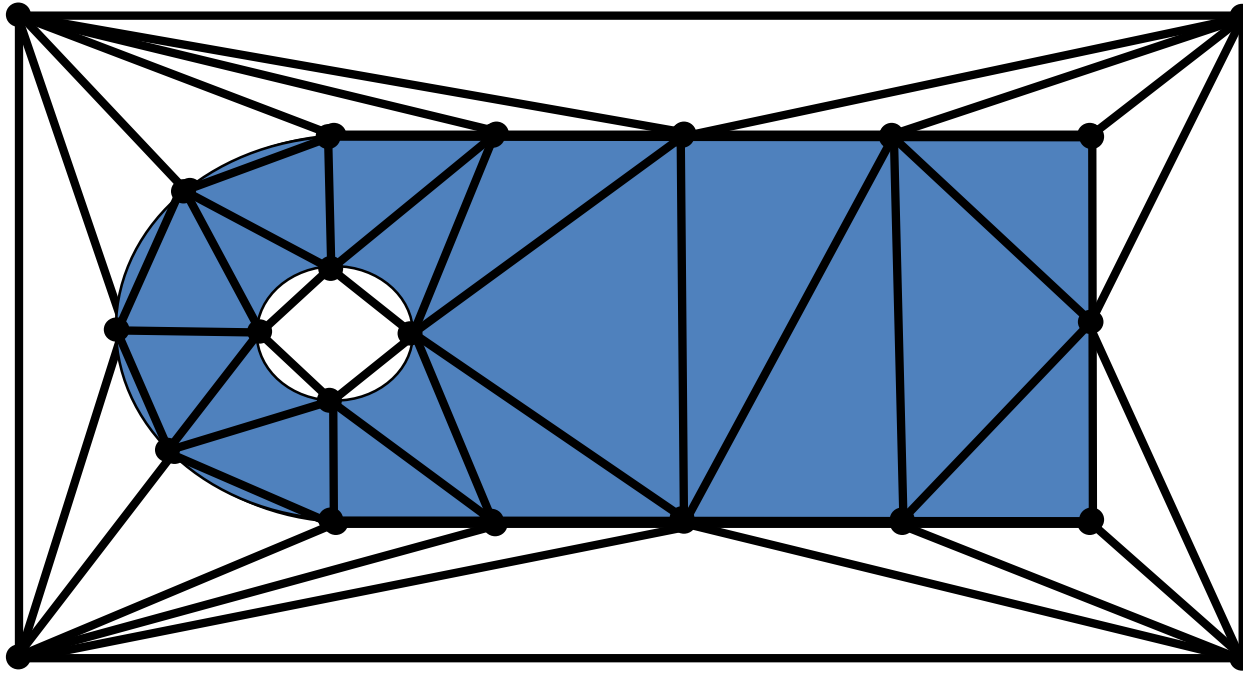


□ Insert vertices using Delaunay method

Vertex Insertion

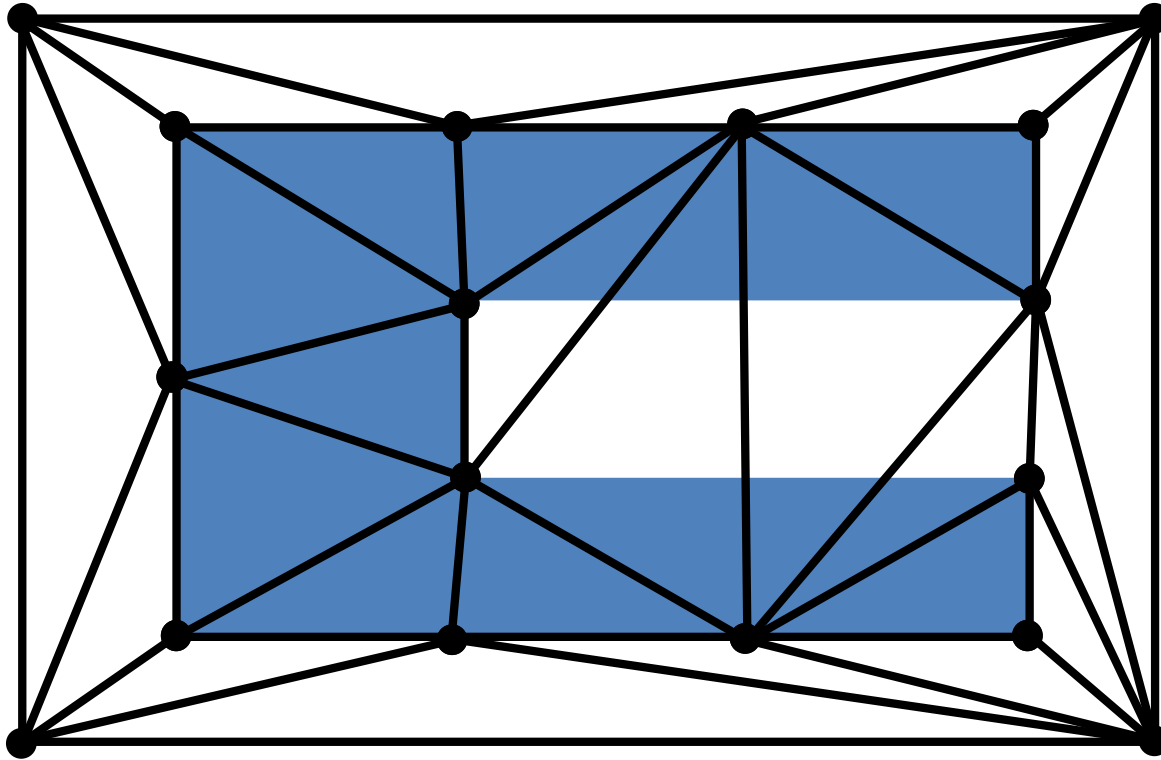
- Start with Delaunay mesh covering domain
 - Typically 2 triangle bounding box
- Insert one vertex at a time
 - Add vertex to mesh (locate triangle to split)
 - Flip edges (locally) to preserve Delaunay property
- **Boundary recovery**

Boundary Recovery



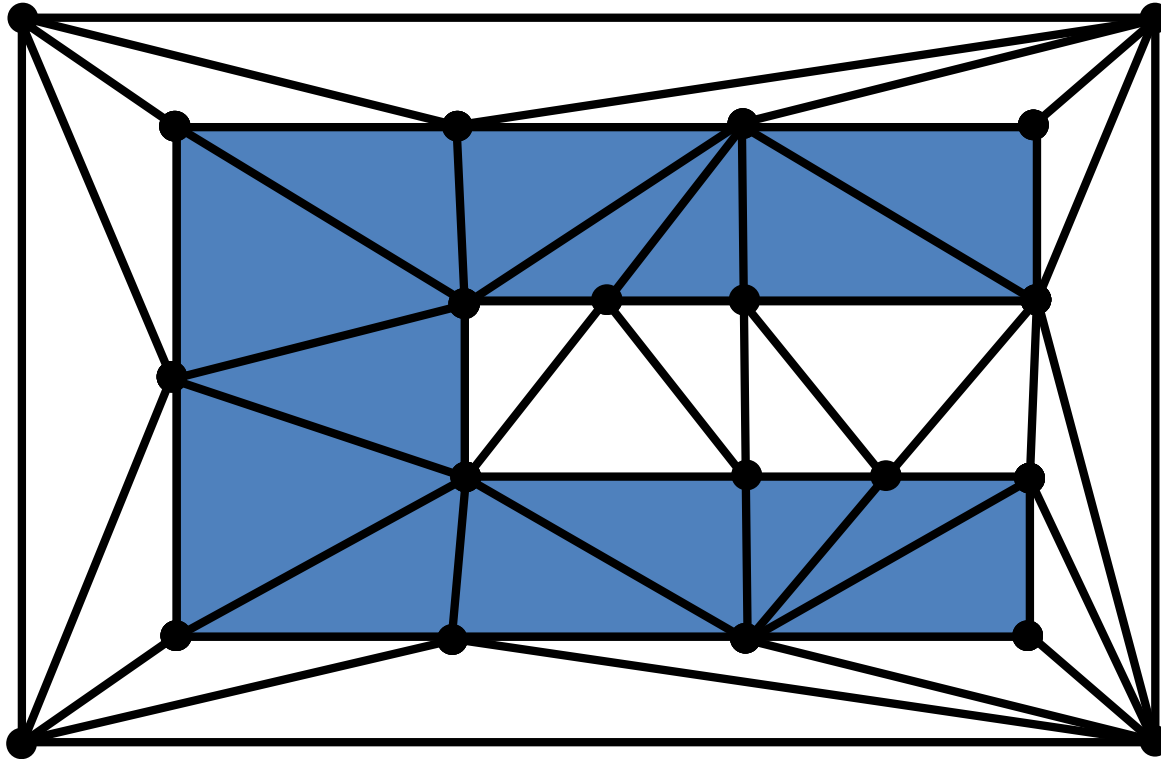
- Delete outside triangles (if can)
 - Delaunay triangulation does not have to obey polygon boundary

Boundary Recovery



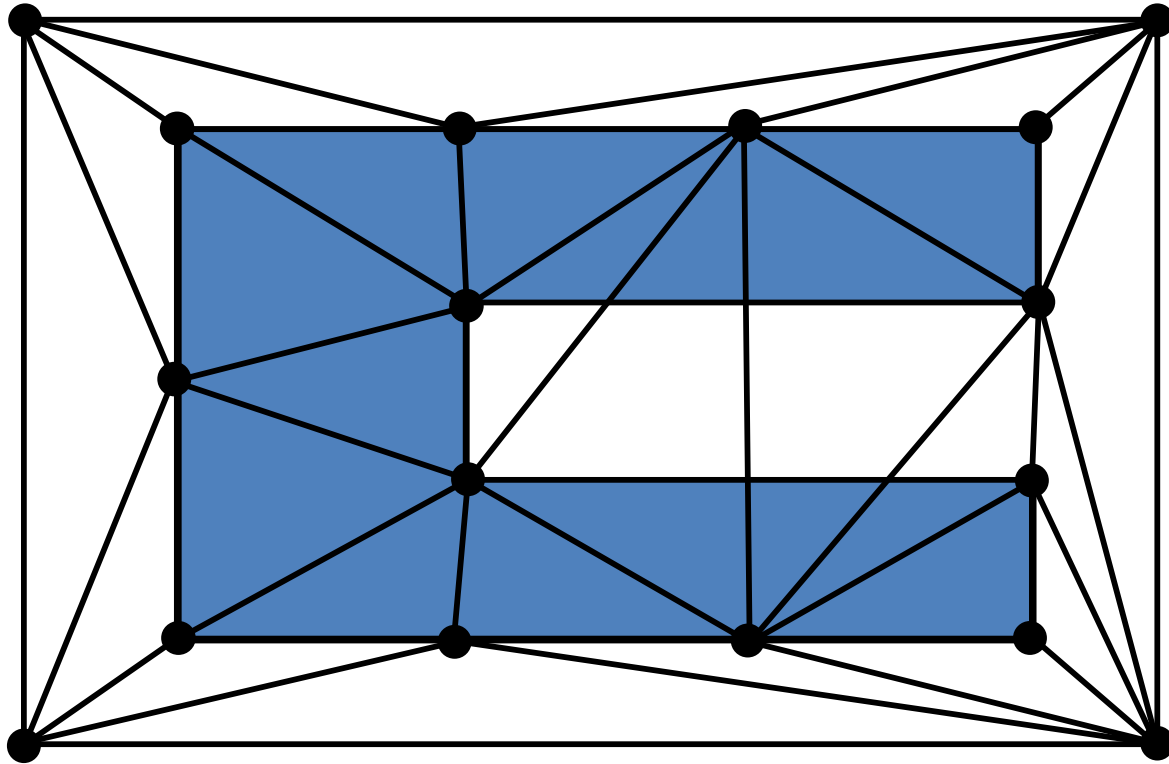
- Delaunay triangulation does not always obey polygon boundary

Boundary Recovery



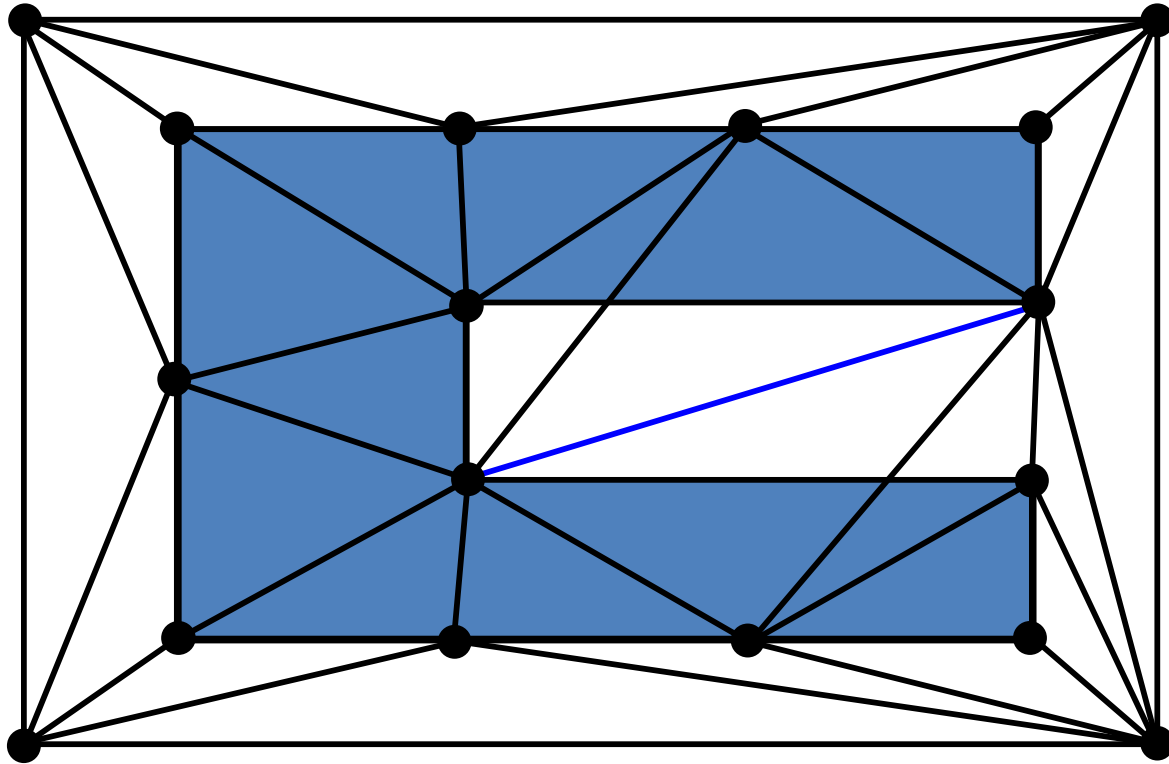
- Boundary Conforming Solution
 - Add vertices at intersections
 - Repeat if necessary

Boundary Recovery - Constrained



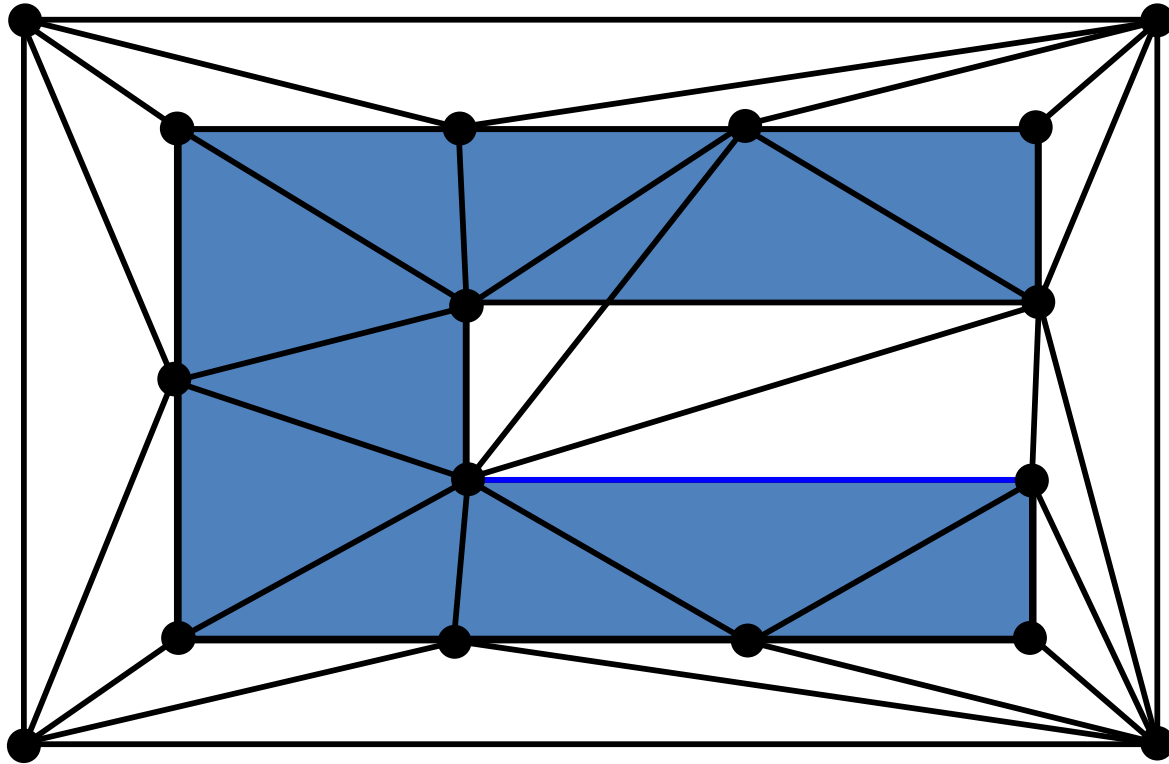
- Not always can add boundary vertices (shared edges)

Boundary Recovery - Constrained



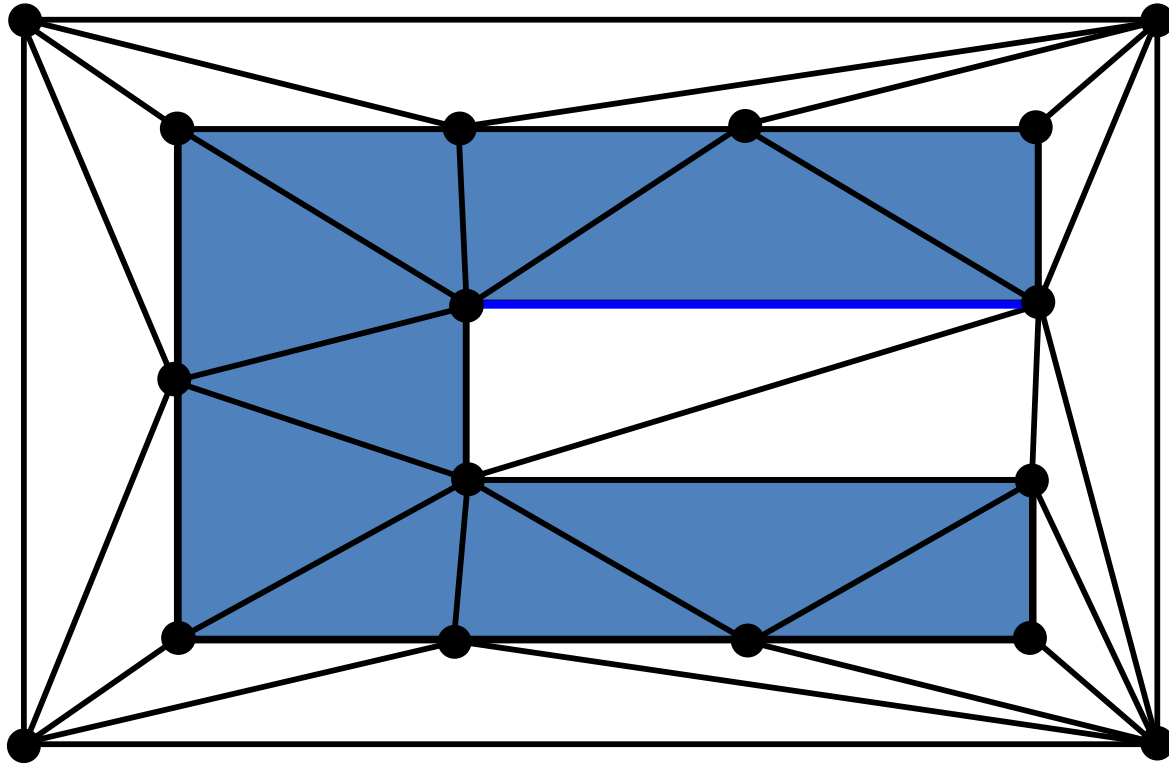
- Swap edges between adjacent pairs of triangles
- Repeat till recover the boundary

Boundary Recovery - Constrained



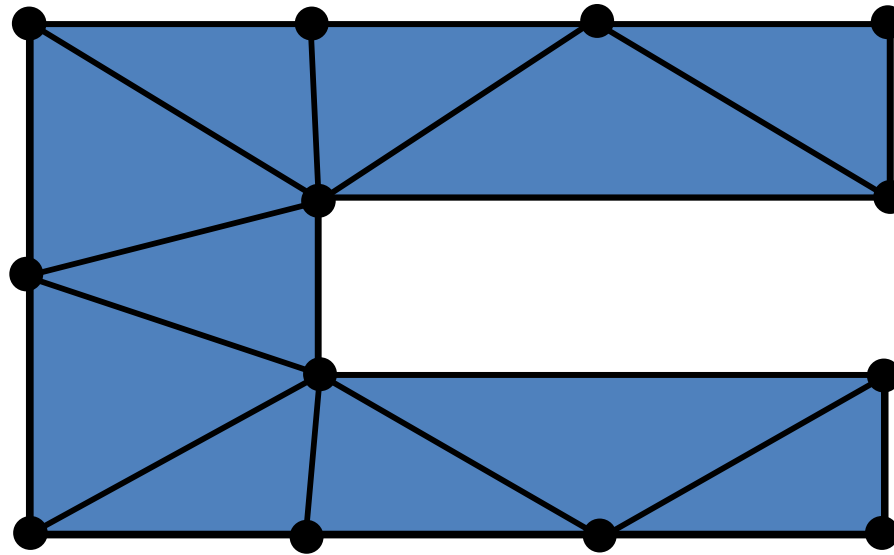
- Swap edges between adjacent pairs of triangles
- Repeat till recover the boundary

Boundary Recovery - Constrained



- Swap edges between adjacent pairs of triangles
- Repeat till recover the boundary

Boundary Recovery - Constrained

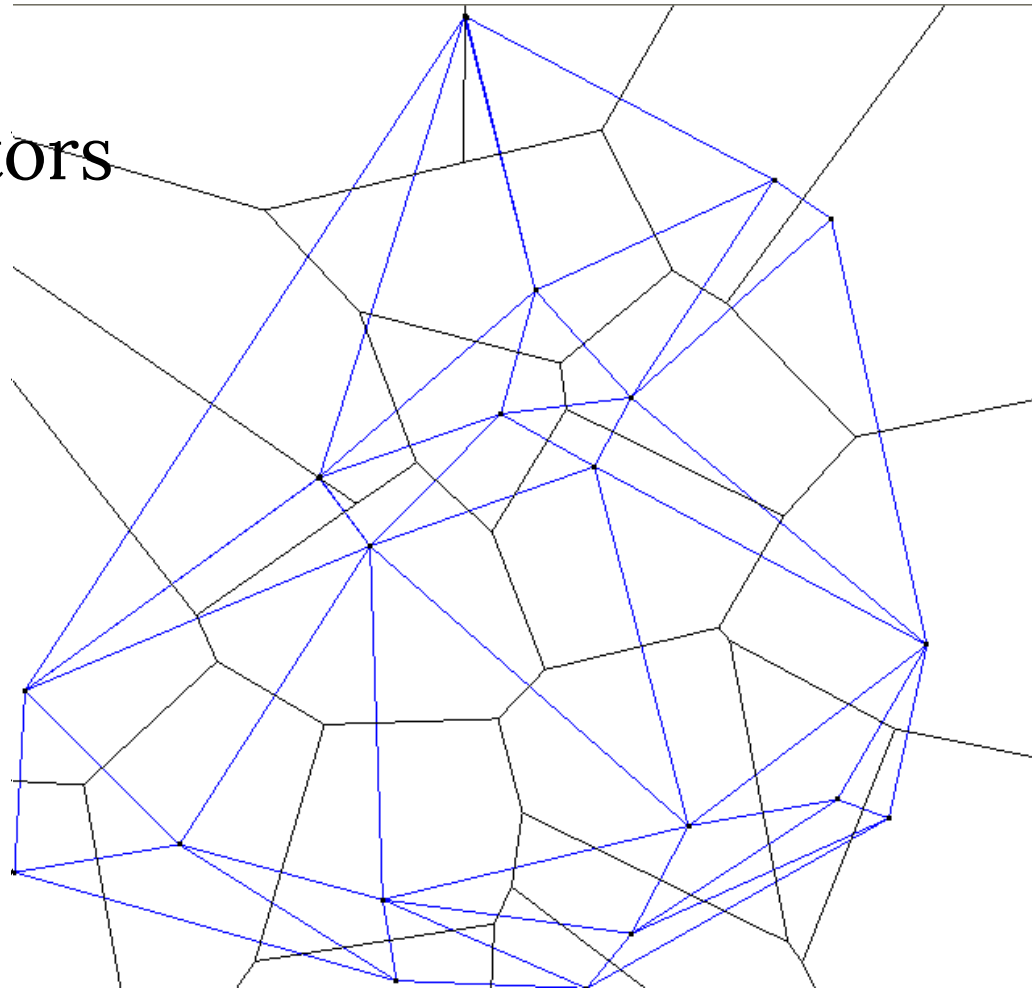


Not actually Delaunay

Voronoi Diagram

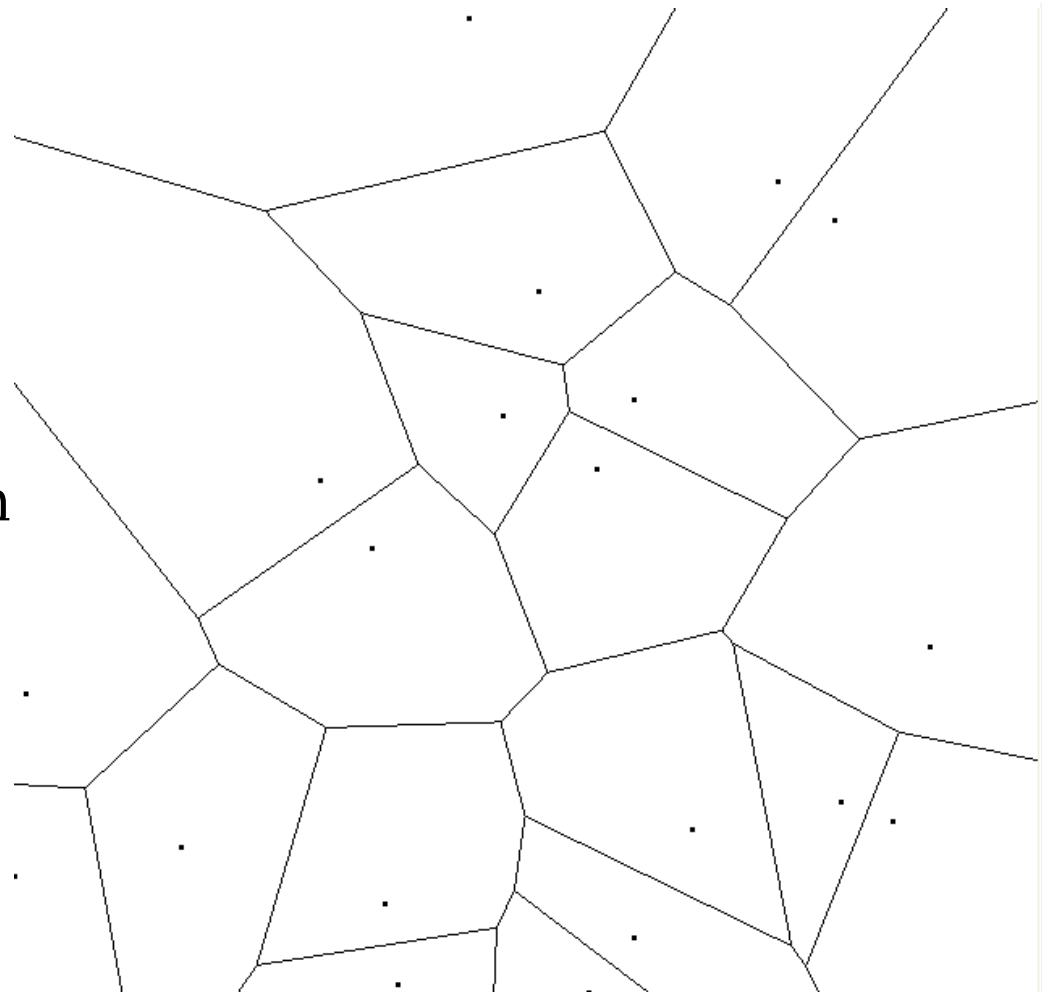
Dual to Delaunay Triangulation

- Vertices \rightarrow faces
- Voronoi edges =
- perpendicular bisectors of Delaunay edges



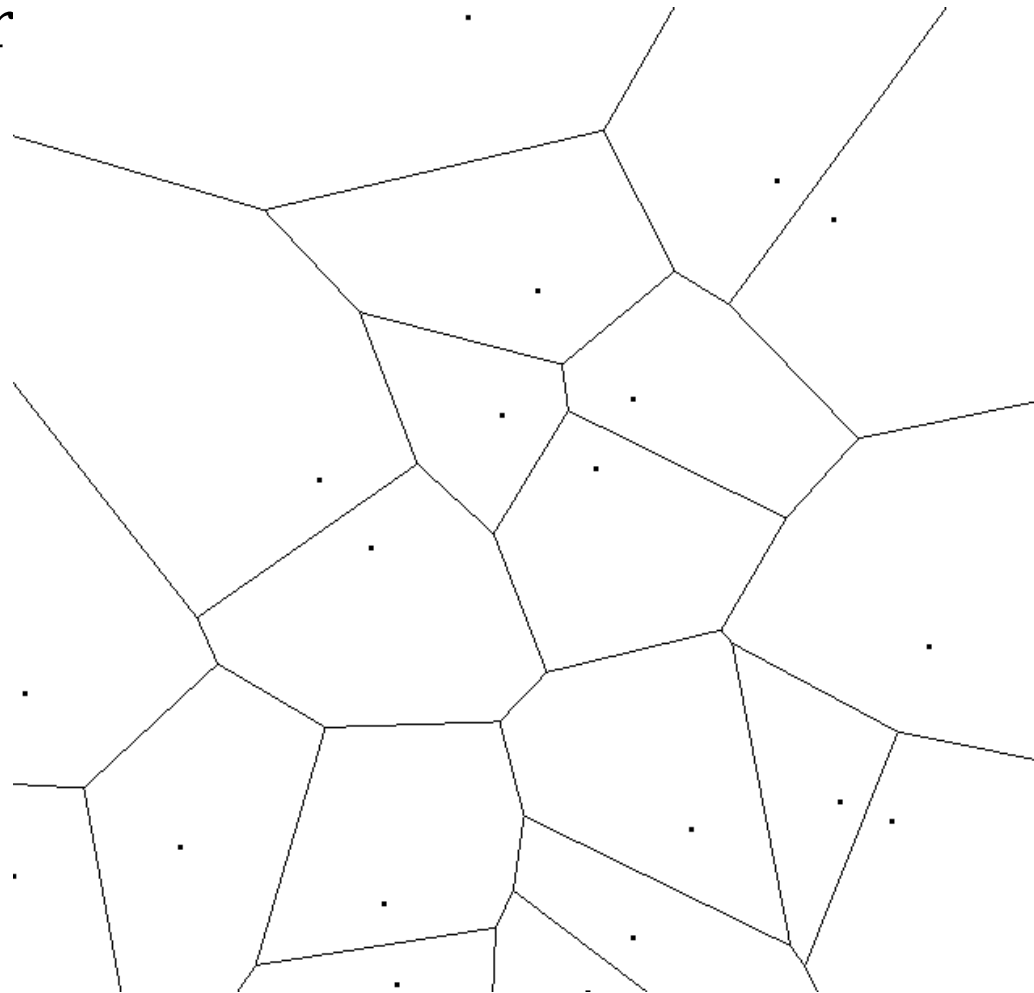
Sampling Using Centroidal Voronoi Diagrams

- *Voronoi Diagram for given set of vertices*: union of all locations at equal distance from two or more vertices
- Consists of
 - straight lines
 - vertex bisectors
 - vertices
 - bisector intersection



Voronoi Diagram

- Diagram partitions space into regions “closer” to one vertex than other
- Can define using distance function



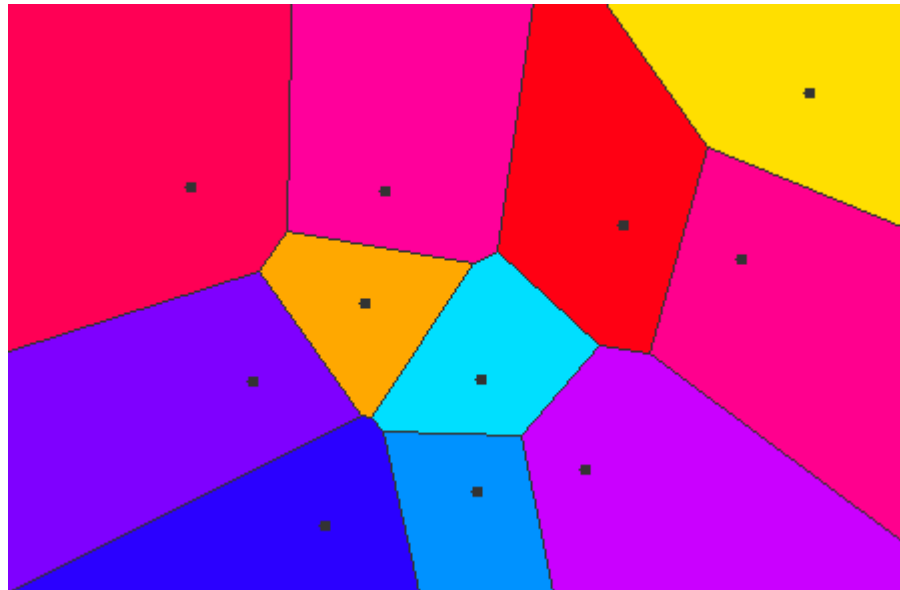
Voronoi Diagrams



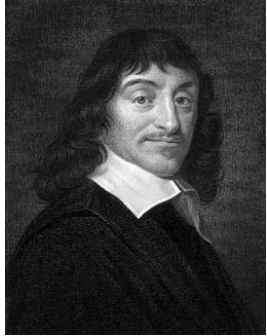
Georgy Voronoi

Георгий Феоодсьевич Вороной

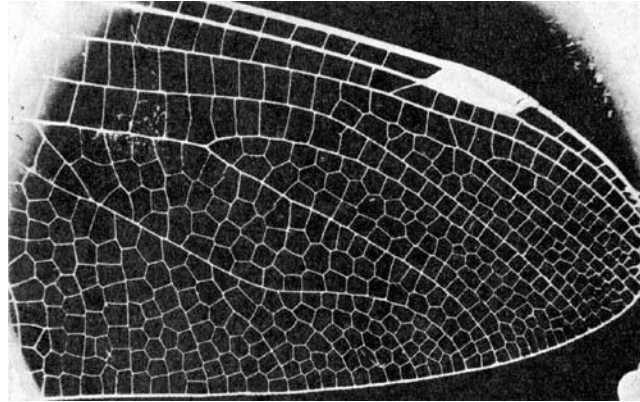
1868-1908



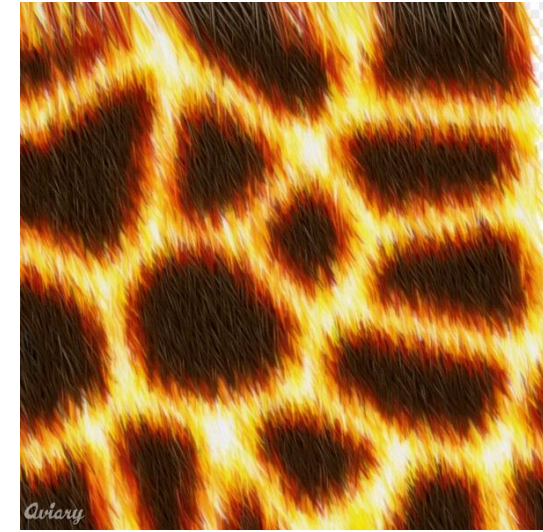
Historical Origins and Diagrams in Nature



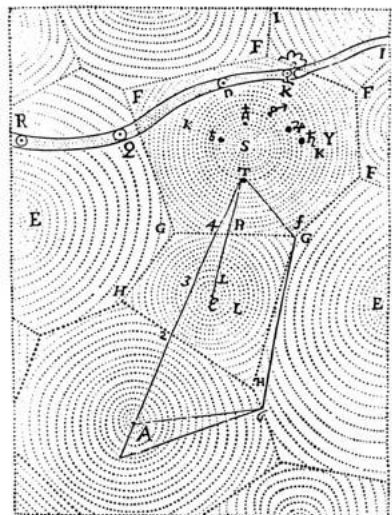
René Descartes
1596-1650
1644: Gravitational
Influence of stars



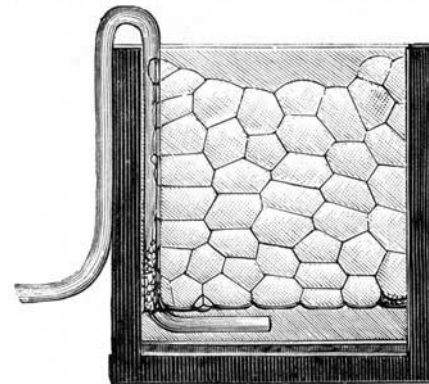
Dragonfly wing



Giraffe pigmentation



Honeycomb

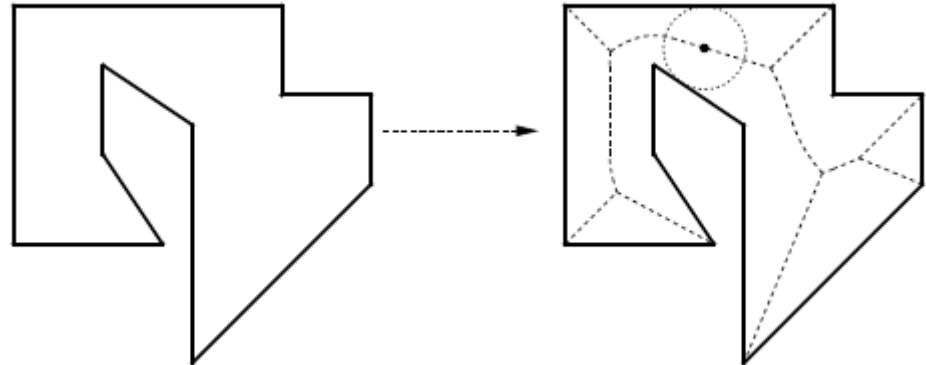


Constrained soap bubbles

Slides from Prof. Joseph S.B. Mitchell's AMS 345 course materials

Voronoi Applications

- Voronoi + point location search: nearest neighbor queries
- Facility location: Largest empty disk (centered at a Voronoi vertex)
- Shape description/approximation: medial axis

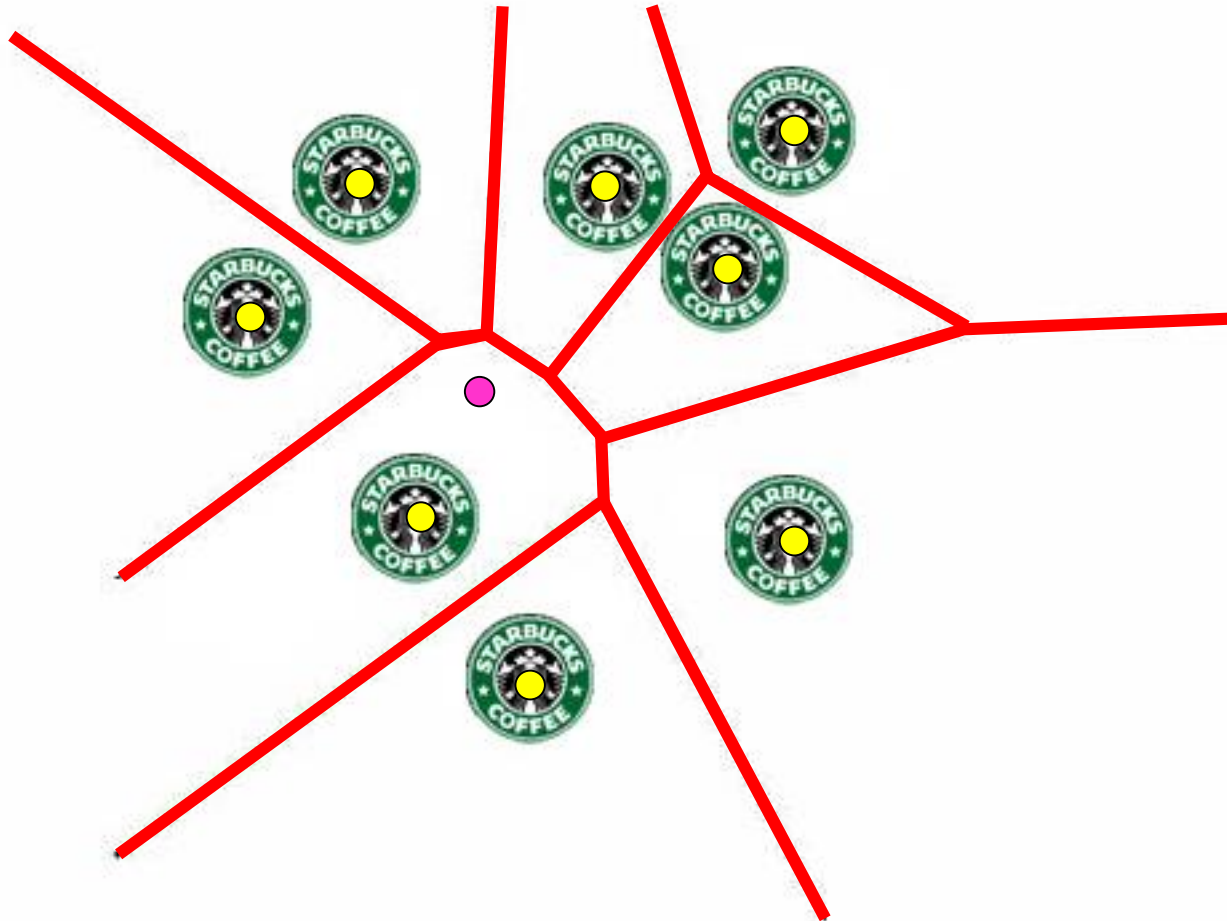


Starbucks

Post Office Problem

● Query point

● Post offices

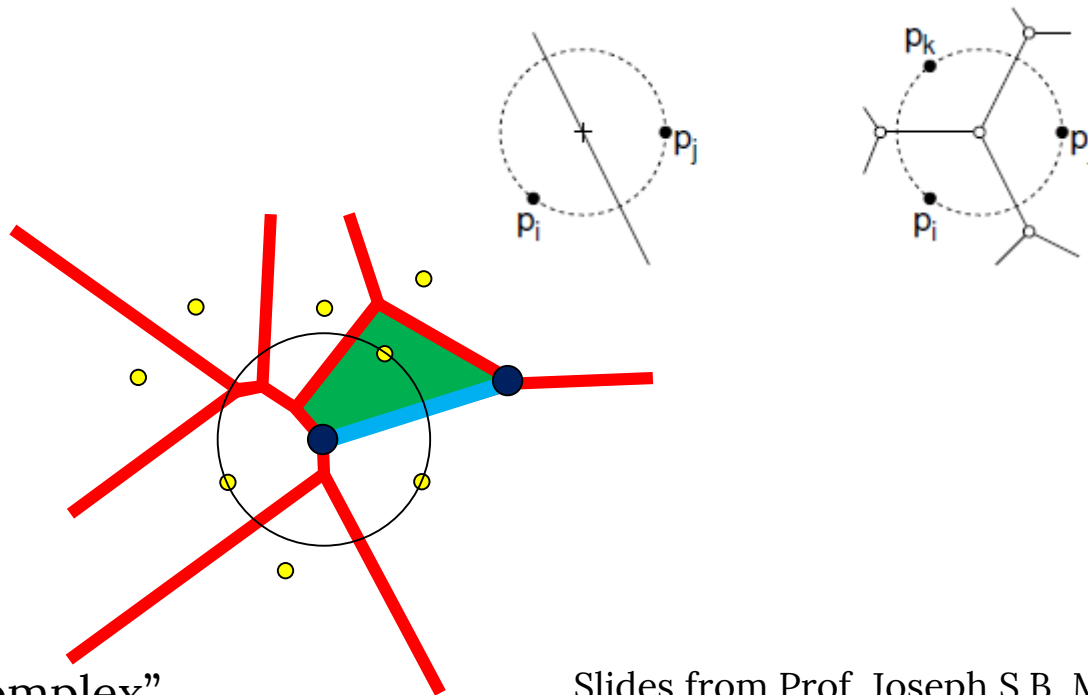


Voronoi Diagram

Partition the plane into cells:

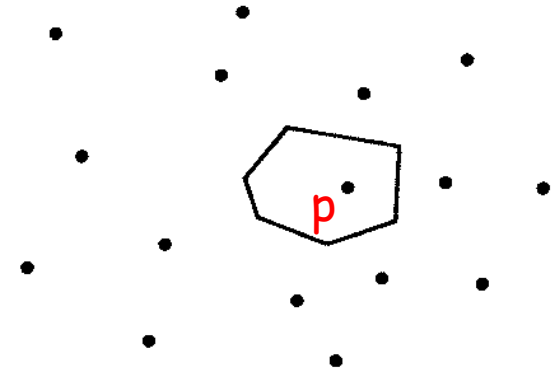
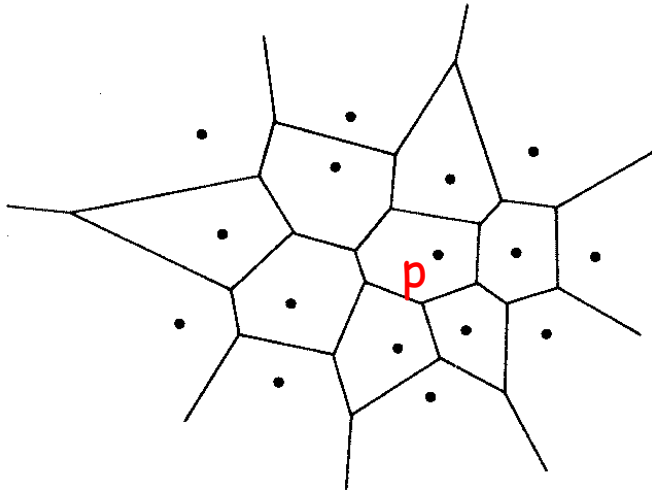
$$\mathcal{V}(p_i) = \{q \mid \|p_i q\| < \|p_j q\|, \forall j \neq i\}$$

Voronoi cell of p_i is open, convex



“cell complex”

Example



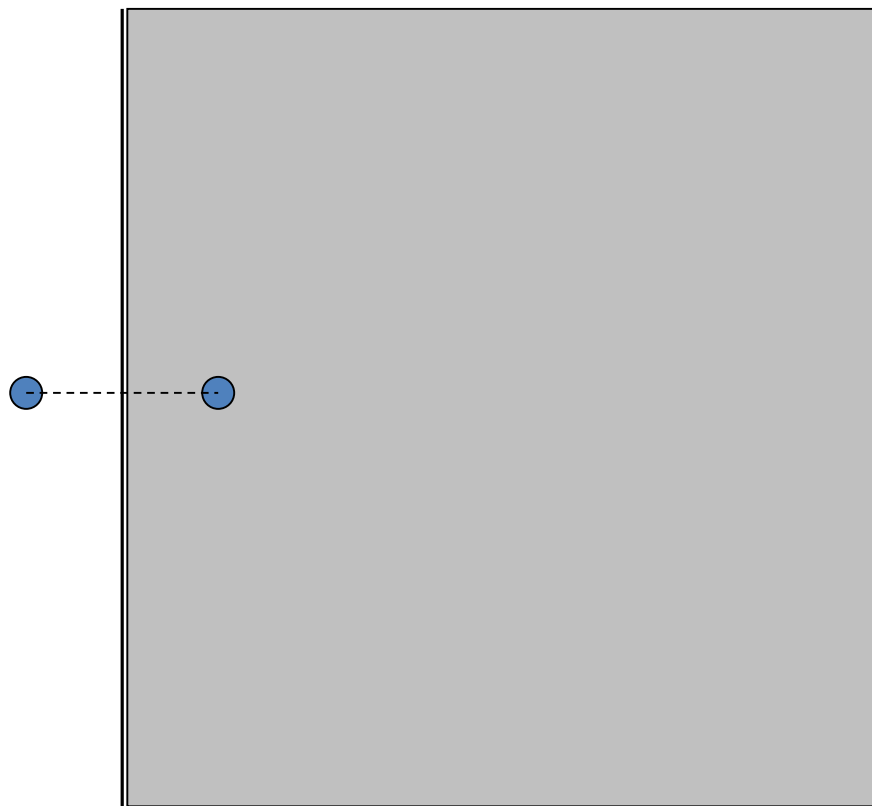
Voronoi cell of p

Constructing Voronoi Diagrams

Given a half plane intersection algorithm...

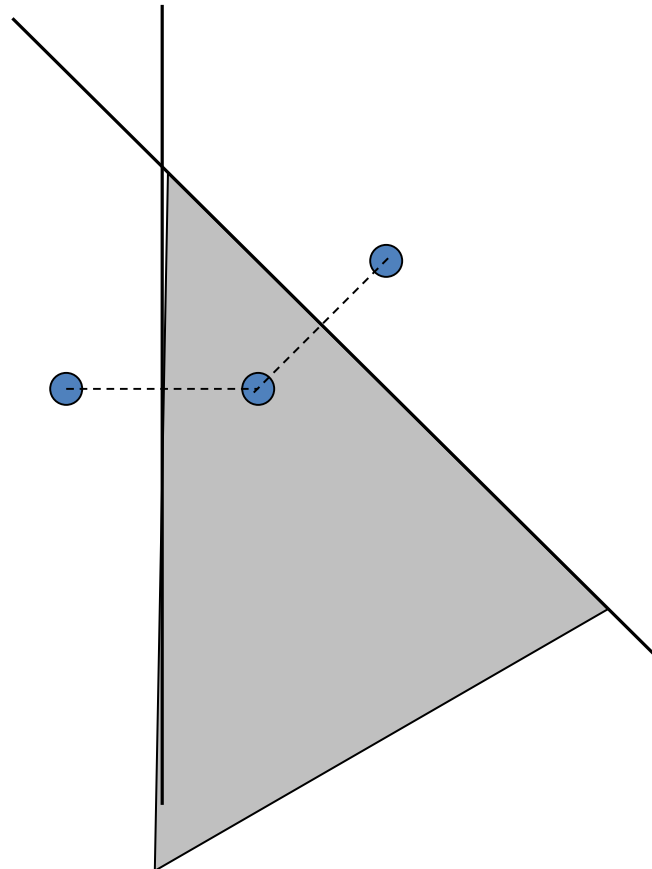
Constructing Voronoi Diagrams

Given a half plane intersection algorithm...



Constructing Voronoi Diagrams

Given a half plane intersection algorithm...

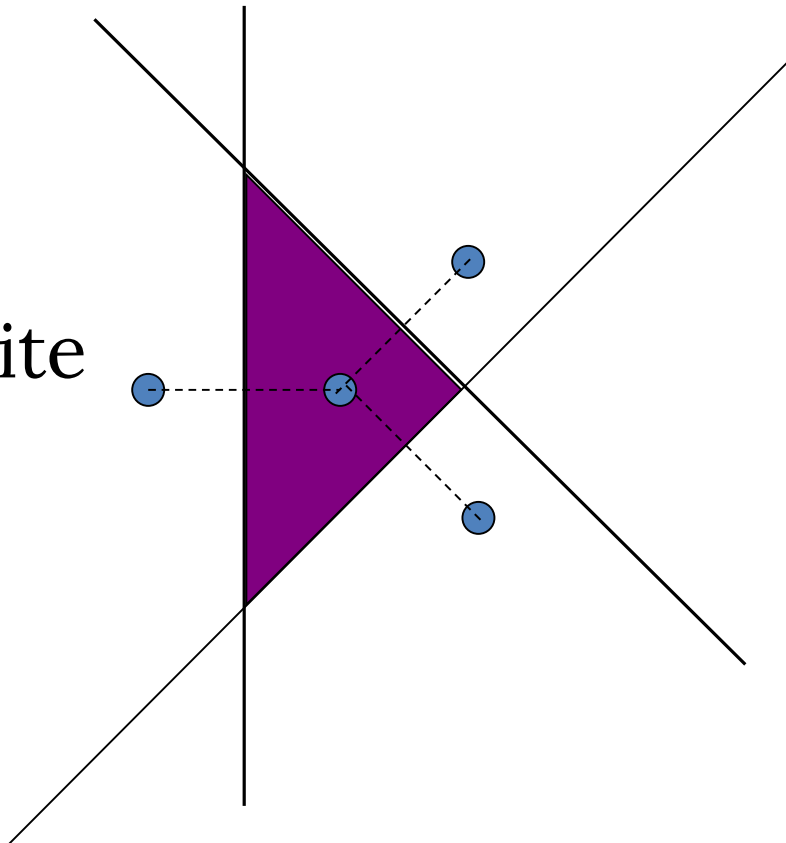


Constructing Voronoi Diagrams

Given a half plane intersection algorithm...

Repeat for each site

Running Time:
 $O(n^2 \log n)$

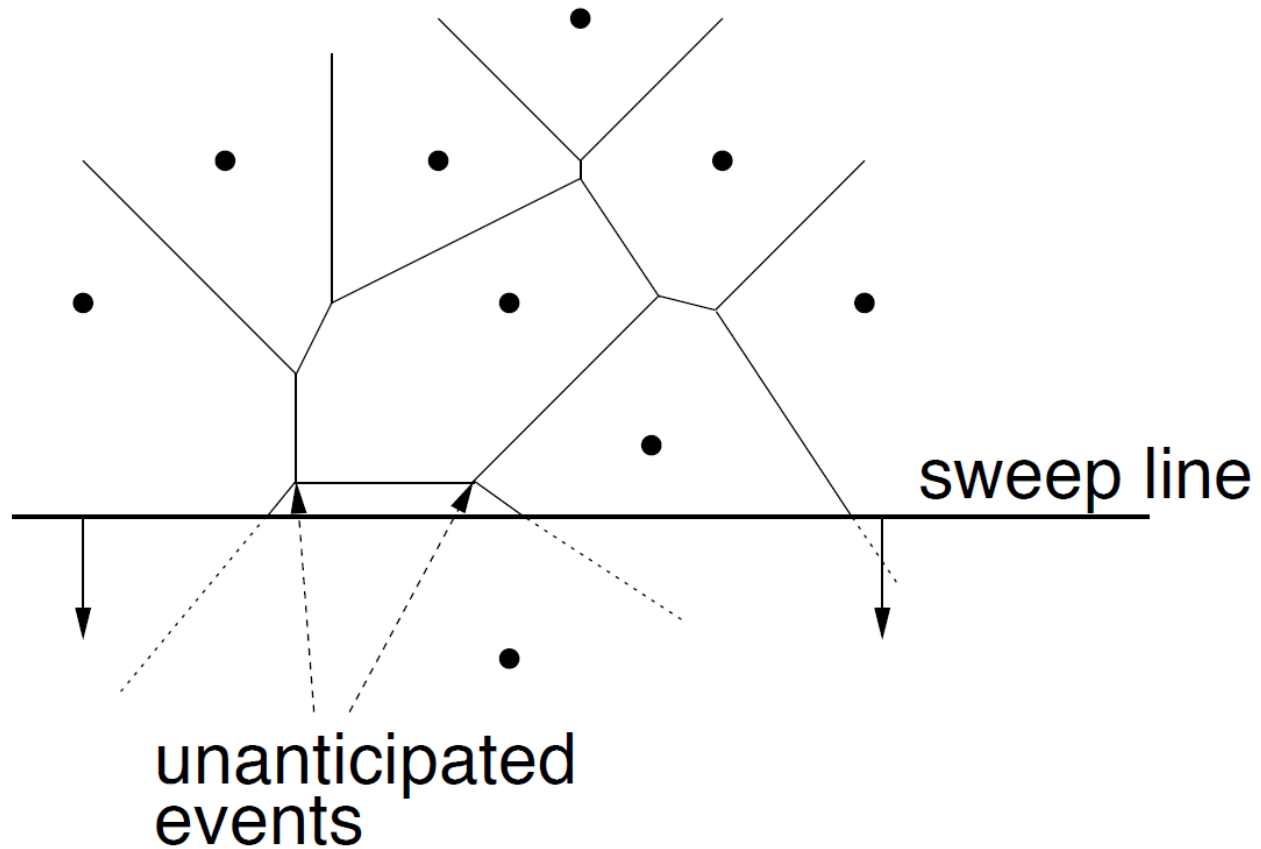


Constructing Voronoi Diagrams

- Half plane intersection $O(n^2 \log n)$
- Fortune's Algorithm
 - Sweep line algorithm
 - Voronoi diagram constructed as horizontal line sweeps the set of sites from top to bottom
 - Incremental construction \rightarrow maintains portion of diagram which doesn't change as we sweep down

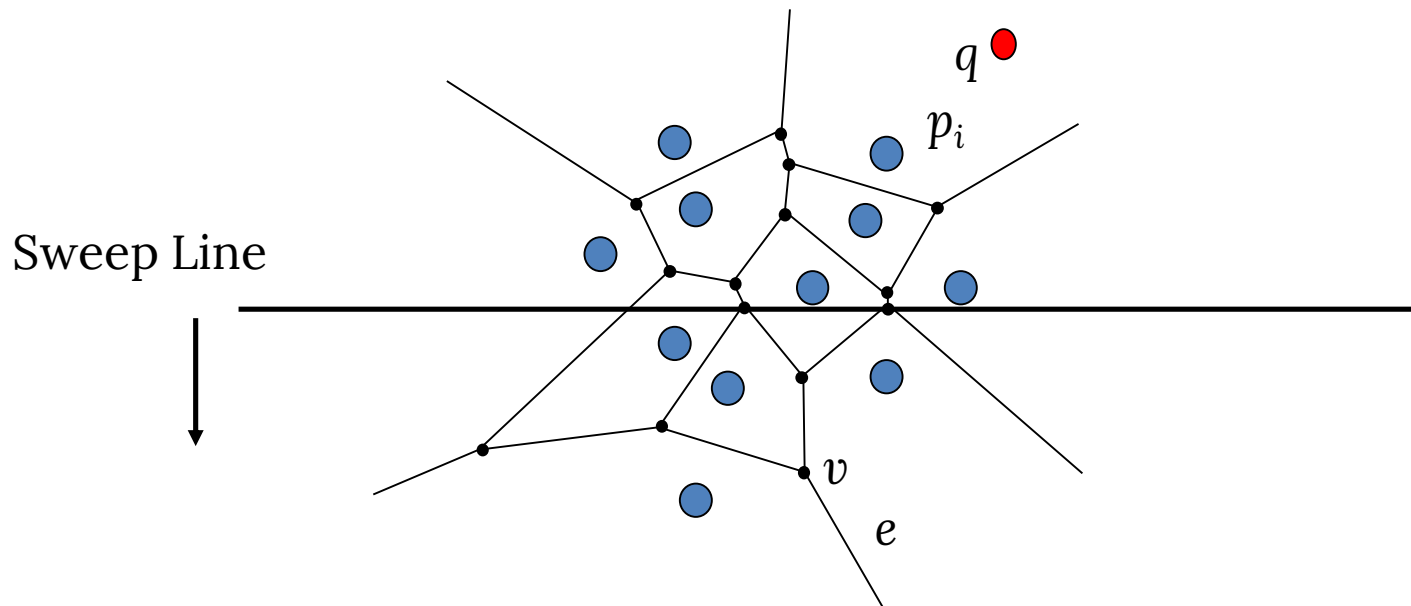
Constructing Voronoi Diagrams

Idea: line sweep?



Constructing Voronoi Diagrams

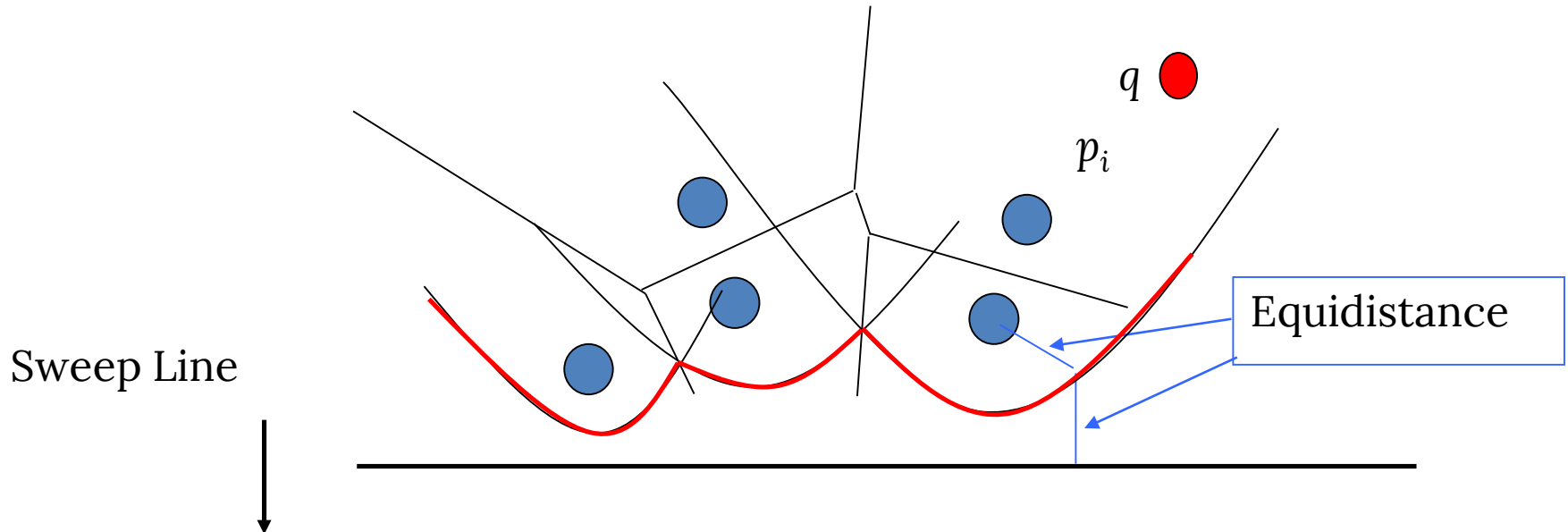
What is the invariant we are looking for?



Maintain a representation of the locus of points q that are closer to some site p_i above the sweep line than to the line itself (and thus to any site below the line).

Constructing Voronoi Diagrams

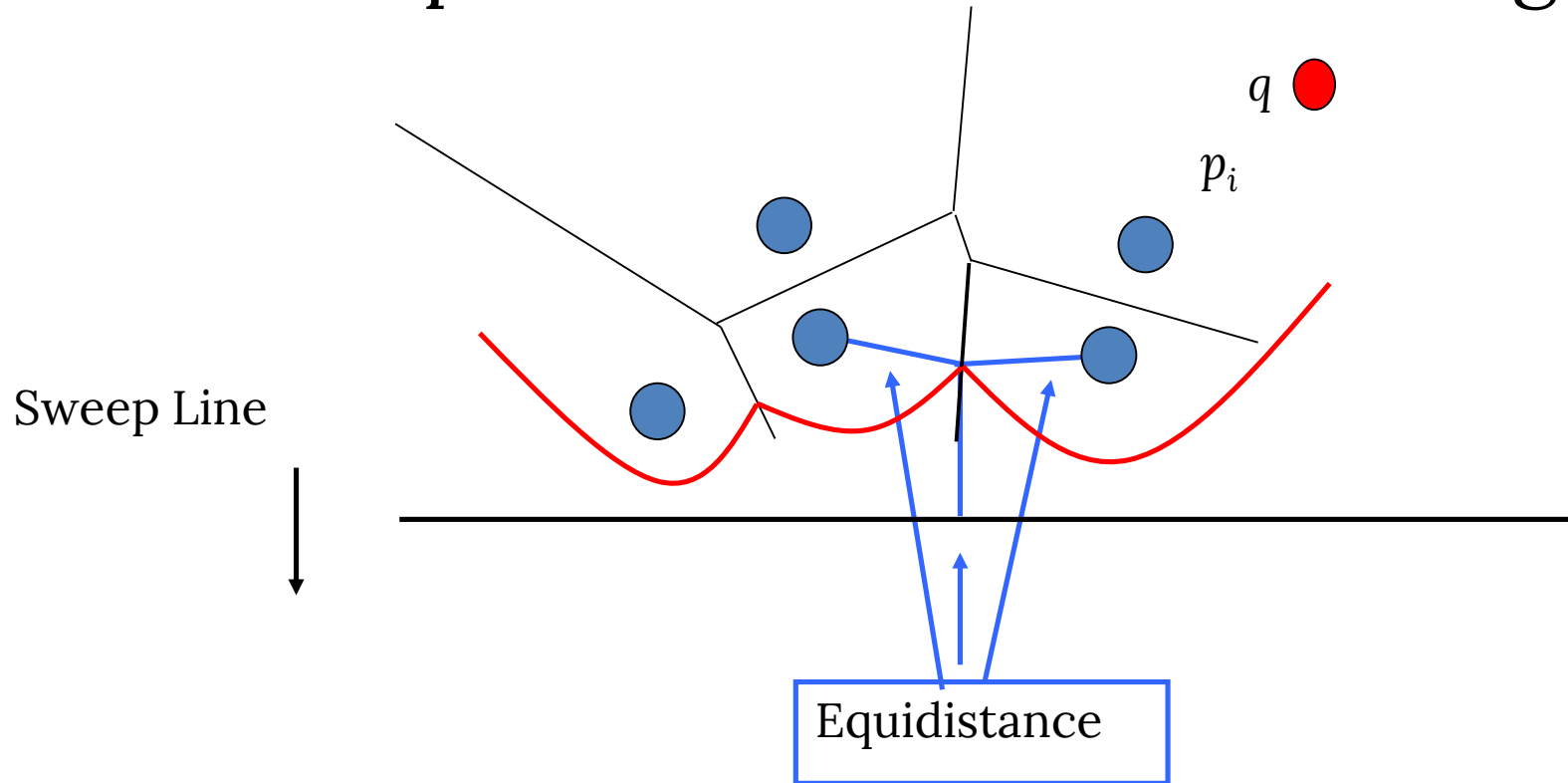
Which points are closer to a site above the sweep line than to the sweep line itself?



Beach line: lower envelope of all parabolas

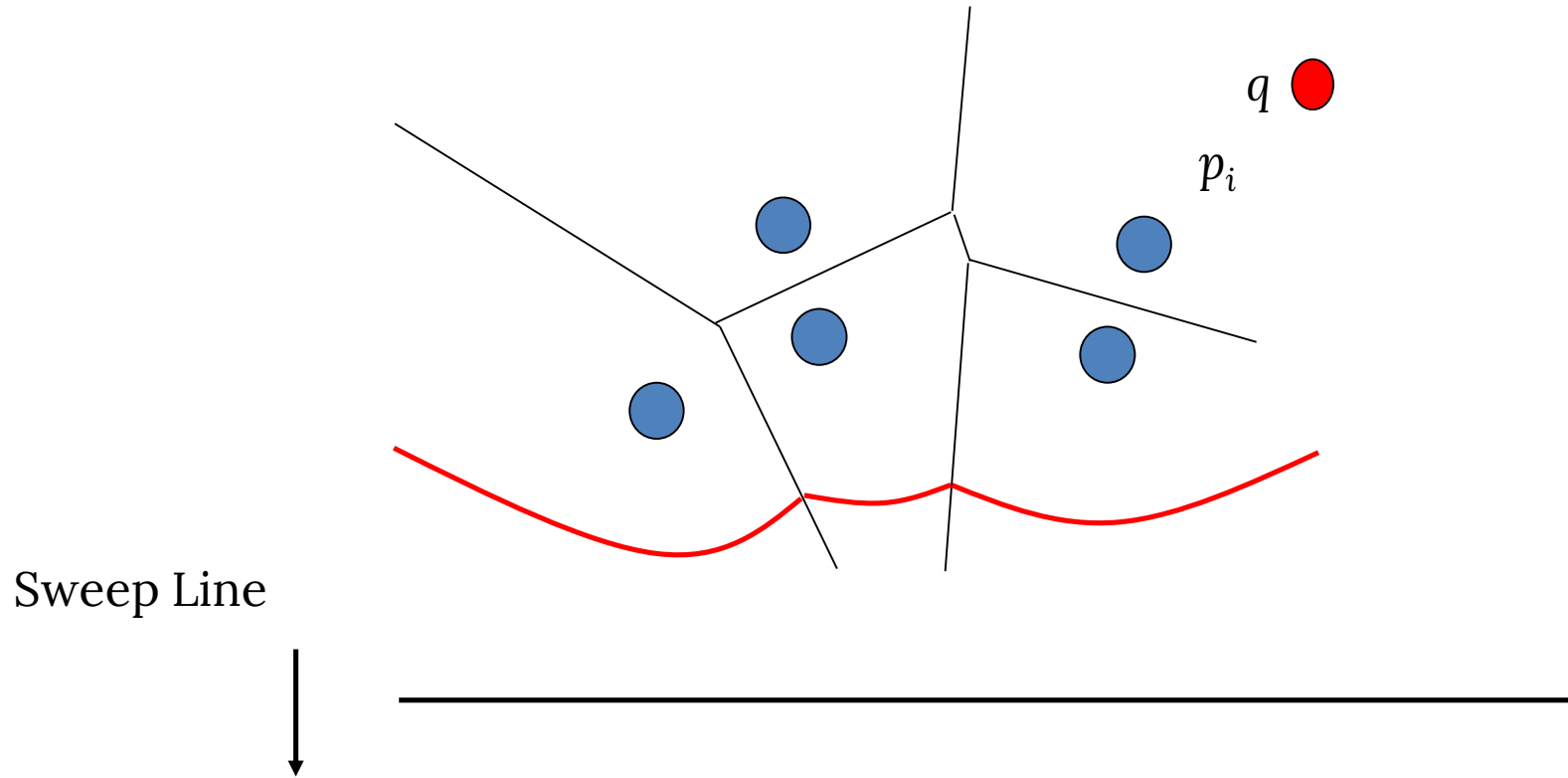
Constructing Voronoi Diagrams

Break points trace out Voronoi edges.



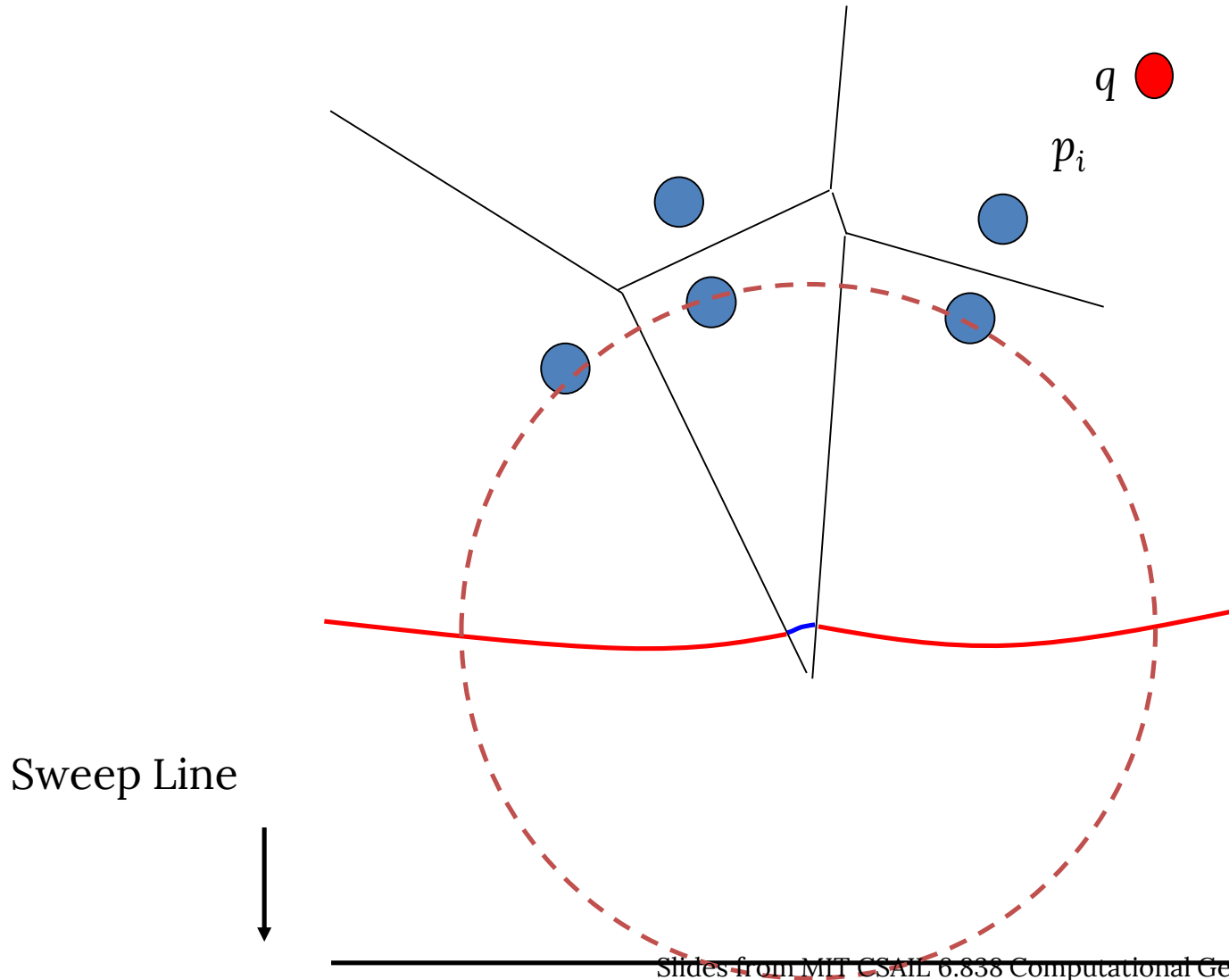
Constructing Voronoi Diagrams

Arcs flatten out as sweep line moves down.



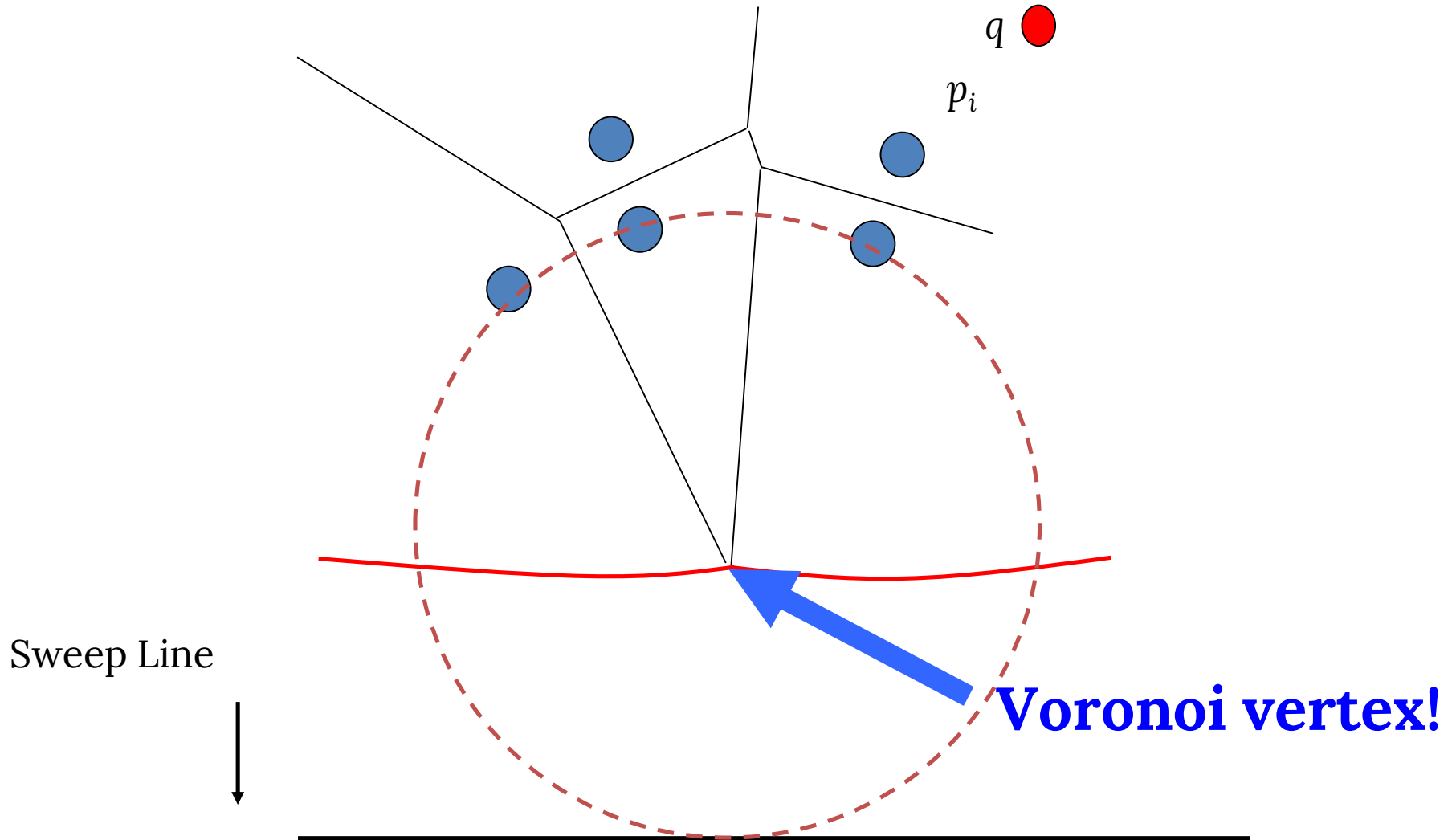
Constructing Voronoi Diagrams

Eventually, the middle arc disappears.



Constructing Voronoi Diagrams

We have detected a circle that is empty (contains no sites) and touches 3 or more sites.



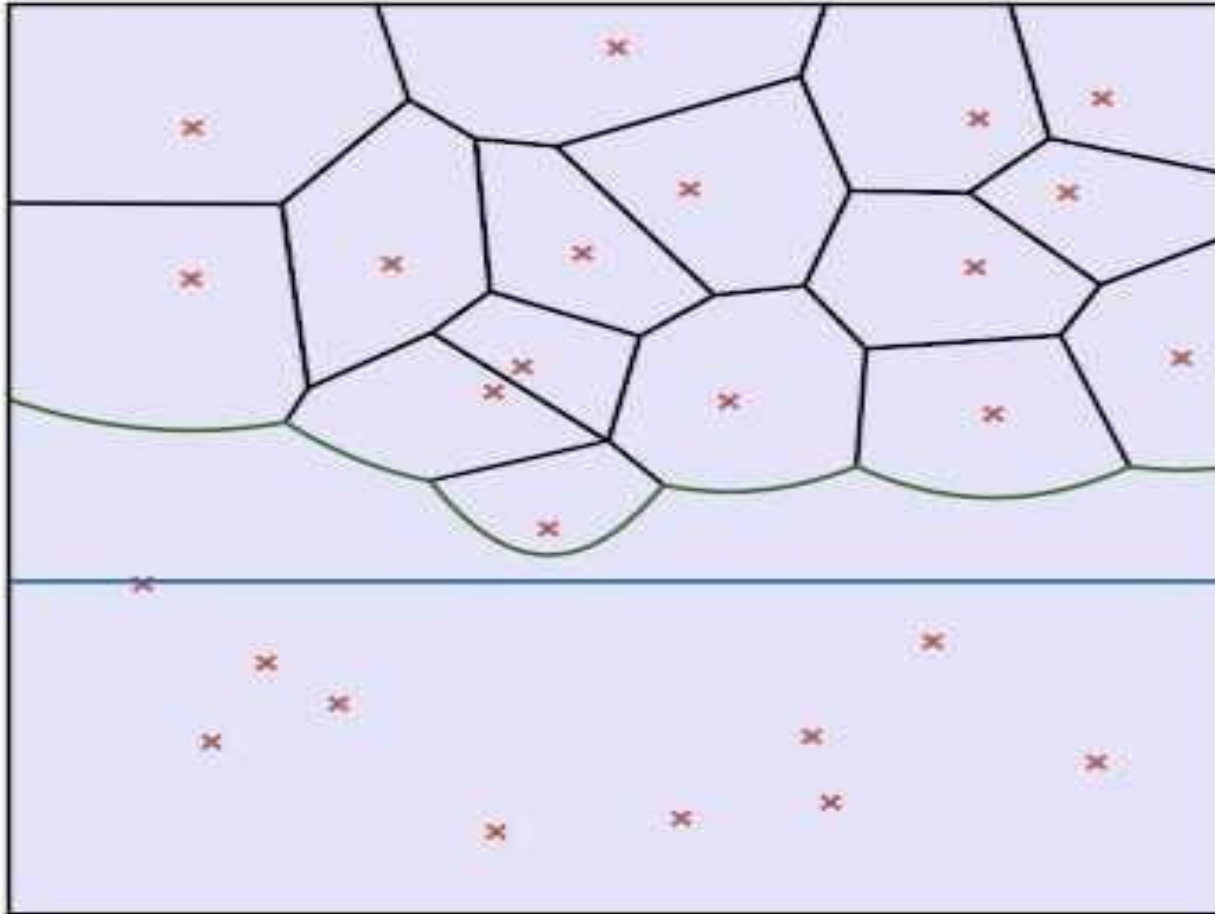
Beach Line properties

- Voronoi edges are traced by the break points
- Voronoi vertices are identified when two break points fuse
 - Decimation of an old arc identifies new vertex

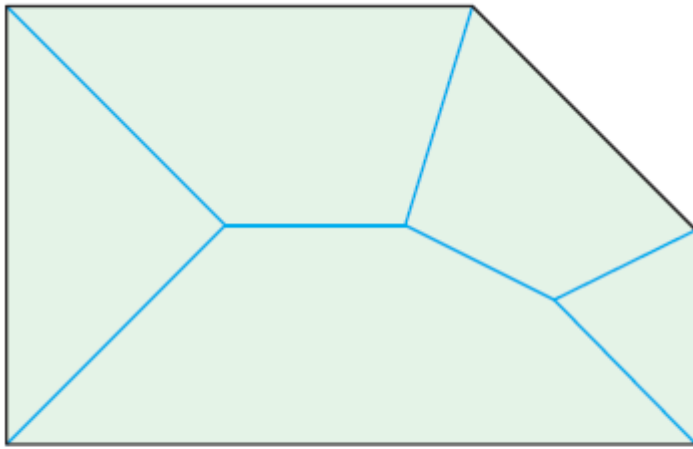
Fortune's Algorithm

- Trace out the cells by line sweep
- Maintain and track the beach line
- No need to store parabolas, just store the participating vertex
- $O(n \log n)$ time

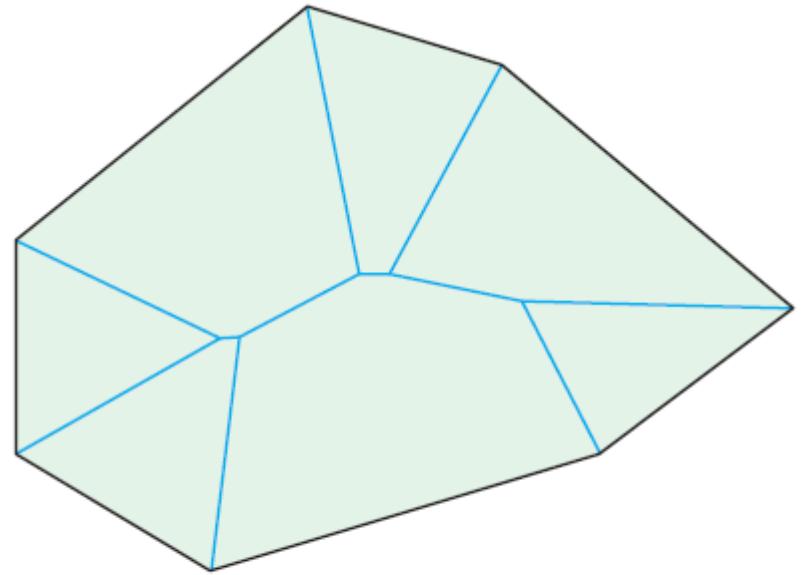
Fortune's Algorithm



Voronoi Diagram and Medial Axis



(a)



(b)

Figure 5.1: Convex polygons and their medial axes marked in blue.

Voronoi Diagram and Medial Axis

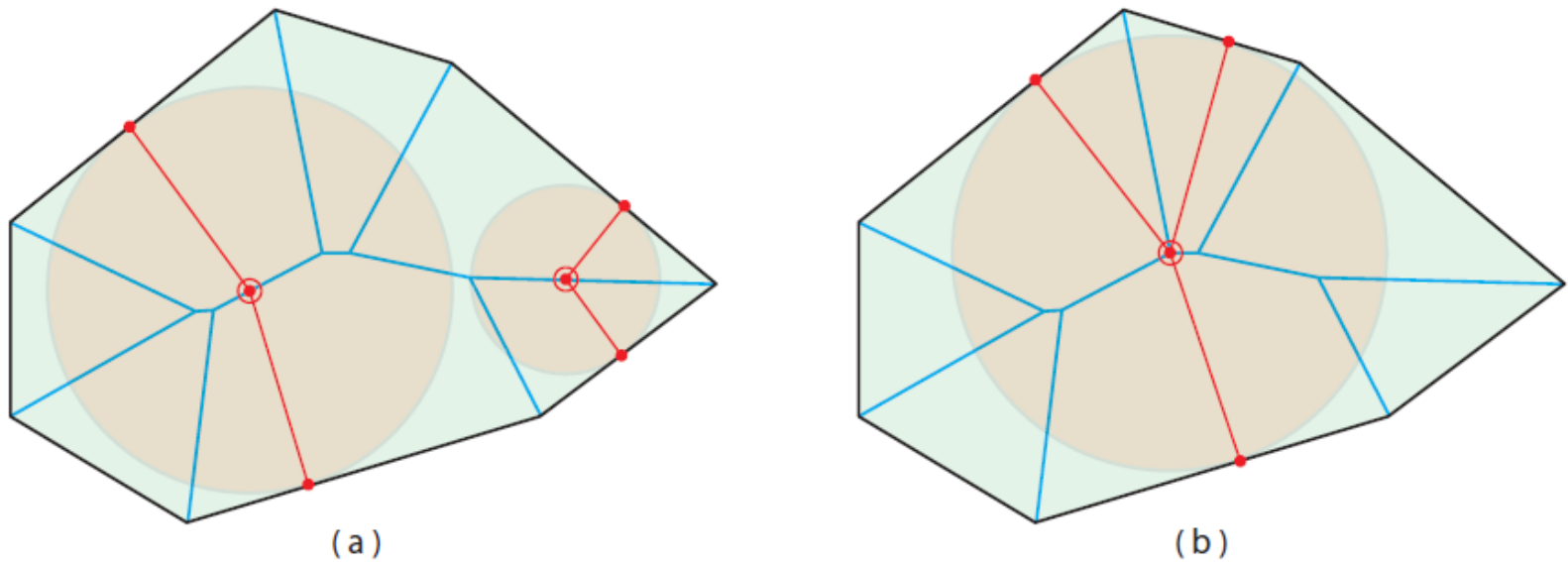
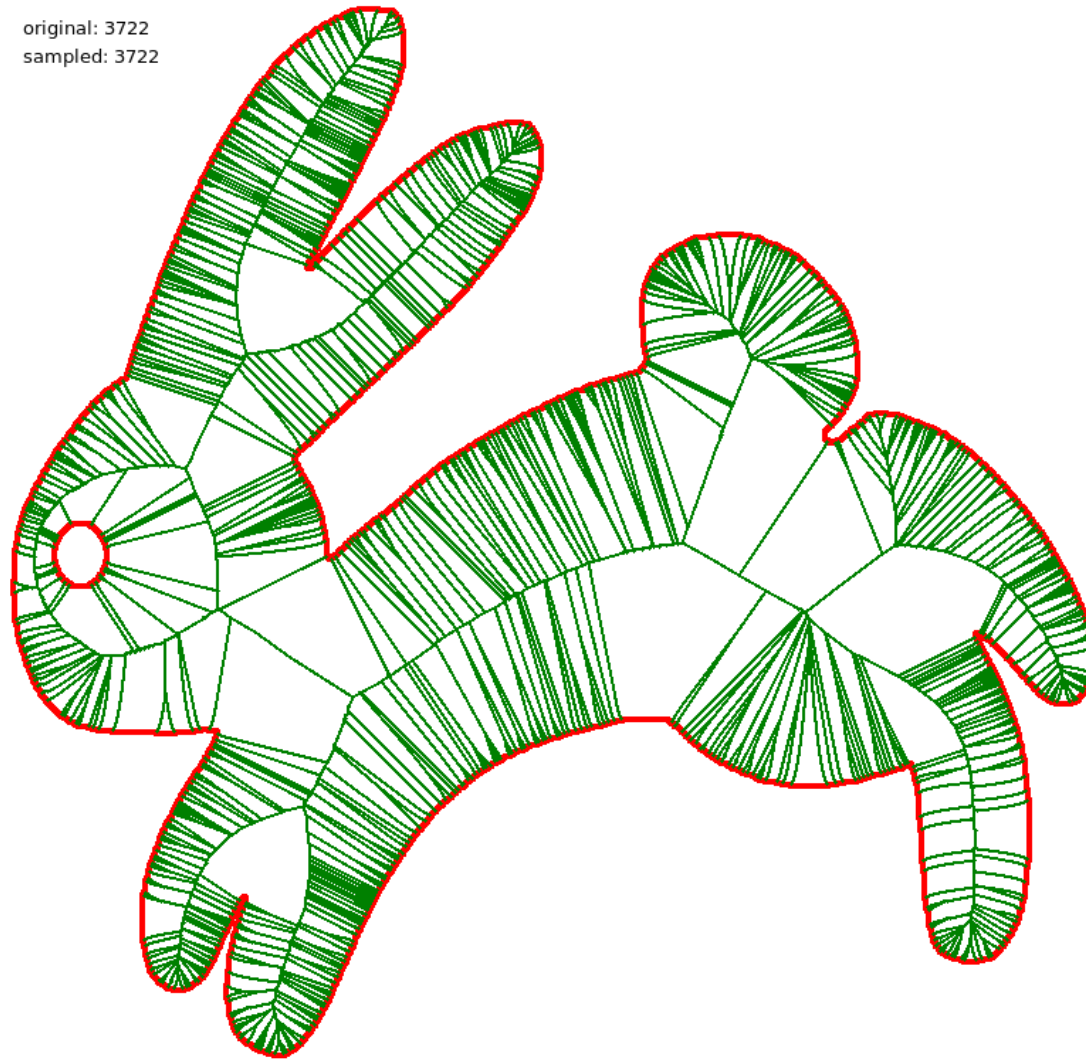


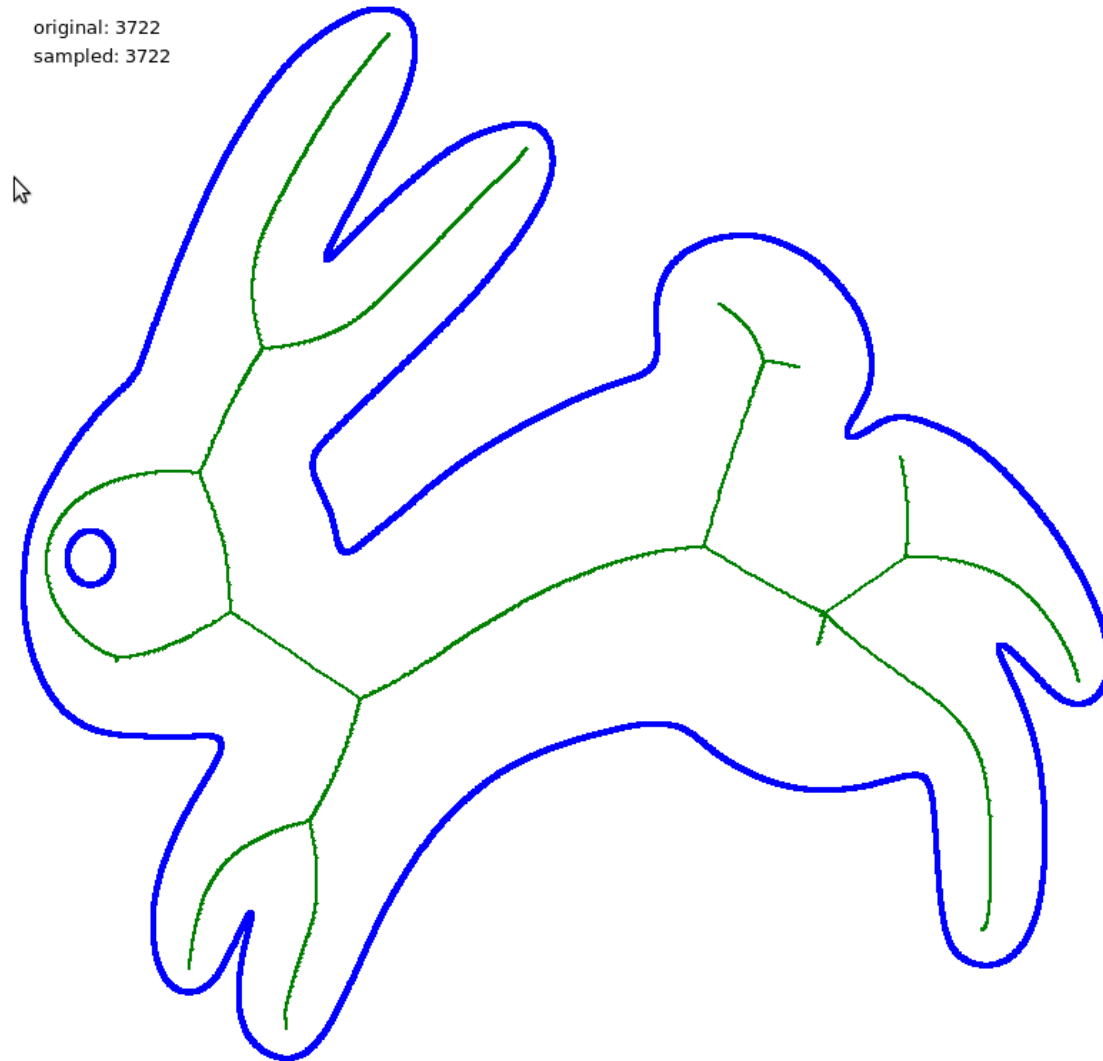
Figure 5.2: The maximal disks associated to (a) interior segments of $M(P)$ and (b) a degree-3 vertex of $M(P)$.

Medial axis vs Voronoi diagram

original: 3722
sampled: 3722



Medial axis vs Voronoi diagram



Looks familiar?

Skeleton Extraction by Mesh Contraction

Oscar Kin-Chung Au* Chiew-Lan Tai* Hung-Kuo Chu† Daniel Cohen-Or‡ Tong-Yee Lee†
*The Hong Kong Univ. of Science and Technology †National Cheng Kung University ‡Tel Aviv University

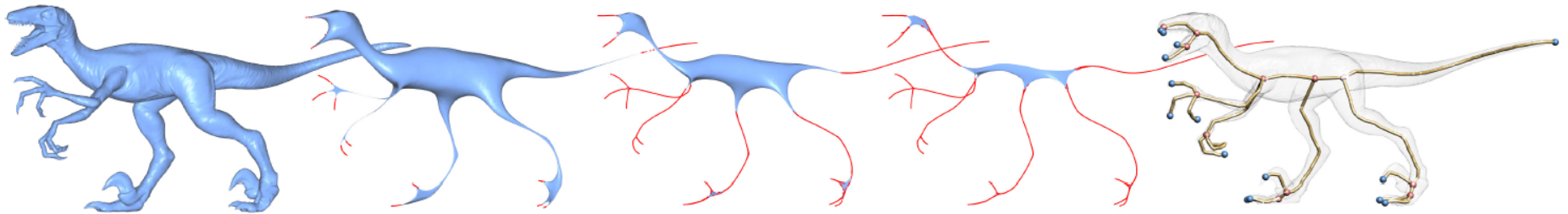


Figure 1: Our method extracts a 1D skeletal shape by performing geometry contraction using constrained Laplacian smoothing. Left to right are the original mesh and the results of the contraction after 1, 2 and 3 iterations. Faces with zero area are drawn in red. The rightmost ray-traced image shows the final skeleton after performing connectivity surgery and embedding refinement.