CS133
Computational Geometry

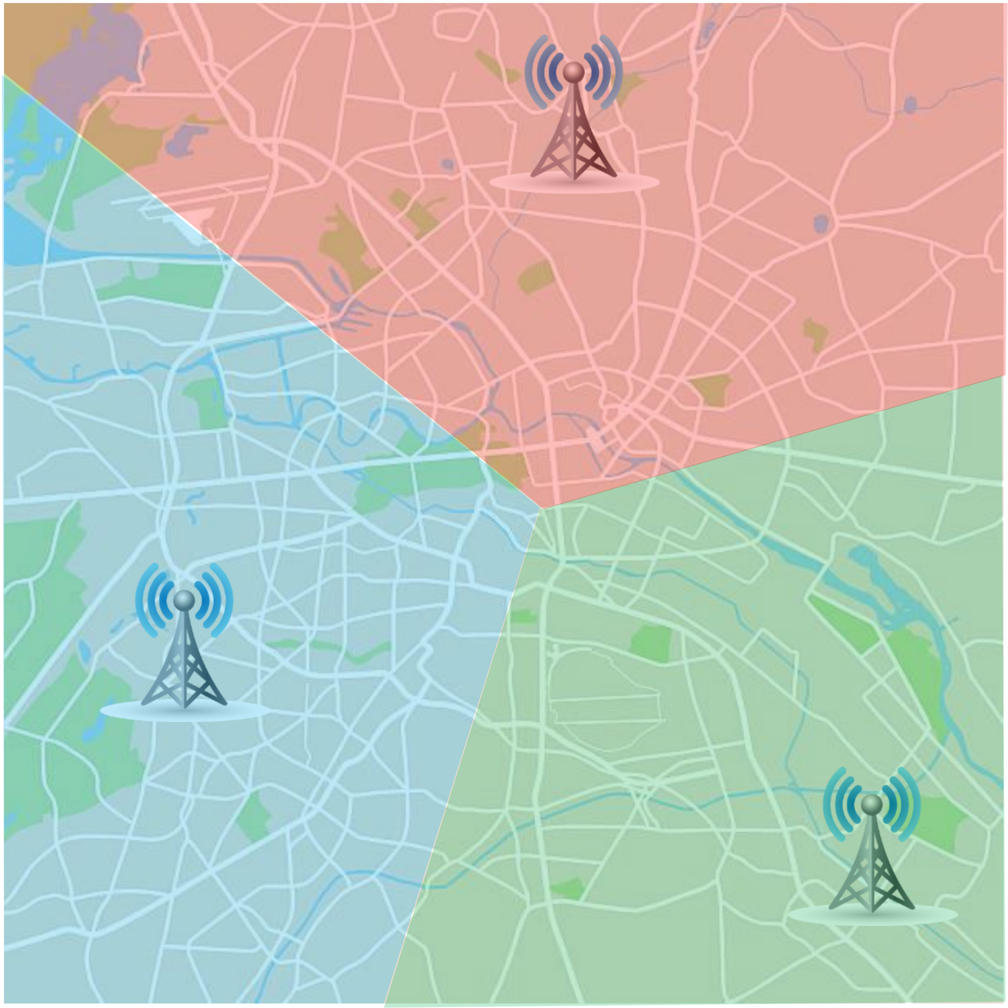
Voronoi Diagram
Delaunay Triangulation

Nearest Neighbor Problem

- Given a set of points P and a query point q, find the closest point $p \in P$ to q
- $\forall p, r \in P, dist(p, q) \leq dist(r, q)$
- Simple algorithm: Scan and find the minimum
- An efficient algorithm: Use a spatial index structure such as K-d tree
- What if we need to repeat this for every point in the space, i.e., an infinite number of points?

Application: Cell Coverage

Voronoi Diagram



Other Applications

- Service coverage for hospitals, post offices, schools, … etc.
- Marketing: Find candidate locations for a new restaurant
- Routing: How an electric vehicle should travel while staying close to charging stations

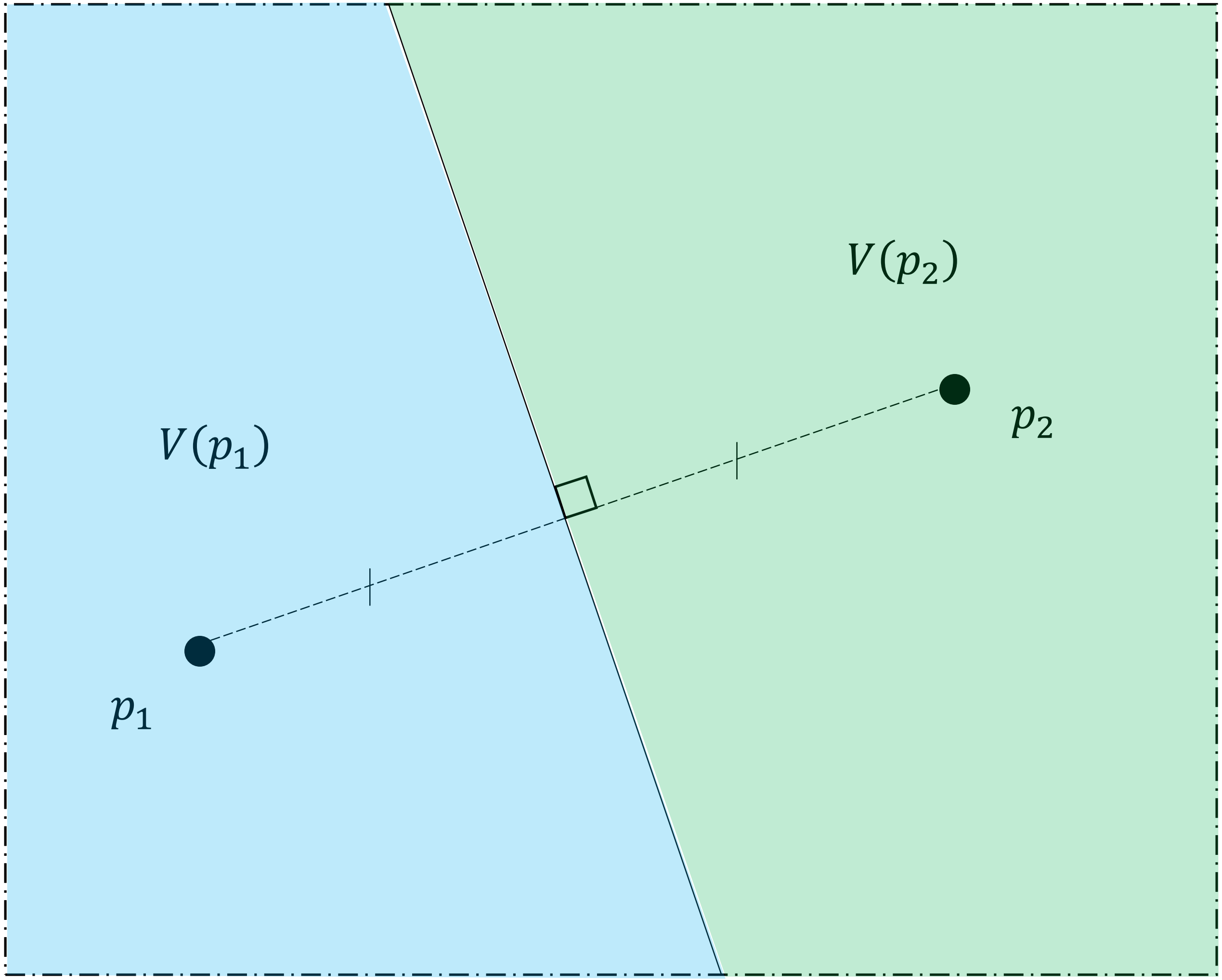
Voronoi Region

- Given a set P of points (also called sites), a Voronoi region (Voronoi face) of a site $p_i \in P$, $V(p_i)$ is the set of points in the Euclidean space where p_i is (one of) the closest sites
- $V(p_i) = \{x: \|p_i - x\| \leq \|p_j - x\| \forall p_j \in P\}$

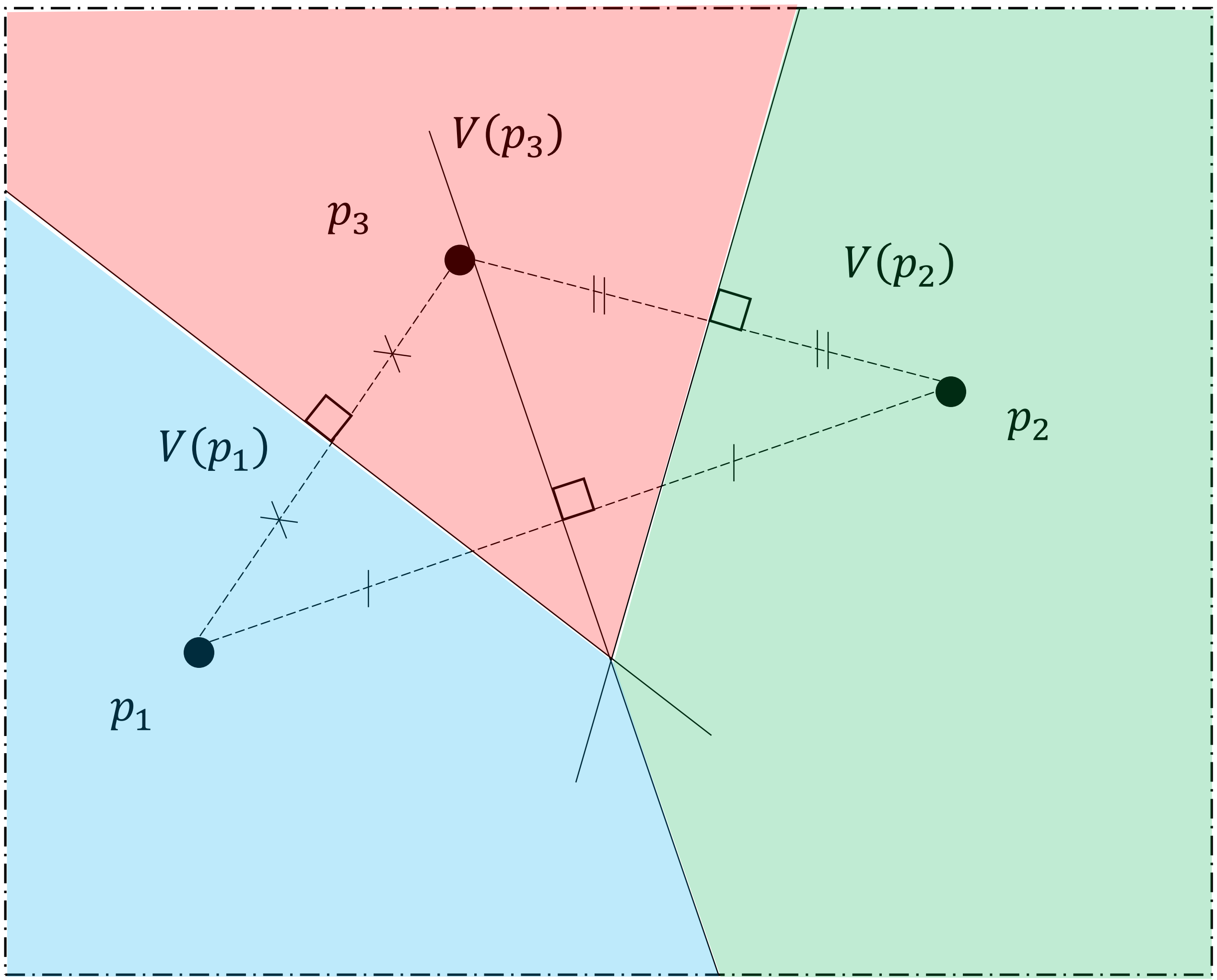
Voronoi Diagram

- The Voronoi diagram is the set of points that belong to two or more Voronoi regions
- Voronoi diagram is a *tessellation* of the space into regions where each region contains all the points that are closest to one site

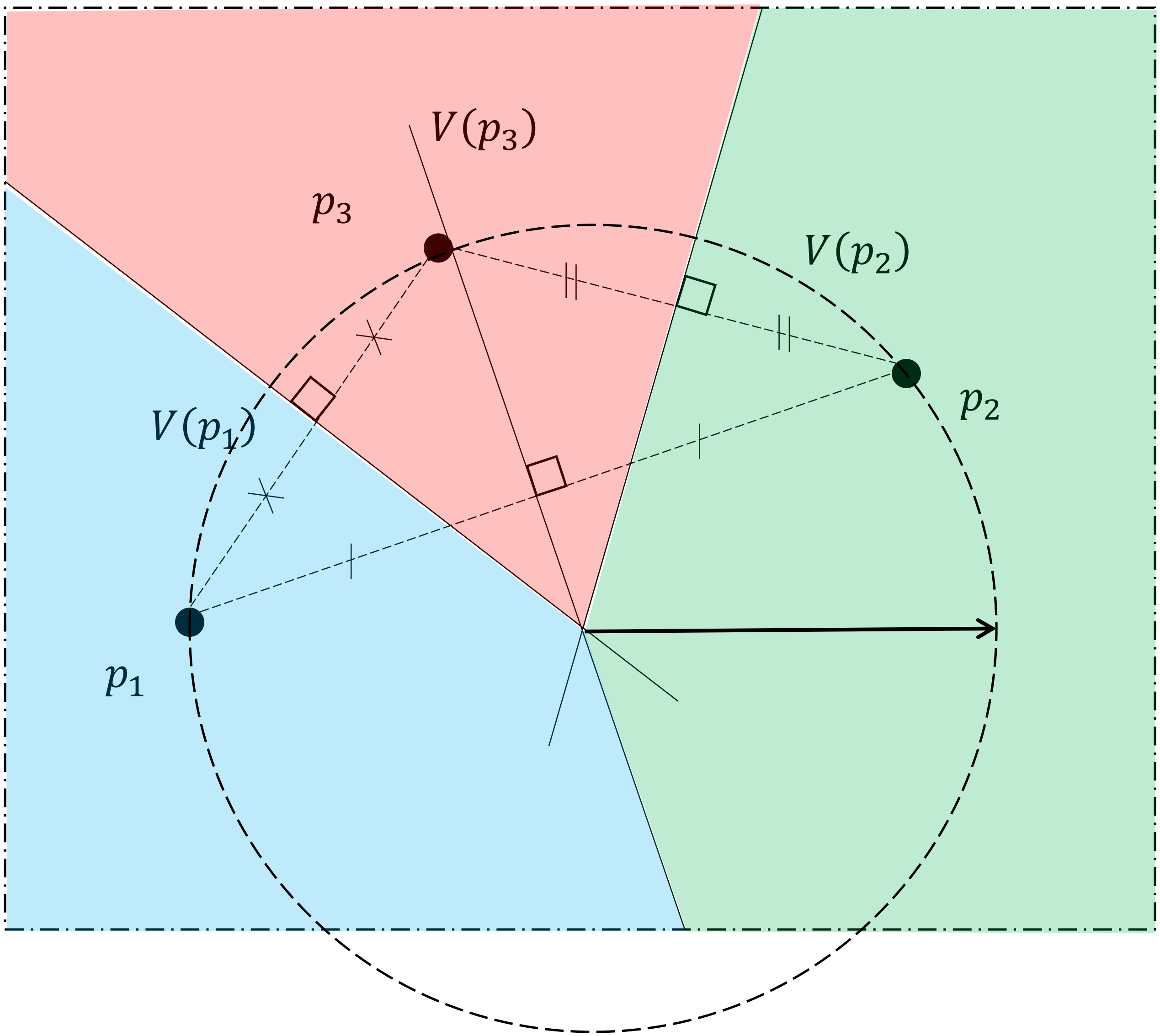
VD of Two Points



VD of Three Points

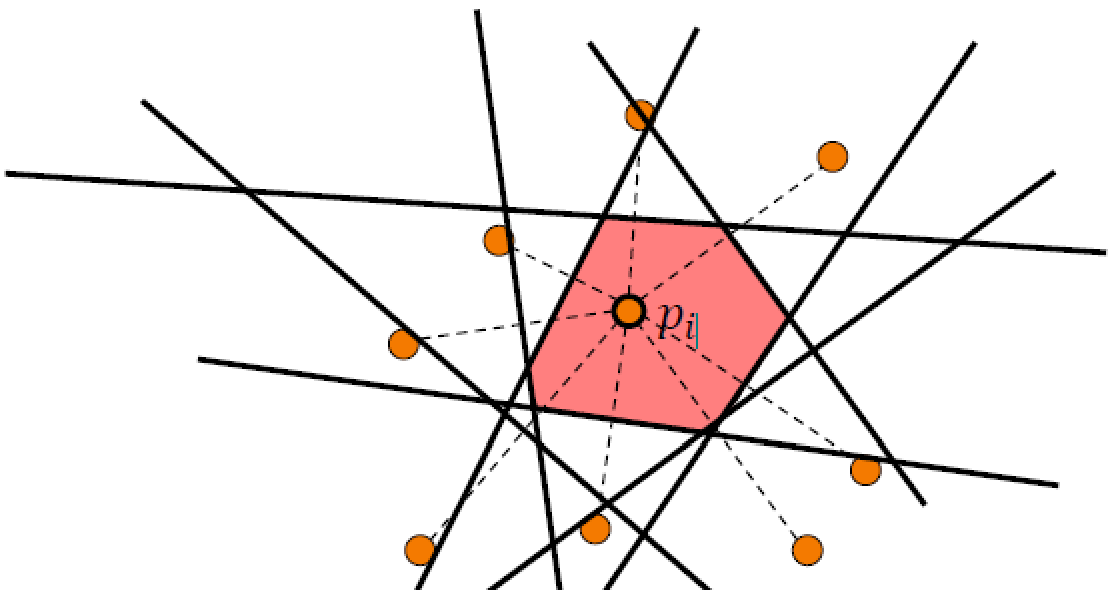


VD of Three Points

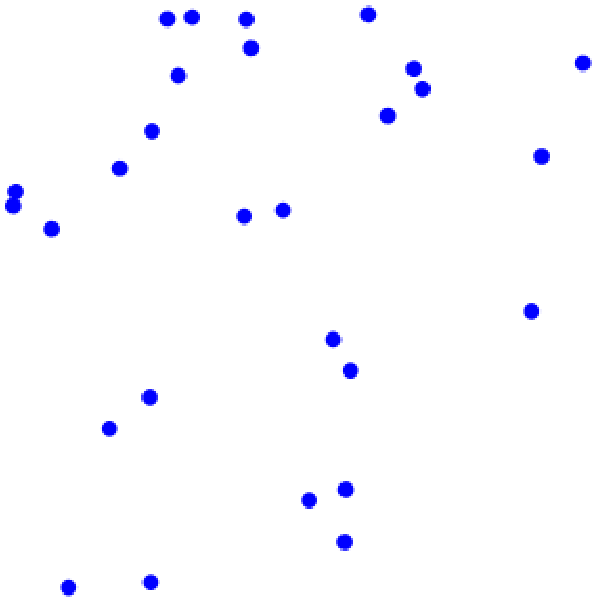


Voronoi Region

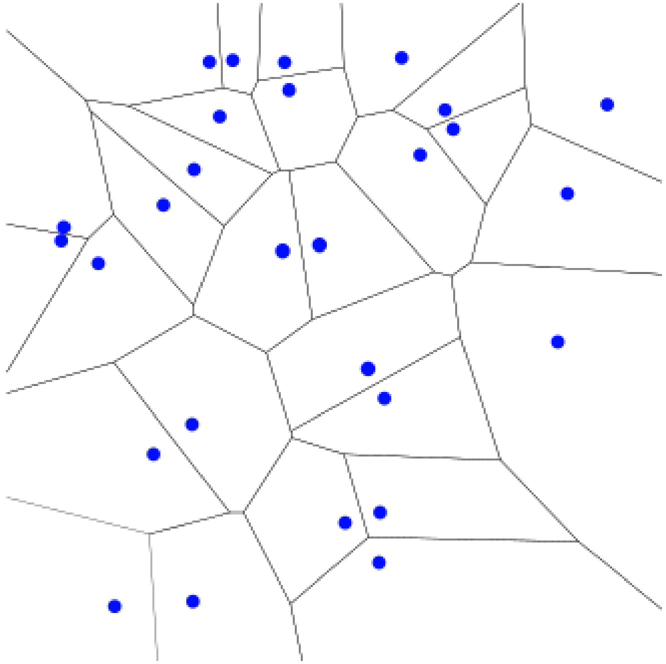
- A Voronoi region of a set p_i is the intersection of all half spaces defined by the perpendicular bisectors
- $V(p_i) = \cap_{j \neq i} H(p_i, p_j)$



VD of a Set of Points



P



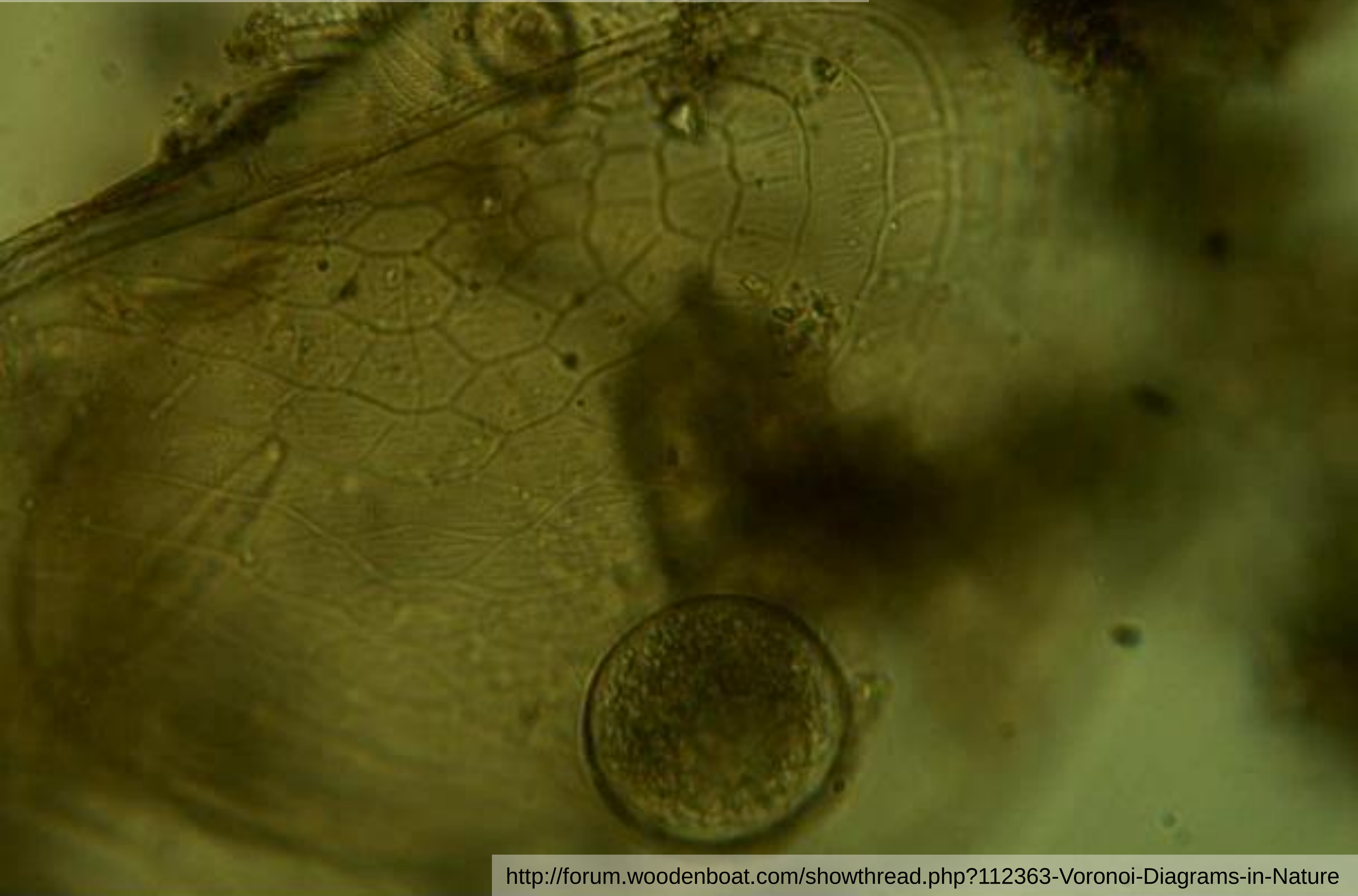
$VD(P)$

Mother Nature Loves VD

http://forum.woodenboat.com/showthread.php?112363-Voronoi-Diagrams-in-Nature

Mother Nature Loves VD

Mother Nature Loves VD



http://forum.woodenboat.com/showthread.php?112363-Voronoi-Diagrams-in-Nature

Mother Nature Loves VD

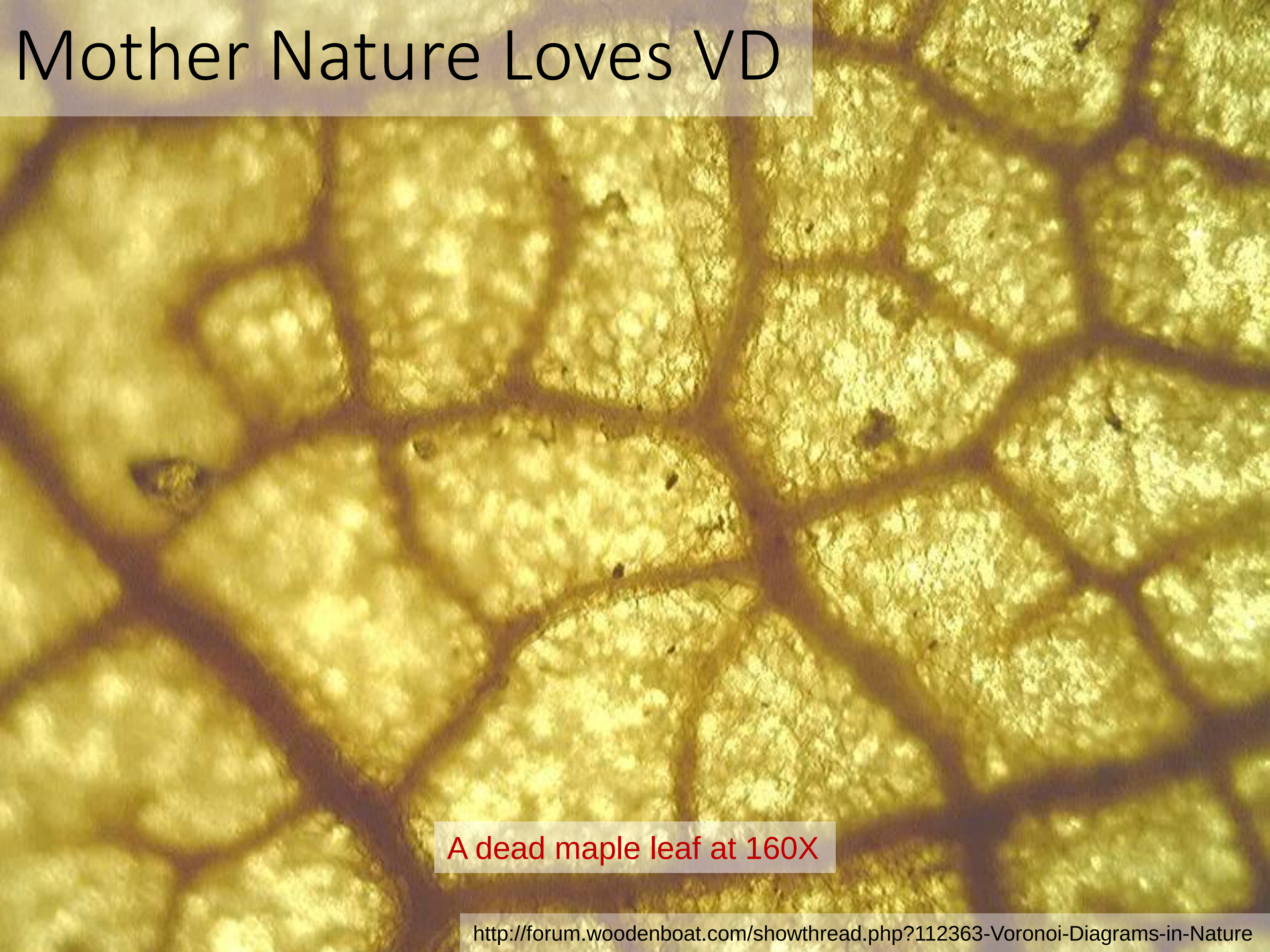
Onion cells under the microscope

Mother Nature Loves VD

A thin slice of carrot under the scope

http://forum.woodenboat.com/showthread.php?112363-Voronoi-Diagrams-in-Nature

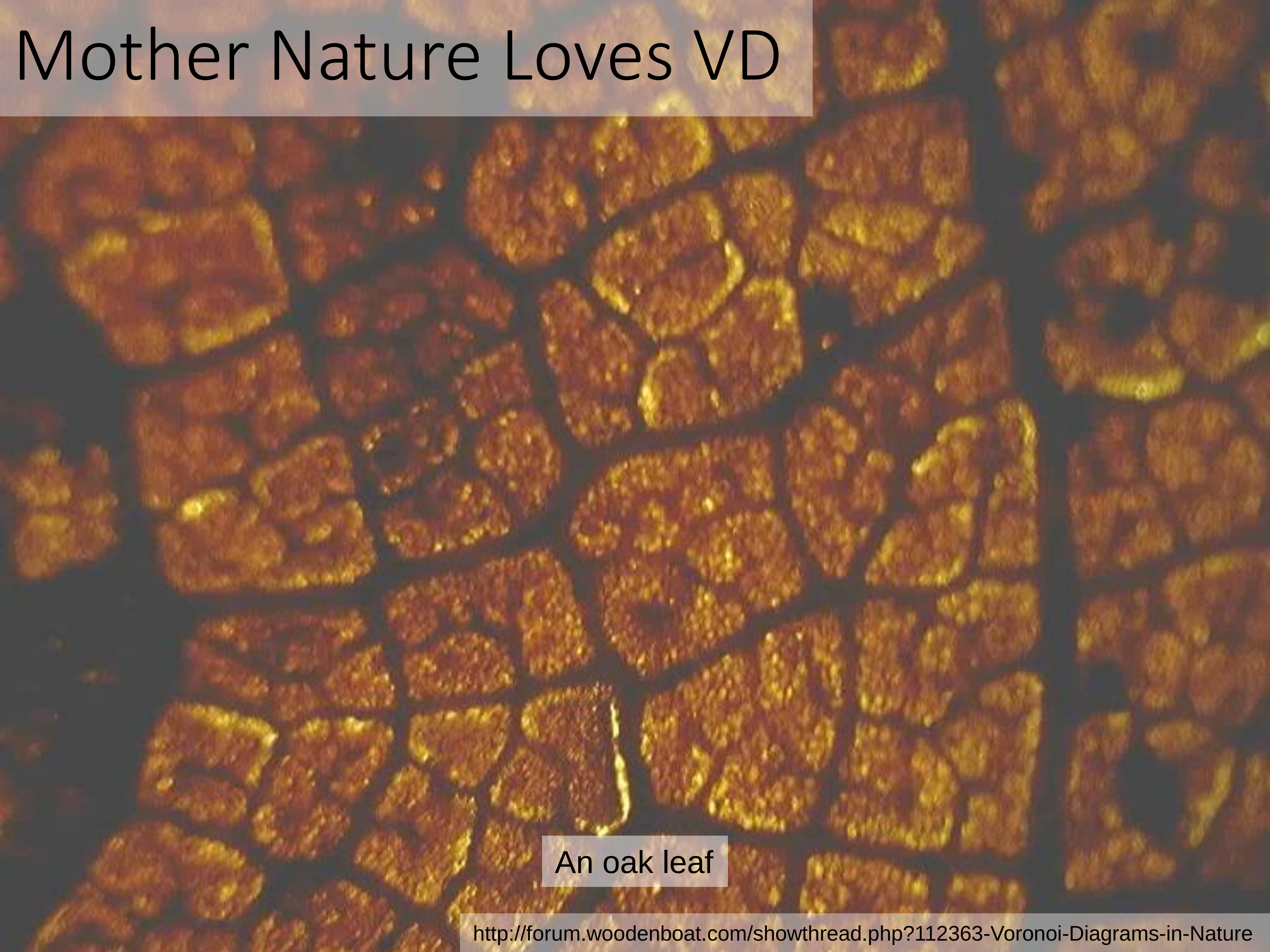
Mother Nature Loves VD



A dead maple leaf at 160X

http://forum.woodenboat.com/showthread.php?112363-Voronoi-Diagrams-in-Nature

Mother Nature Loves VD

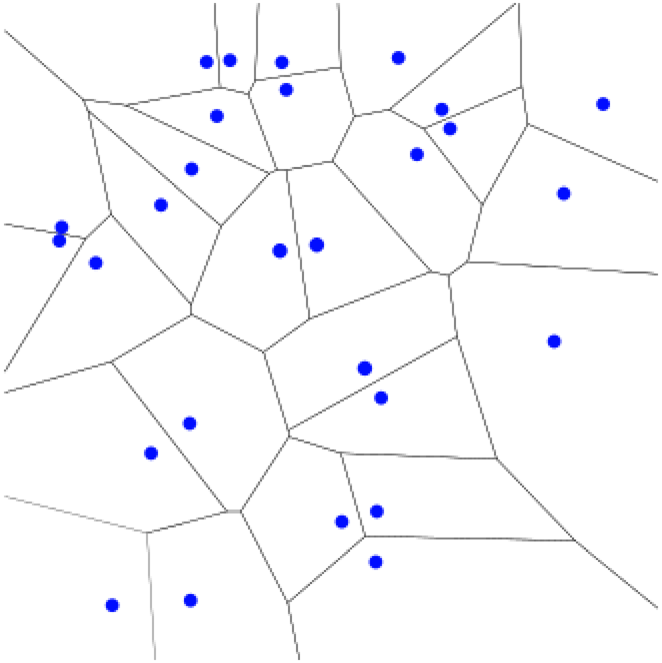


An oak leaf

http://forum.woodenboat.com/showthread.php?112363-Voronoi-Diagrams-in-Nature

VD Properties

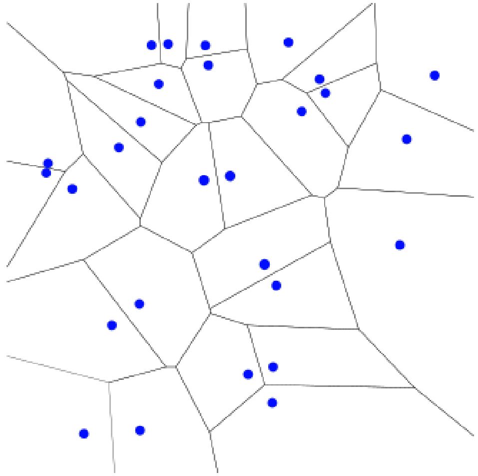
- Voronoi regions are convex
- Each Voronoi region contains a single site
- Voronoi regions (faces) can be unbounded
- Most intersection points connect three segments



Voronoi Diagram

VD Properties

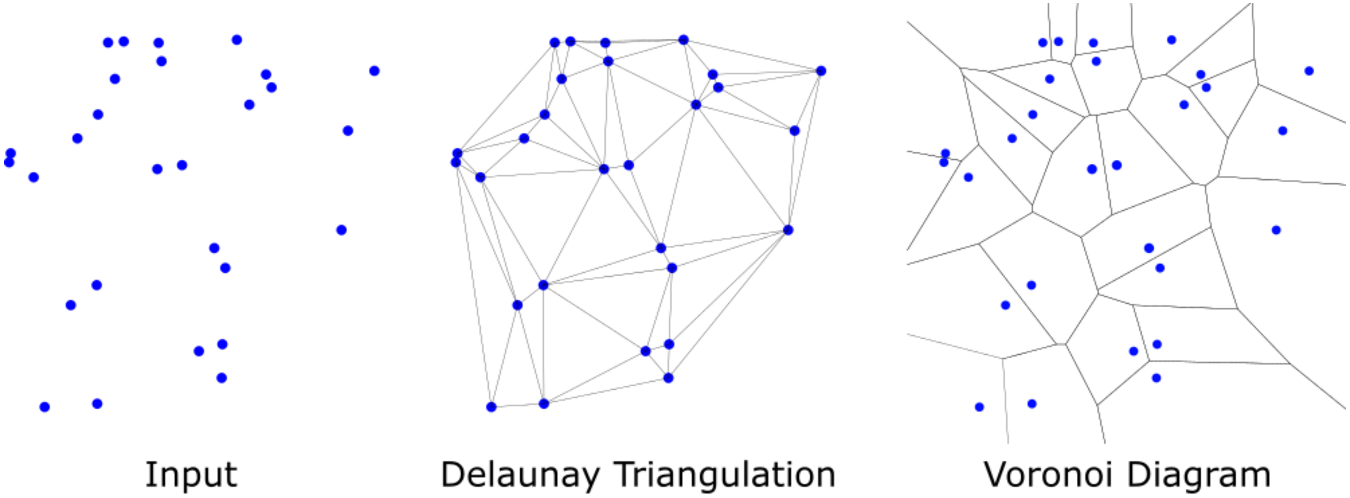
- $V(p_i)$ is unbounded iff $p_i \in \mathcal{CH}(P)$
- If a point x is at the intersection of three or more Voronoi regions, say $V(p_1), V(p_2), \dots, V(p_k)$, then x is the center of a circle C that have $p_1, \dots, p_k$ at its boundary
- C contains no other sites
- VD is unique



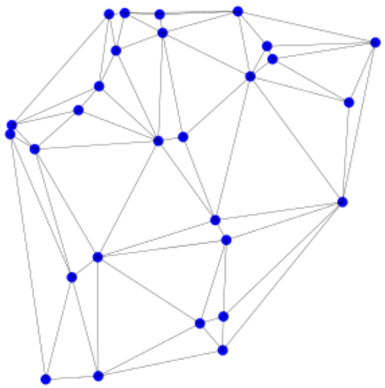
Voronoi Diagram

Delaunay Triangulation (DT)

- Delaunay triangulation is the straight-line dual of the Voronoi diagram
- Each site is a corner of at least one triangle
- Each two Voronoi regions that share an edge are connected with an edge in DT



DT Properties



- The edges of $D(P)$ do not intersect
- Is $D(P)$ unique?
  - Yes, if no four sites are co-circular
- If p_i and p_j are the closest pair of sites, they are connected with an edge in DT
- If p_i and p_j are nearest neighbors, they are connected with an edge in DT
- The circumcircle of p_i, p_j, and p_k is empty $\Leftrightarrow$ (p_i, p_j, p_k) is a triangle in DT

DT is a Planar Graph

- Since the edges in DT do not intersect, they form a planar graph
  - The number of edges/faces in a Delaunay Triangulation is linear in the number of vertices.
  - The number of edges/vertices in a Voronoi Diagram is linear in the number of faces.
  - The number of vertices/edges/faces in a Voronoi Diagram is linear in the number of sites.

Theorem 7.3

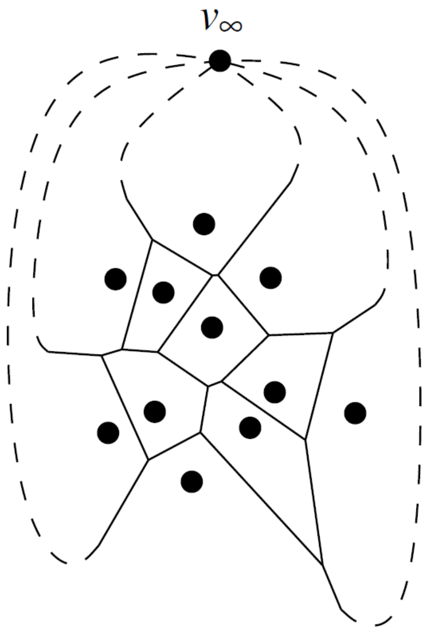
- For $n \geq 3$, the number of vertices in the Voronoi diagram (n_v) of a set of n point sites in the plane is at most $2n - 5$, and the number of edges n_e is at most $3n - 6$

Proof

- For any connected graph G
- Euler's rule: $m_v - m_e + m_f = 2$
  - m_v: Number of vertices (nodes)
  - m_e: Number of edges (arcs)
  - m_f: Number of faces
- $(n_v + 1) - n_e + n = 2$
- Each edge connects two vertices
- The sum of degrees of vertices

$$\sum d(v_i) = 2n_e$$

- $d(v_i) \geq 3$

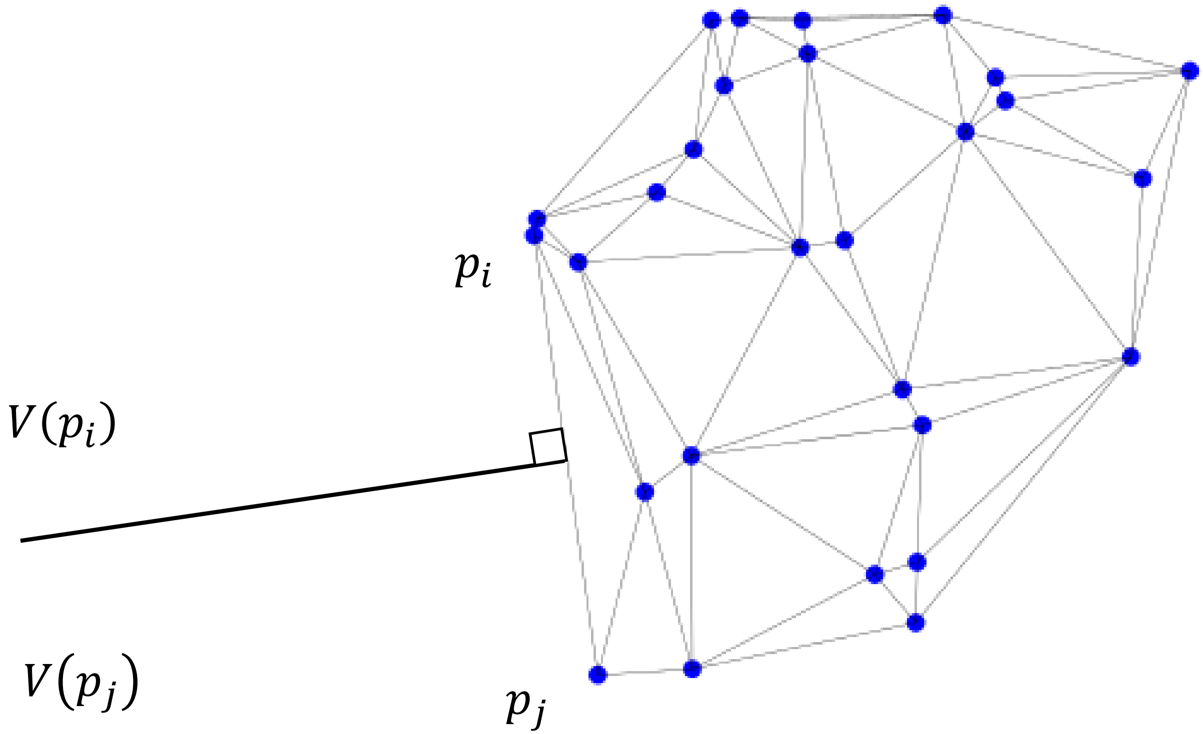


Proof (cont'd)

- $3n_v \le \sum d(v_i)$
- $3(n_v + 1) \le 2n_e$
- $(n_v + 1) \le \frac{2}{3} n_e$
- But: $(n_v + 1) - n_e + n = 2$
- $(n_v + 1) = 2 - n + n_e \le \frac{2}{3} n_e$
- $\frac{1}{3} n_e \le n - 2$
- $n_e \le 3n - 6$
- $n_v \le 2n - 5$

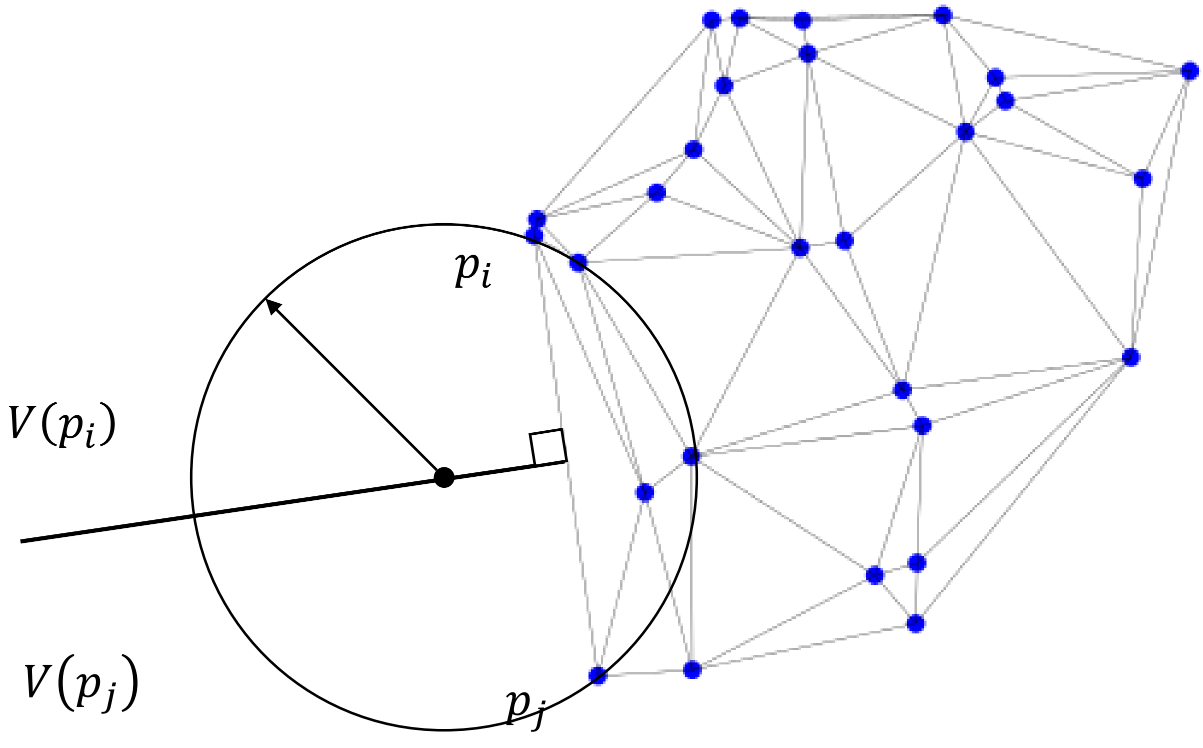
DT Properties

- The boundary of $D(P)$ is the convex hull of P



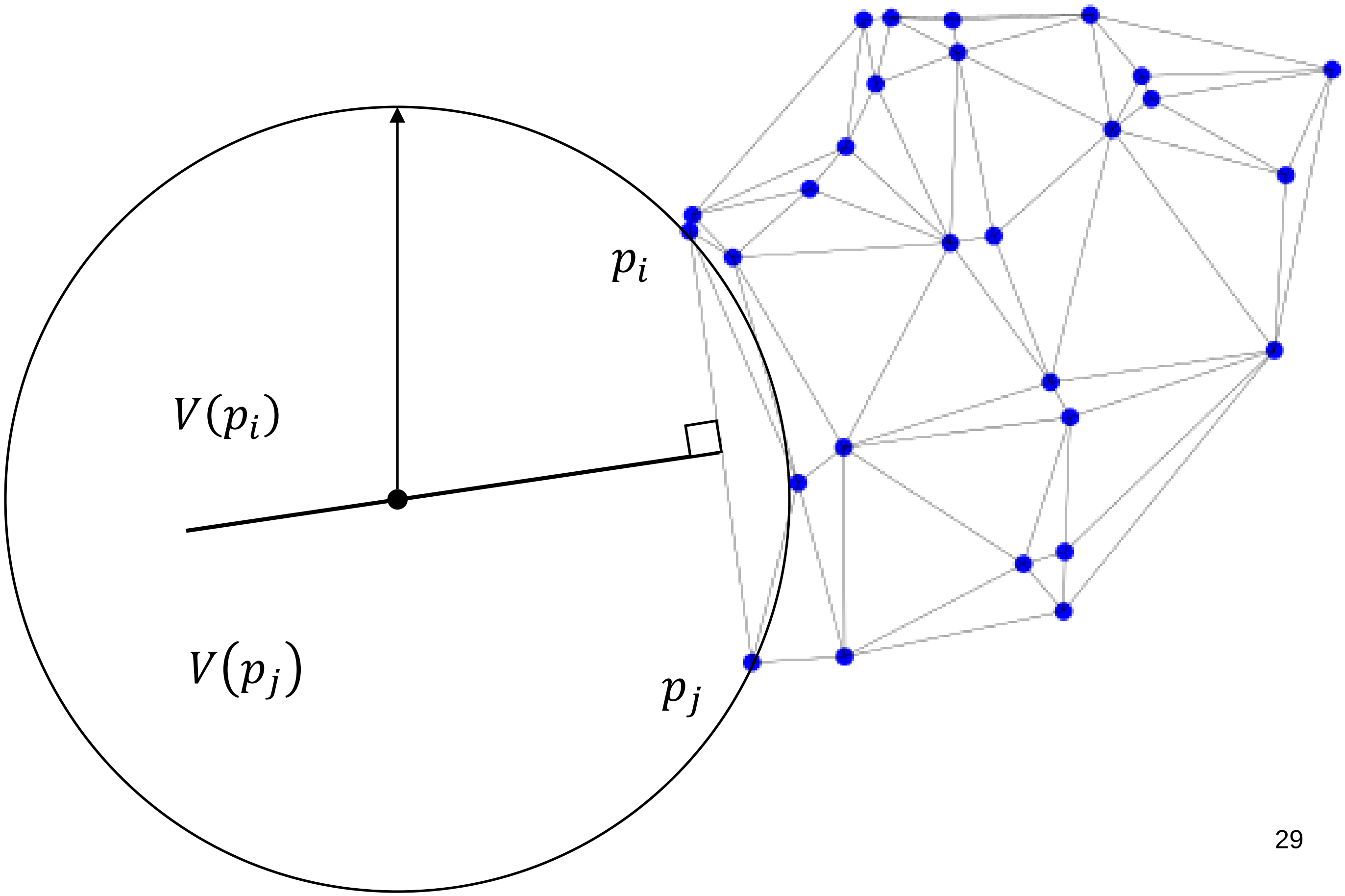
DT Properties

- The boundary of $D(P)$ is the convex hull of P



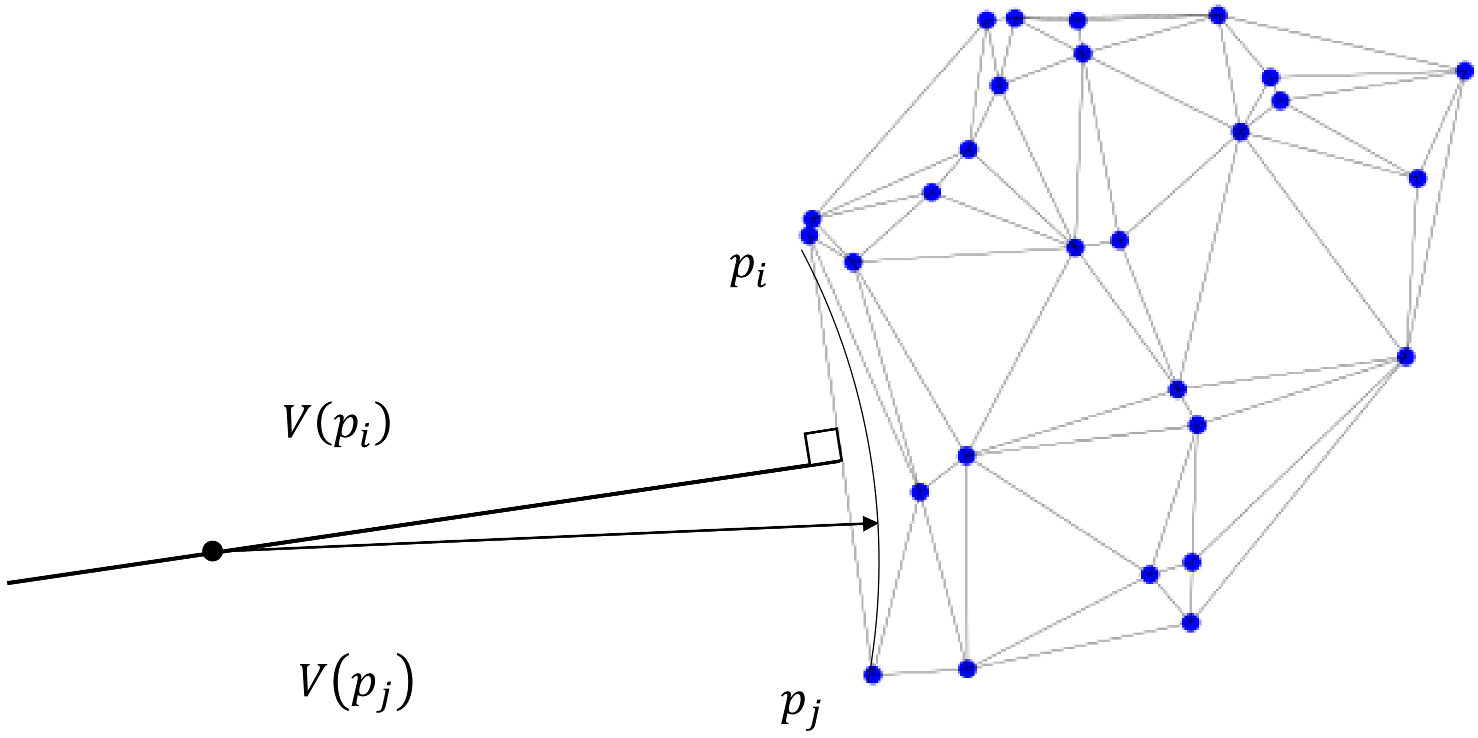
DT Properties

- The boundary of $D(P)$ is the convex hull of P



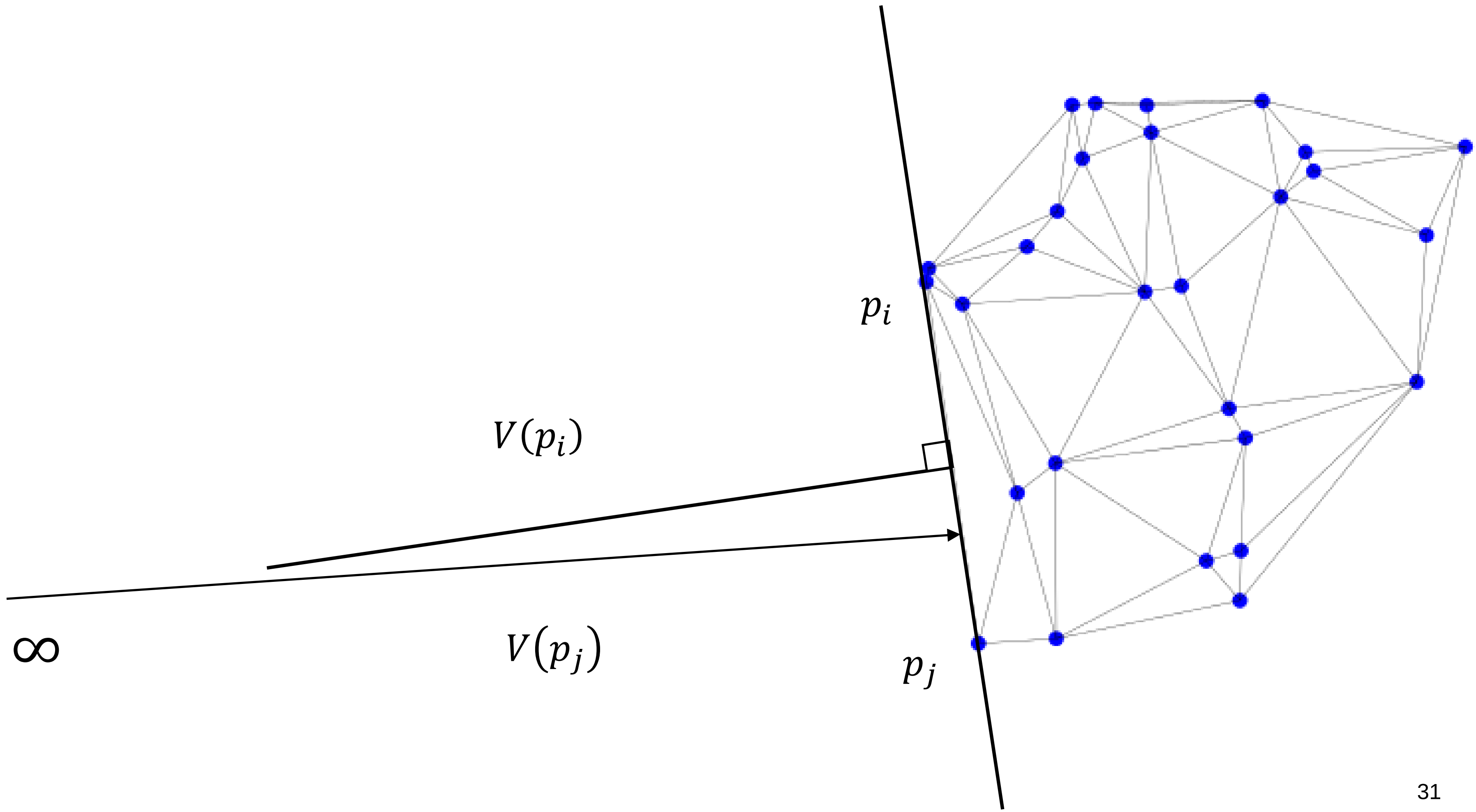
DT Properties

- The boundary of $D(P)$ is the convex hull of P



DT Properties

- The boundary of $D(P)$ is the convex hull of P

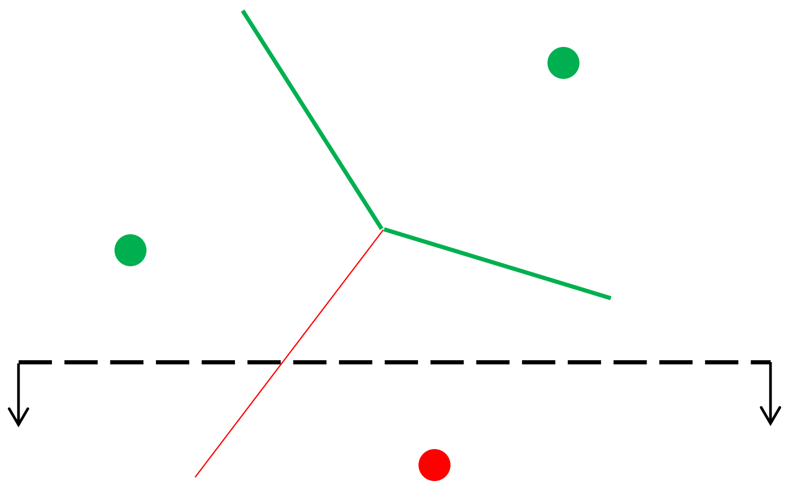


DT Properties

- If p_j is the nearest neighbor of p_i then $\overline{p_i p_j}$ is a Delaunay edge
- p_j is the nearest neighbor of p_i iff. the circle around p_i with radius $\left|p_i - p_j\right|$ is empty of other points.
- $\Rightarrow$The circle through $\left(p_i + p_j\right)/2$ with radius $|p_i - p_j|/2$ is empty of other points.
- $\Rightarrow \left(p_i + p_j\right)/2$ is on the Voronoi diagram.
- $\Rightarrow \left(p_i + p_j\right)/2$ is on a Voronoi edge.

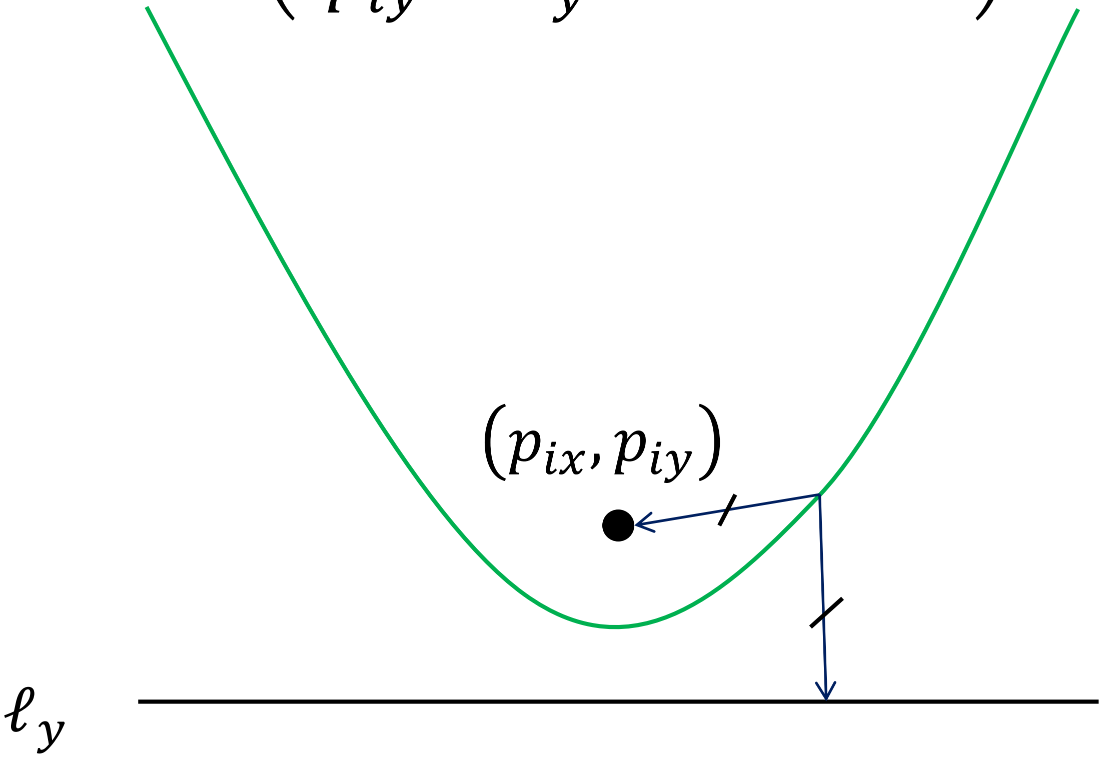
VD Plane Sweep

- Scan the plane from top to bottom
- Compute the VD of the points above the sweep line
- Is it that simple?

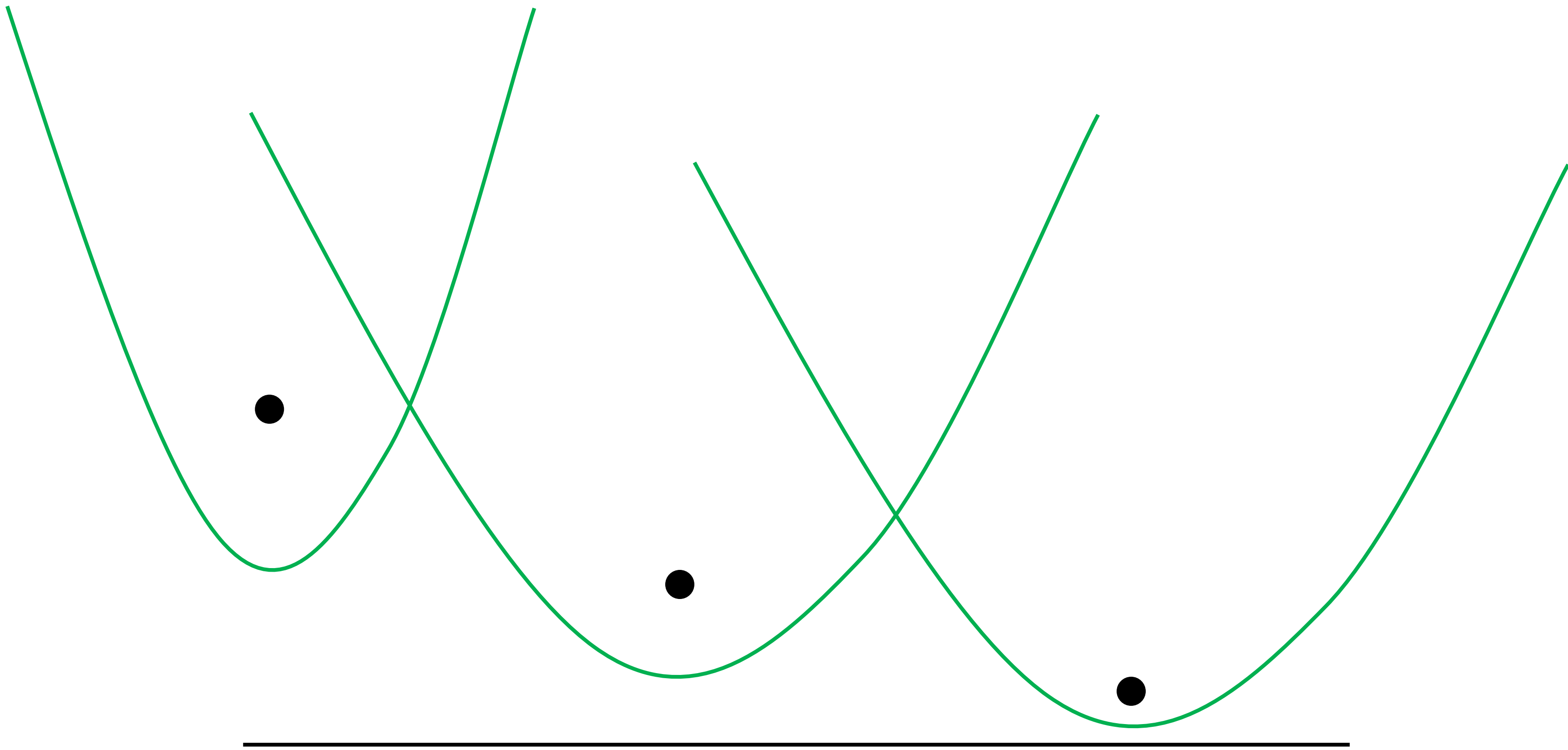


VD of a Line and a Point

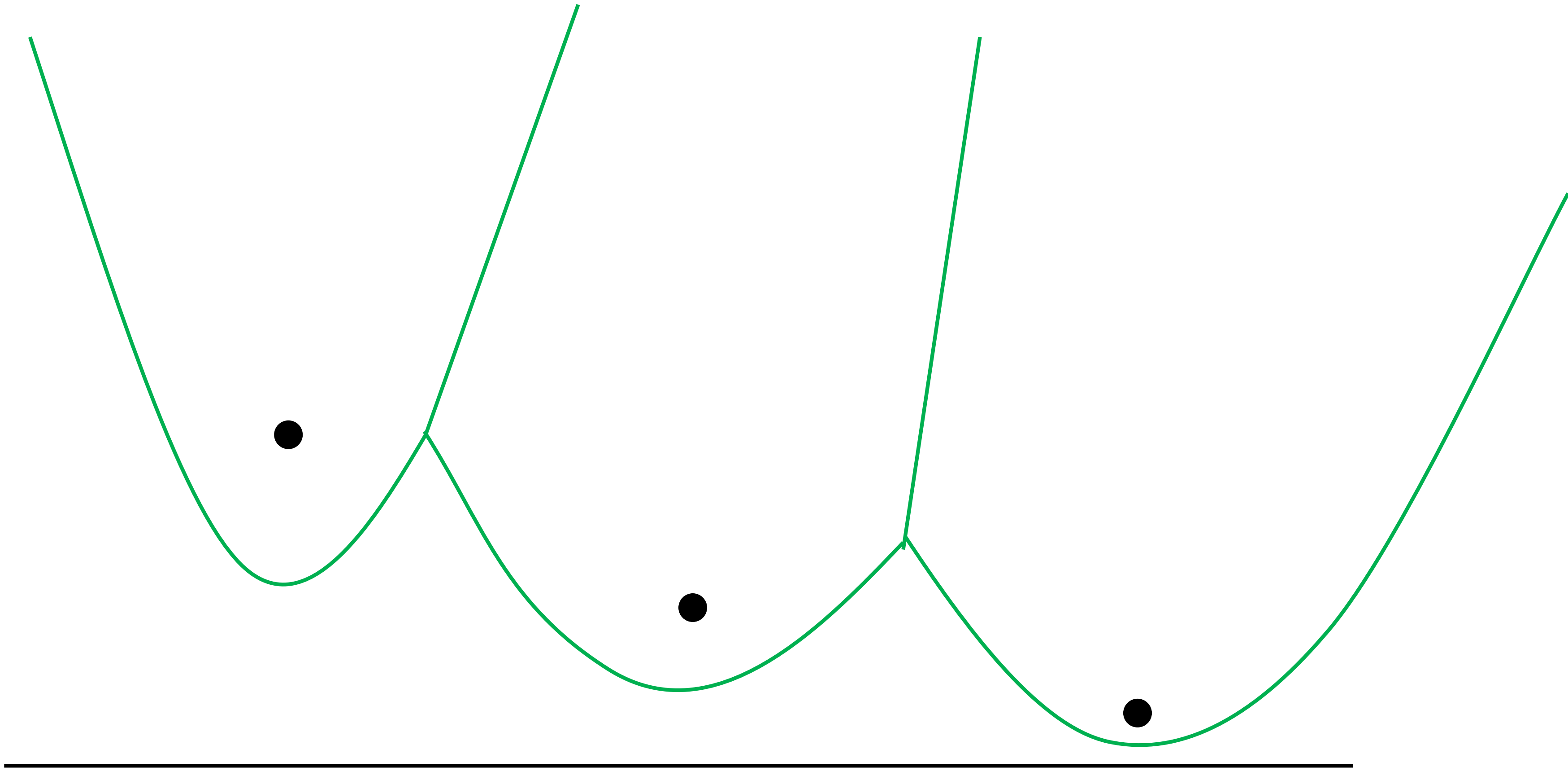
$$y = \frac{1}{2}\left(\frac{(x - p_{ix})^2}{p_{iy} - \ell_y} + \ell_y + p_{iy}\right)$$



VD of a Line and a n Points

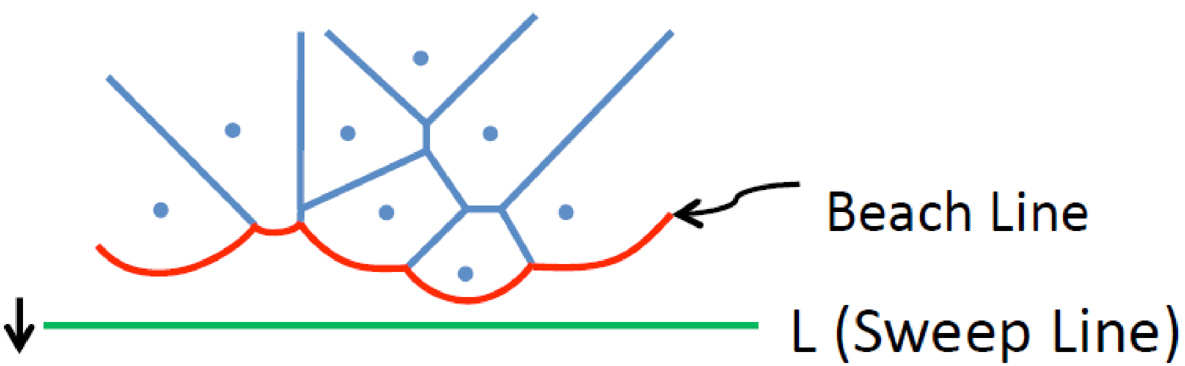


VD of a Line and a n Points

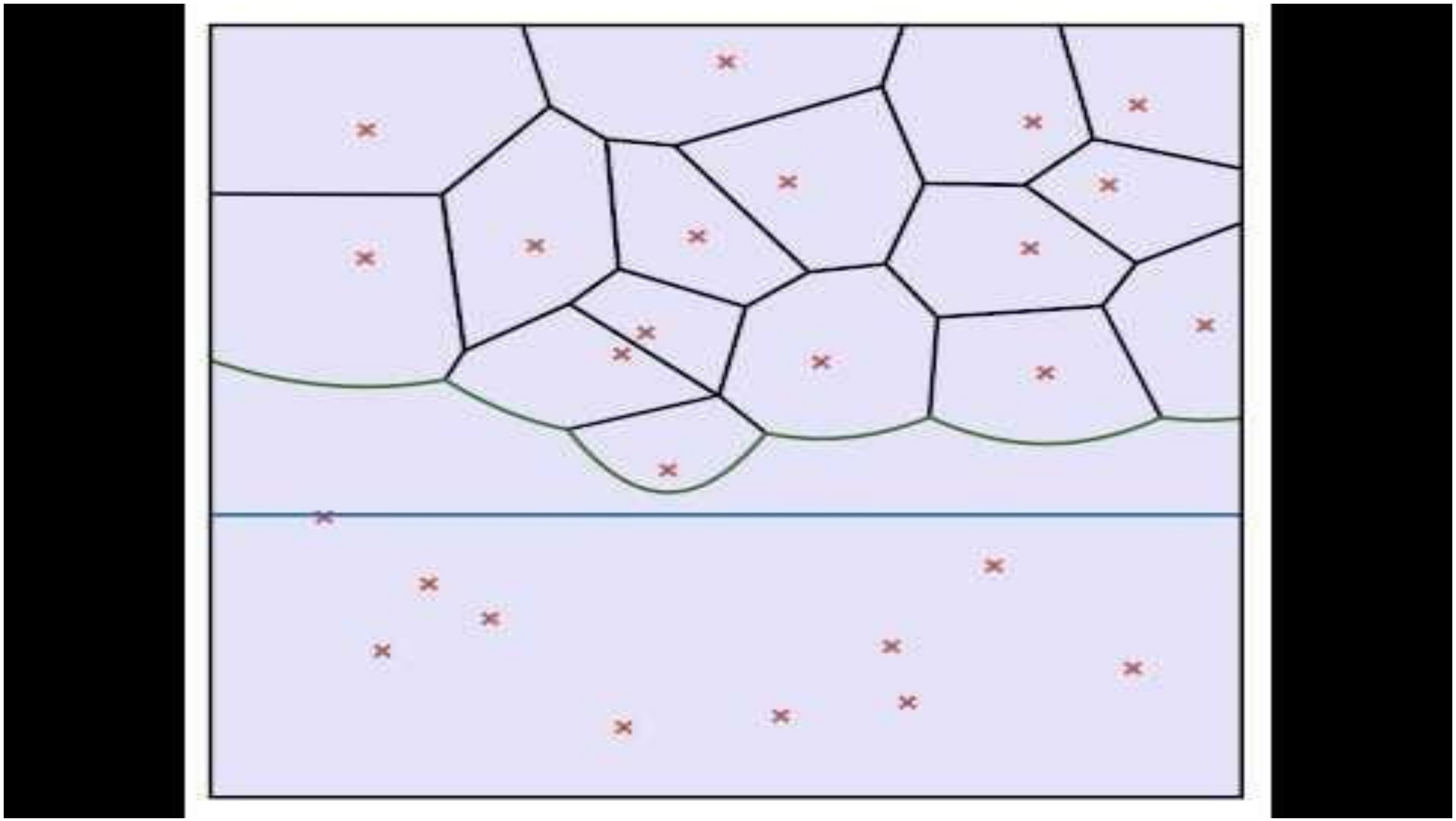


Fortune's Algorithm

- As the line sweeps the plane, the algorithm maintains the VD of the set of points and the sweep line
- Since the sweep line is closer than any future point, it acts as a *barrier* that isolates the VD from all future points



Fortune's Algorithm in Action



VD Properties

- The VD part above the beach line (blue) is final. Why?
  - This area is closer to some site than the beach line
  - … closer to some site than any future site
  - We already know the nearest site to those areas

VD Properties

- The beach line is x-monotone. Why?
  - Each parabola is x-monotone
  - At each x-coordinate, the beach line takes one value which is the minimum of all the parabolas
  - Therefore, it is x-monotone

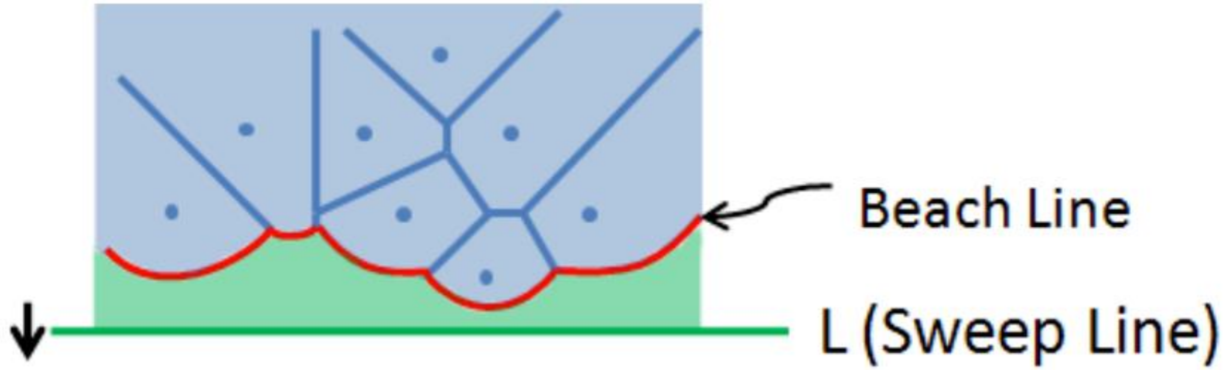
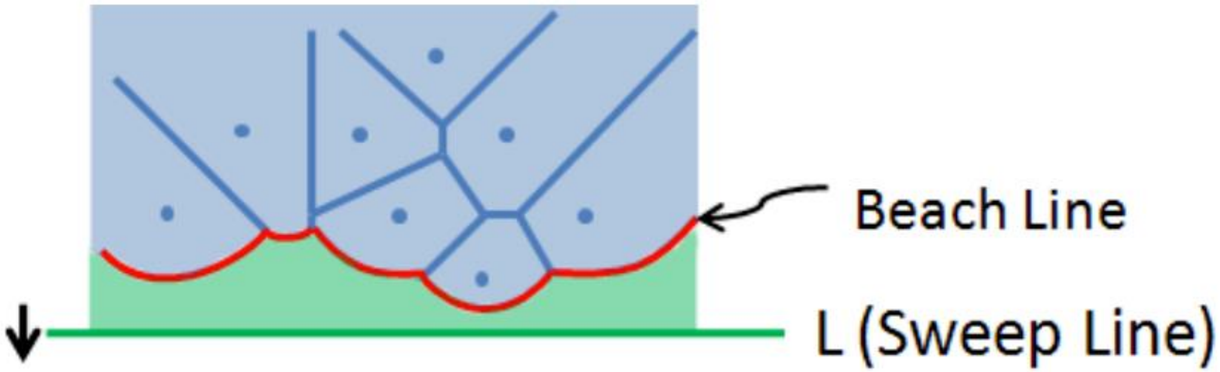


Figure Credits: http://www.cs.sfu.ca/~binay/813.2011/Fortune.pdf

VD Properties

- The breakpoints of the beach line lie on Voronoi edges of the final diagram
  - Each breakpoint is equidistant from two sites
  - A breakpoint is as close to some site as to the sweep line
  - The sweep line is (closer) to the blue sites than future sites



Fortune’s Algorithm

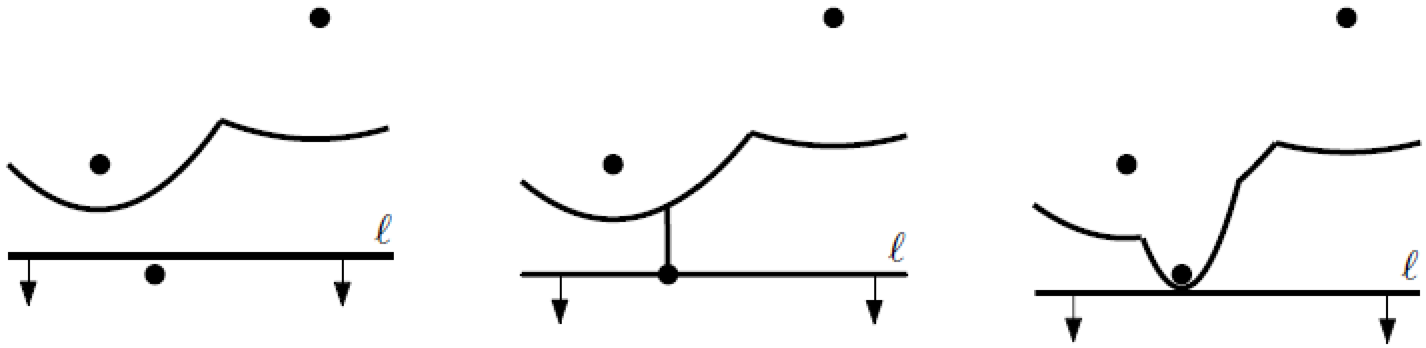
- Move the sweep line downwards and update the VD as the line moves
- When the line reaches $-\infty$, we will have our final VD. (Because any point in the space is closer to some site than $y = -\infty$)
- Note: We never create the beach line explicitly. We only maintain enough information that allows us to reconstruct parts of it when we need them

Beach Line Changes

- How can the beach line change (topologically)
  - A new arc appears
  - An existing arc is removed

Site Event

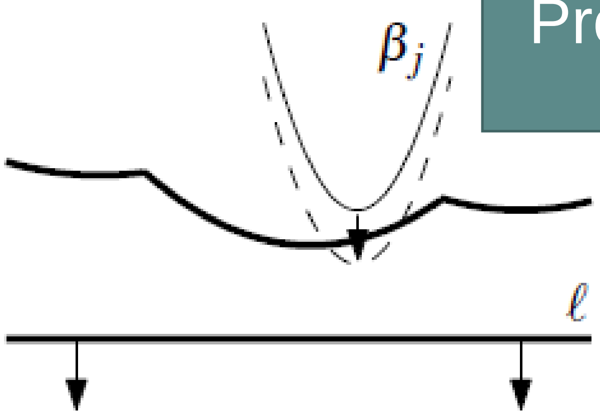
- When the sweep line hits a new site
- Where are the points that are equi-distant from the new site and the sweep line?
- A vertical line that crosses the new site



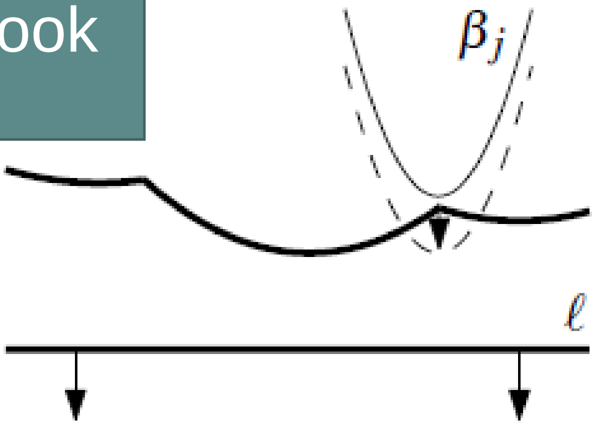
Site Event

- Lemma: *The only way in which a new arc can appear on the beach line is through a site event*
- Proof by contradiction

Proof is in the book



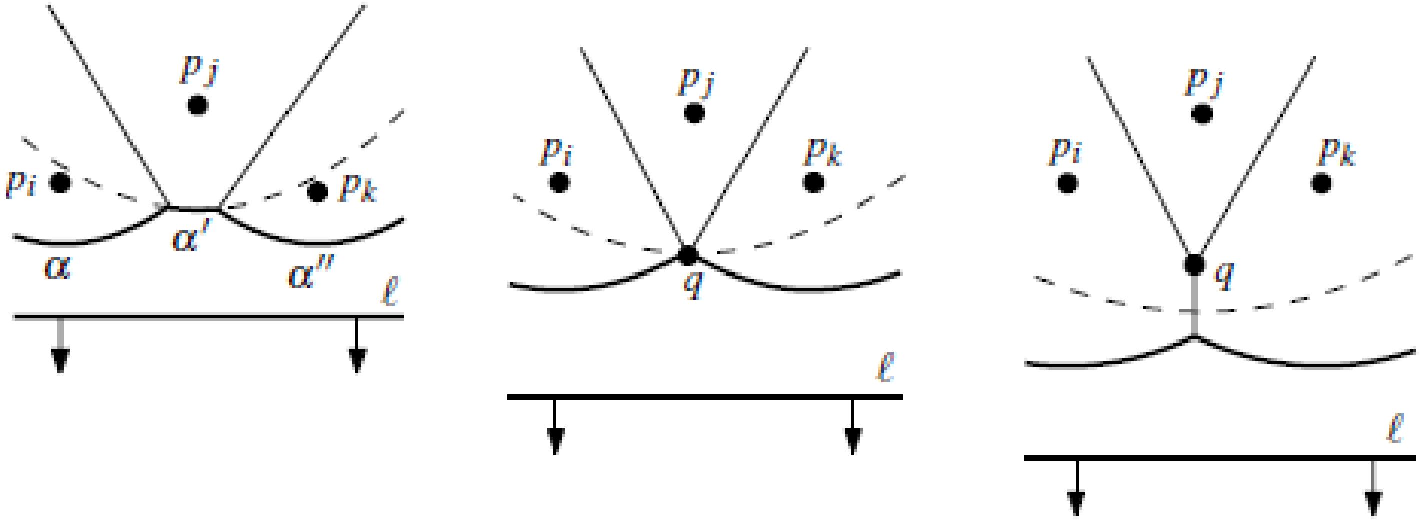
Case 1: An existing arc β_j breaks through the middle of an existing arc β_i



Case 2: An existing arc β_j appears in between two arcs

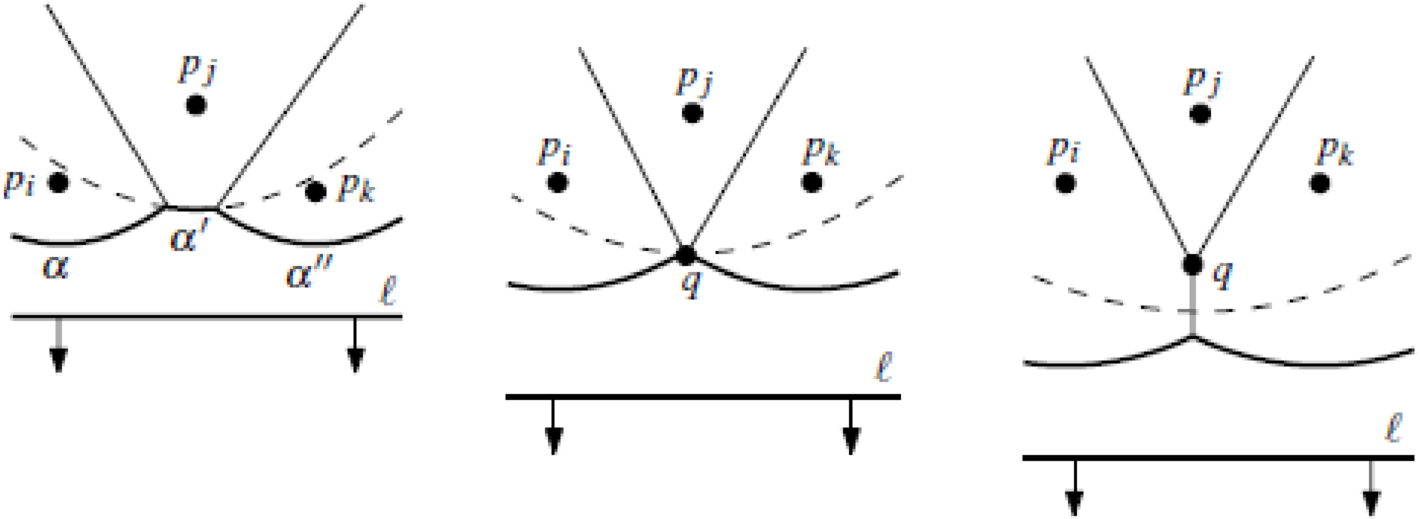
Circle (Vertex) Event

- An existing arc shrinks into a point and disappears
- This happens when three (or more) sites become closer to a point than the sweep line *shielding* the point from the sweep line



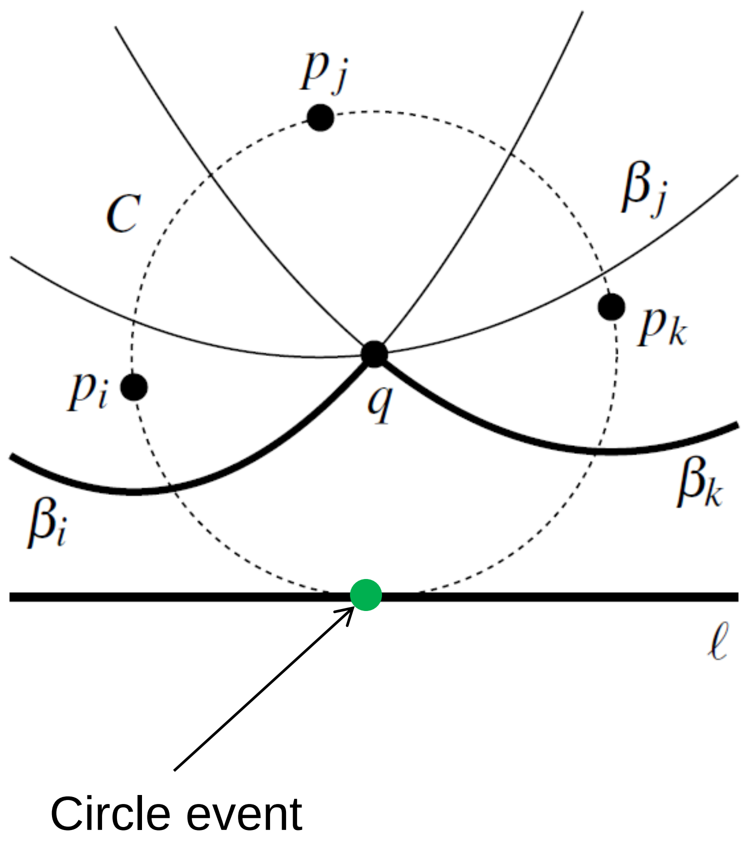
Circle (Vertex) Event

- The sweep line will only go further down while the points stay
- This results in a vertex on the Voronoi Diagram
- Lemma: The only way in which an existing arc can disappear from the beach line is through a circle event



Circle (Vertex) Event

- A circle event happens between three adjacent arcs of three different sites
- A circle event is added at the lowest point of the circle and is associated with the point of the disappearing arc



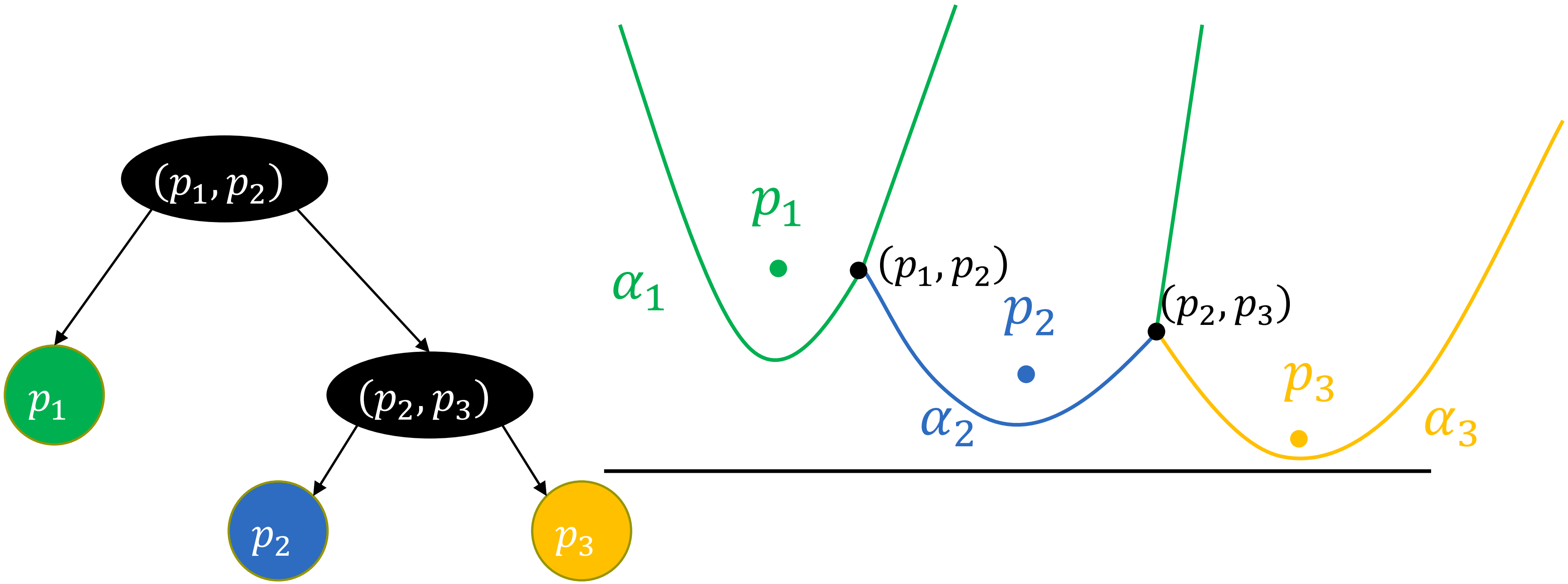
Plane Sweep Constructs

- Sweep line status: The VD of the sites and the sweep line. In other words, the final part of the VD + the beach line in non-decreasing x order
- Event points:
  - Site event: A new site that adds a new arc to the VD. 1-to-1 mapping to an input site
  - Circle event: The disappearance of an arc resulting in a vertex in VD. Can only be discovered along the way

Sweep Line Status

- The final part of VD is stored in the Doubly-Connected Edge List (DCEL) data structure
- The beach line is stored as a BST (τ) of arcs sorted by x
  - Leaves store arcs
  - Internal nodes store the breakpoints as a pair of sites (p_i, p_j)

Sweep Line Status



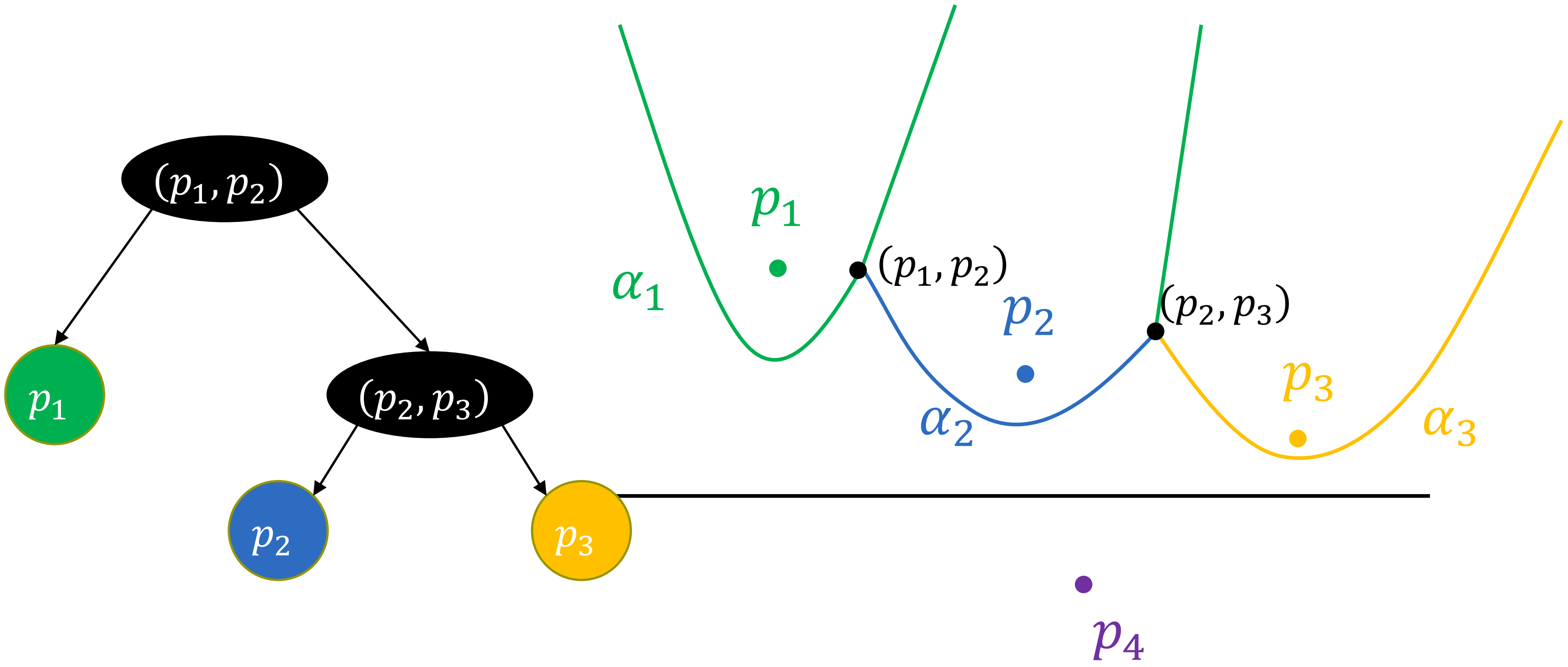
Event Points

- Stored in a priority queue Q as a max-heap ordered by y
- Q is initialized with all sites

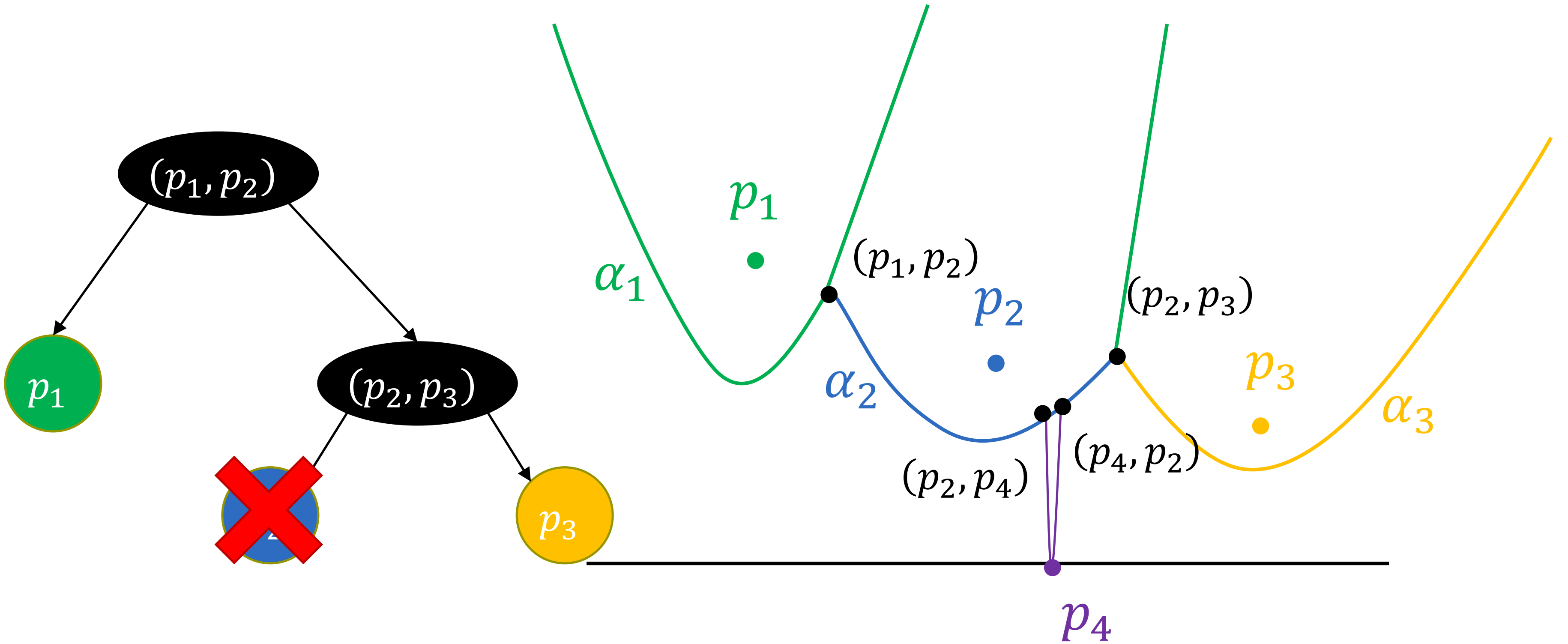
Handle Site Event ($\boldsymbol{p_i}$)

- If τ is empty, add the site to it and return
- Search in τ for the arc α vertically above p_i
- If exists, delete a circle event linked with α
- Split α into two arcs and insert a new arc α_i corresponding to p_i
- The new intersections are (α, α_i) and (α_i, α)
- Check the new triples of arcs and add their corresponding circle event to Q

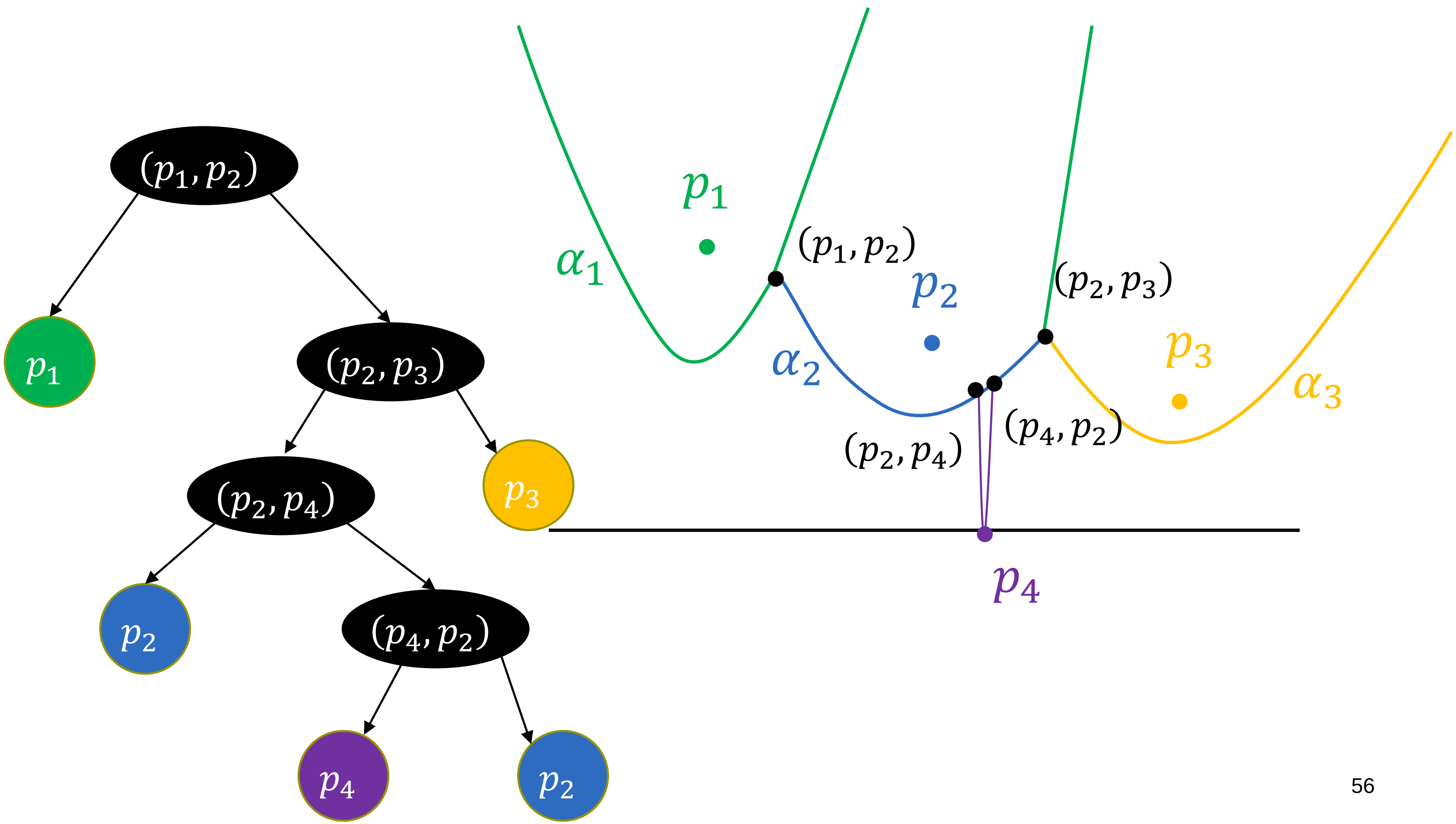
Handle Site Event (p_i)



Handle Site Event (p_i)



Handle Site Event (p_i)

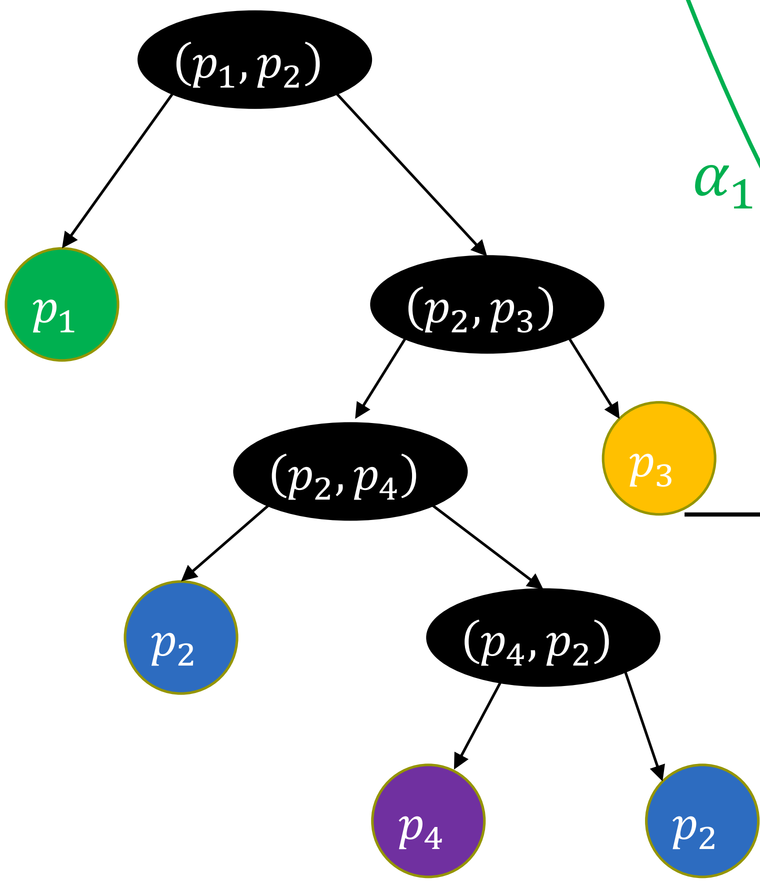


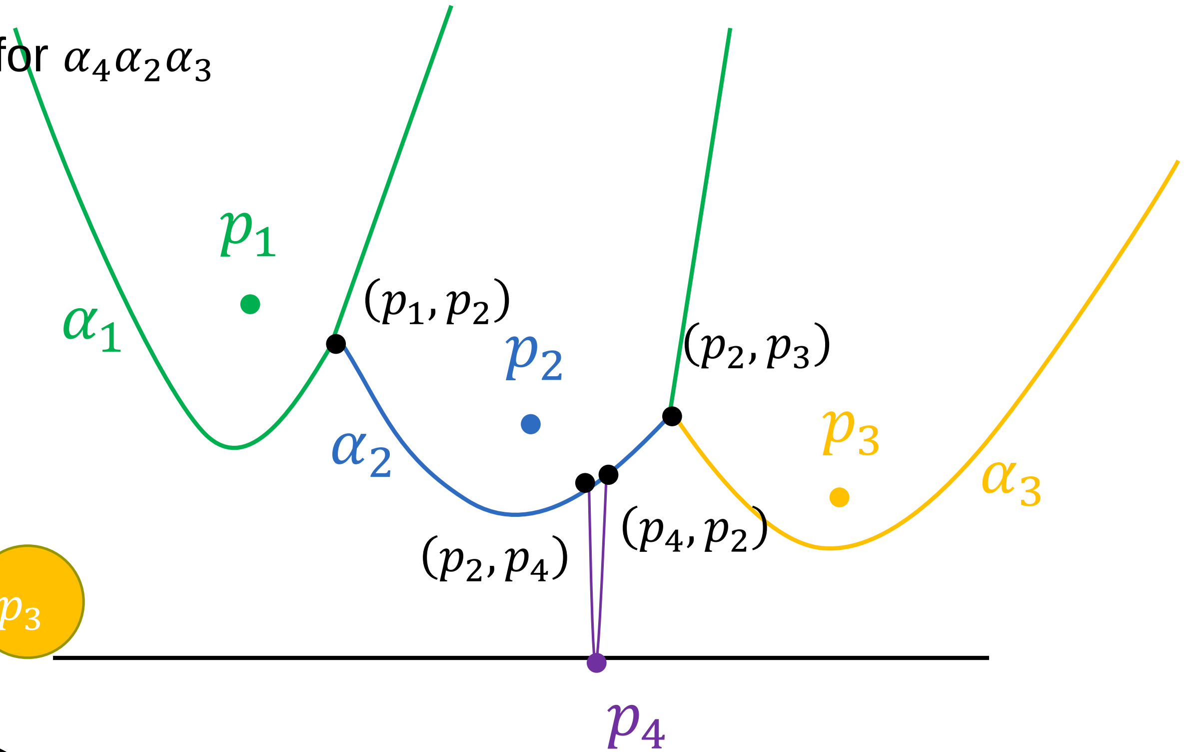
Handle Site Event (p_i)

$\alpha_1\alpha_2\alpha_3$ are no longer adjacent ➔ Remove the circle event that corresponds to α_2

$\alpha_1\alpha_2\alpha_4$ are now adjacent ➔ Create a new circle event for them

Similarly, create a circle event for $\alpha_4\alpha_2\alpha_3$





No circle event for the triple $\alpha_2\alpha_4\alpha_2$ because they don't correspond tot three different sites

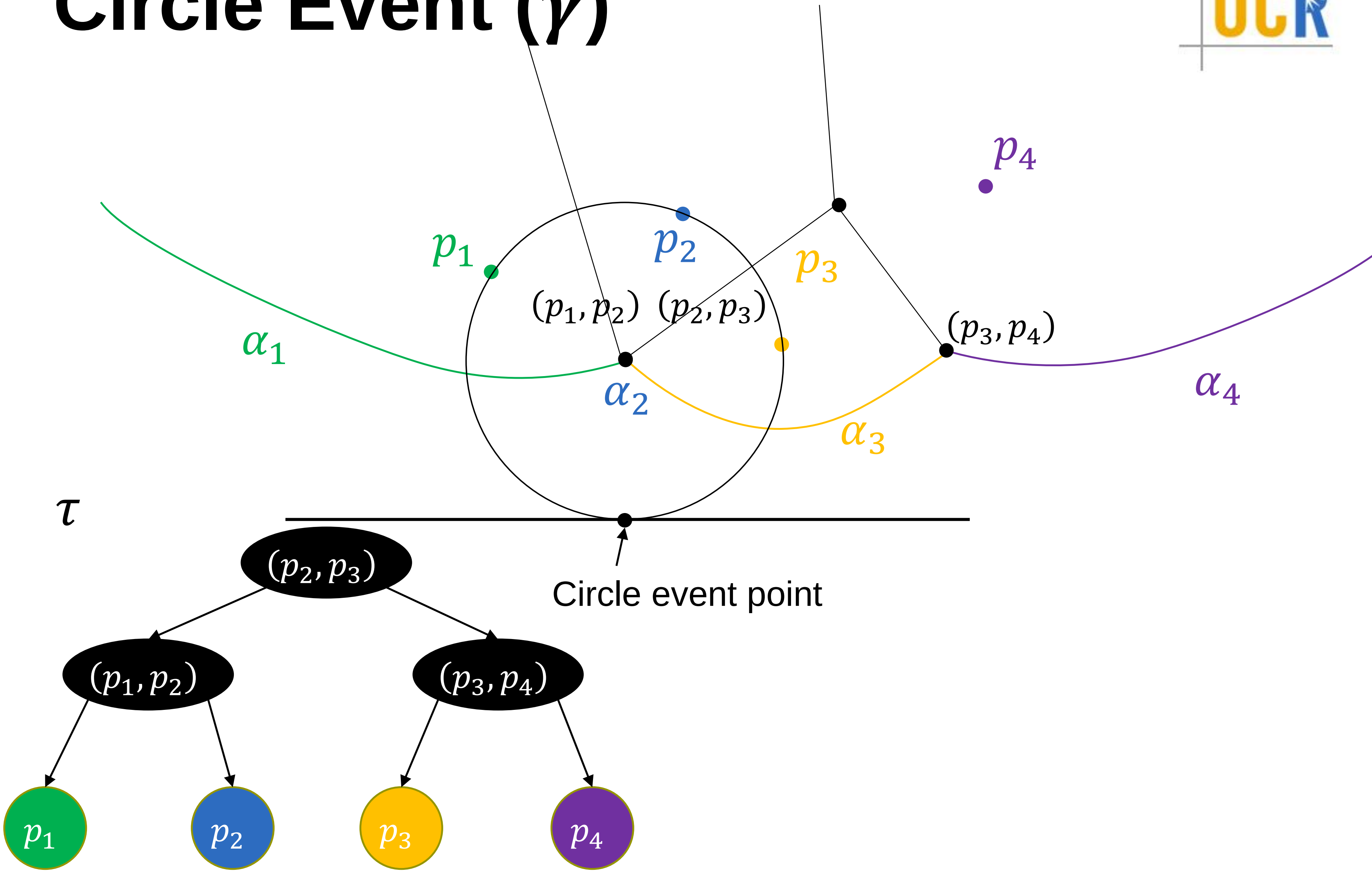
Creating a Circle Event

- Given three sites (p_i, p_j, p_k) that have three adjacent arcs, we first compute the center of their circumcircle, i.e., the intersection of the two perpendicular bisectors to $\overline{p_i p_j}$ and $\overline{p_j p_k}$
- Compute the bottom point of the circle as $(x_c, y_c - r)$ where
  - (x_c, y_c) are the coordinates of the circle center and r is the circle radius
- Associate the circle event with the middle site in the tree order

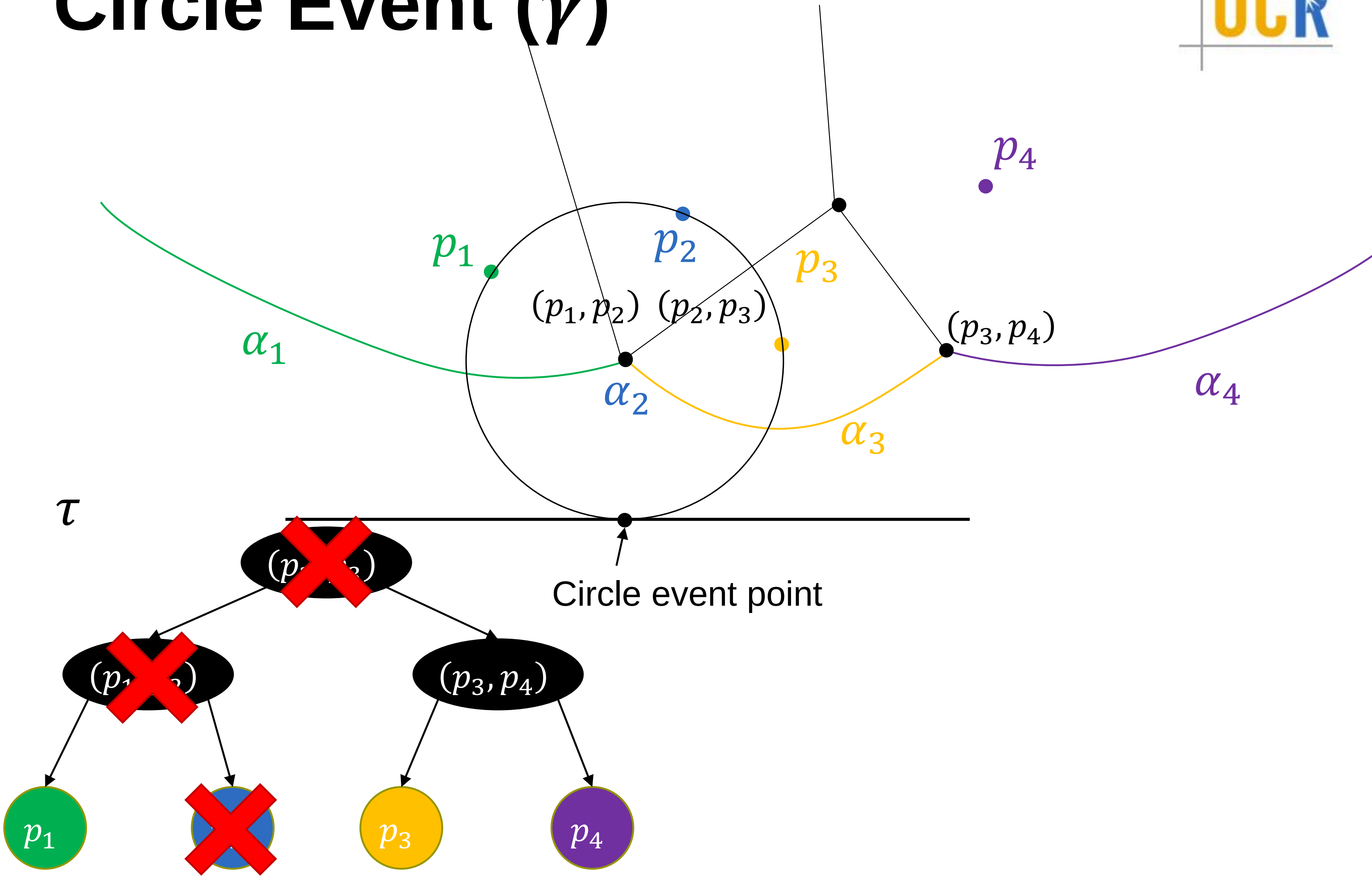
Handle Circle Event (γ)

- Delete the leaf γ that corresponds to the disappearing arc α_i from τ
- Delete the two breakpoints that involve α_i
- Insert a new break point
- Add the center of the circle event as a vertex in VD. This center is one side of two half-edges
- Check for any new circle events caused by the now adjacent triples of arcs
- Running time: $O(n \log n)$

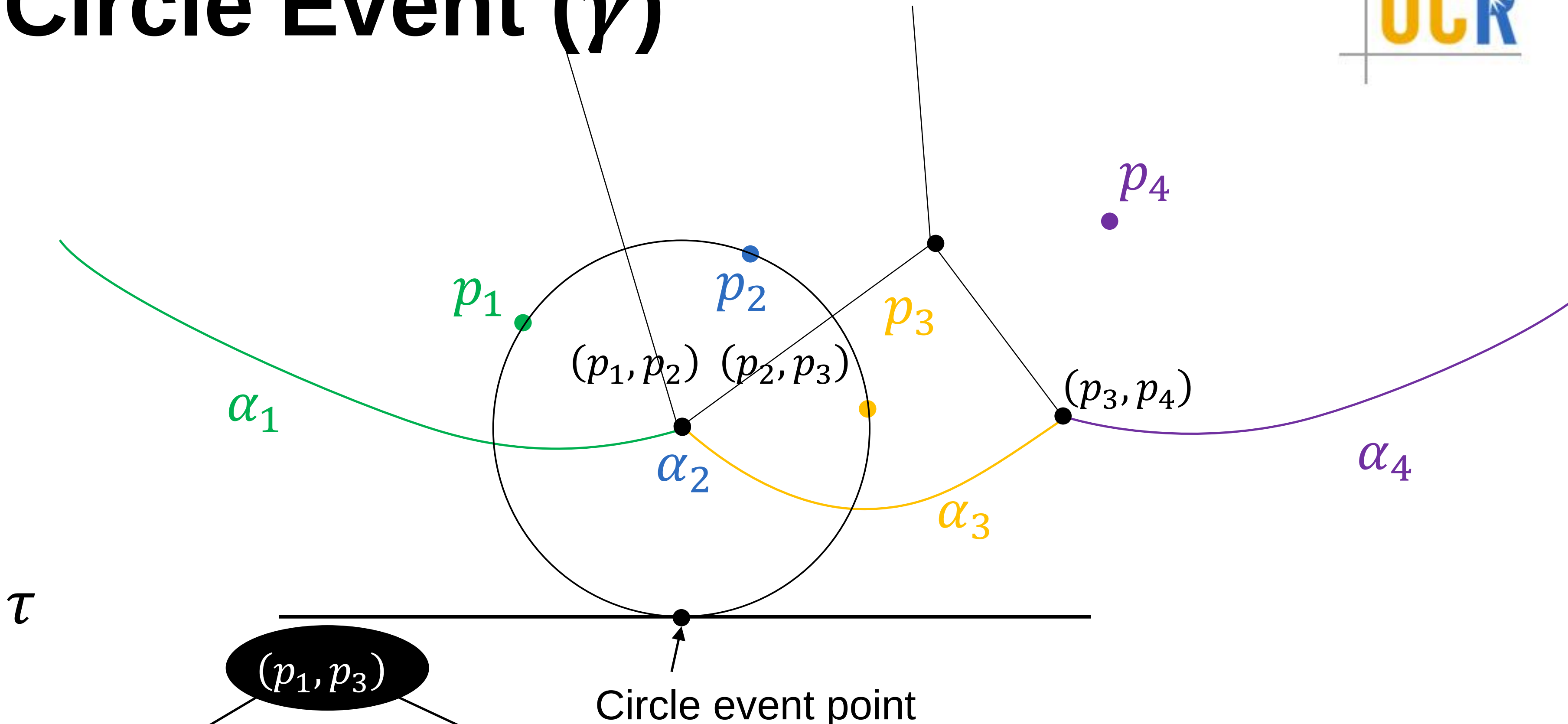
Circle Event (γ)

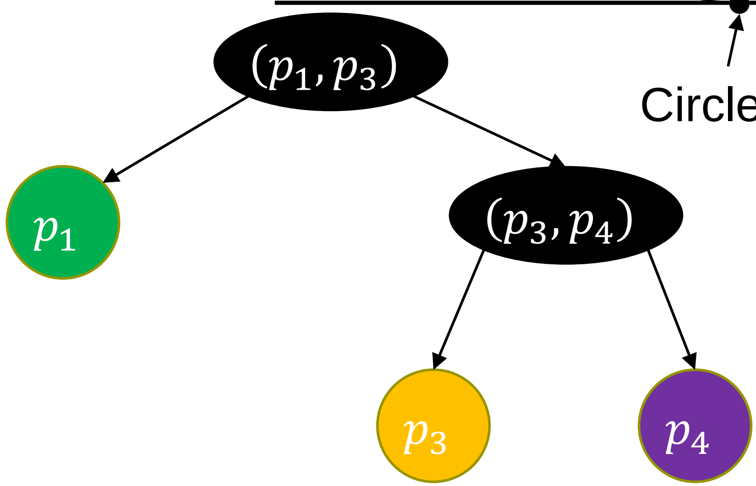


Circle Event (γ)



Circle Event (γ)





(p_1, p_3, p_4) are now adjacent in the tree, create a corresponding circle event

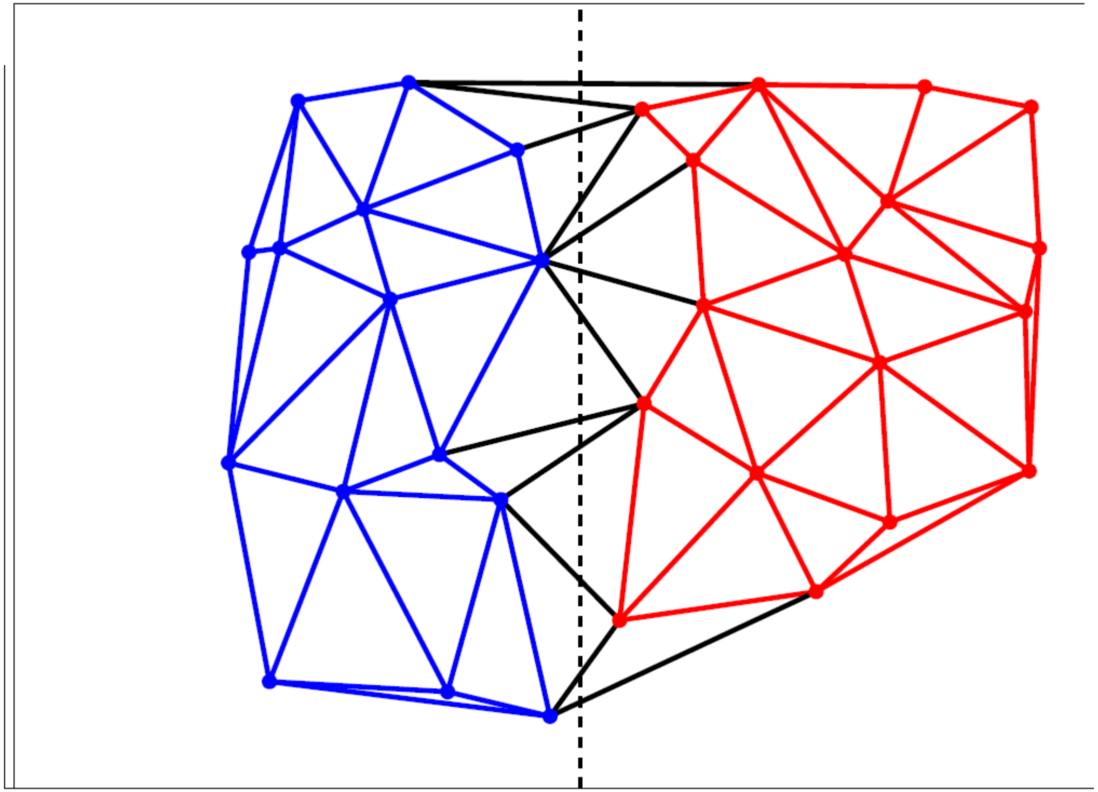
Delaunay Triangulation

Delaunay Triangulation

- A Delaunay triangulation can be defined as the (unique) triangulation in which the circumcircle of each triangle has no other sites
- Naïve algorithm:
  - Consider all possible triangles $O(n^3)$
    - Check if the circumcircle of the triangle is empty $O(n)$
  - Running time $O(n^4)$

Guibas and Stolfi's Algorithm

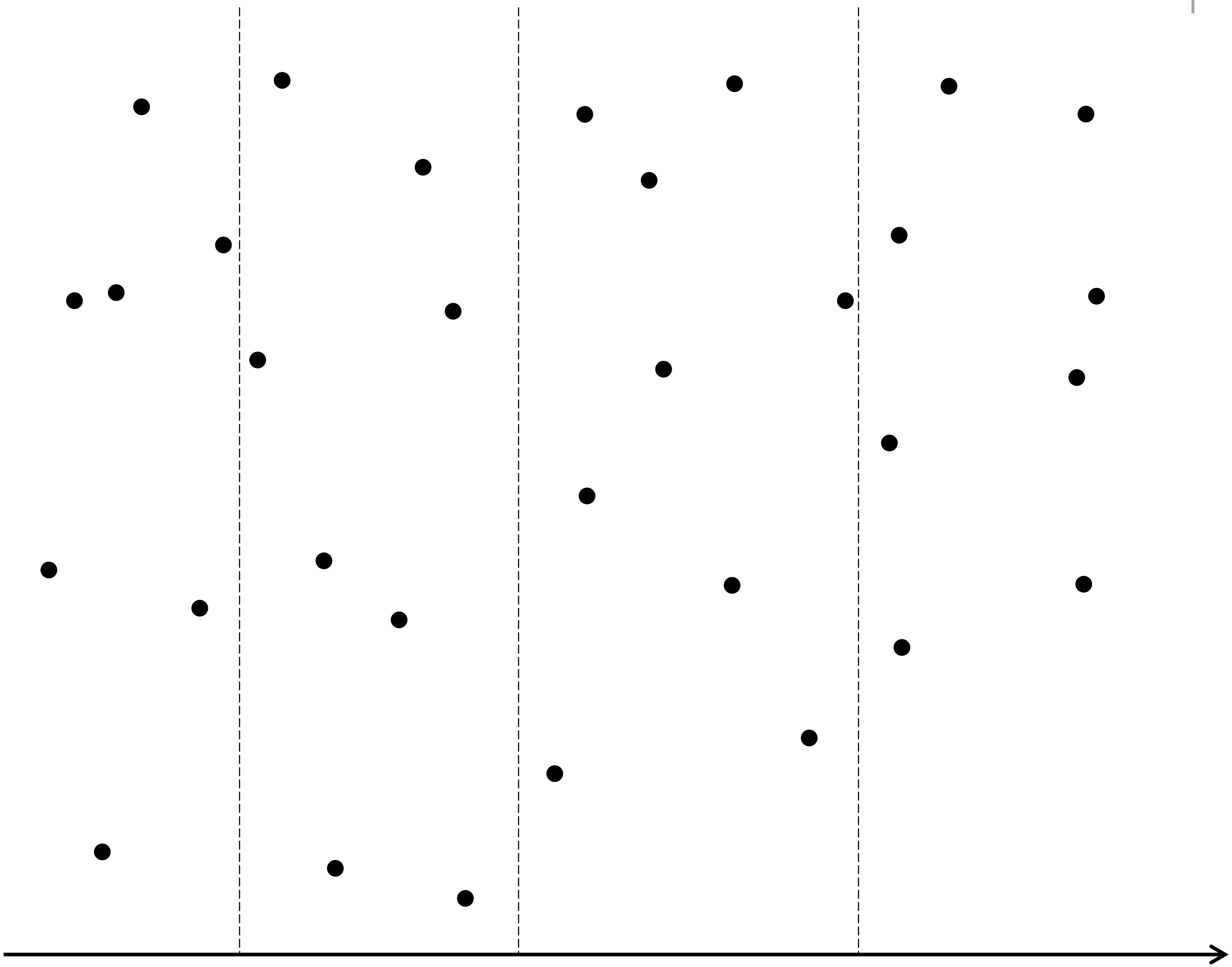
- A divide and conquer algorithm



Algorithm Outline

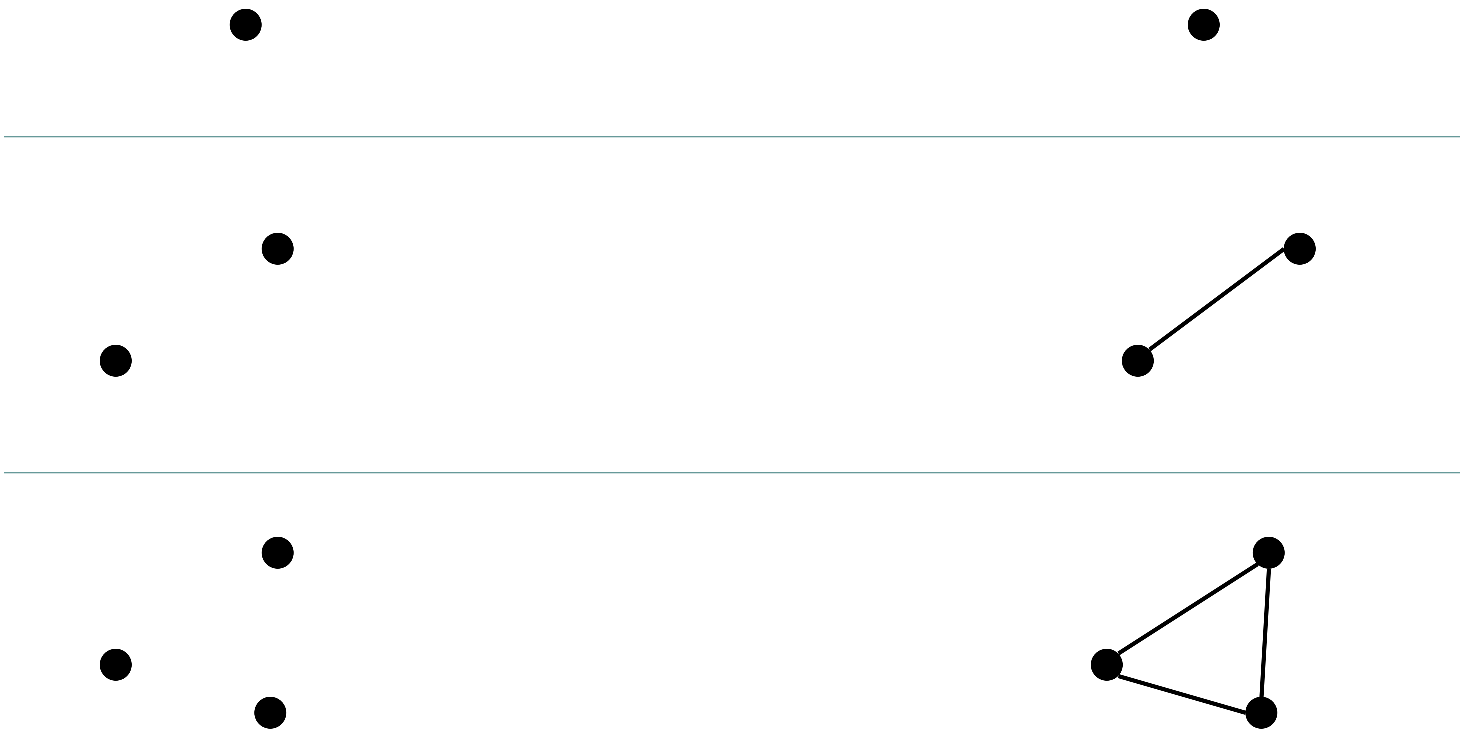
- DelaunayTriangulation(P)
  - If (|P| <= 3)
    - return TrivialDT(P)
  - Split P into P1 and P2
  - DT1 = DelaunayTriangulation(P1)
  - DT2 = DelaunayTriangulation(P1)
  - Merge(DT1, DT2)

Split

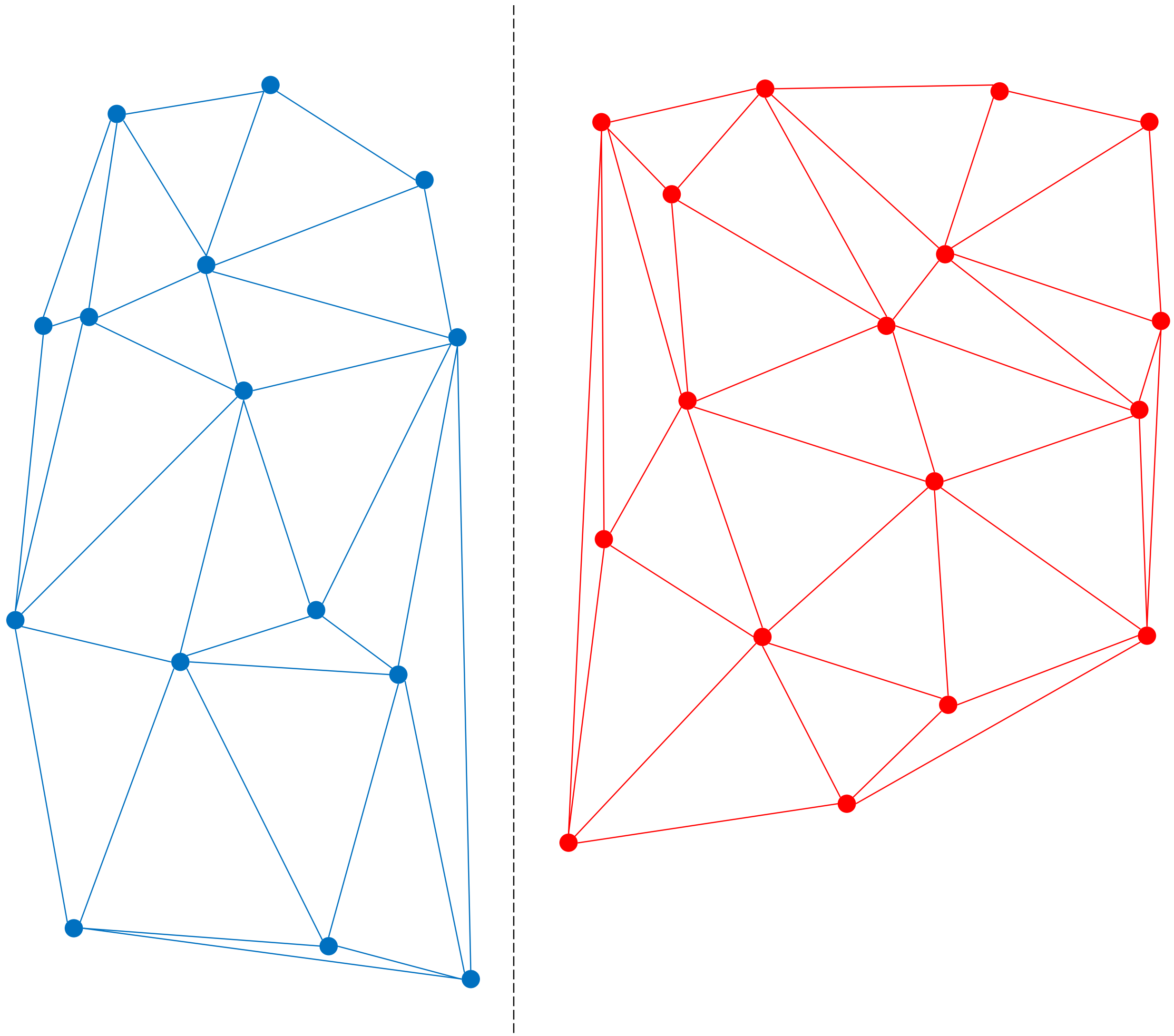


TrivialDT(P)

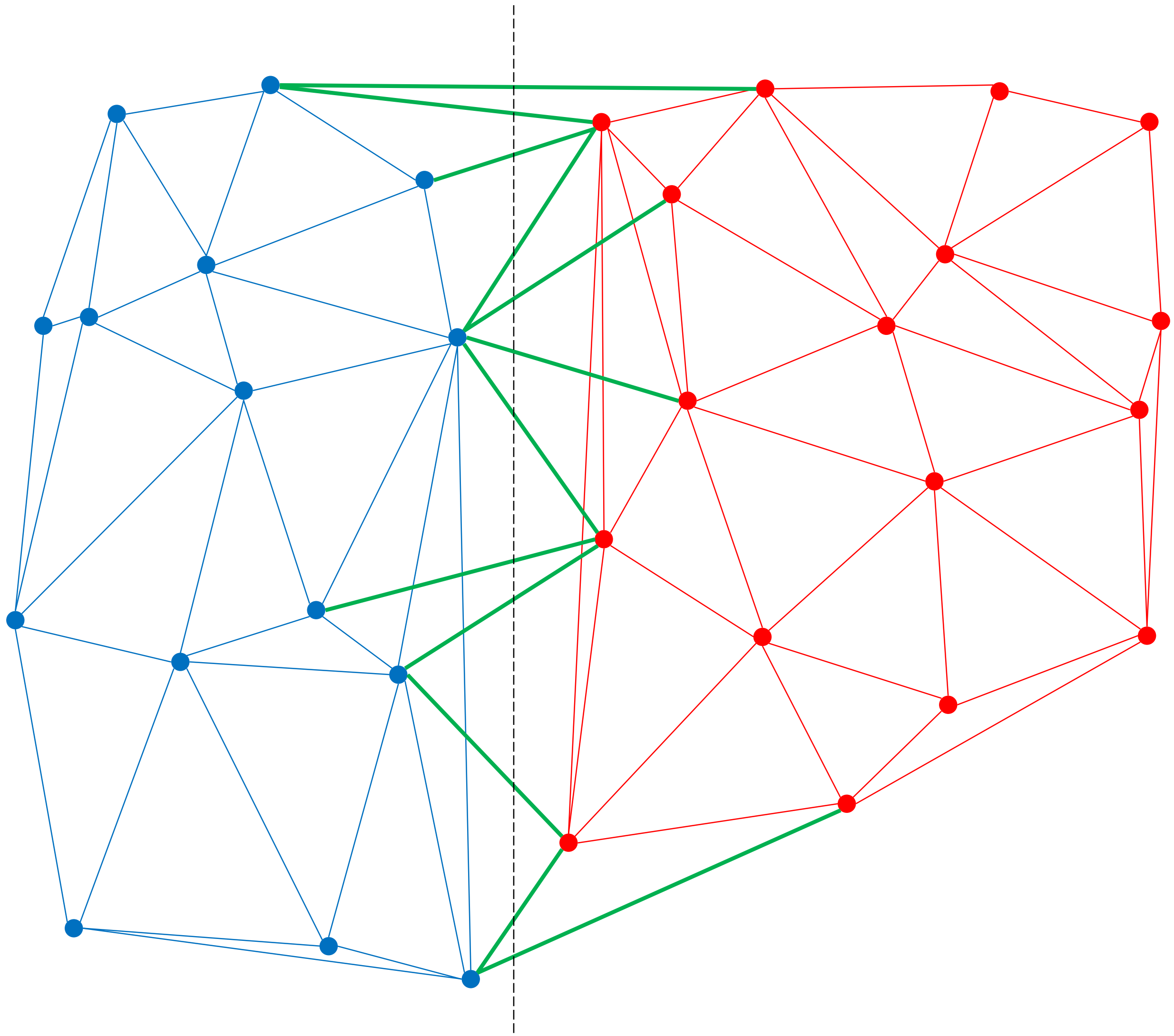
P TrivialDT(P)



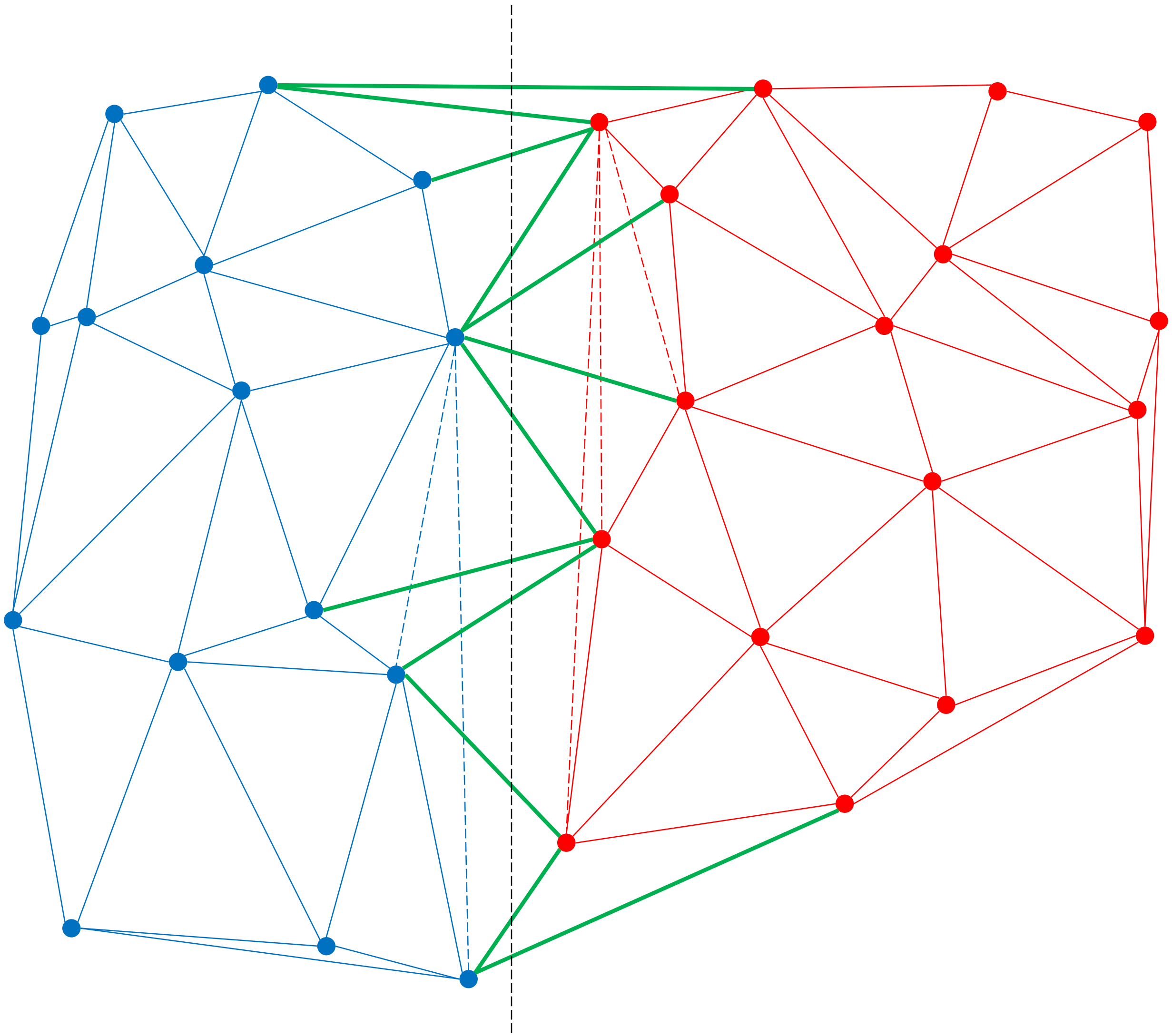
Merge(P1, P2)



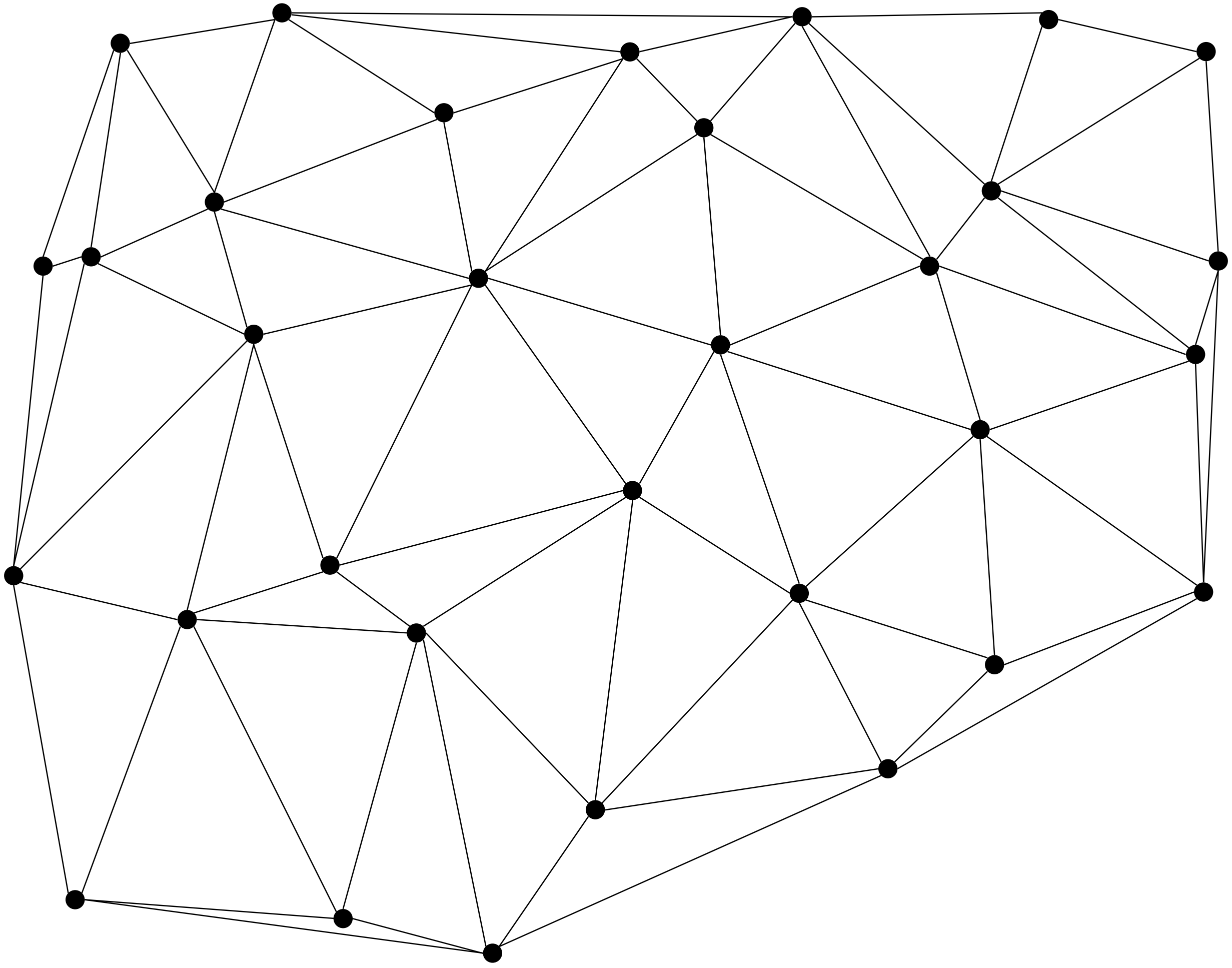
Merge(P1, P2)



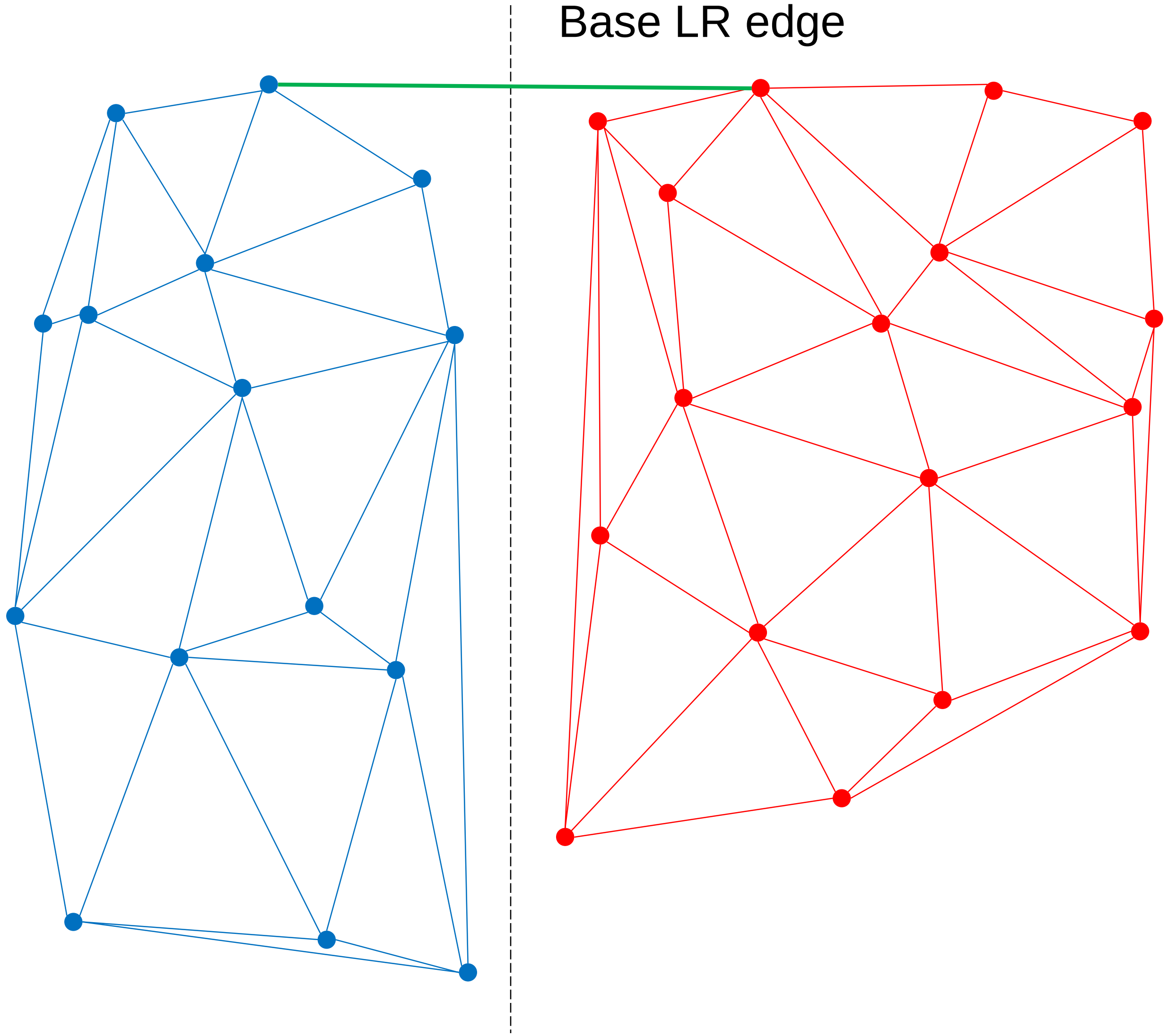
Merge(P1, P2)



Merge(P1, P2)

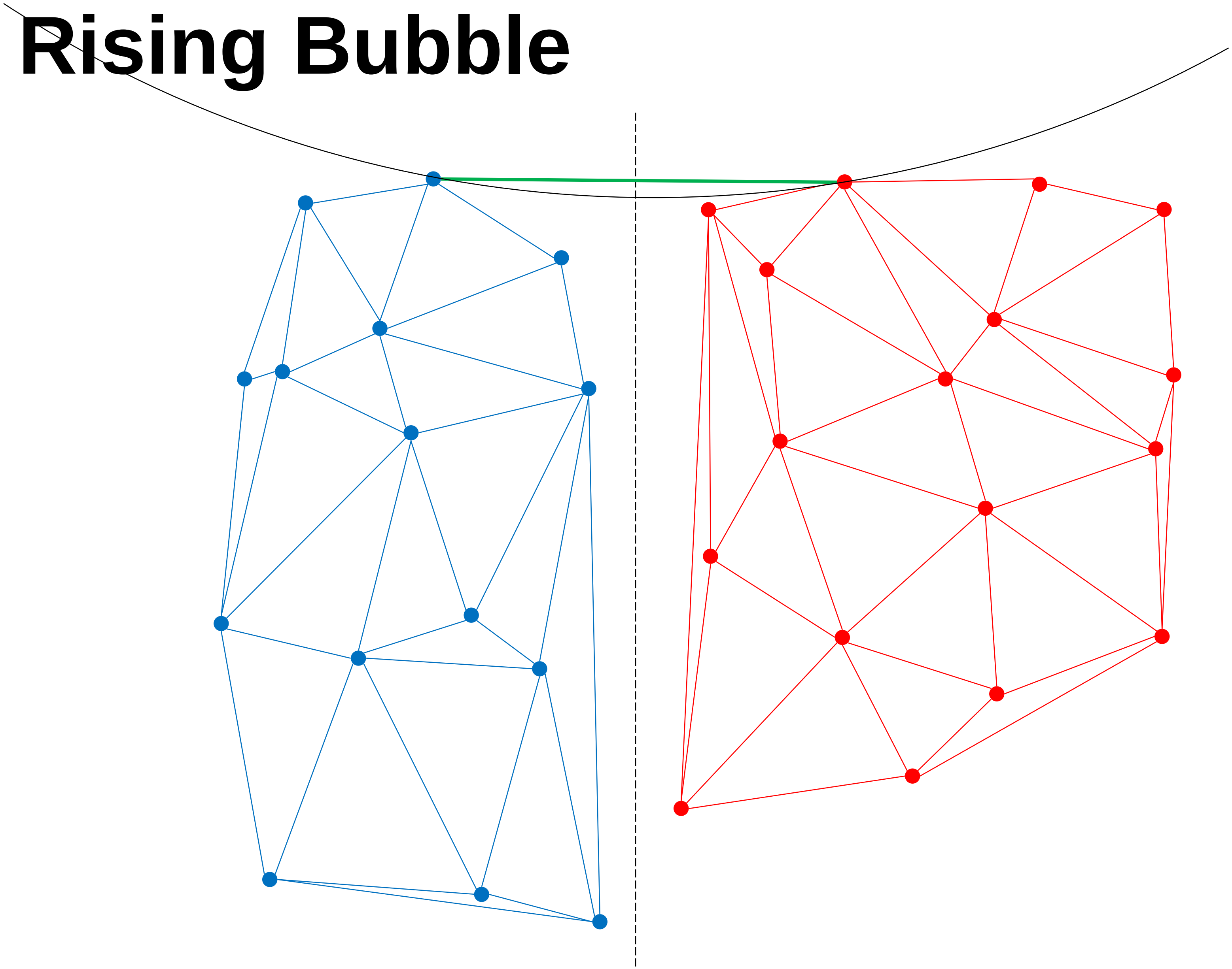


Find the First LR edge

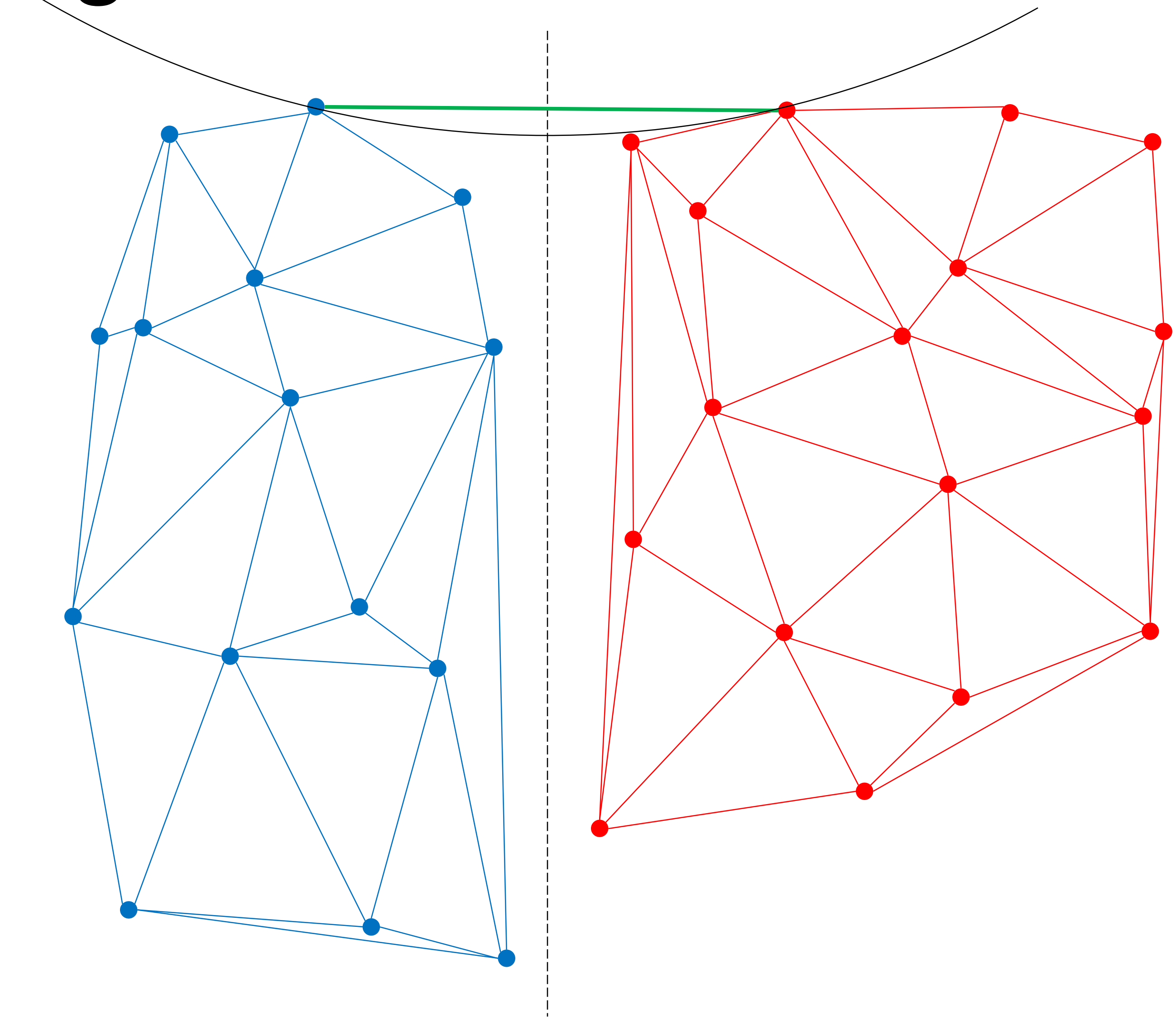


Upper tangent of $\mathcal{CH}(P_1), \mathcal{CH}(P_2)$

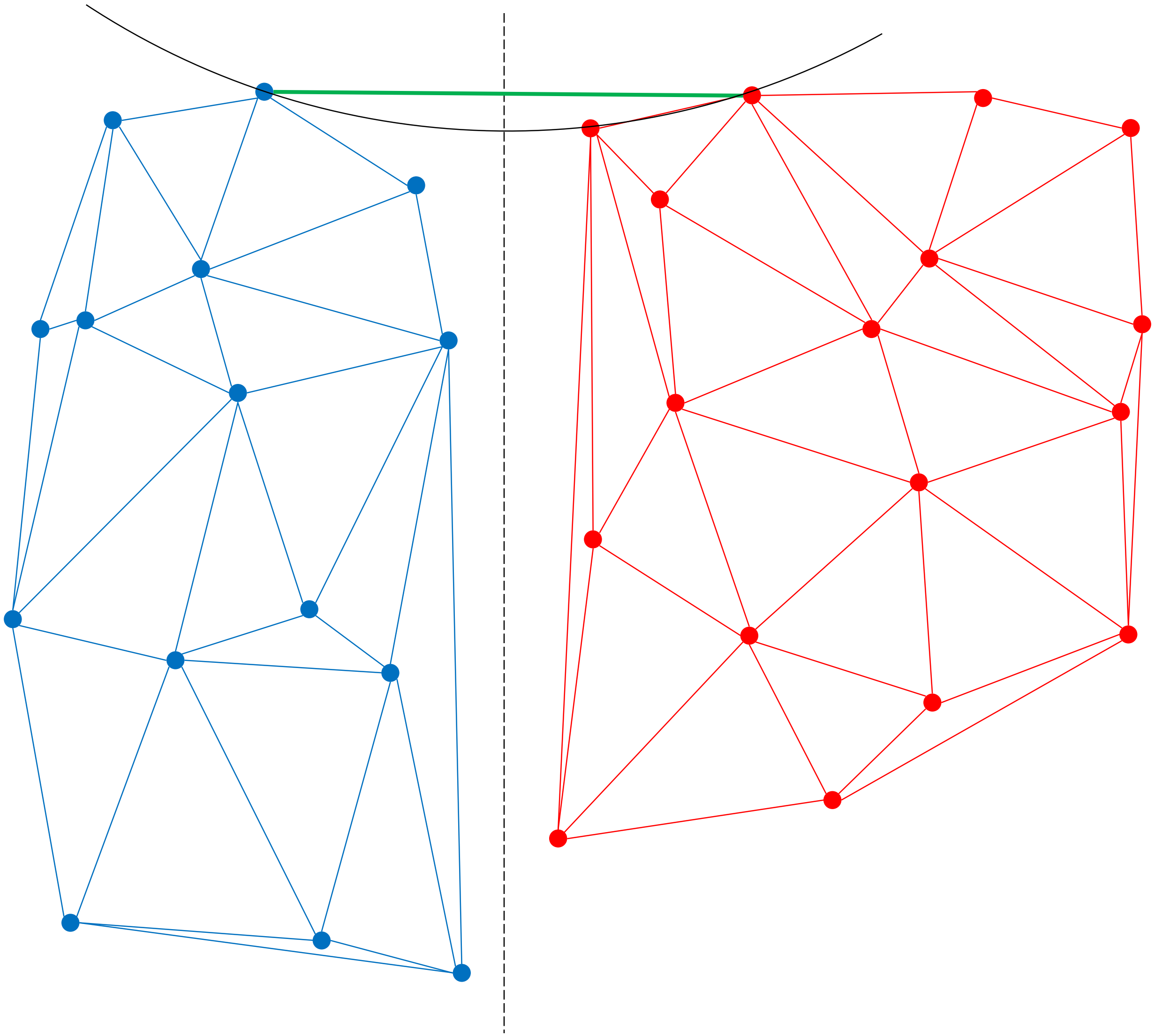
Rising Bubble



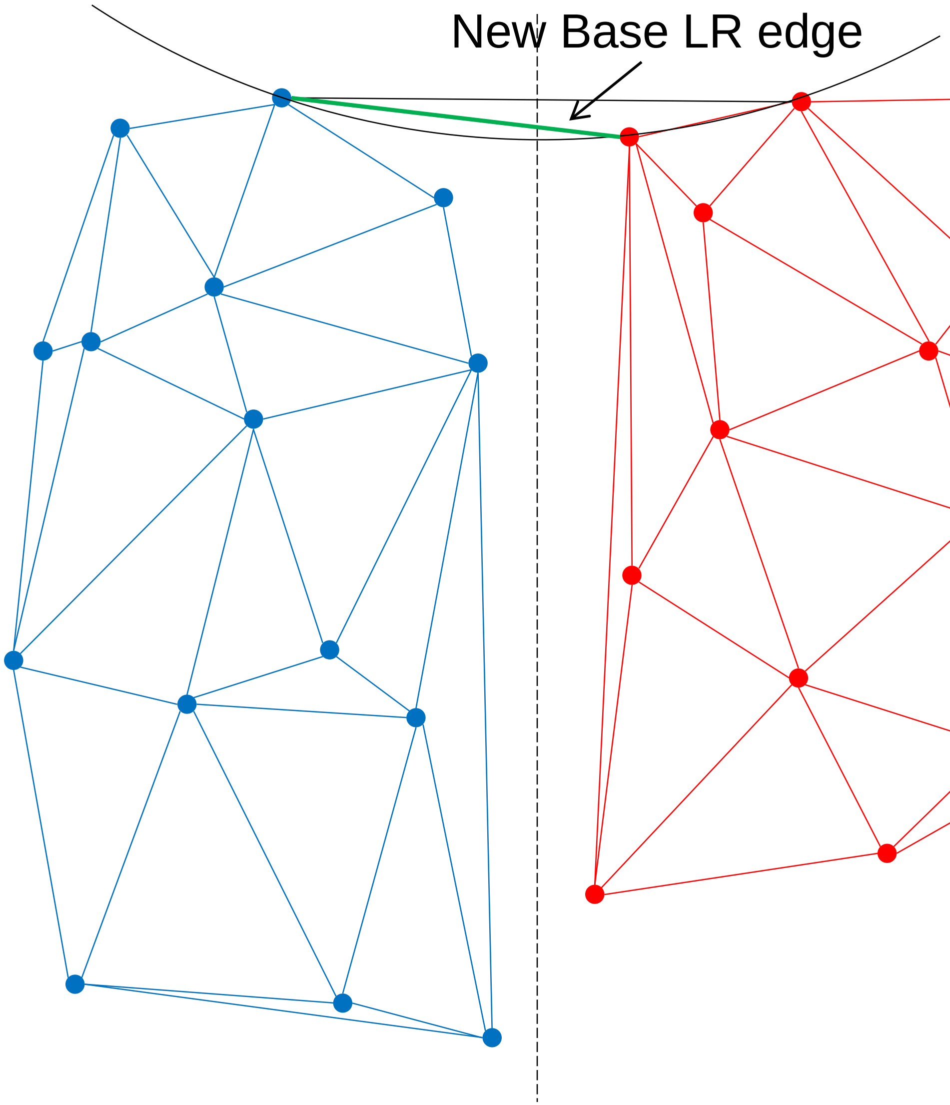
Rising Bubble



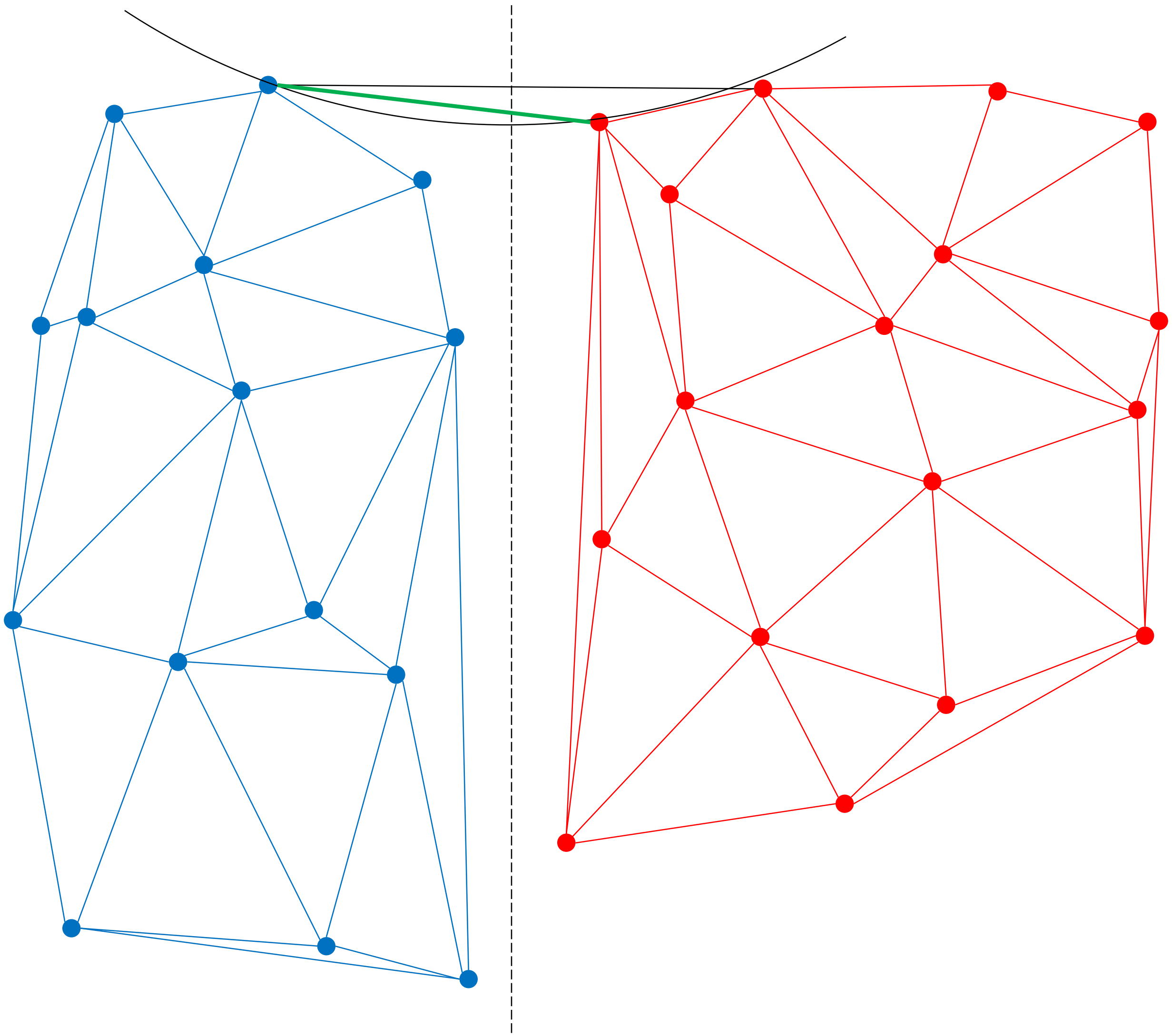
Rising Bubble



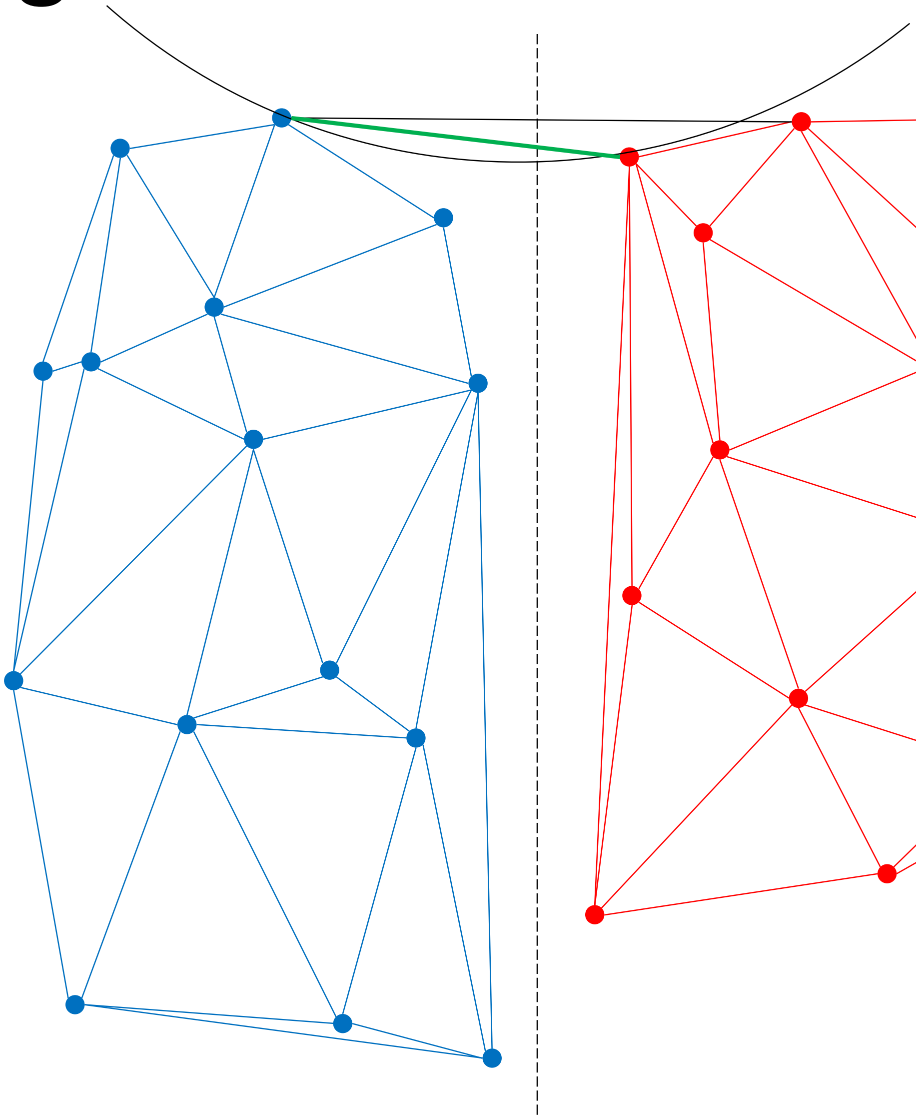
Rising Bubble



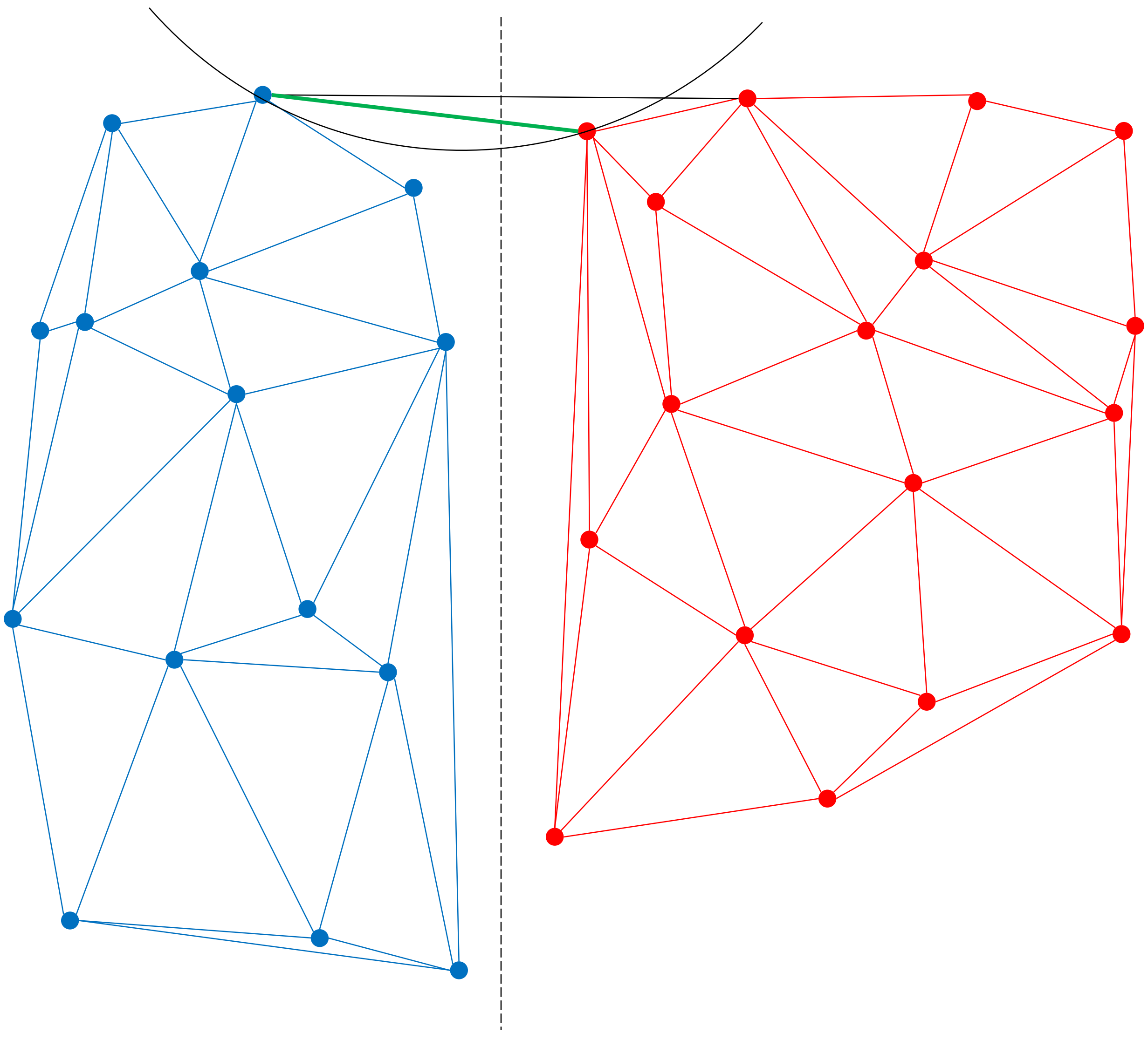
Rising Bubble



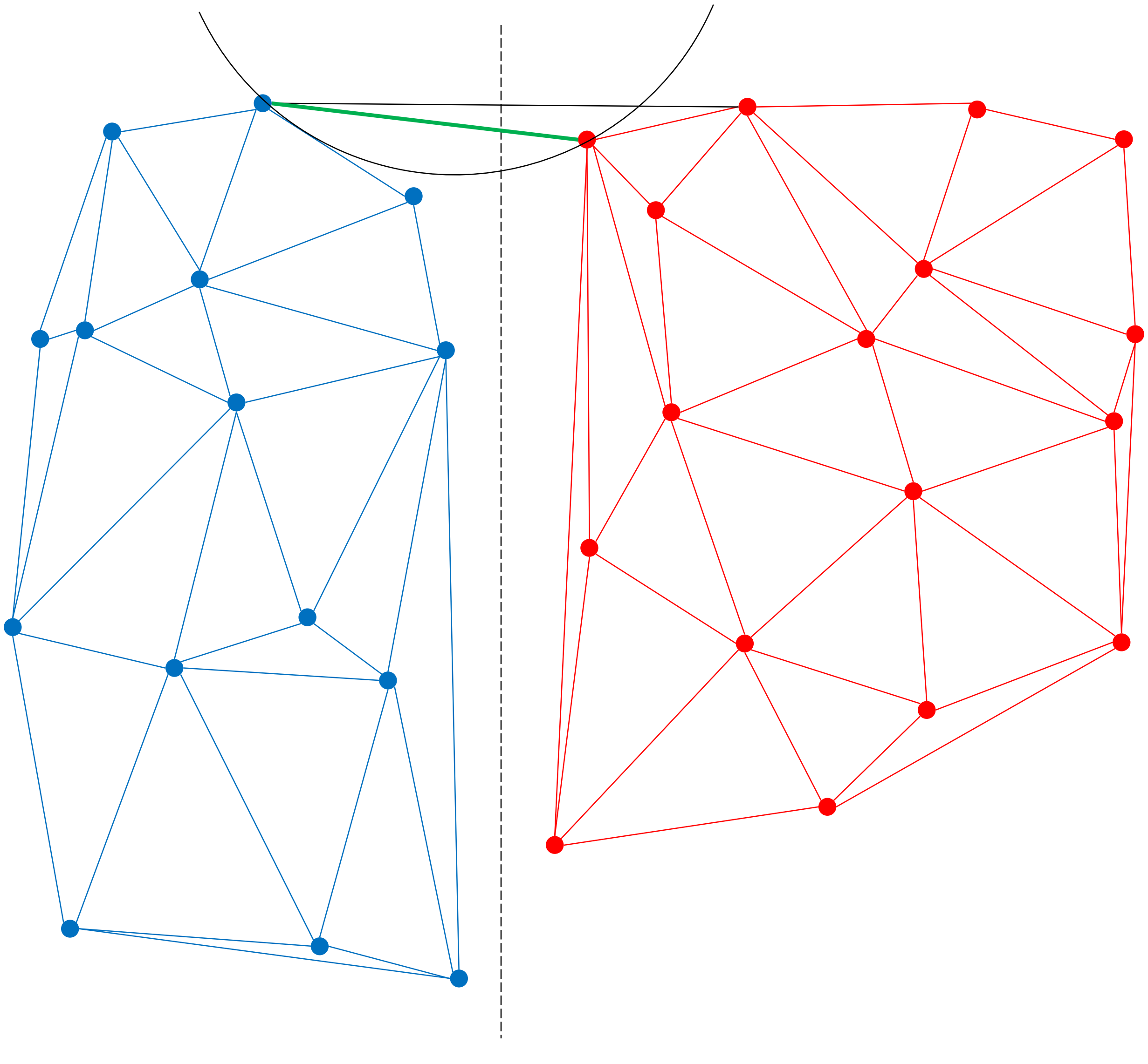
Rising Bubble



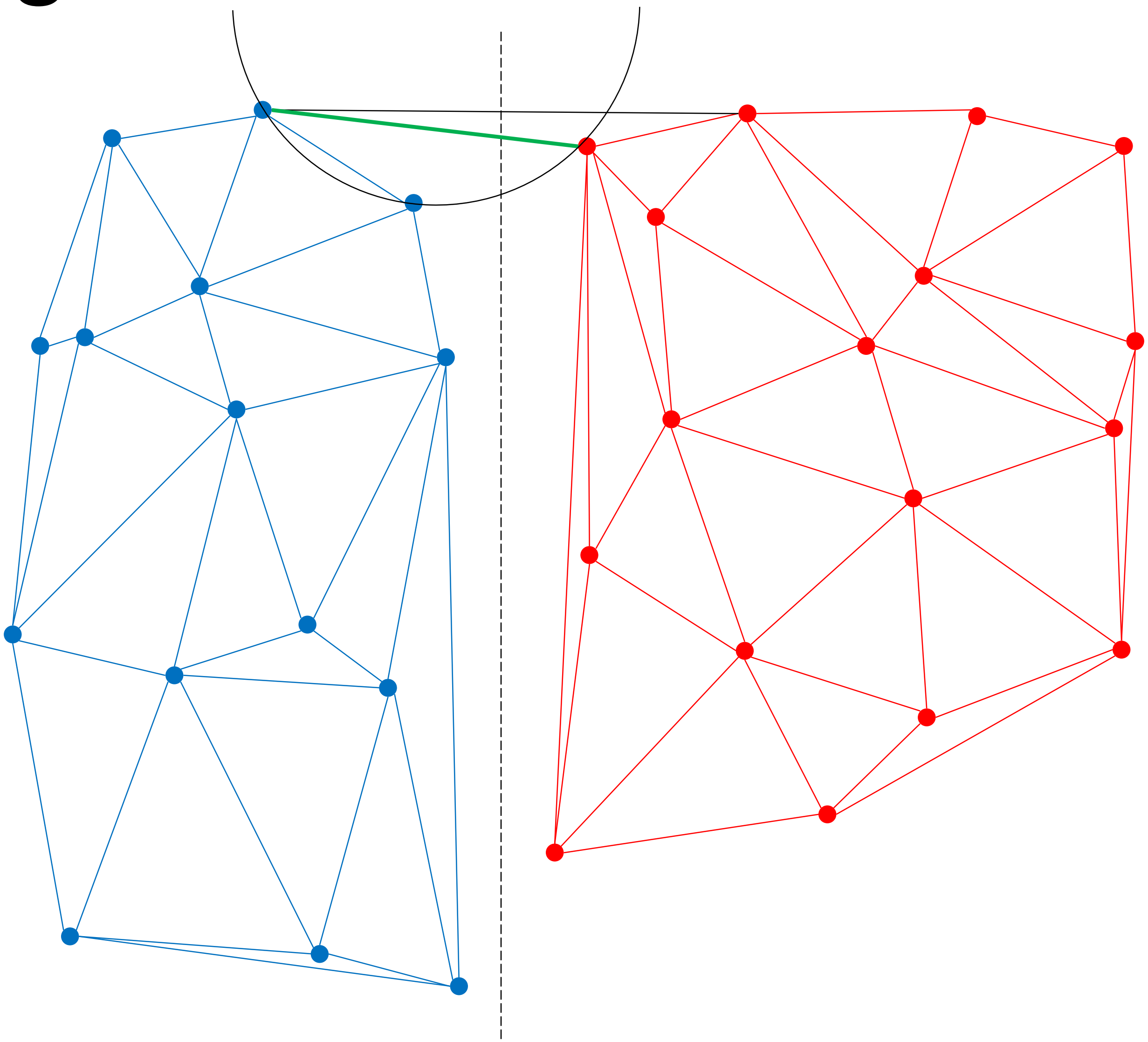
Rising Bubble



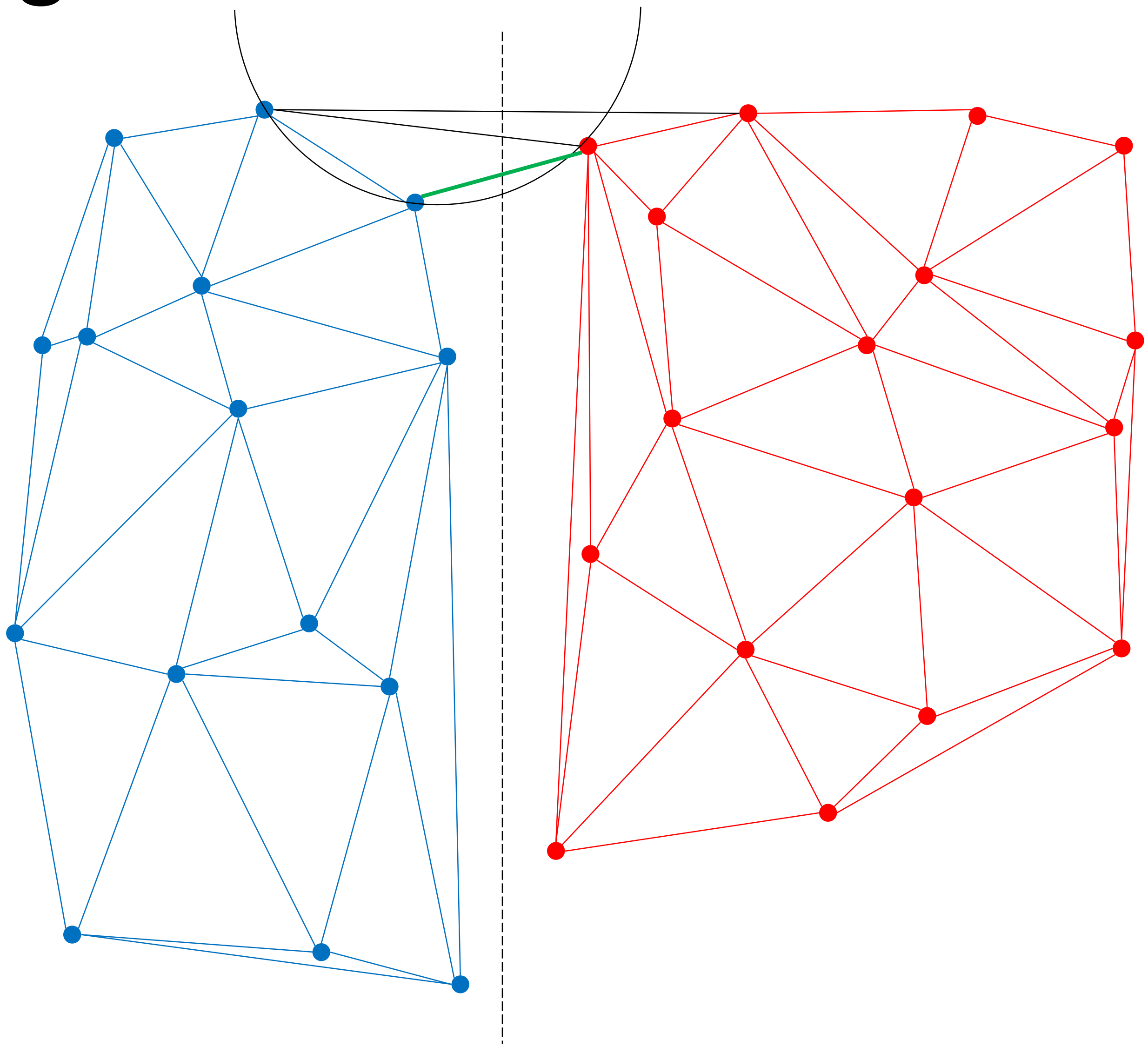
Rising Bubble



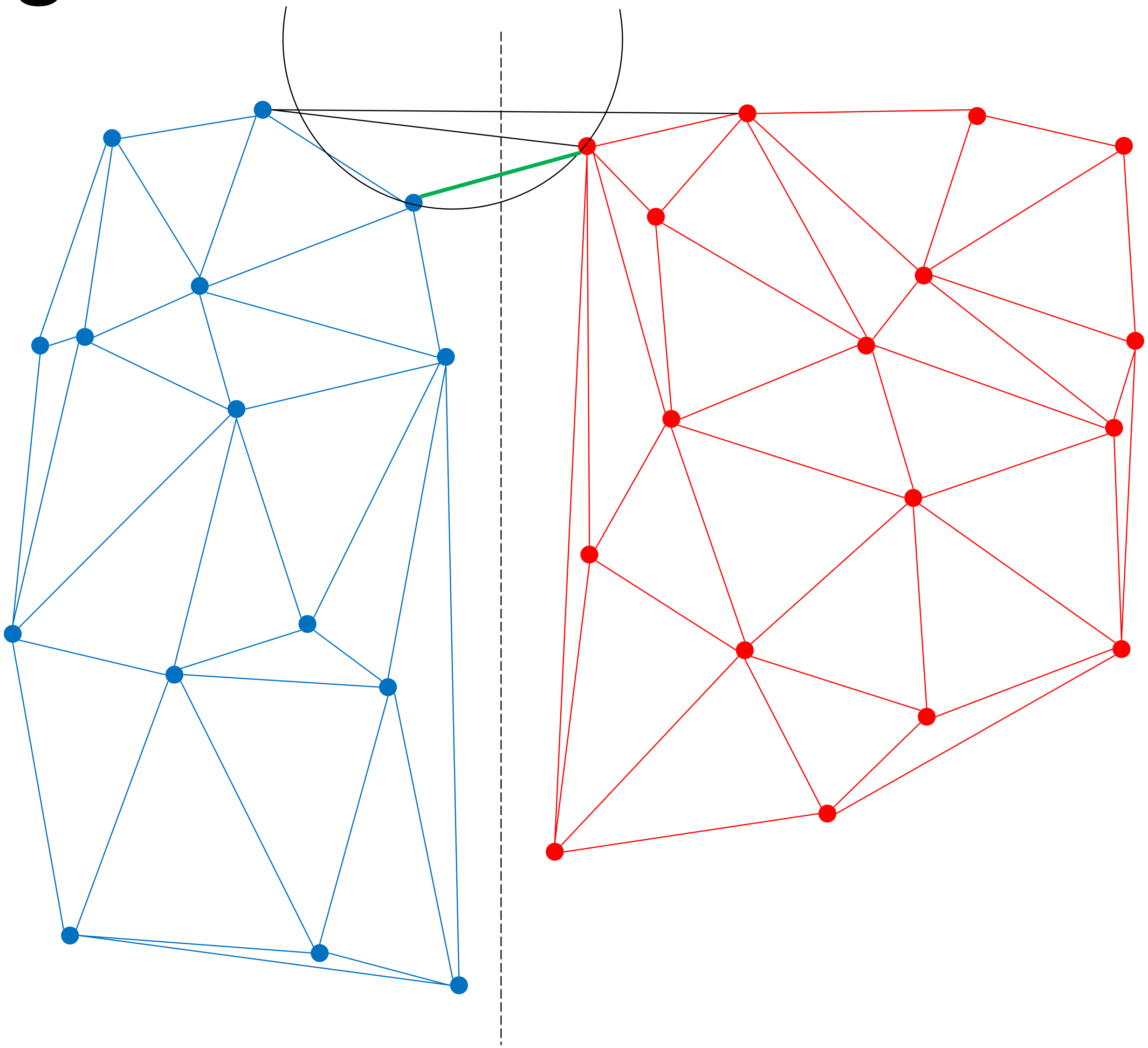
Rising Bubble



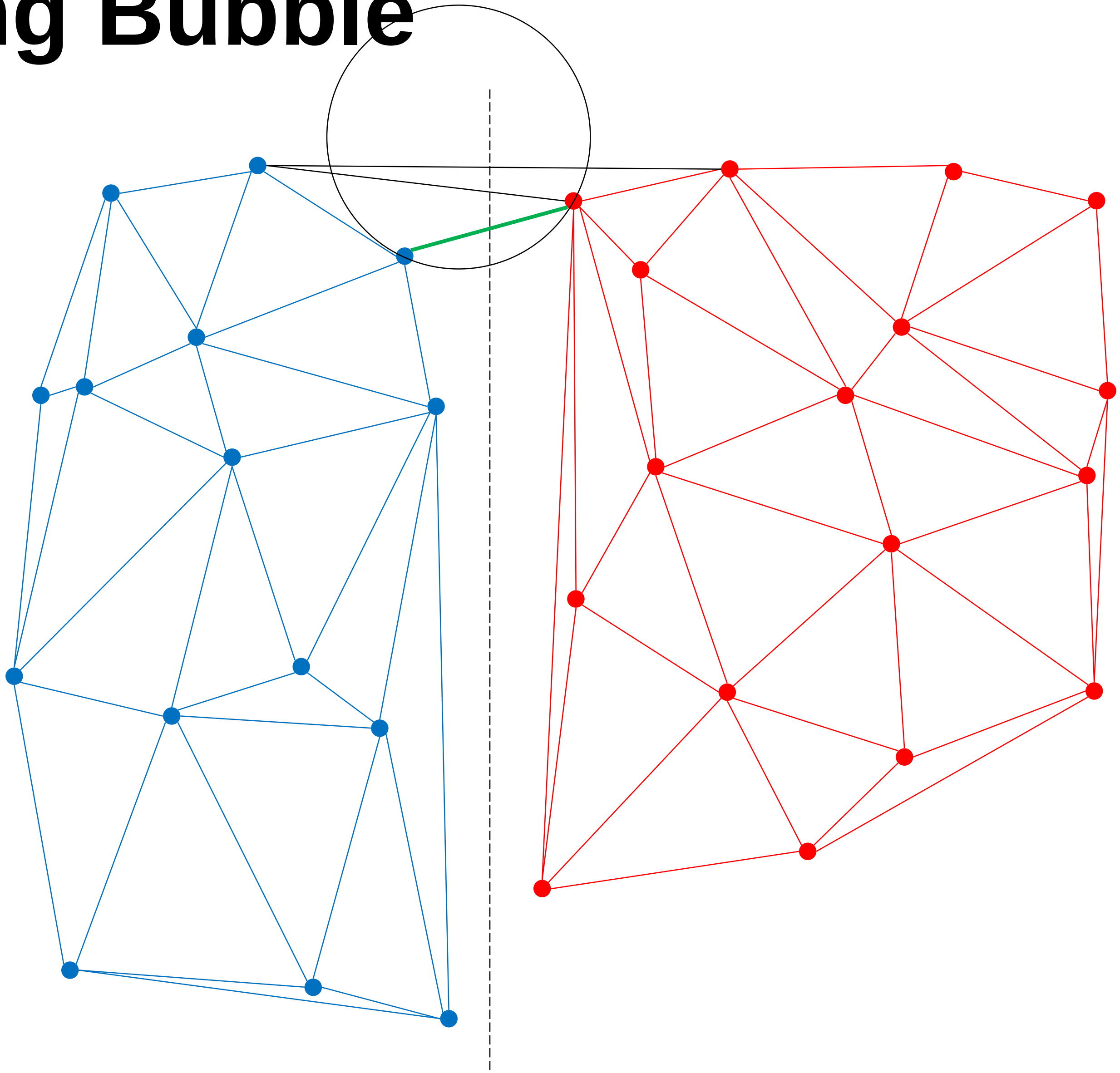
Rising Bubble



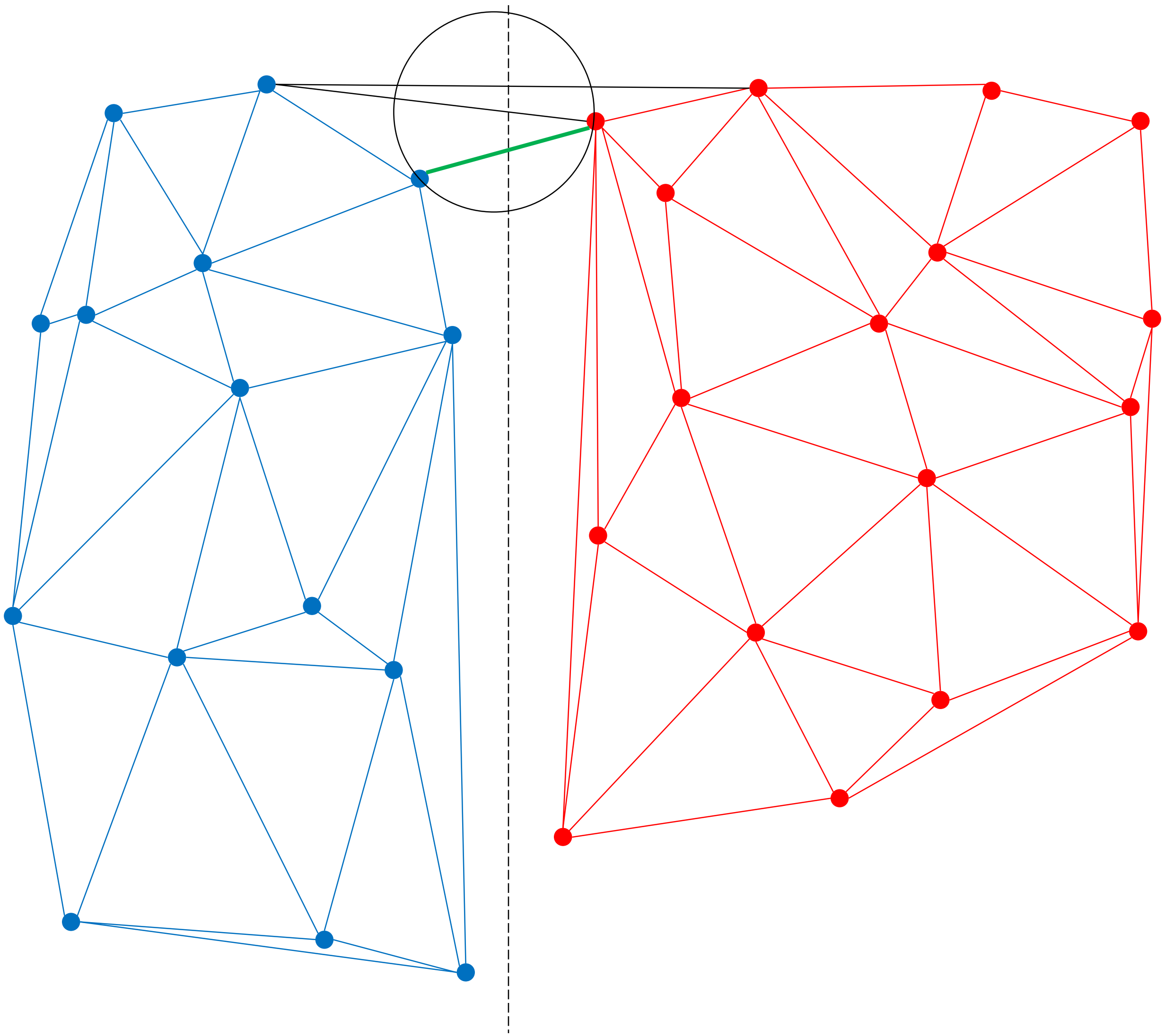
Rising Bubble



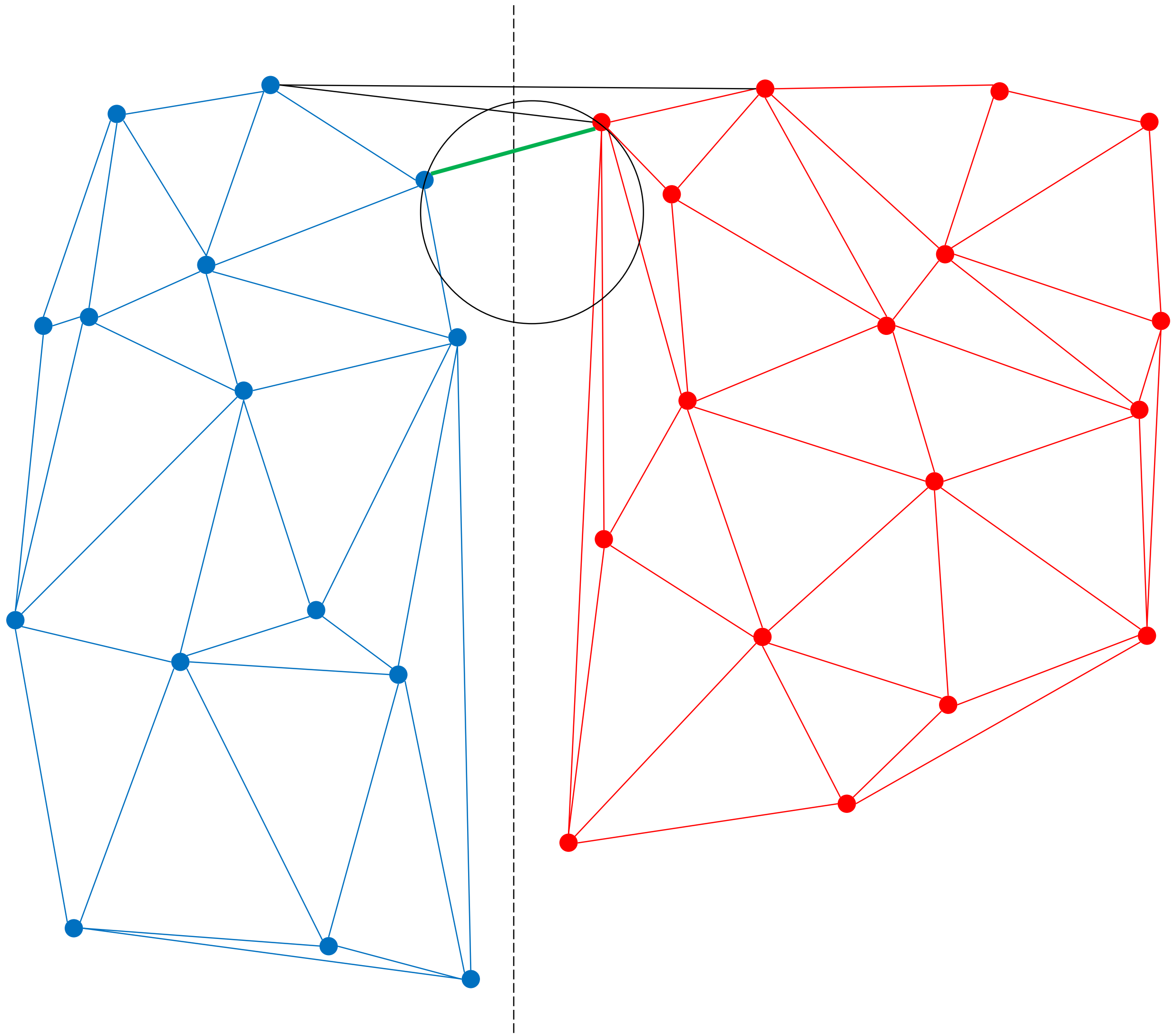
Rising Bubble



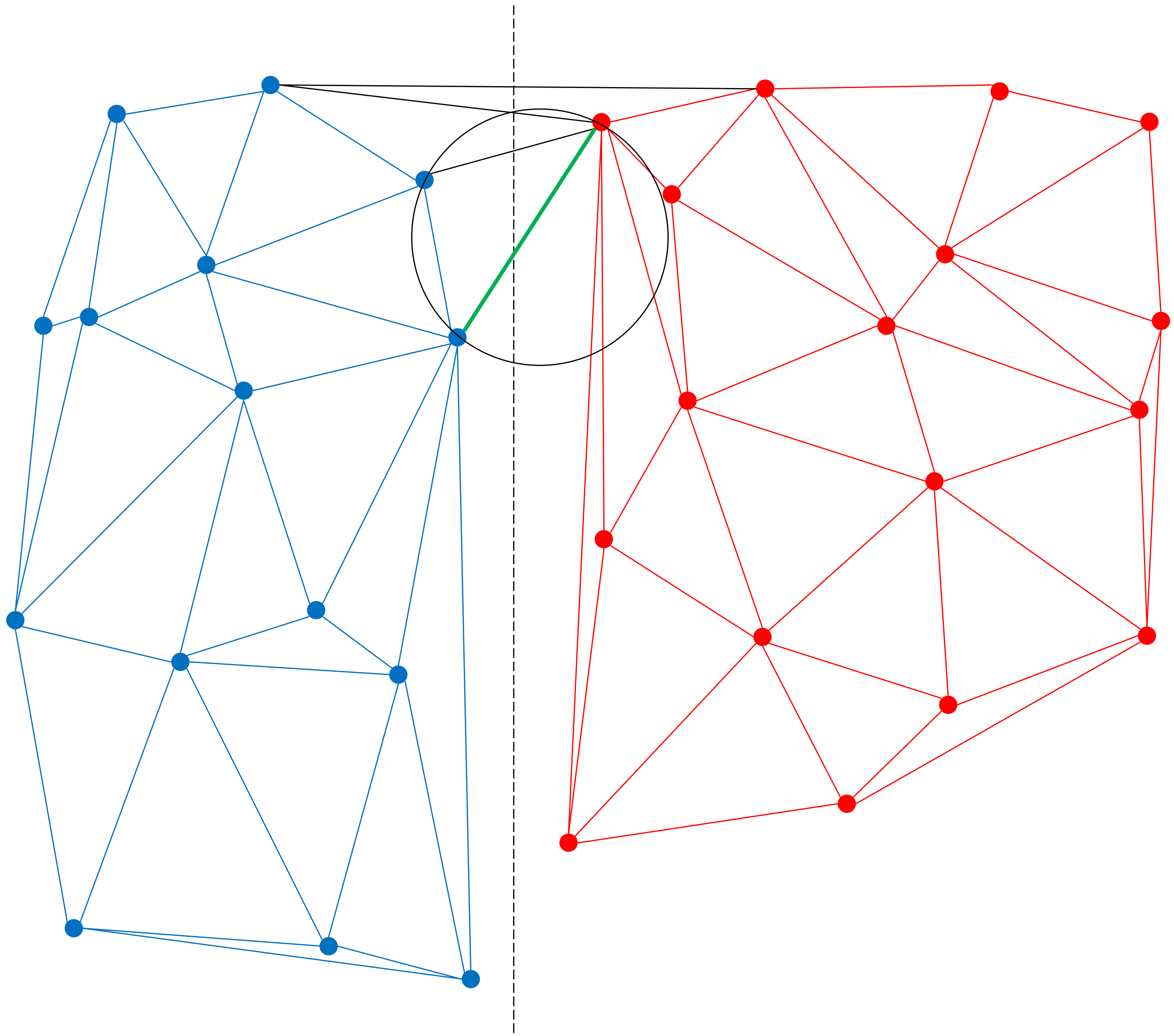
Rising Bubble



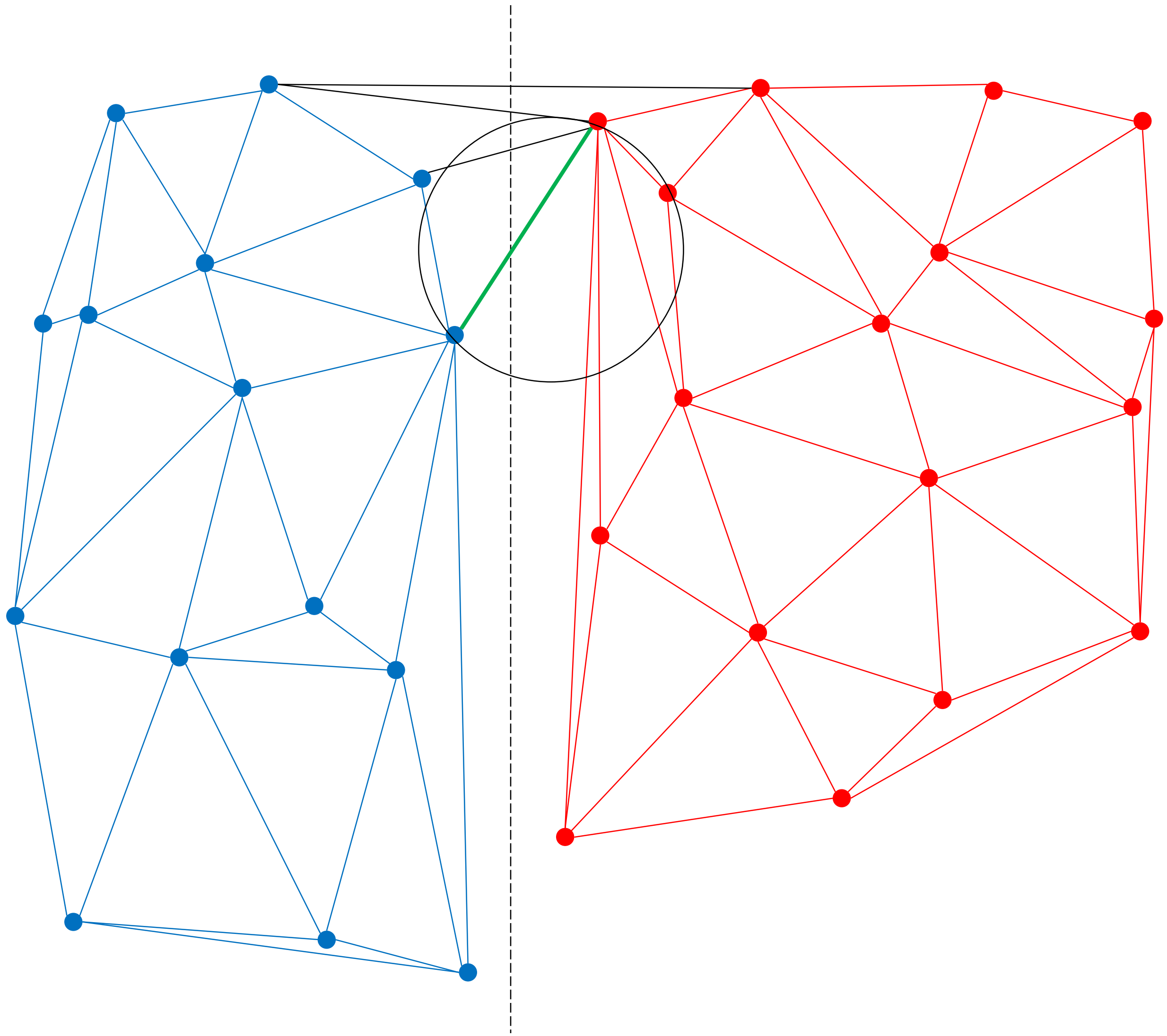
Rising Bubble



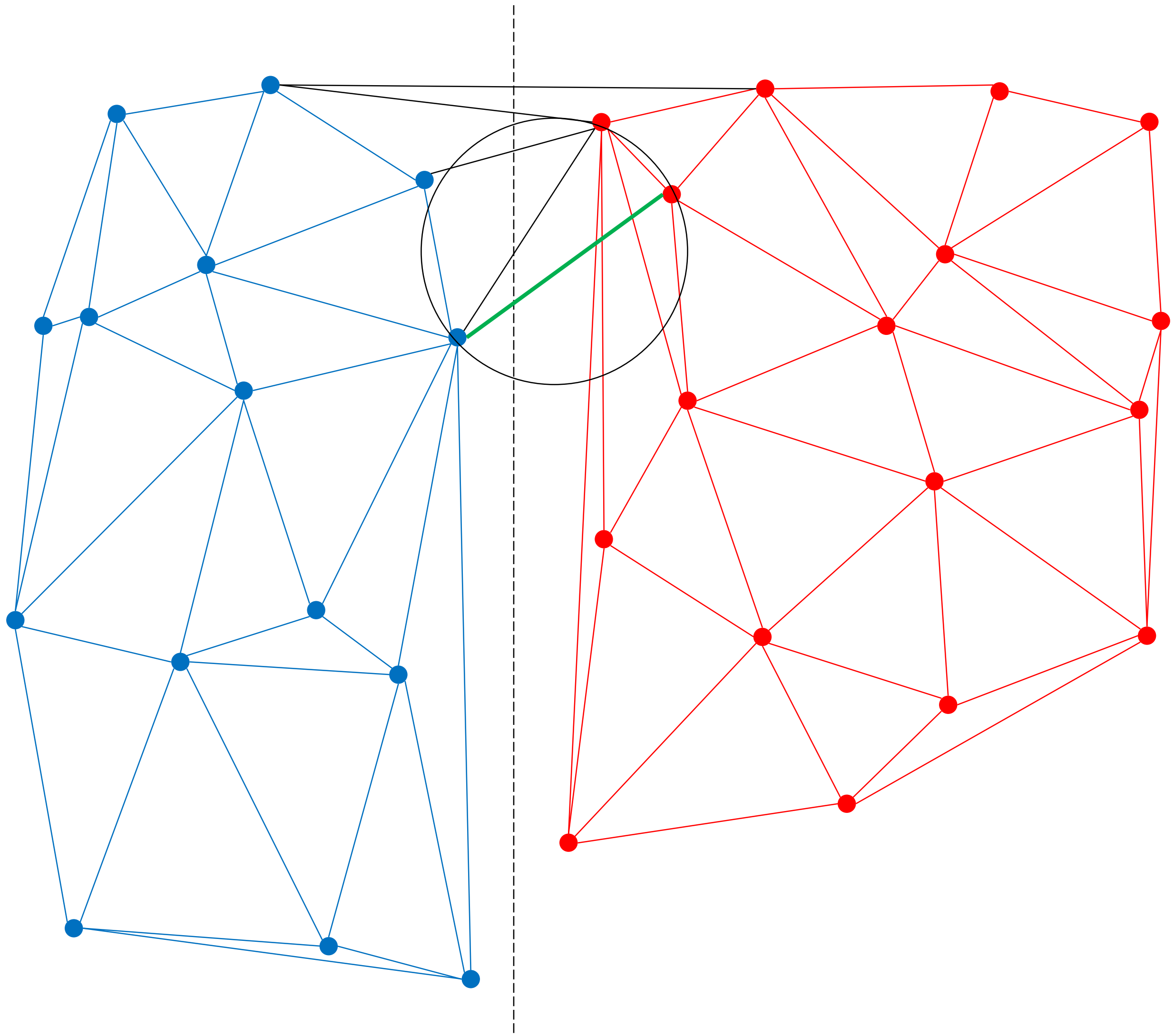
Rising Bubble



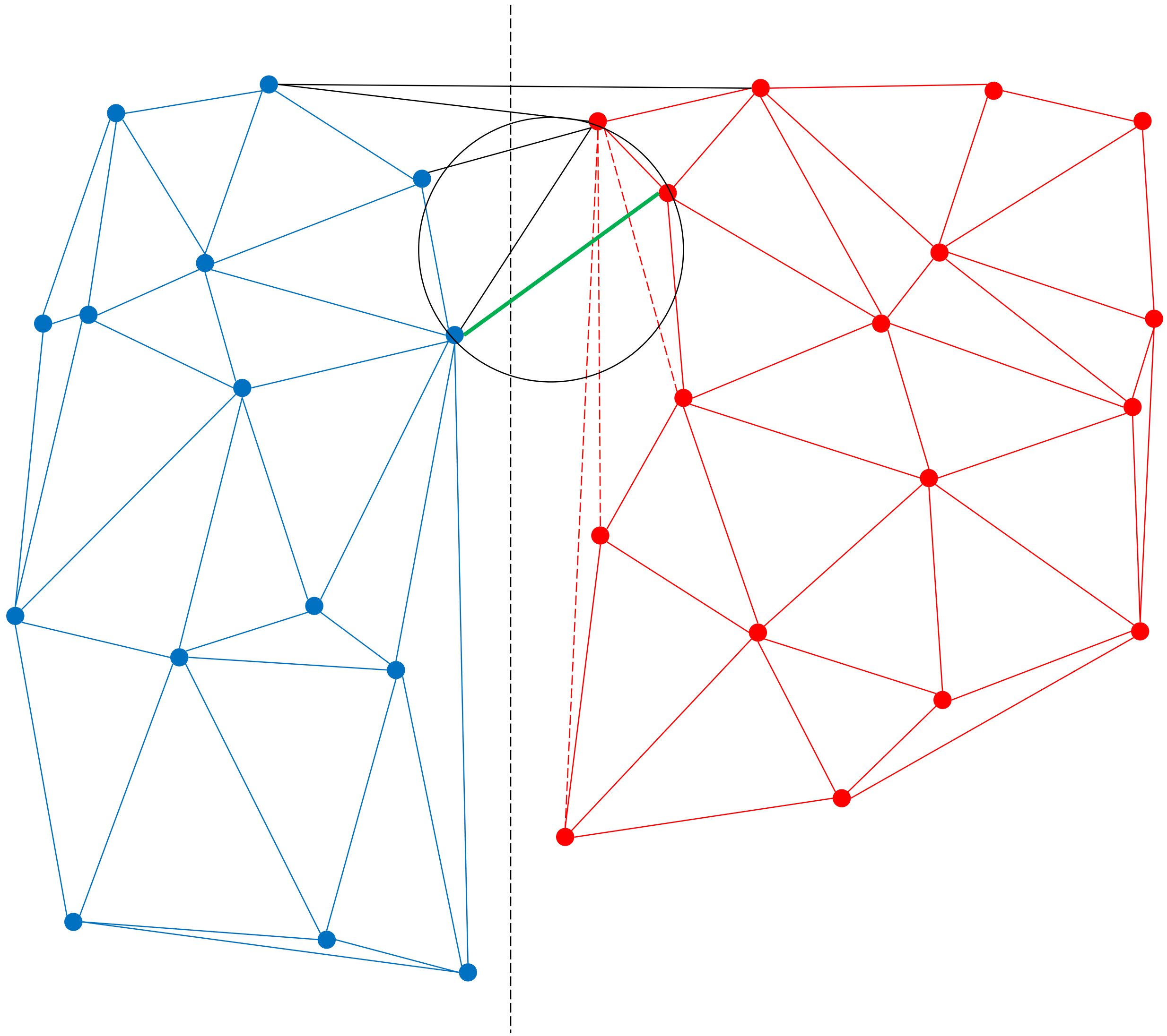
Rising Bubble



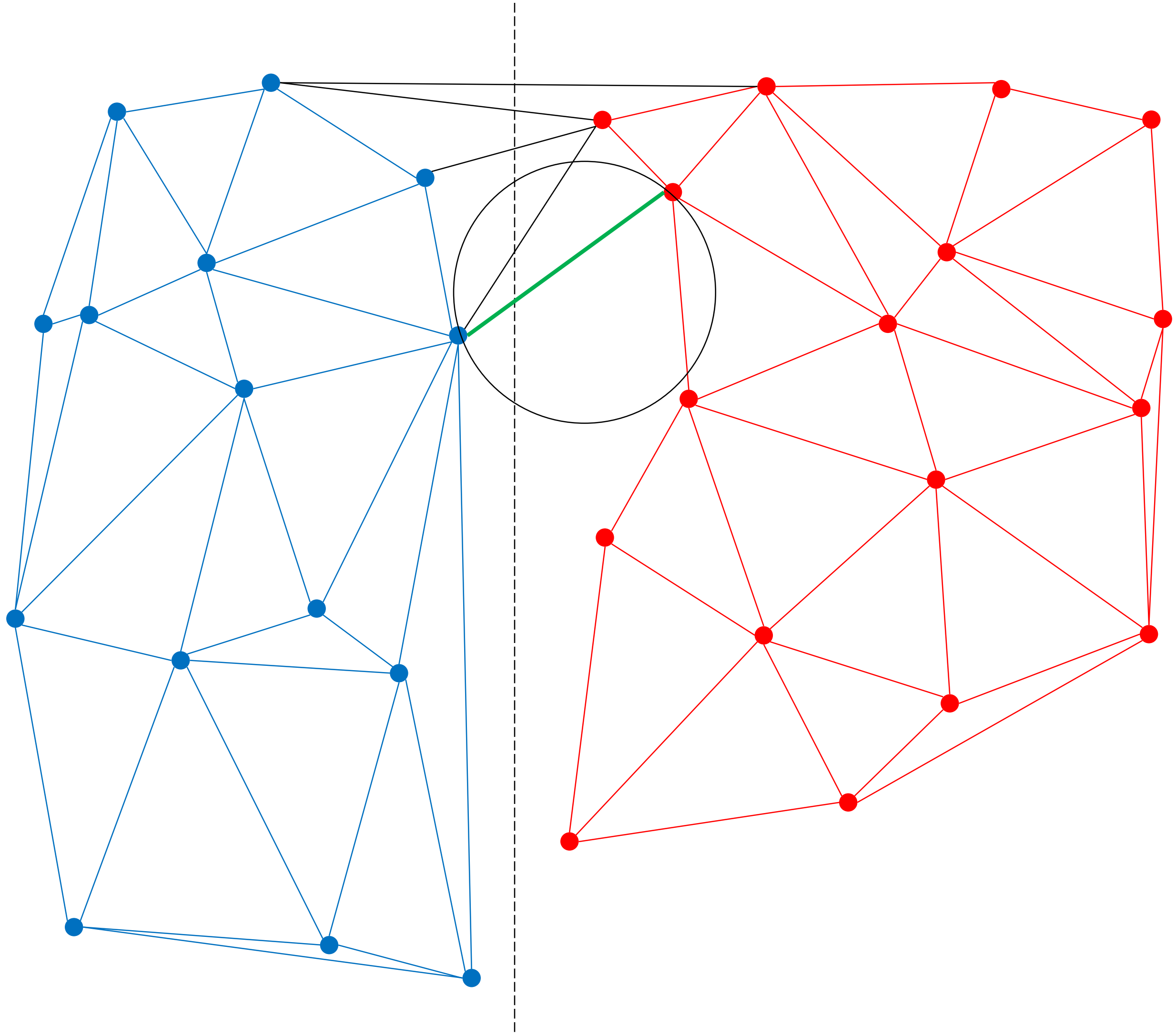
Rising Bubble



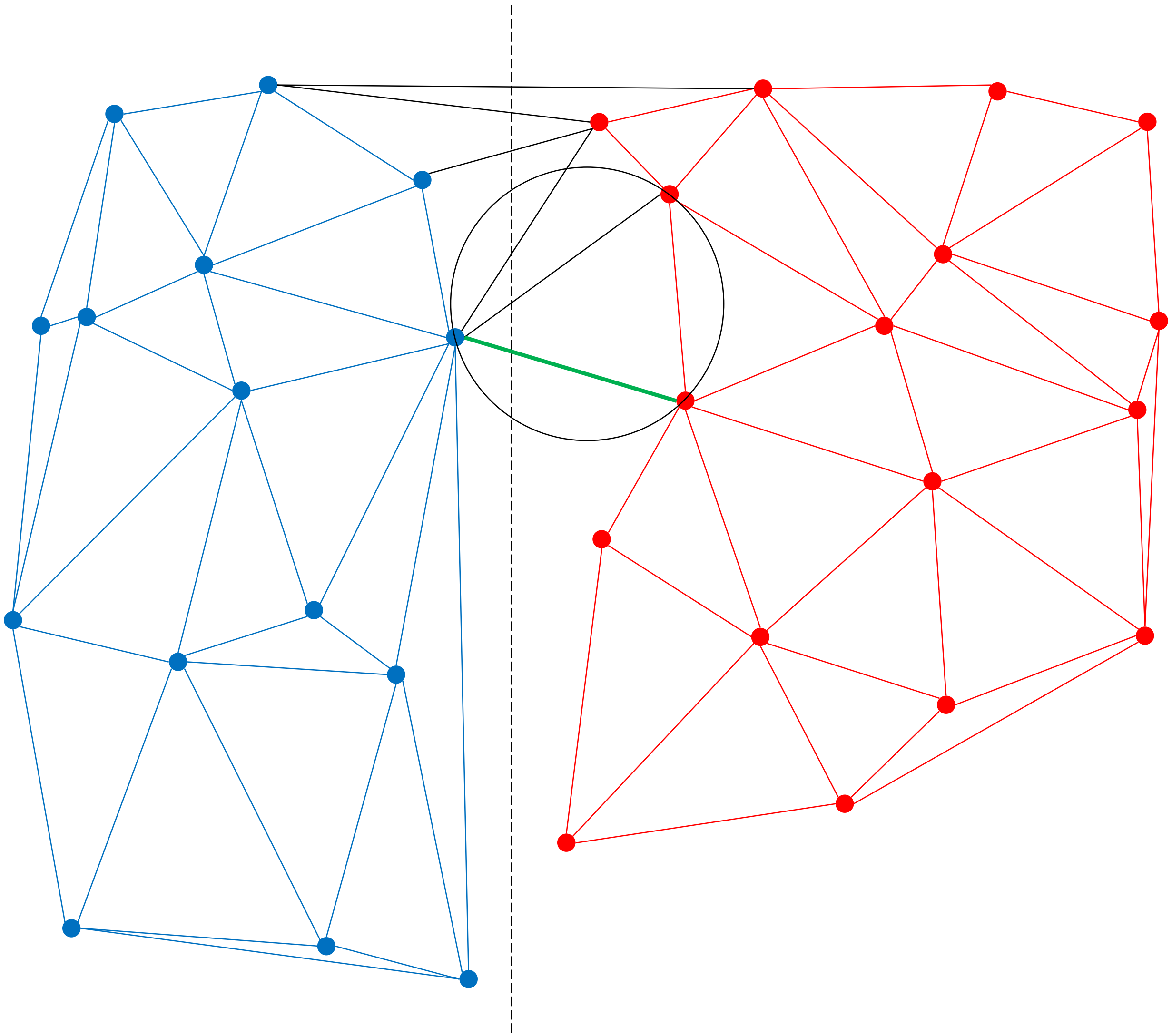
Rising Bubble



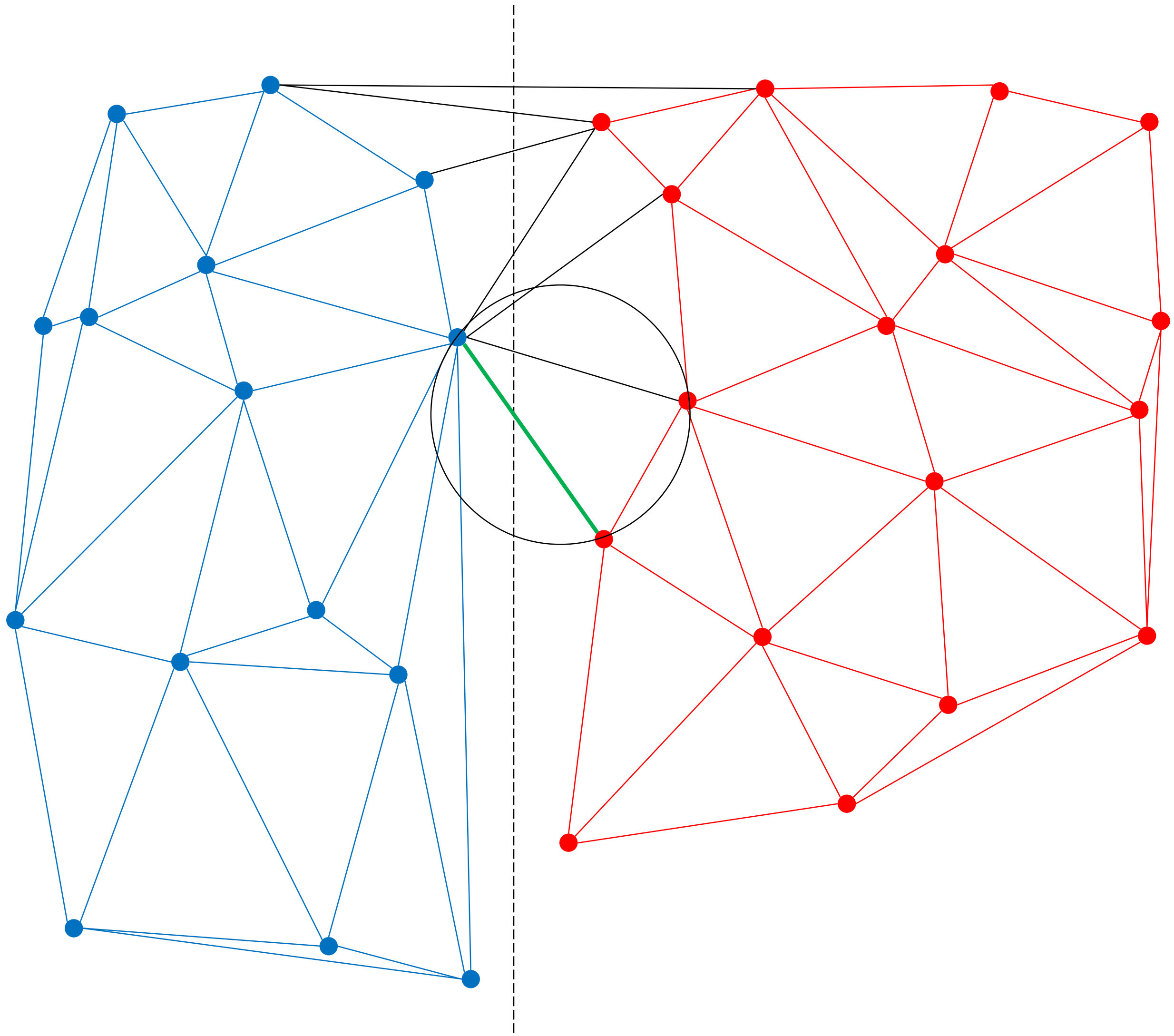
Rising Bubble



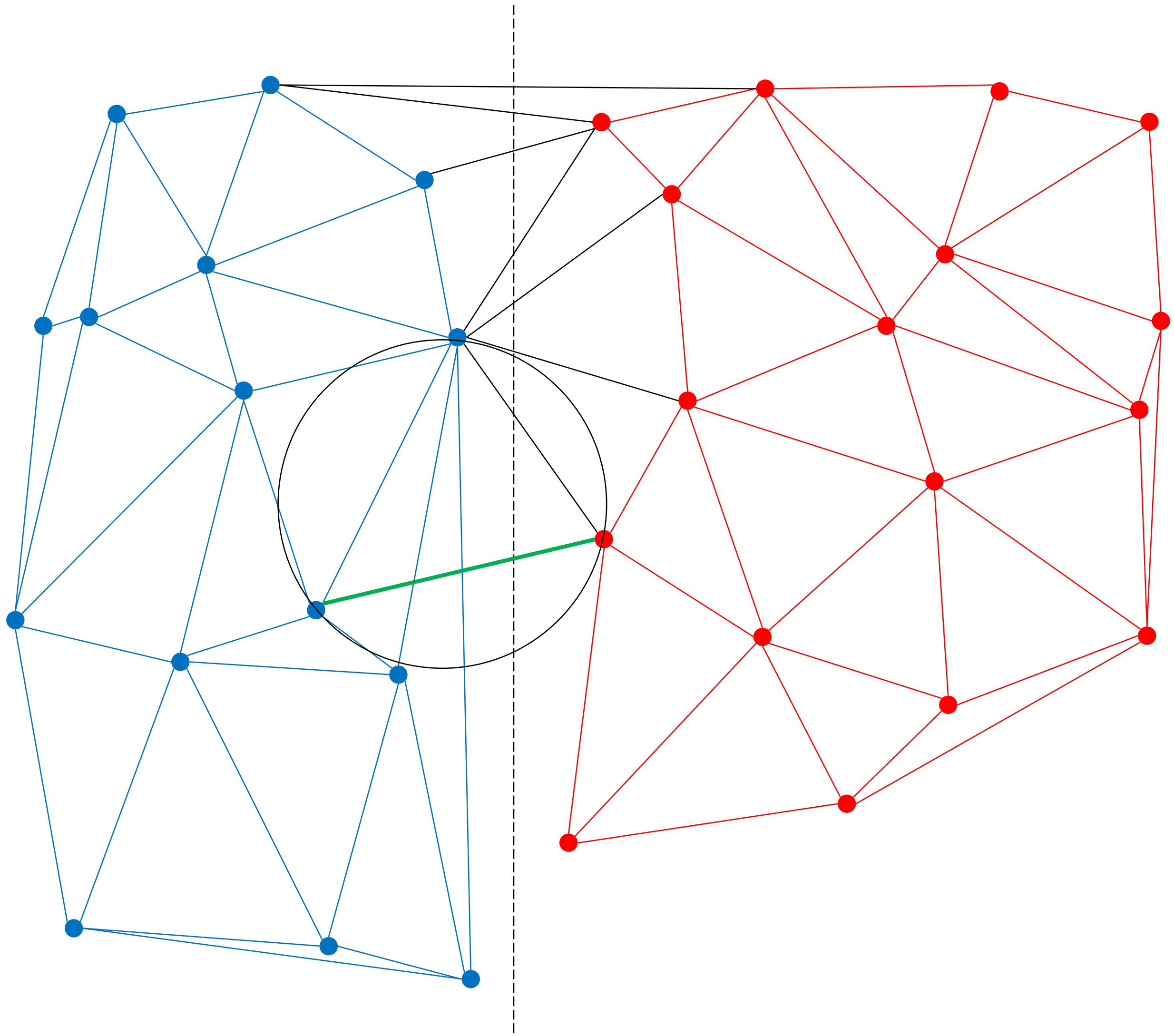
Rising Bubble



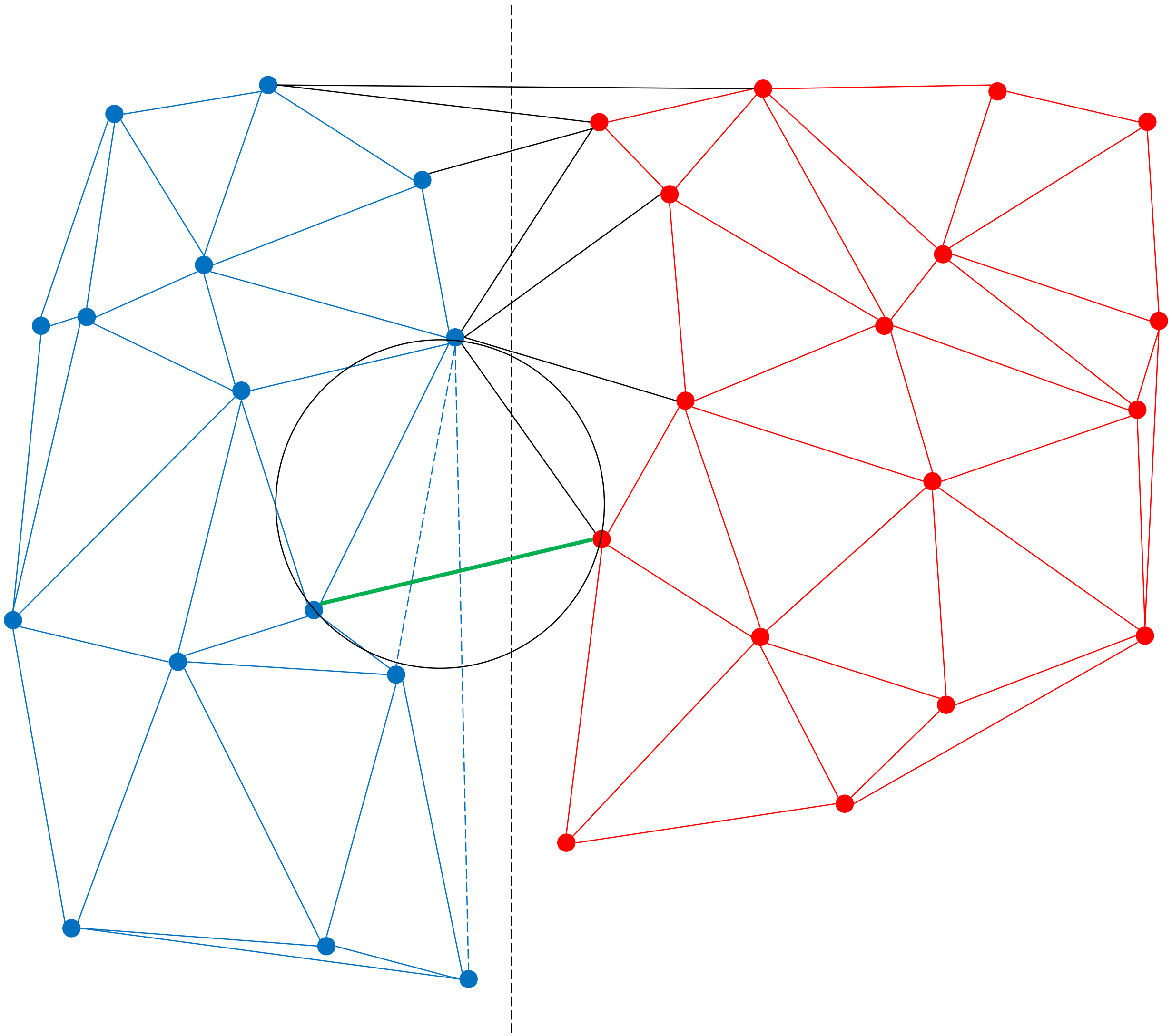
Rising Bubble



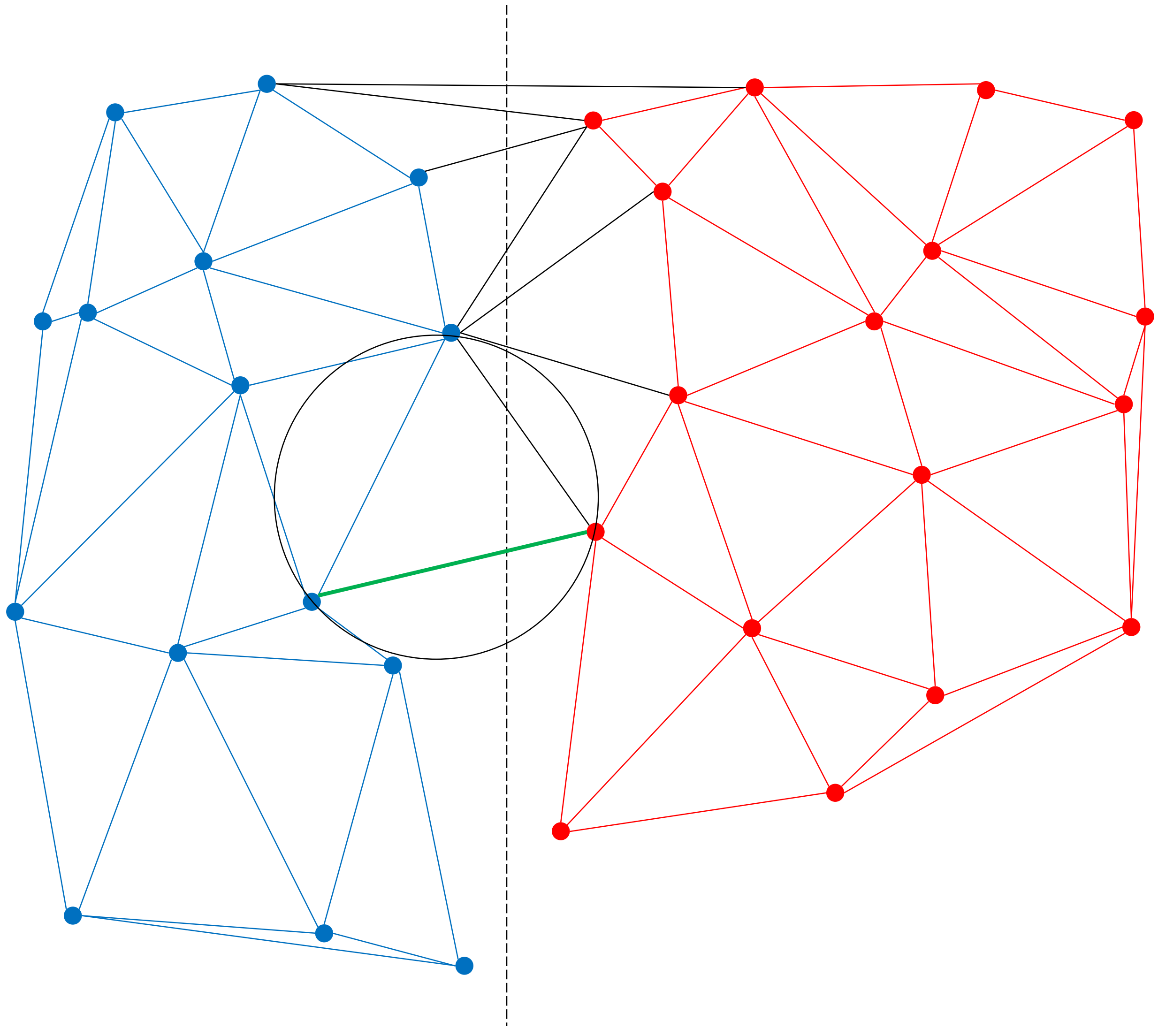
Rising Bubble



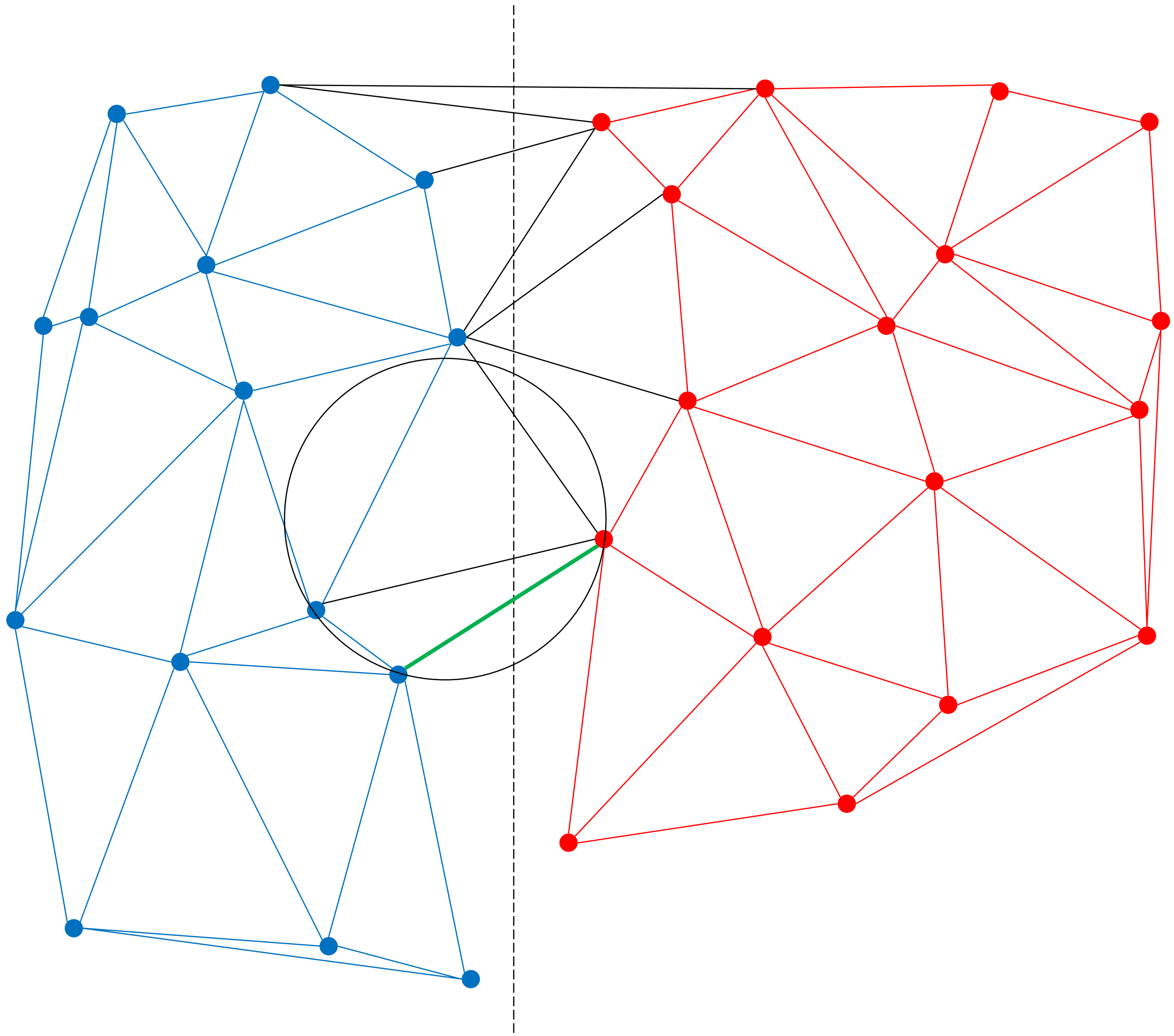
Rising Bubble



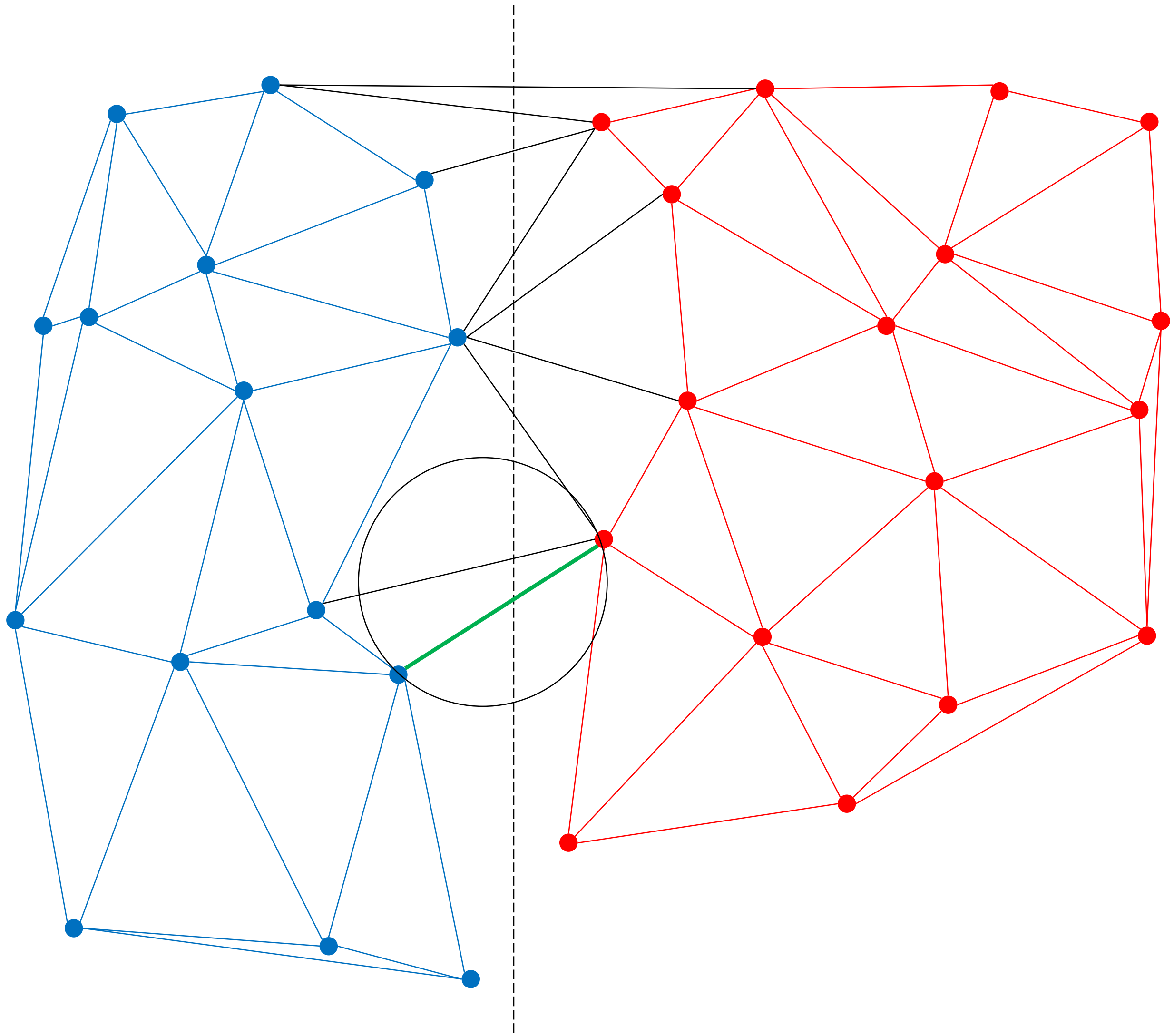
Rising Bubble



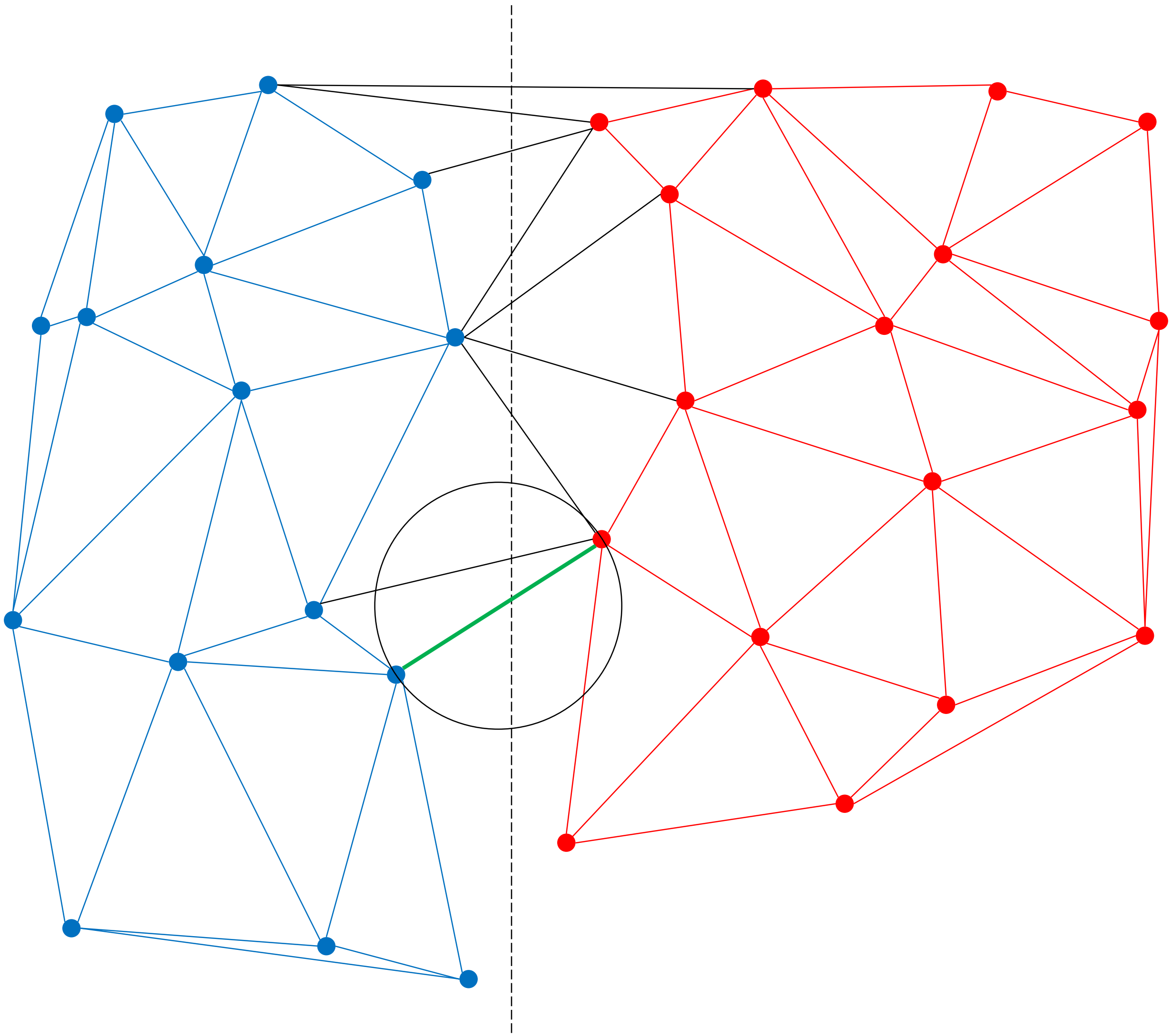
Rising Bubble



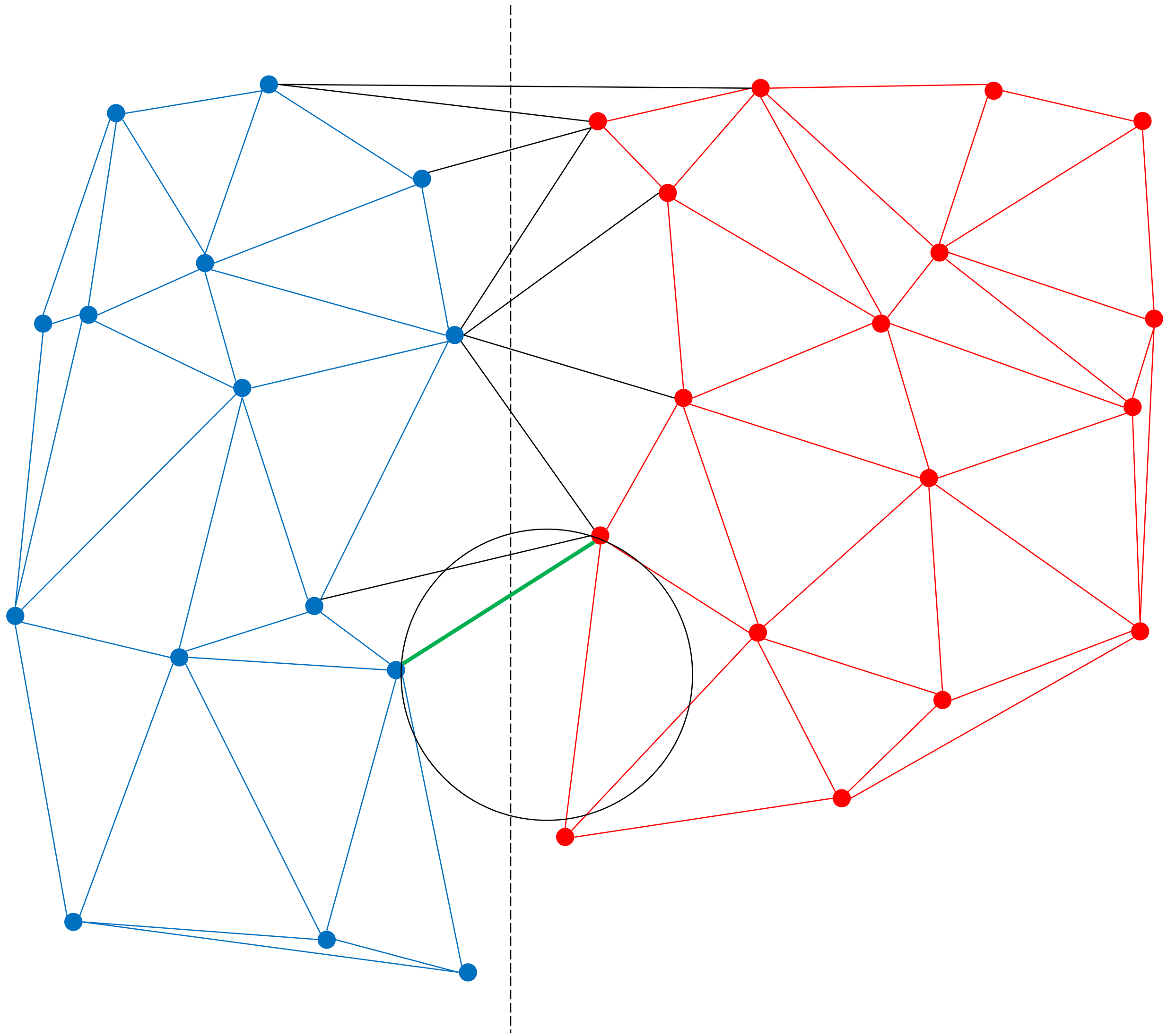
Rising Bubble



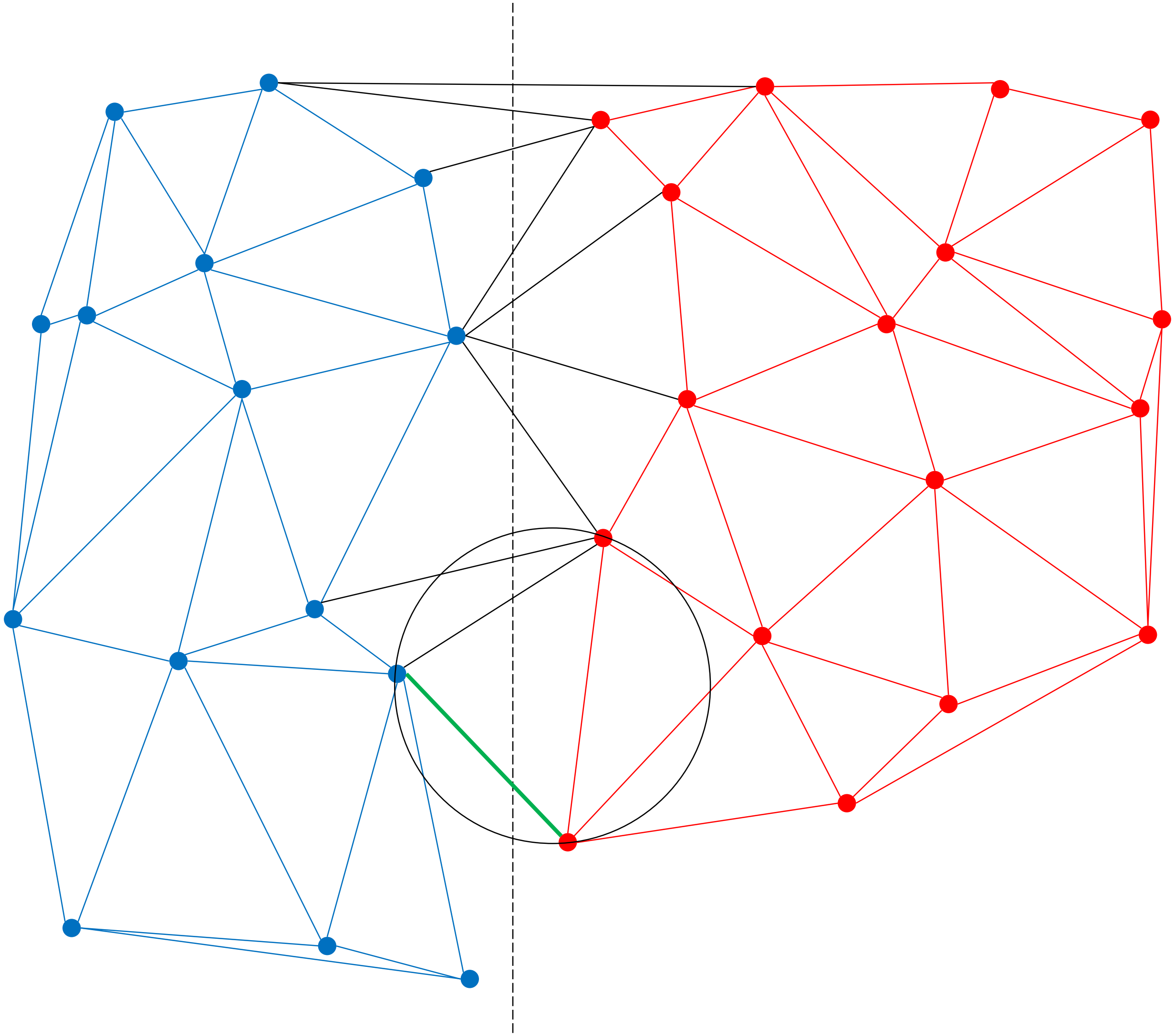
Rising Bubble



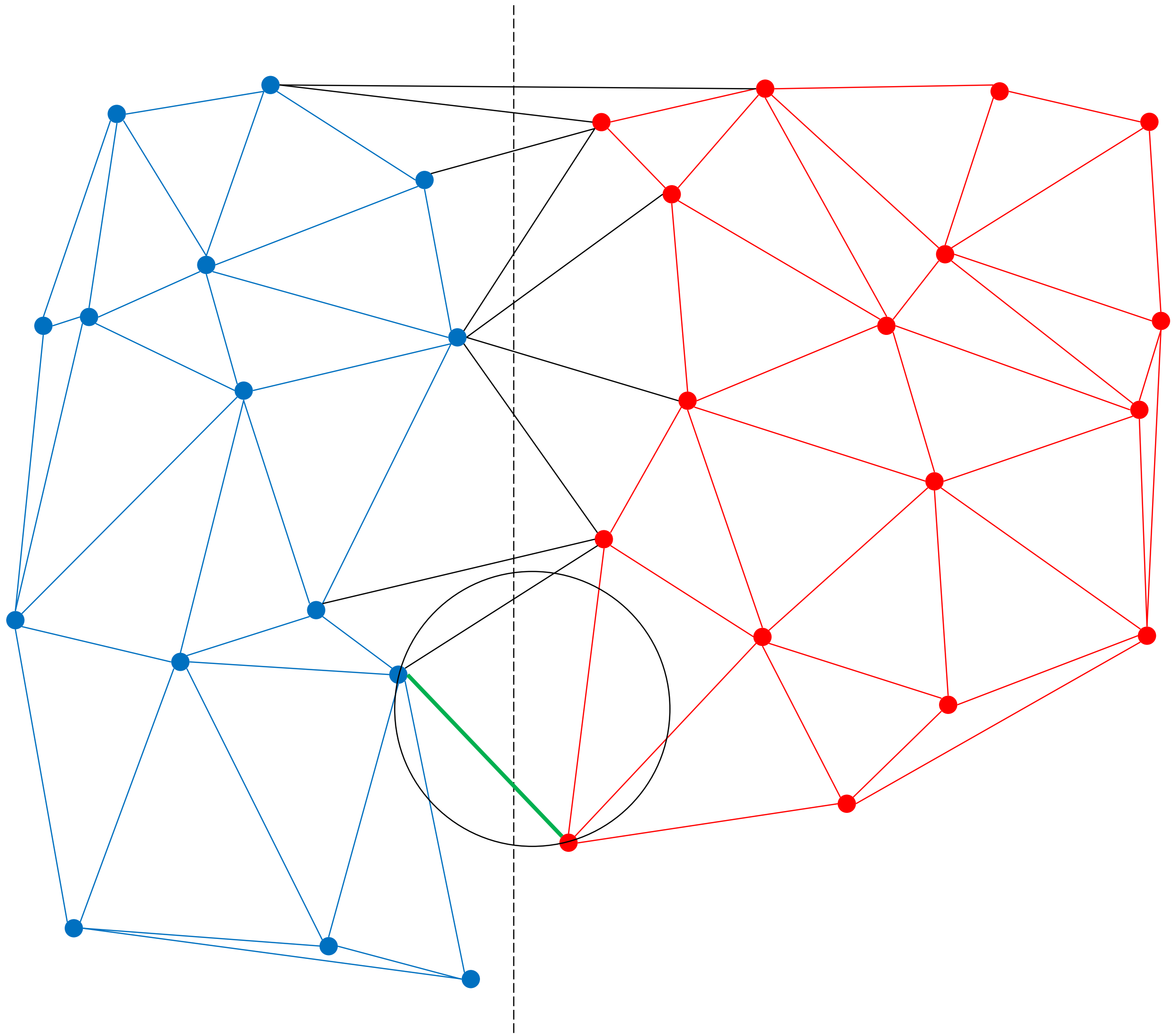
Rising Bubble



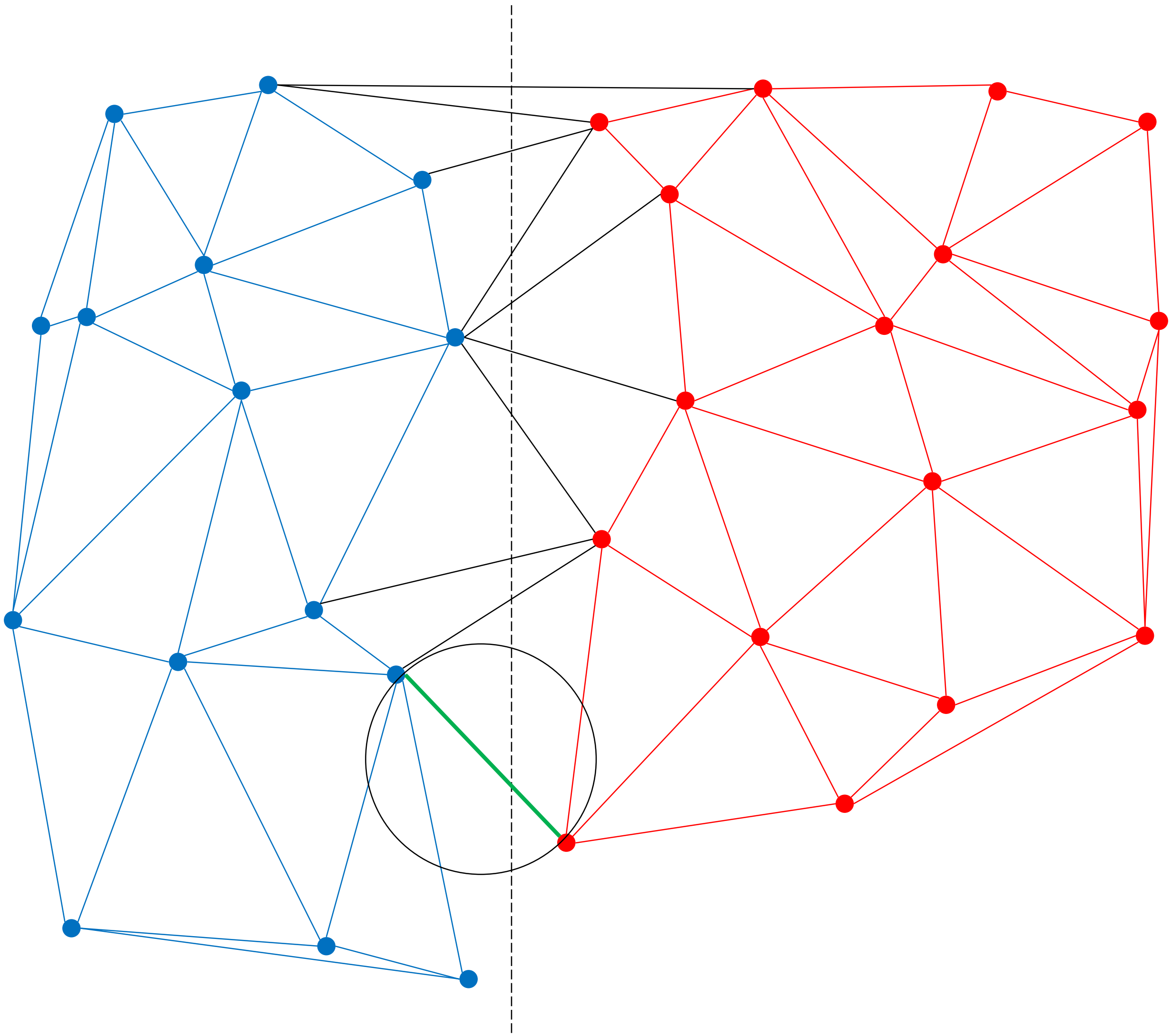
Rising Bubble



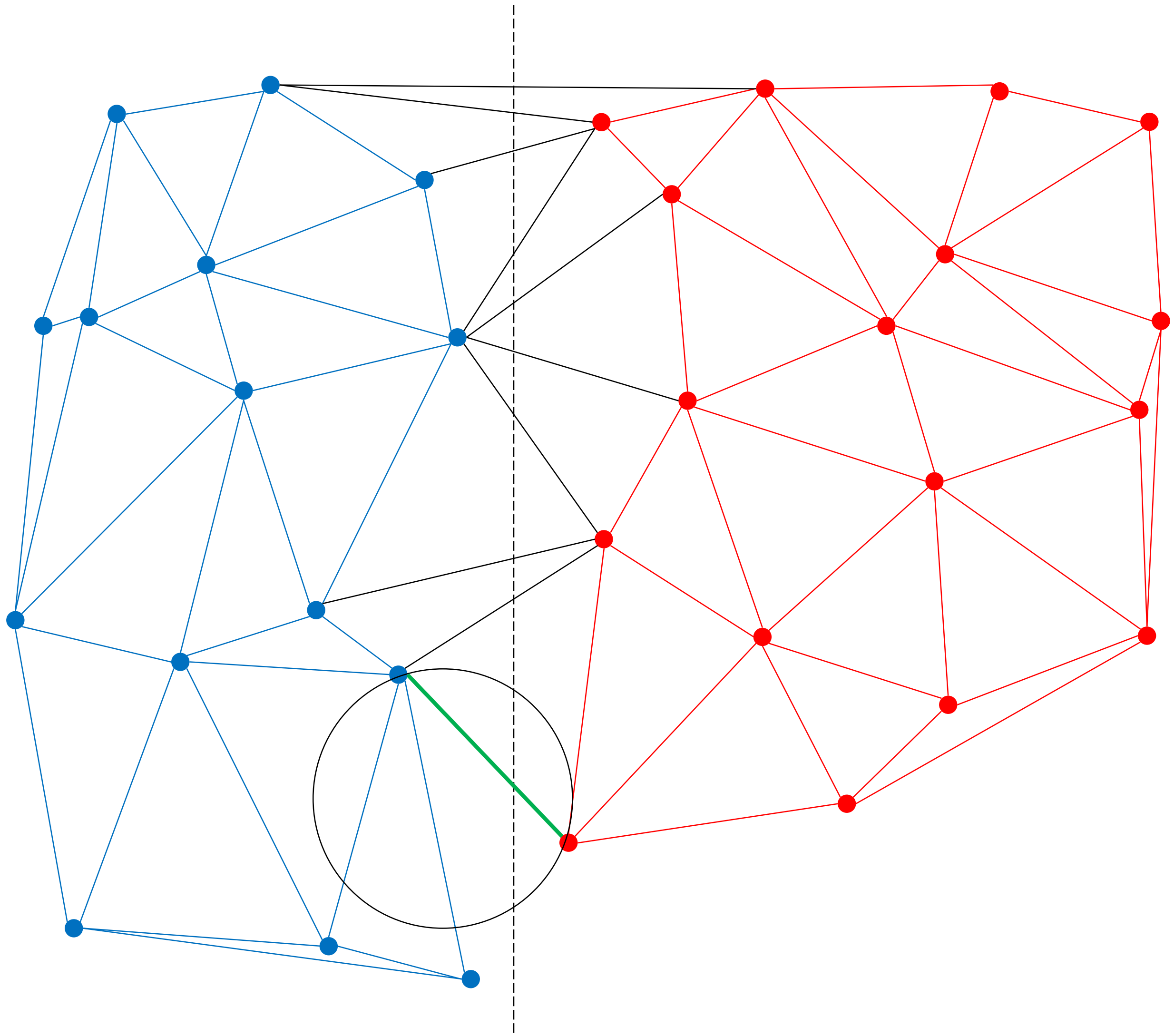
Rising Bubble



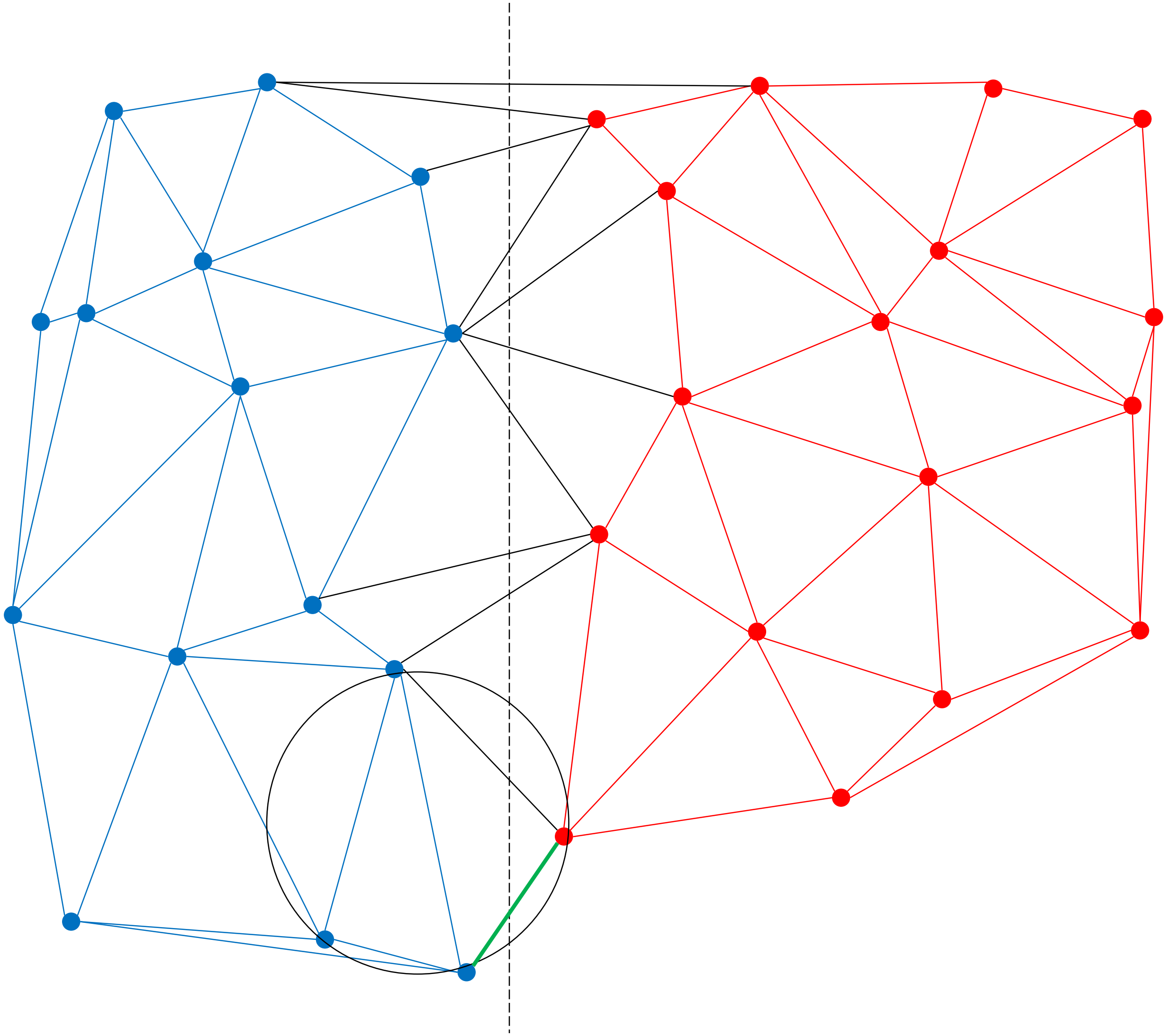
Rising Bubble



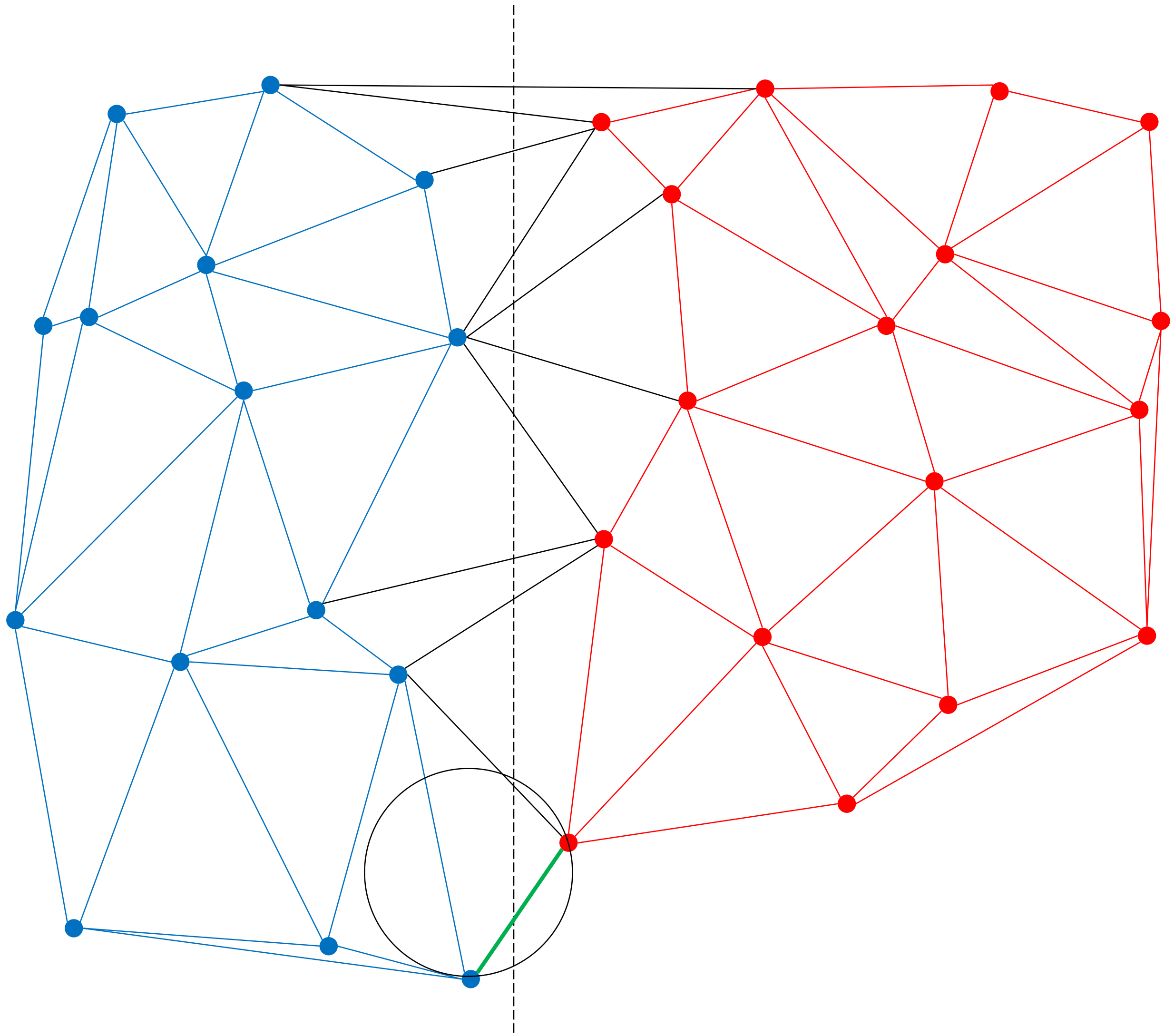
Rising Bubble



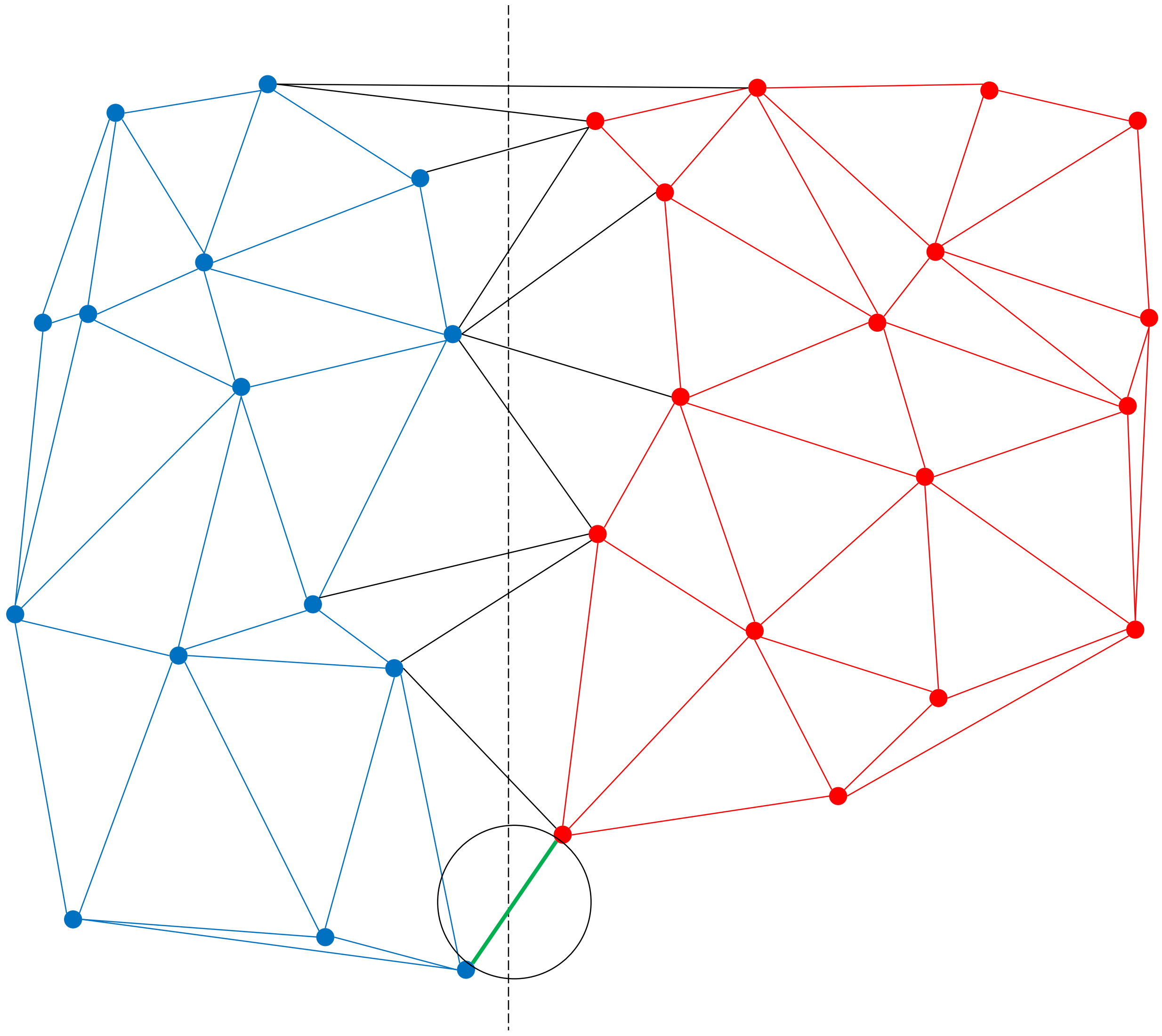
Rising Bubble



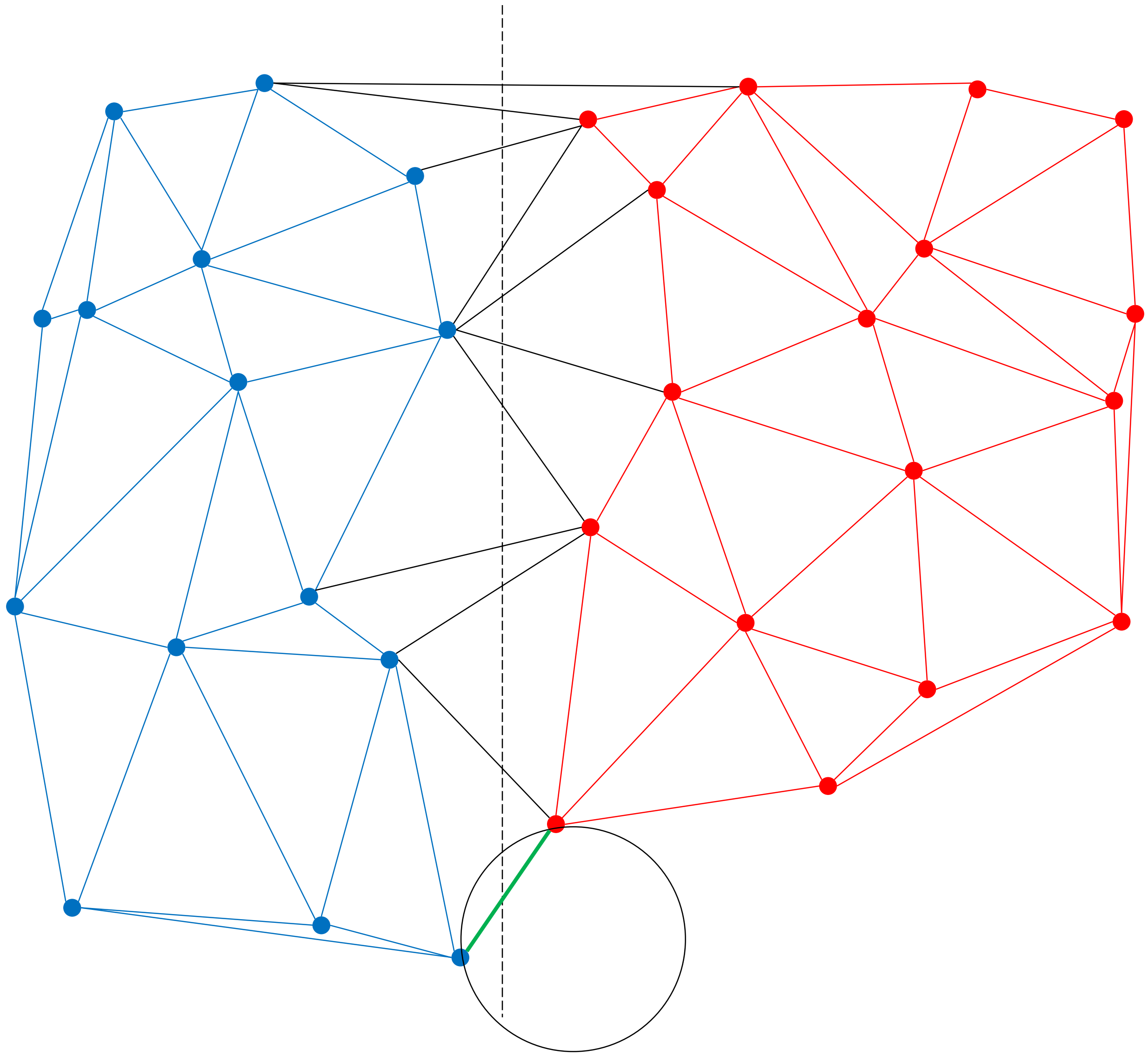
Rising Bubble



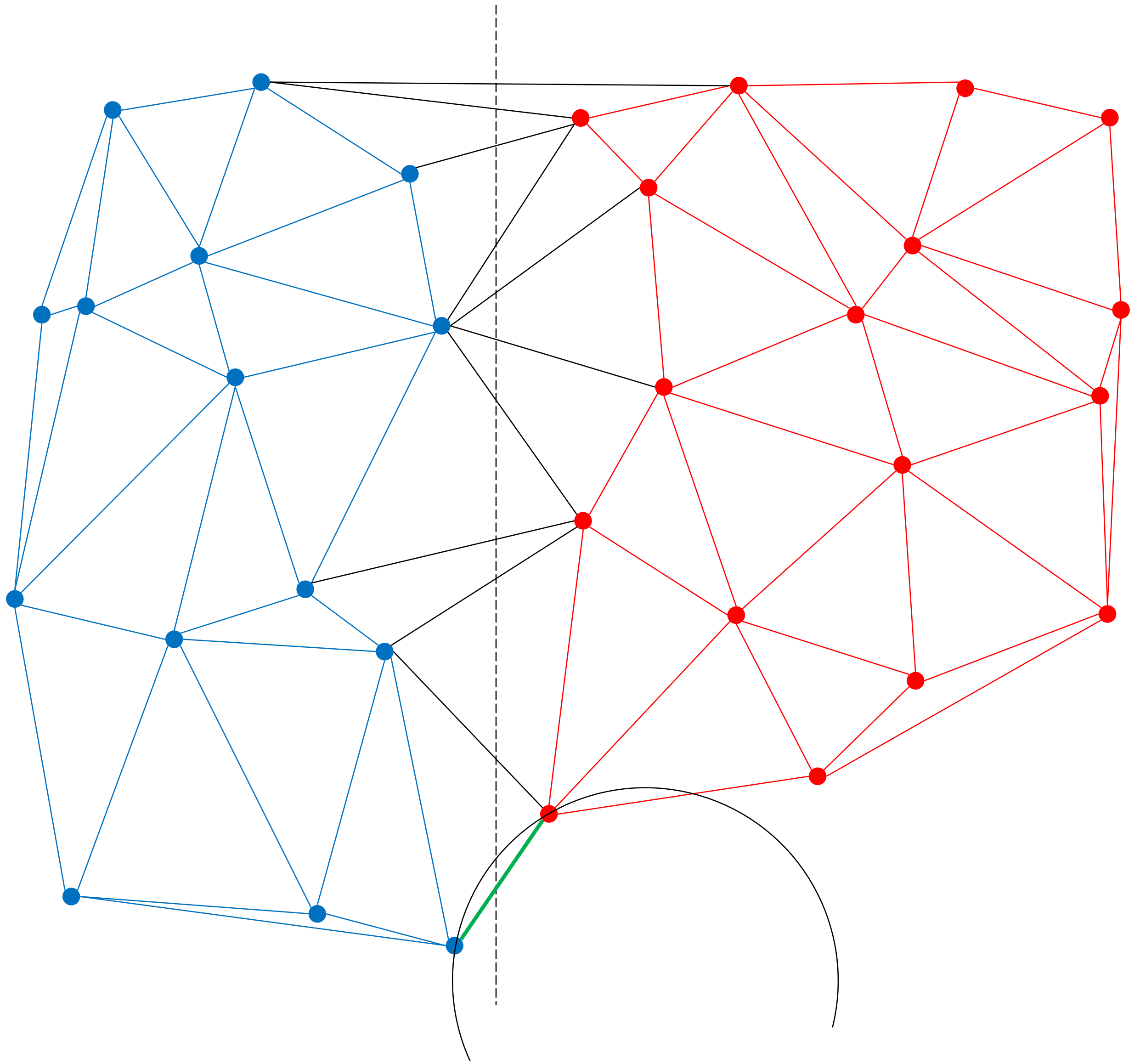
Rising Bubble



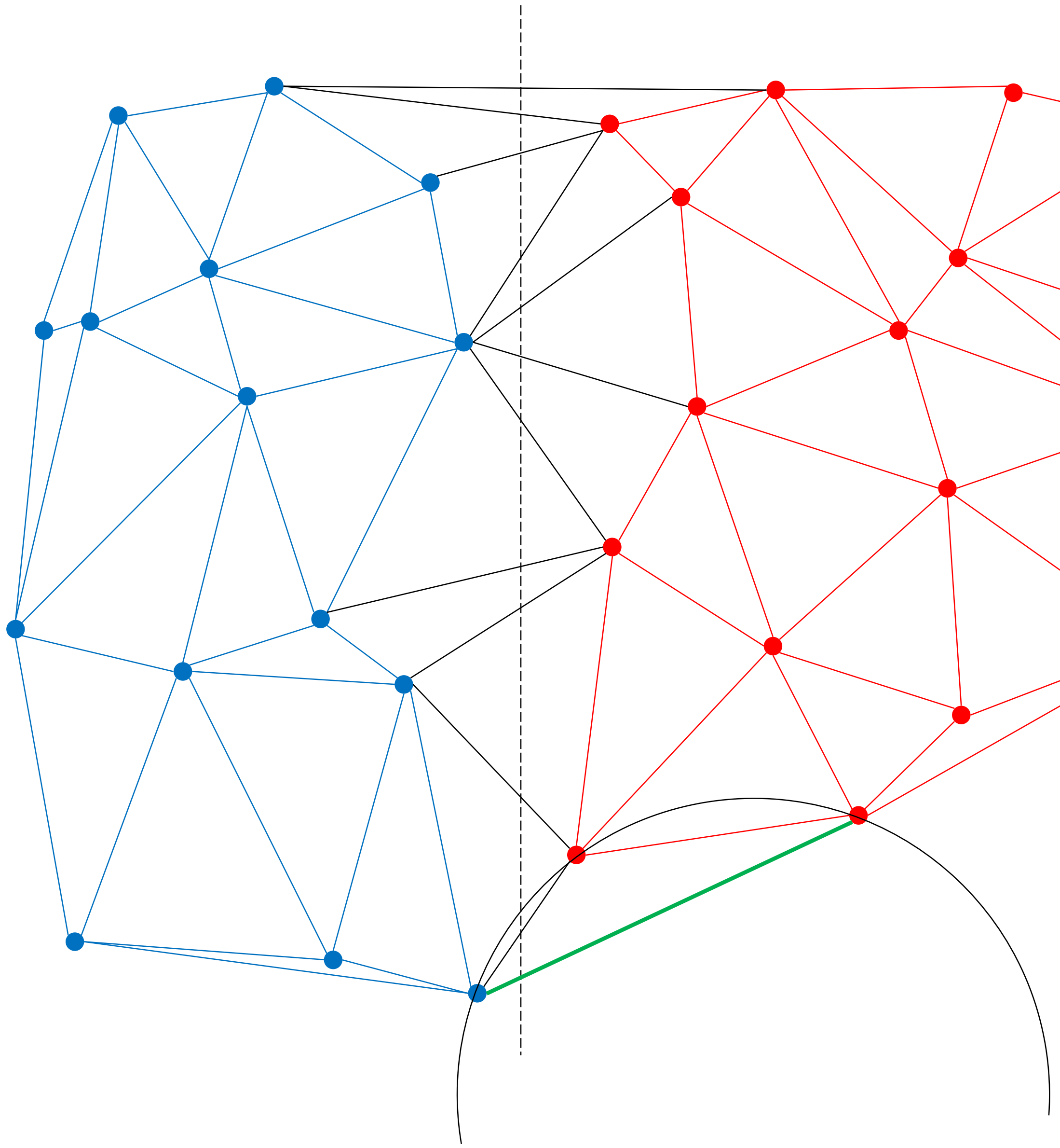
Rising Bubble



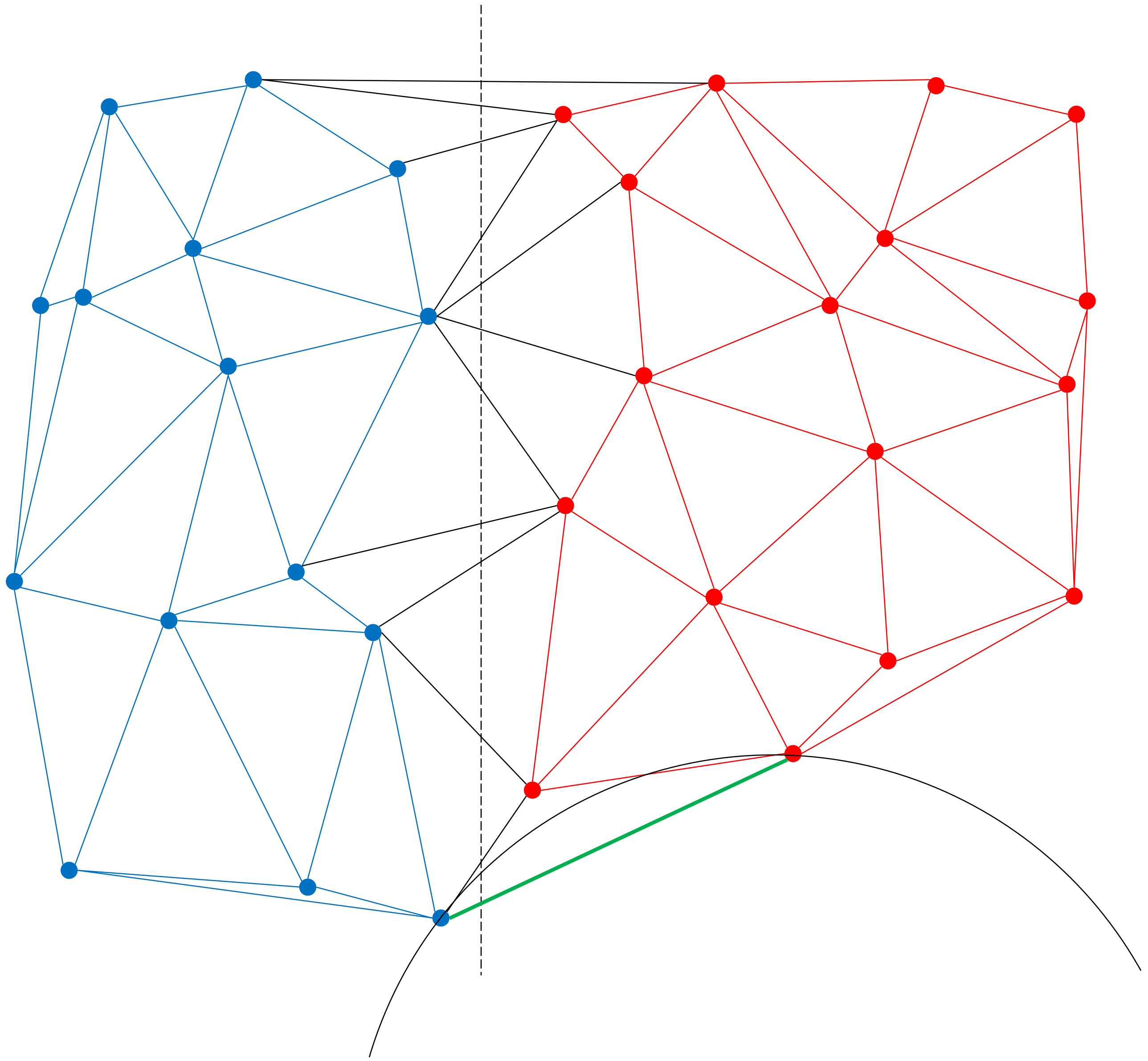
Rising Bubble



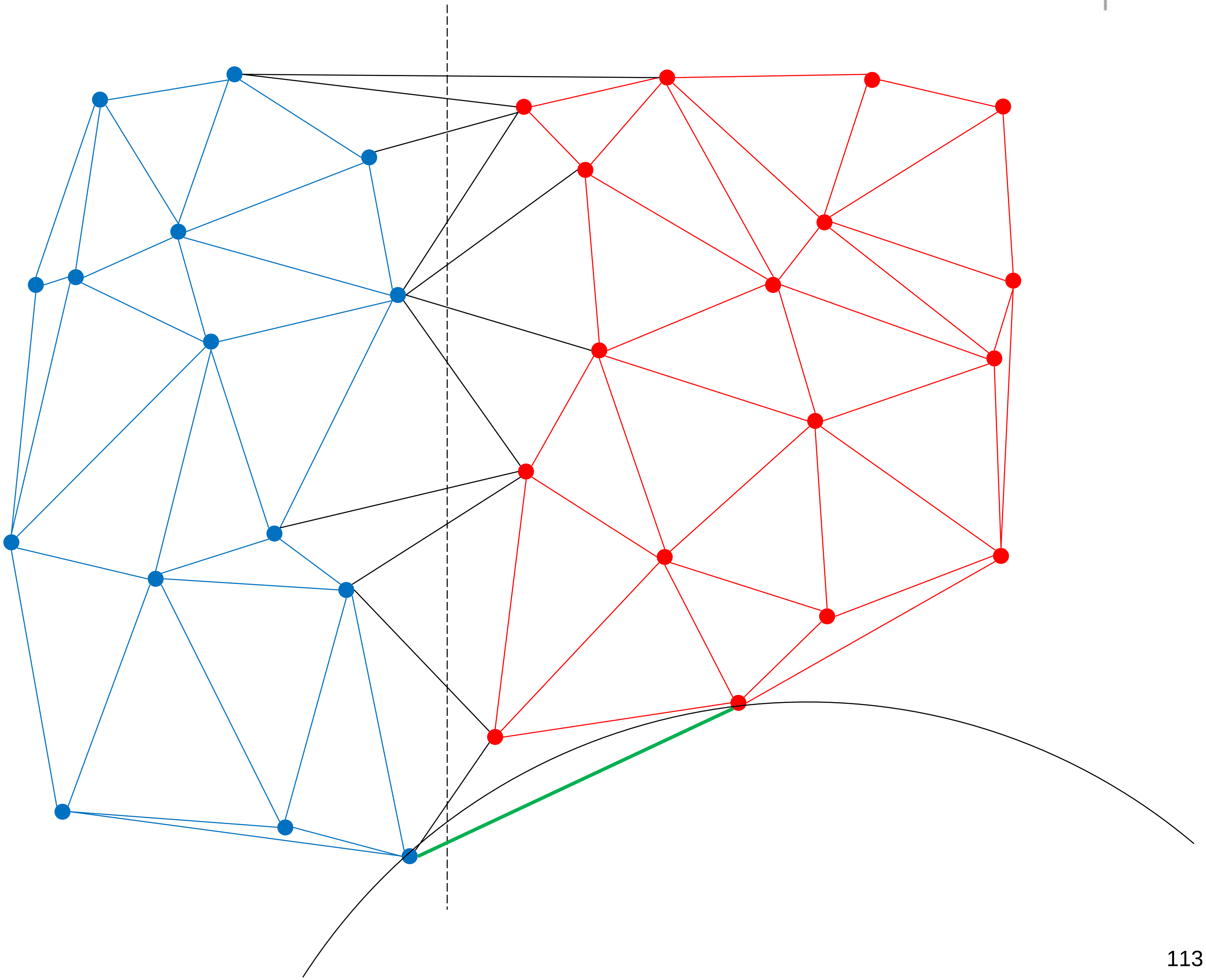
Rising Bubble



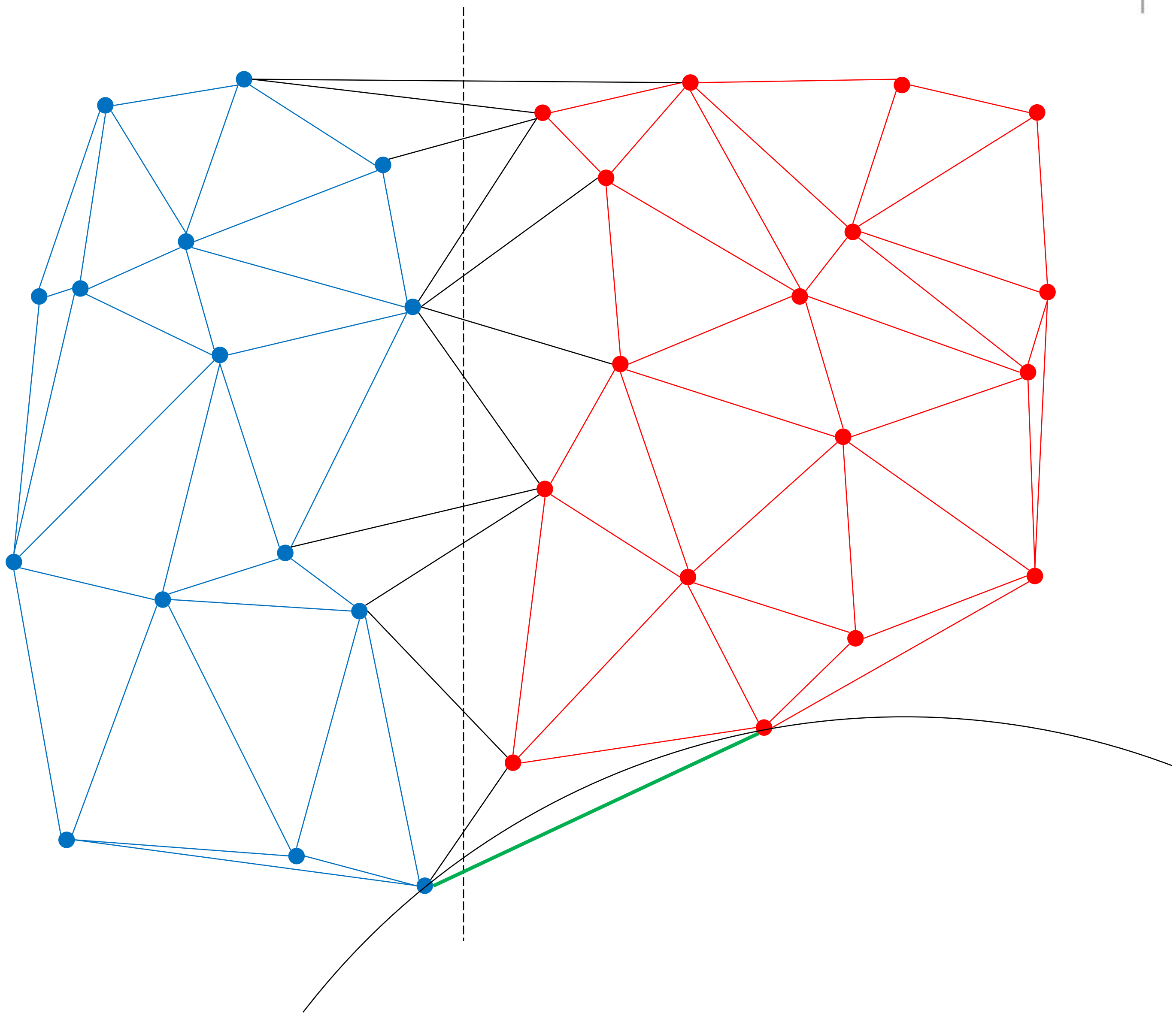
Rising Bubble



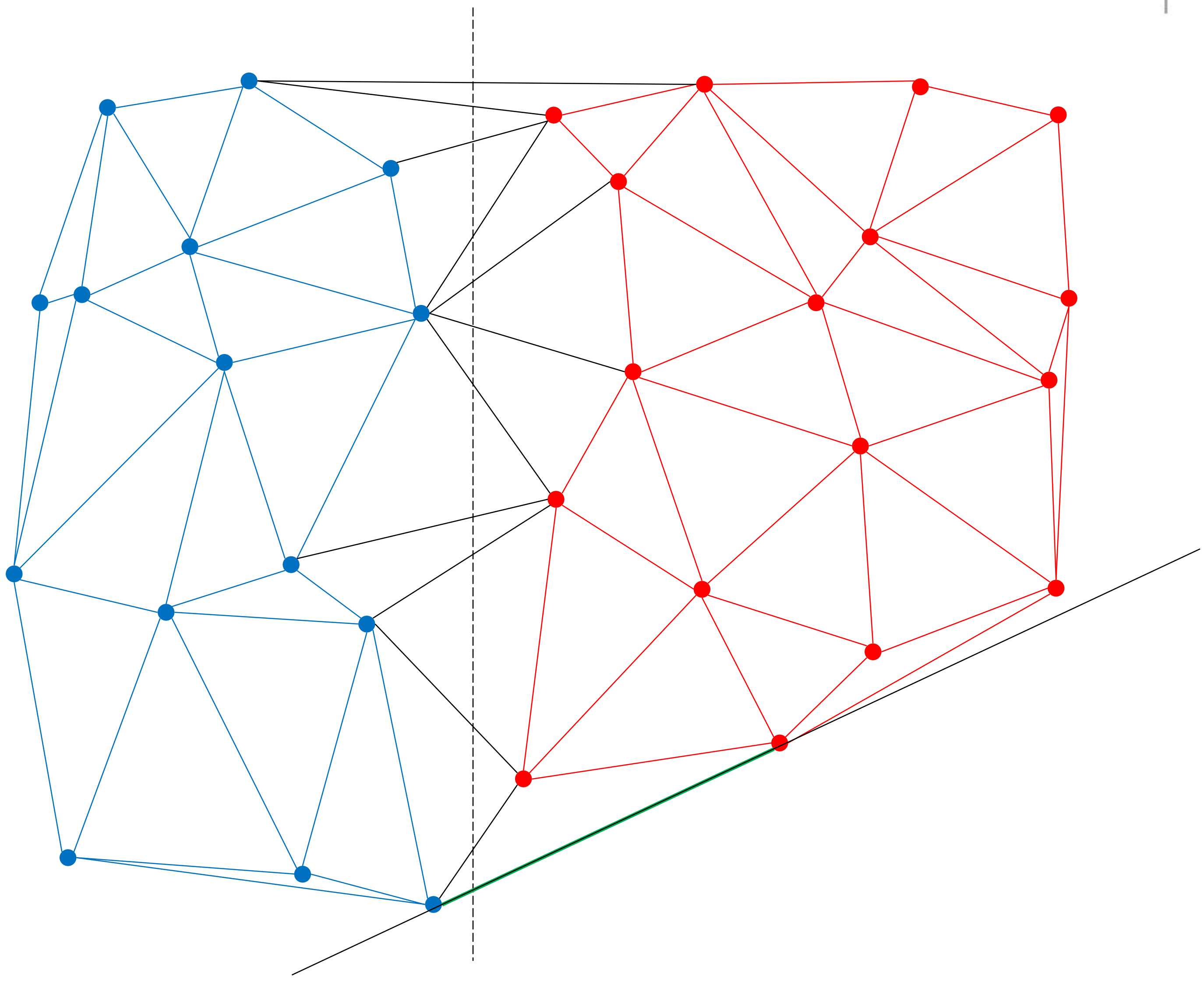
Rising Bubble



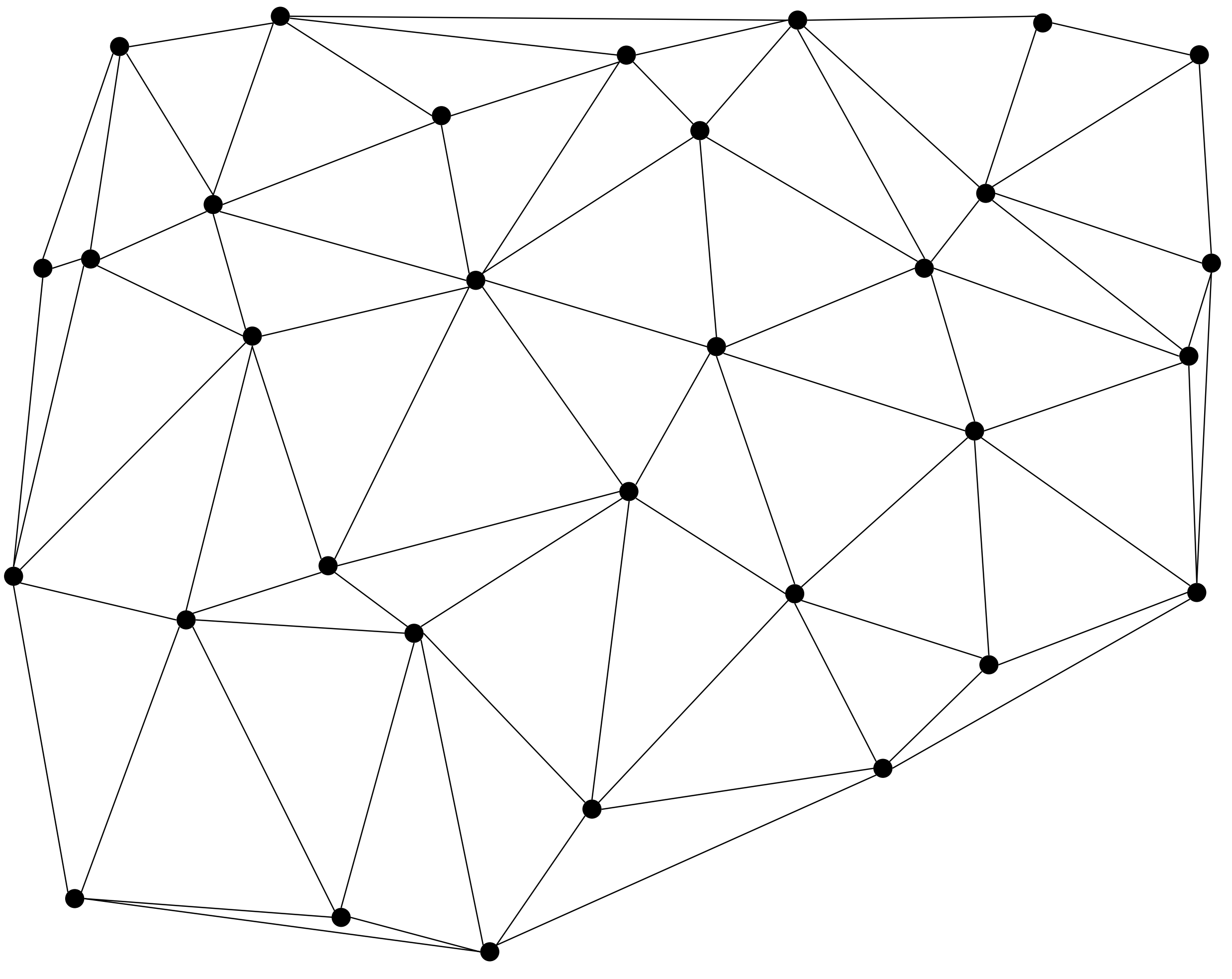
Rising Bubble



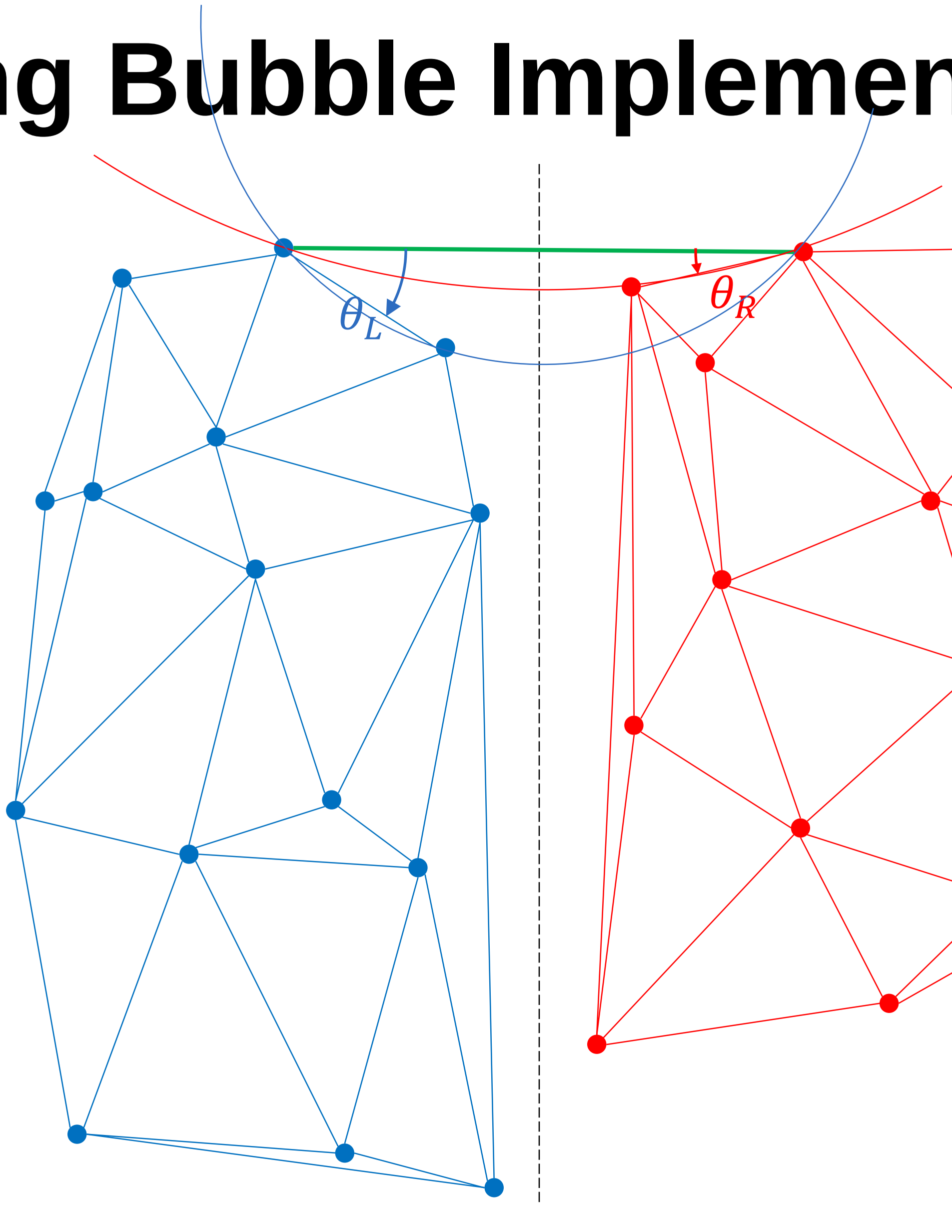
Rising Bubble



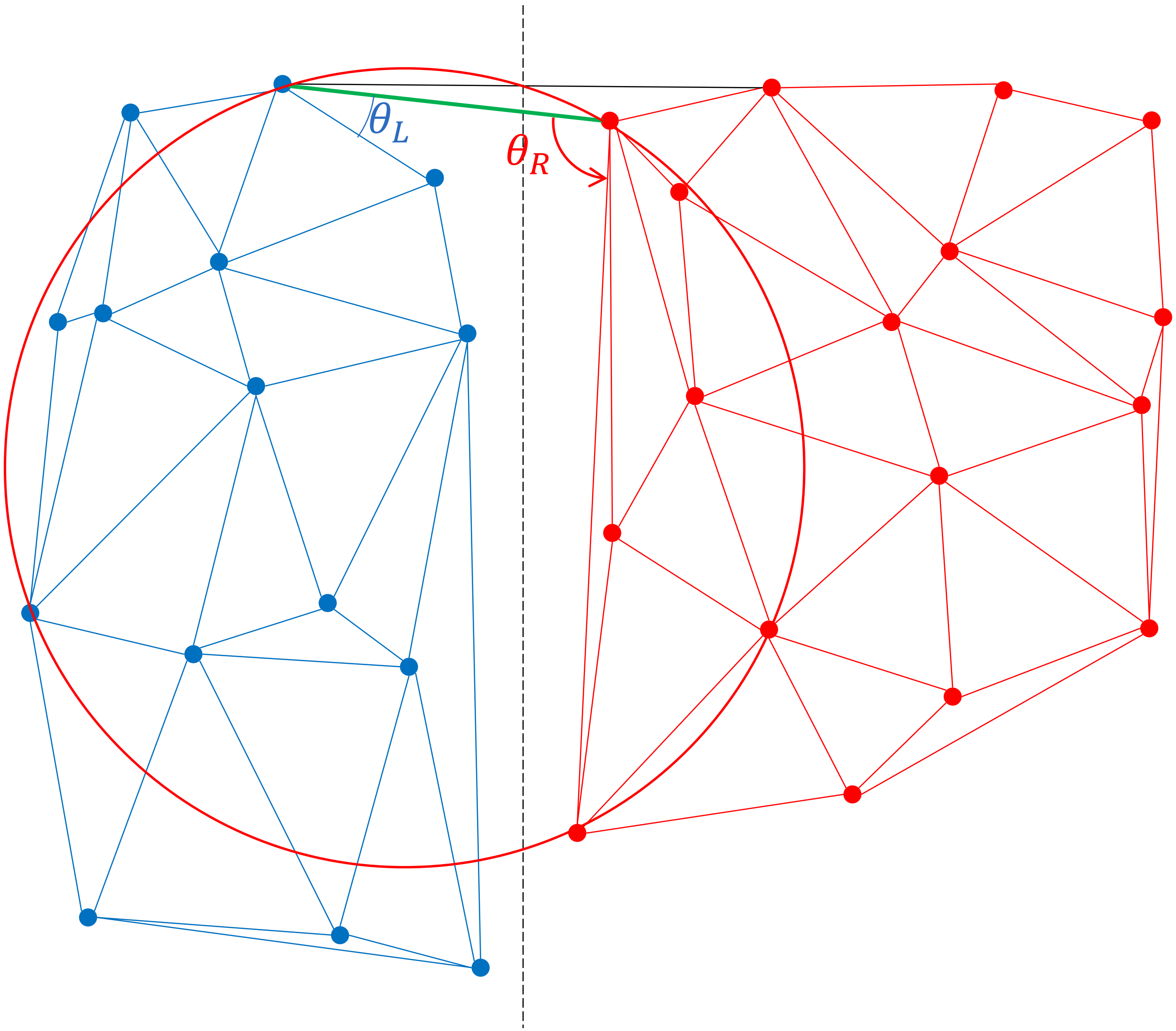
Terminate



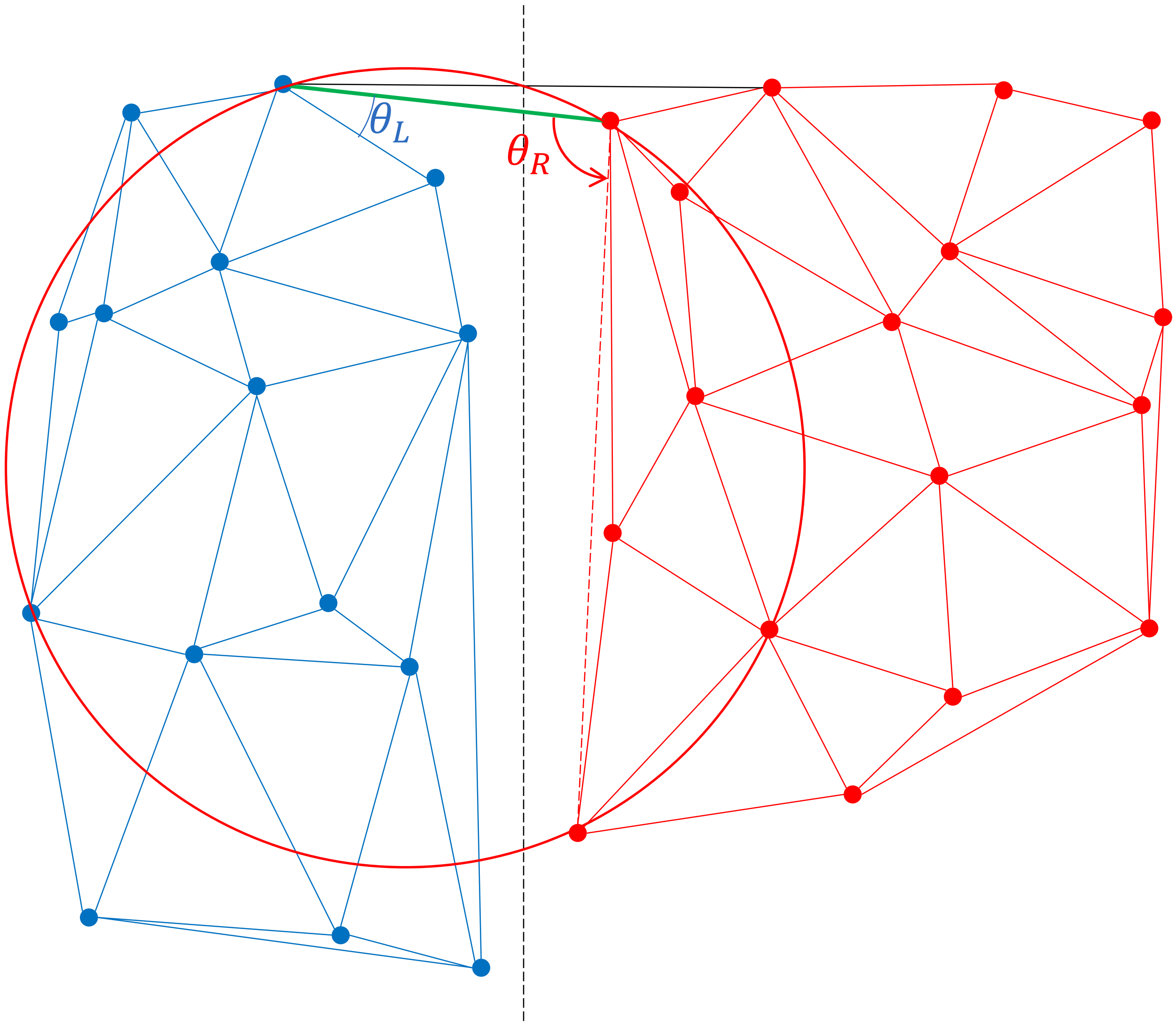
Rising Bubble Implementation



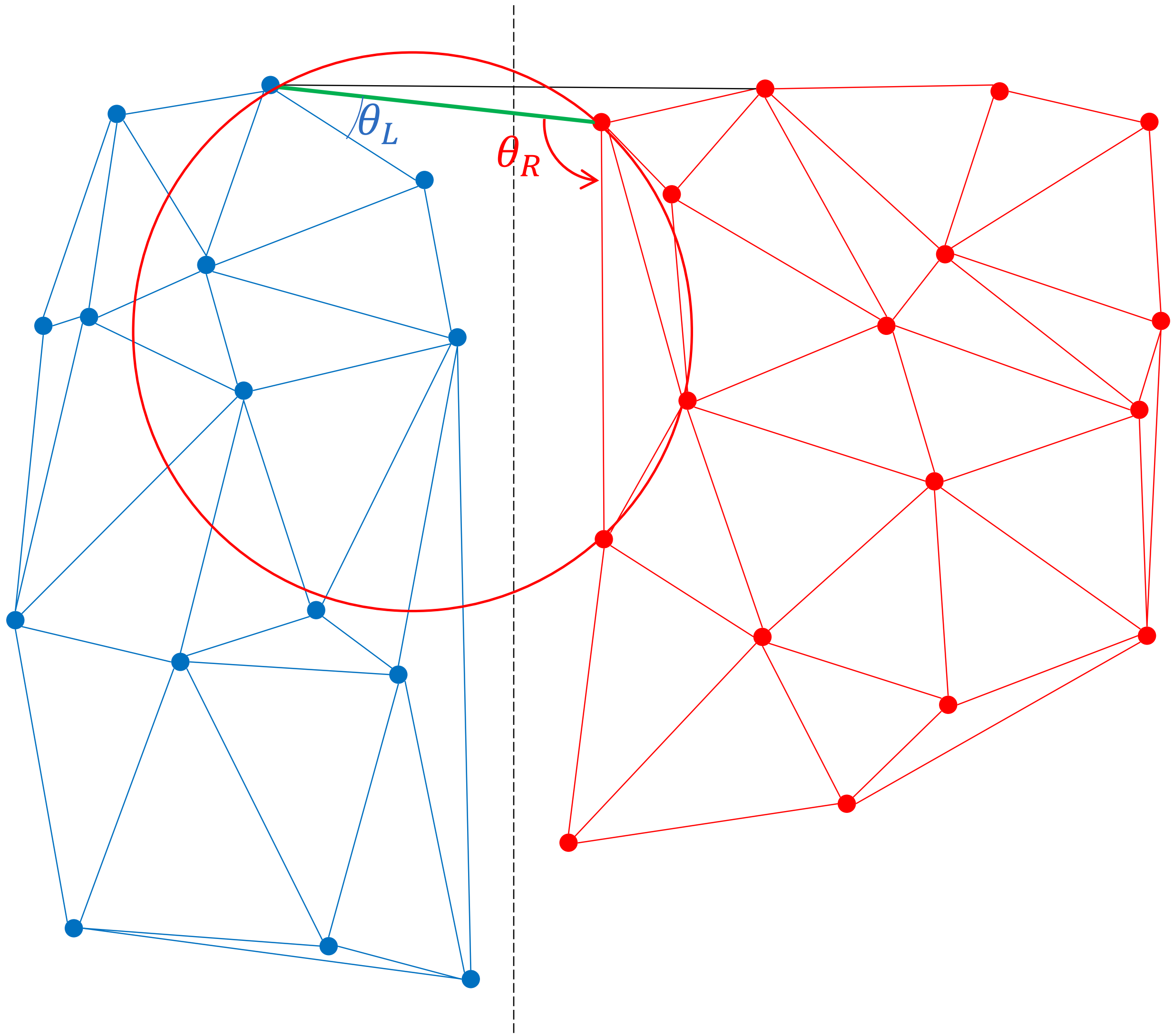
Rising Bubble Implementation



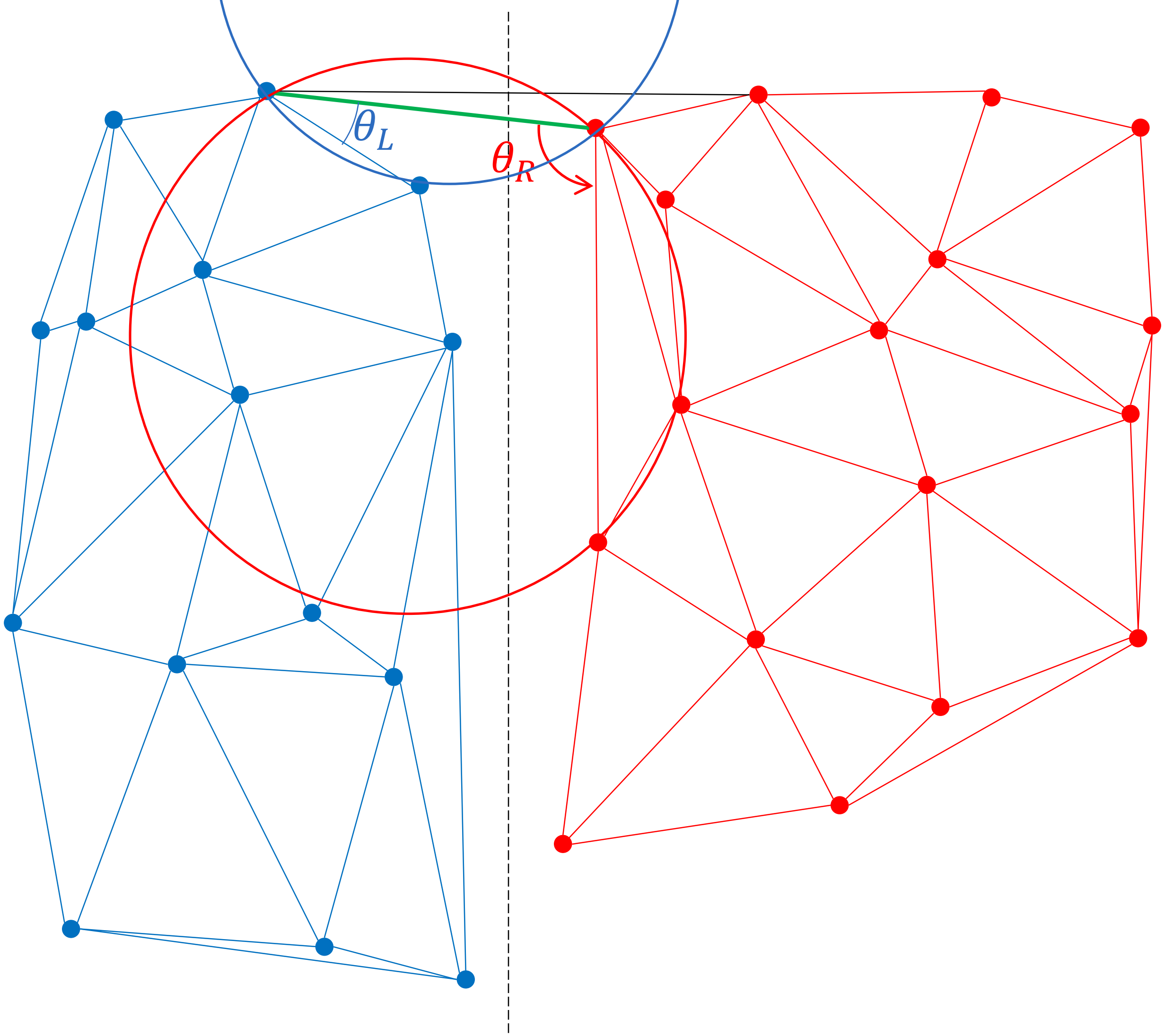
Rising Bubble Implementation



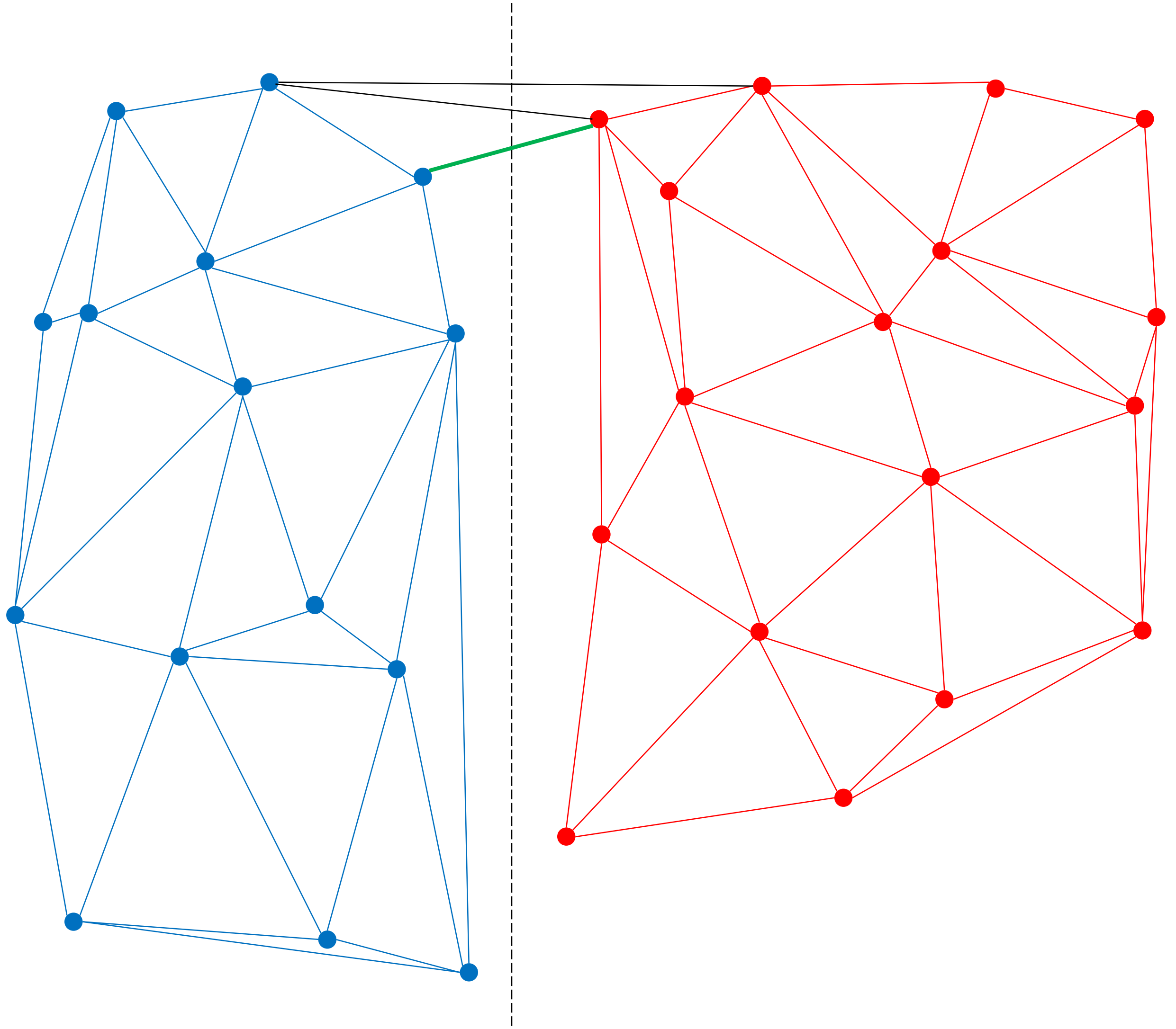
Rising Bubble Implementation

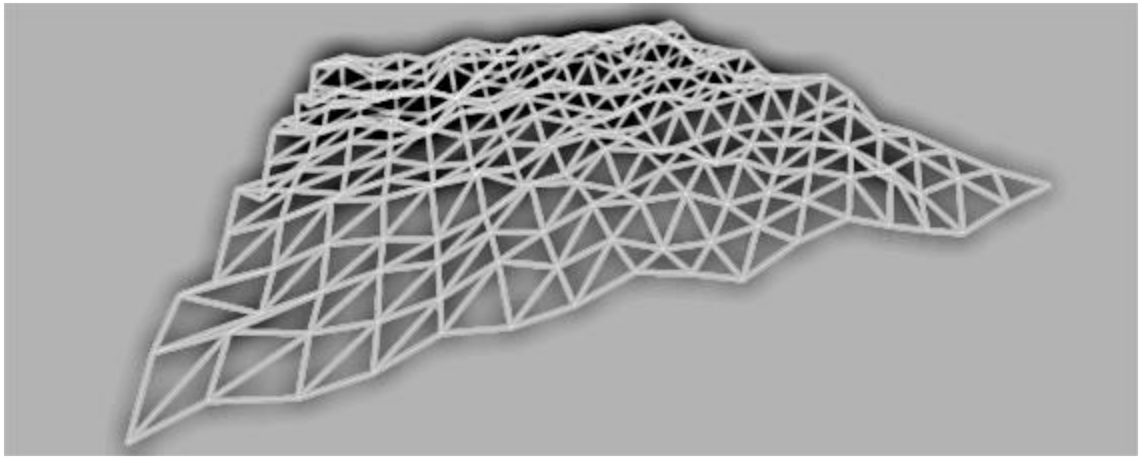


Rising Bubble Implementation



Rising Bubble Implementation





Terrain Problem

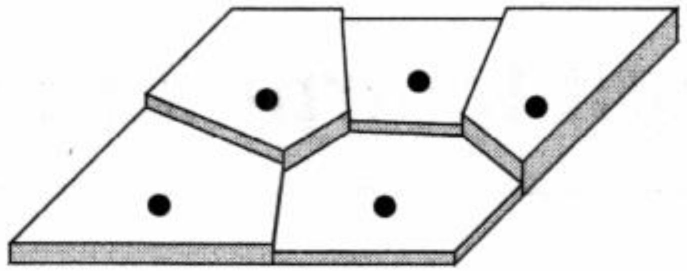
Terrain Problem

- We would like to build a model for the Earth terrain
- We can measure the altitude at some points
- How to approximate the altitude for non-measured points?

Nearest Neighbor

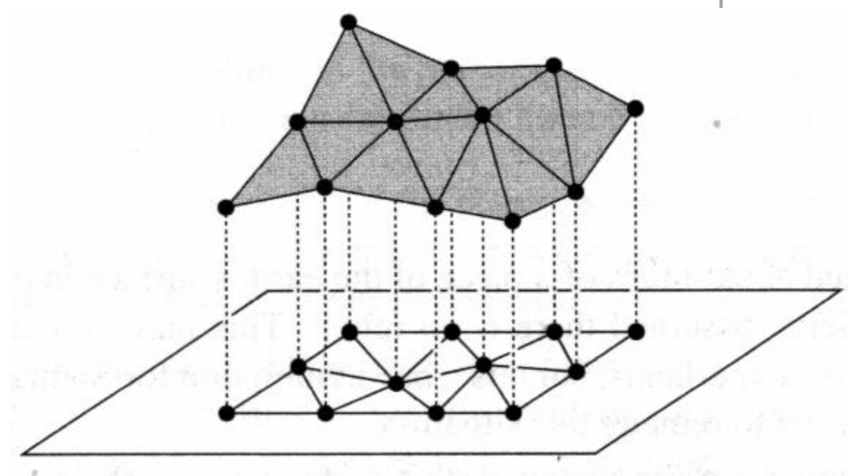
- One possibility, approximate it to the nearest measured point

- Does not look natural

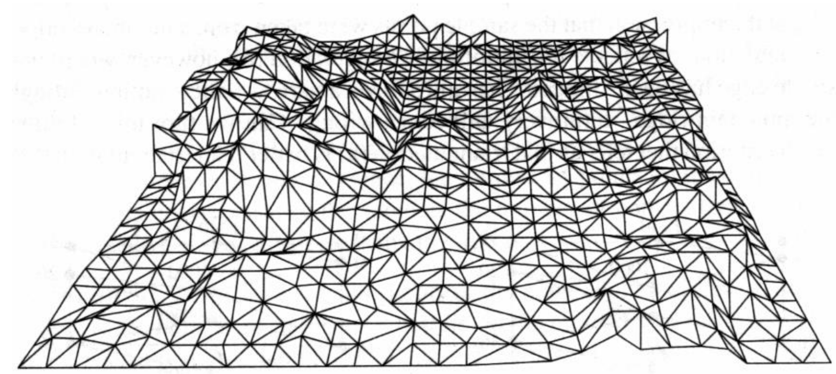


Triangulation

- Determine a triangulation
- Raise each point to its altitude

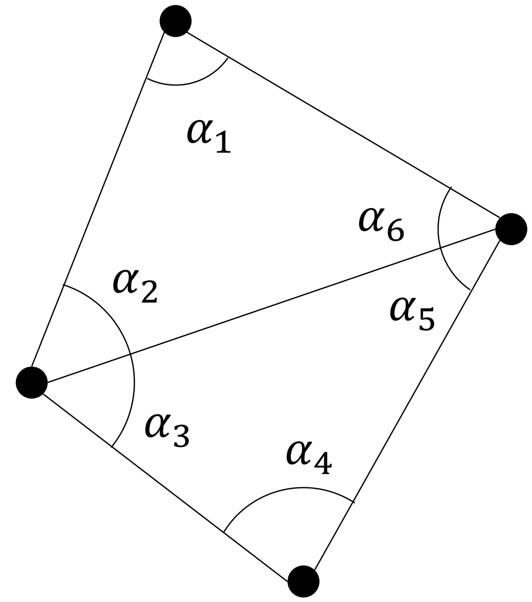


- Question: Which triangulation?

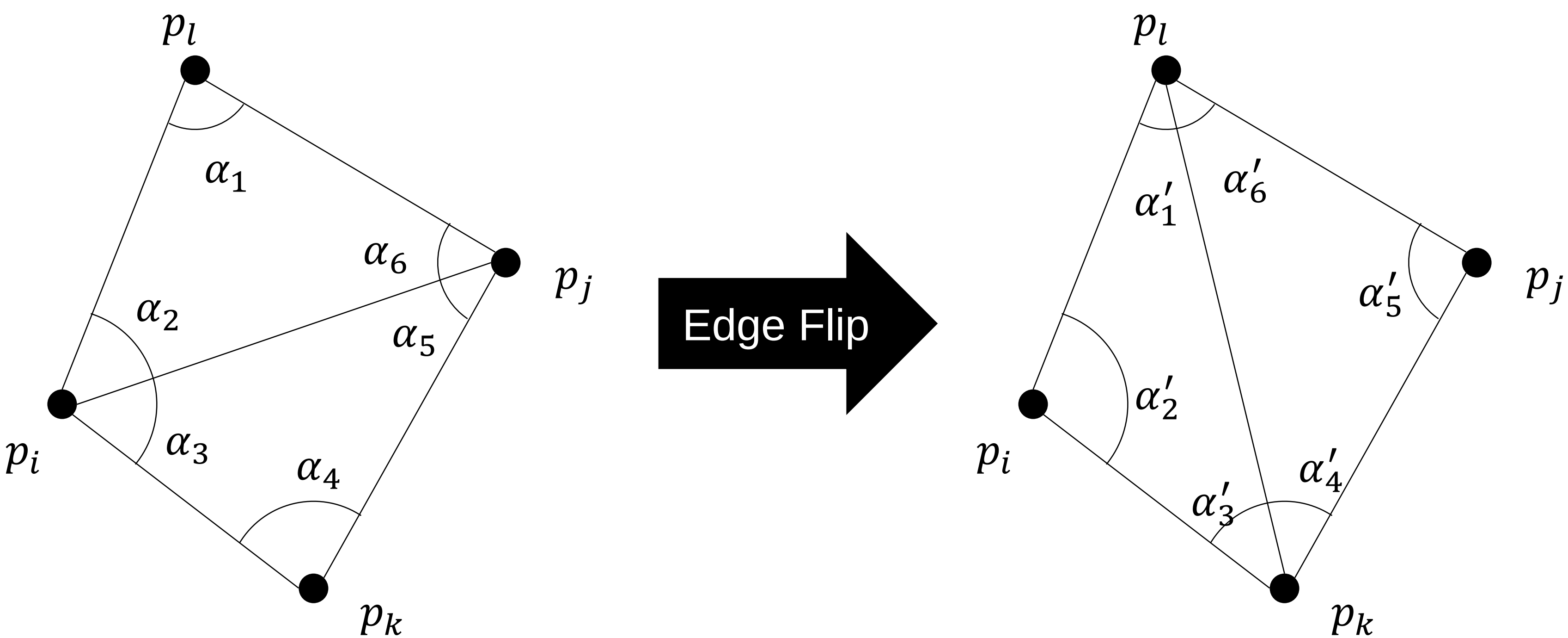


Angle-optimal Triangulation

- For a triangulation $\mathcal{T}$
- $A(\mathcal{T})$: is the angle vector which consists of the angles α's in sorted order
$$\alpha_1 \le \alpha_2 \le \cdots \le \alpha_n$$
- We say that $A(\mathcal{T}) > A(\mathcal{T}')$ if $A(\mathcal{T})$ is lexicographically larger than $A(\mathcal{T}')$
- $\mathcal{T}$ is angle optimal if $A(\mathcal{T}) \ge A(\mathcal{T}')$ for all triangulations $\mathcal{T}'$



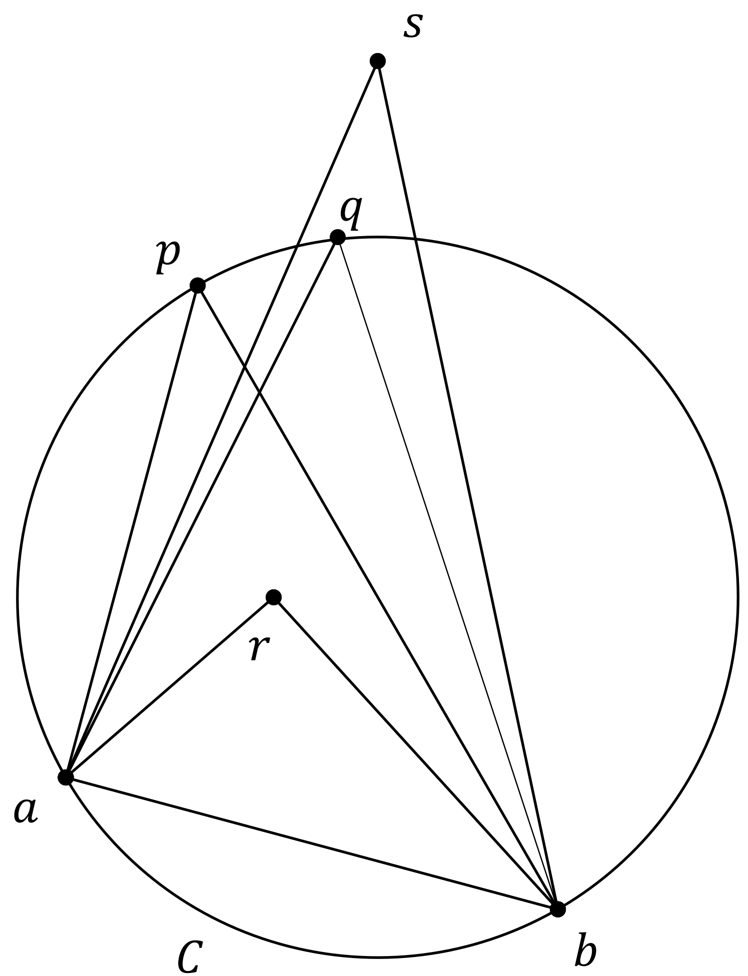
Edge Flip



- The edge $\overline{p_i p_j}$ is illegal if $\min_{1 \le i \le 6} \alpha_i < \min_{1 \le i \le 6} \alpha'_i$
- Flipping an edge increases the angle vector

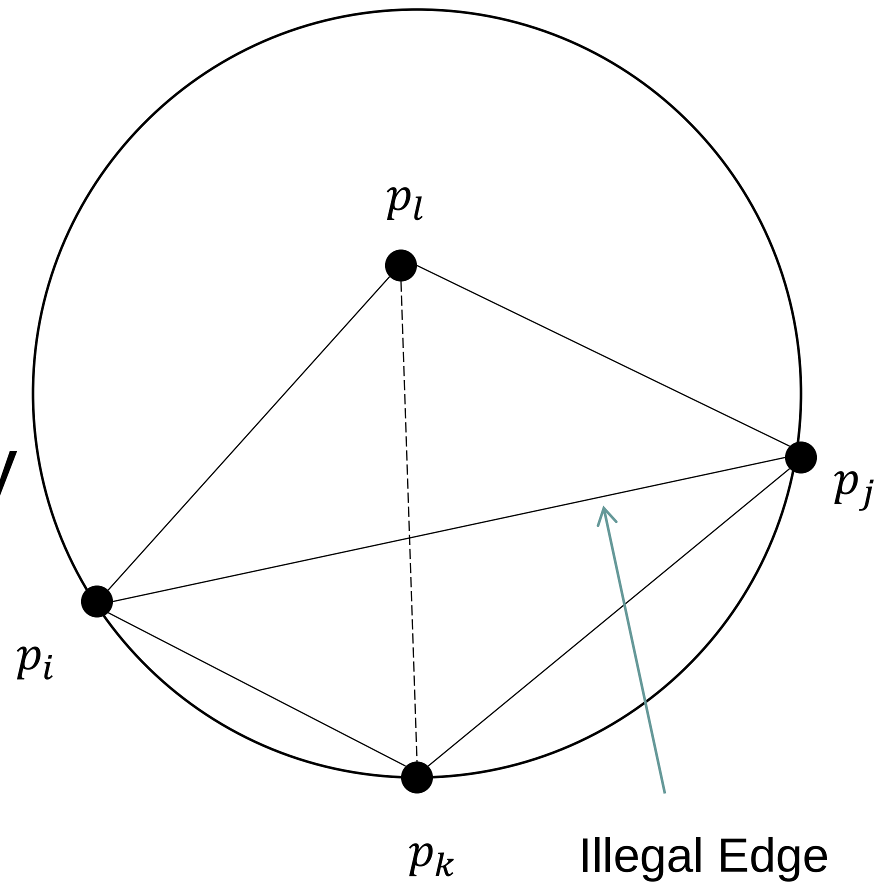
Detect Illegal Edges

- Thale's Theorem
- $\overline{ab}$ is a chord in C
- $\measuredangle arb > \measuredangle apb$
- $\measuredangle apb = \measuredangle aqb$
- $\measuredangle aqb > \measuredangle asb$



Detect Illegal Edges

- By Thale’s Theorem
- $\measuredangle p_i p_j p_k < \measuredangle p_i p_l p_k$
- $\measuredangle p_j p_i p_k < \measuredangle p_j p_l p_k$
- An angle-optimal triangulation is equivalent to Delauany Triangulation



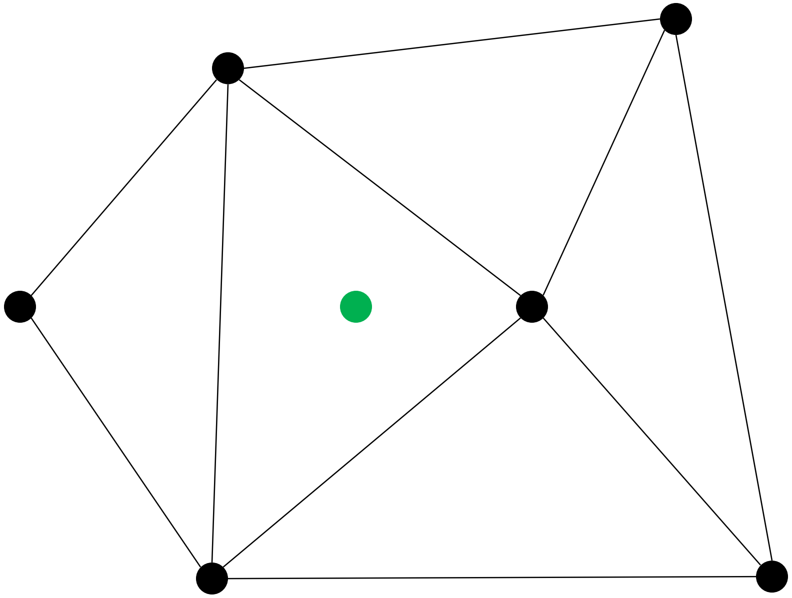
Delaunay Triangulation

1. Start with any valid triangulation
2. If no illegal edges found, terminate
3. Pick an illegal edge and flip it
4. Go to 2

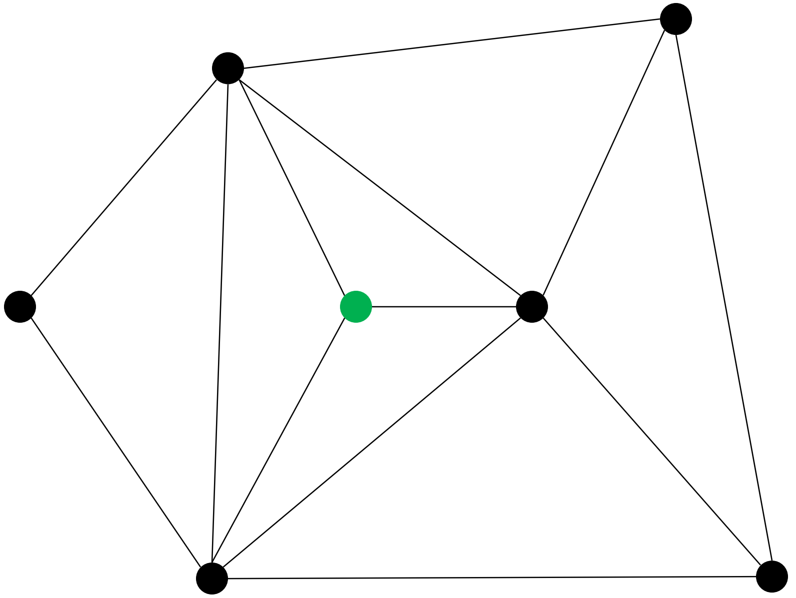
- Does this algorithm terminate?
- Running time: $O(n^2)$

Incremental Algorithm

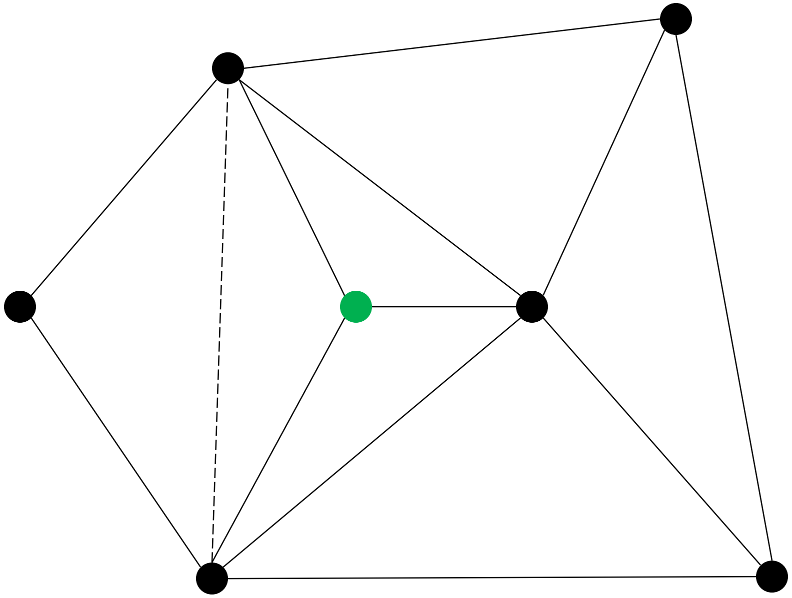
- Given an existing Delaunay triangulation $DT(P)$
- We need to add a point p_i to DT



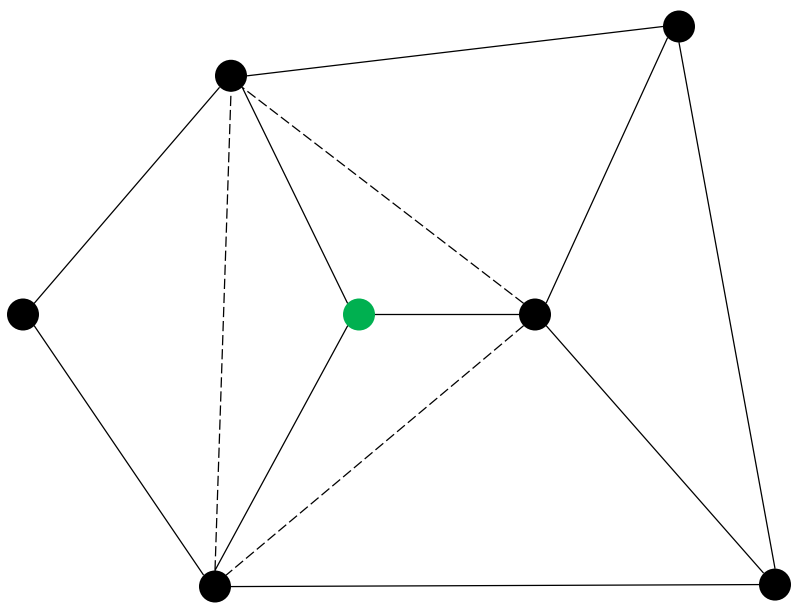
Incremental Algorithm



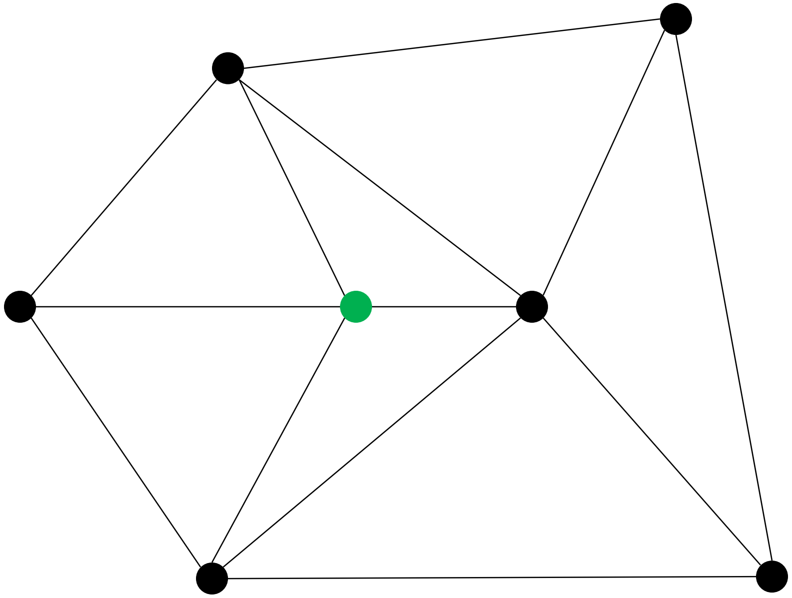
Incremental Algorithm



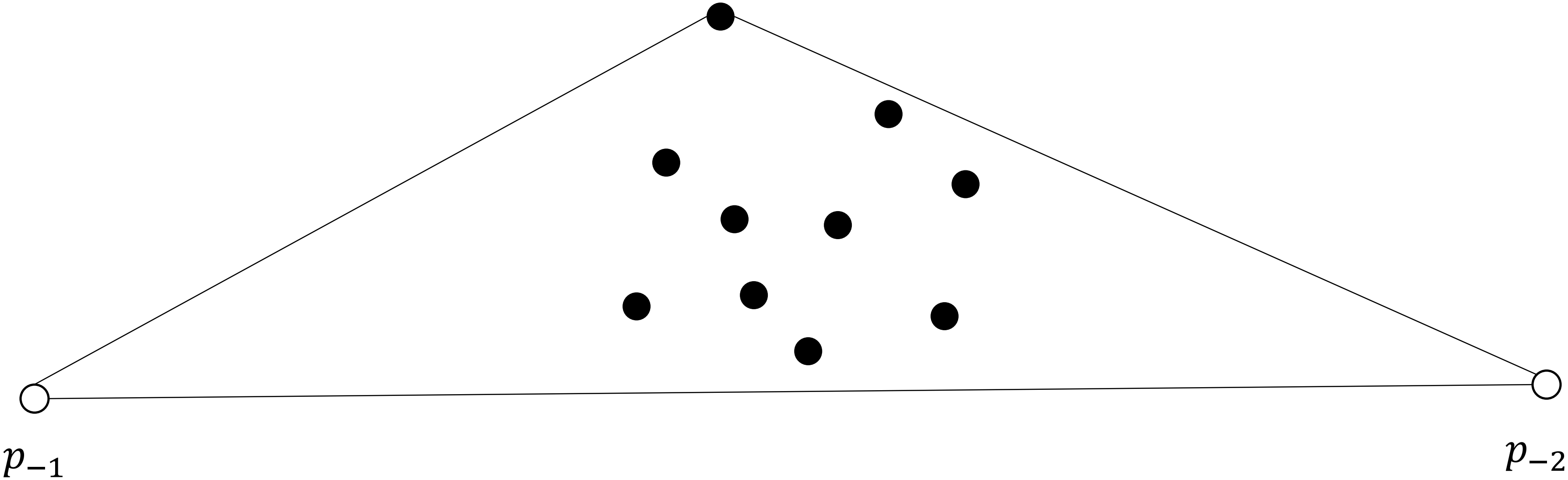
Incremental Algorithm



Incremental Algorithm



Incremental Algorithm



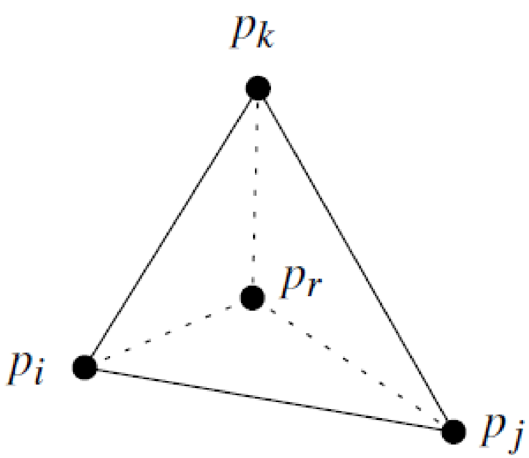
Incremental Algorithm

Algorithm $\text{DELAUNAYTRIANGULATION}(P)$
Input. A set P of $n+1$ points in the plane.
Output. A Delaunay triangulation of P.

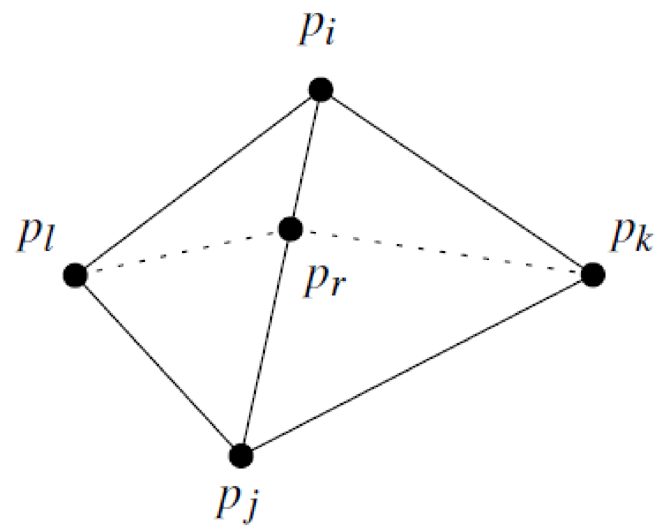
1. Initialize $\mathcal{T}$ as the triangulation consisting of an outer triangle $p_0p_{-1}p_{-2}$ containing points of P, where p_0 is the lexicographically highest point of P.
2. Compute a random permutation $p_1, p_2, \ldots, p_n$ of $P \setminus \{p_0\}$.
3. **for** $r \leftarrow 1$ **to** n
4.   **do**
5. $\text{LOCATE}(p_r, \mathcal{T})$
6. $\text{INSERT}(p_r, \mathcal{T})$
7. Discard p_{-1} and p_{-2} with all their incident edges from $\mathcal{T}$.
8. **return** $\mathcal{T}$

Incremental Algorithm

p_r lies in the interior of a triangle



p_r falls on an edge



Insert

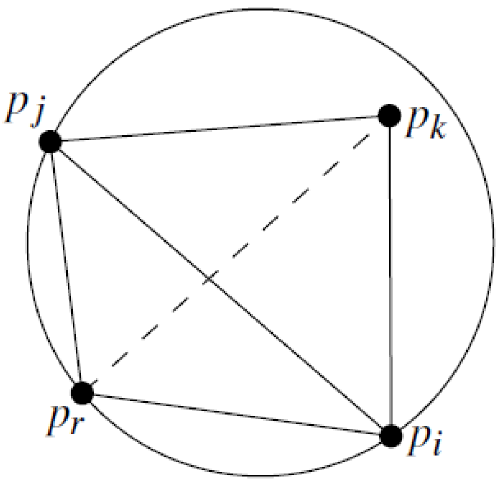
INSERT($p_r, \mathcal{T}$)

if p_r lies in the interior of the triangle $p_i p_j p_k$
    then Add edges from p_r to the three vertices of $p_i p_j p_k$, thereby splitting $p_i p_j p_k$ into three triangles.
        LEGALIZEEDGE($p_r, \overline{p_i p_j}, \mathcal{T}$)
        LEGALIZEEDGE($p_r, \overline{p_j p_k}, \mathcal{T}$)
        LEGALIZEEDGE($p_r, \overline{p_k p_i}, \mathcal{T}$)
    else (* p_r lies on an edge of $p_i p_j p_k$, say the edge $\overline{p_i p_j}$ *)
        Add edges from p_r to p_k and to the third vertex p_l of the other triangle that is incident to $\overline{p_i p_j}$, thereby splitting the two triangles incident to $\overline{p_i p_j}$ into four triangles.
        LEGALIZEEDGE($p_r, \overline{p_i p_l}, \mathcal{T}$)
        LEGALIZEEDGE($p_r, \overline{p_l p_j}, \mathcal{T}$)
        LEGALIZEEDGE($p_r, \overline{p_j p_k}, \mathcal{T}$)
        LEGALIZEEDGE($p_r, \overline{p_k p_i}, \mathcal{T}$)

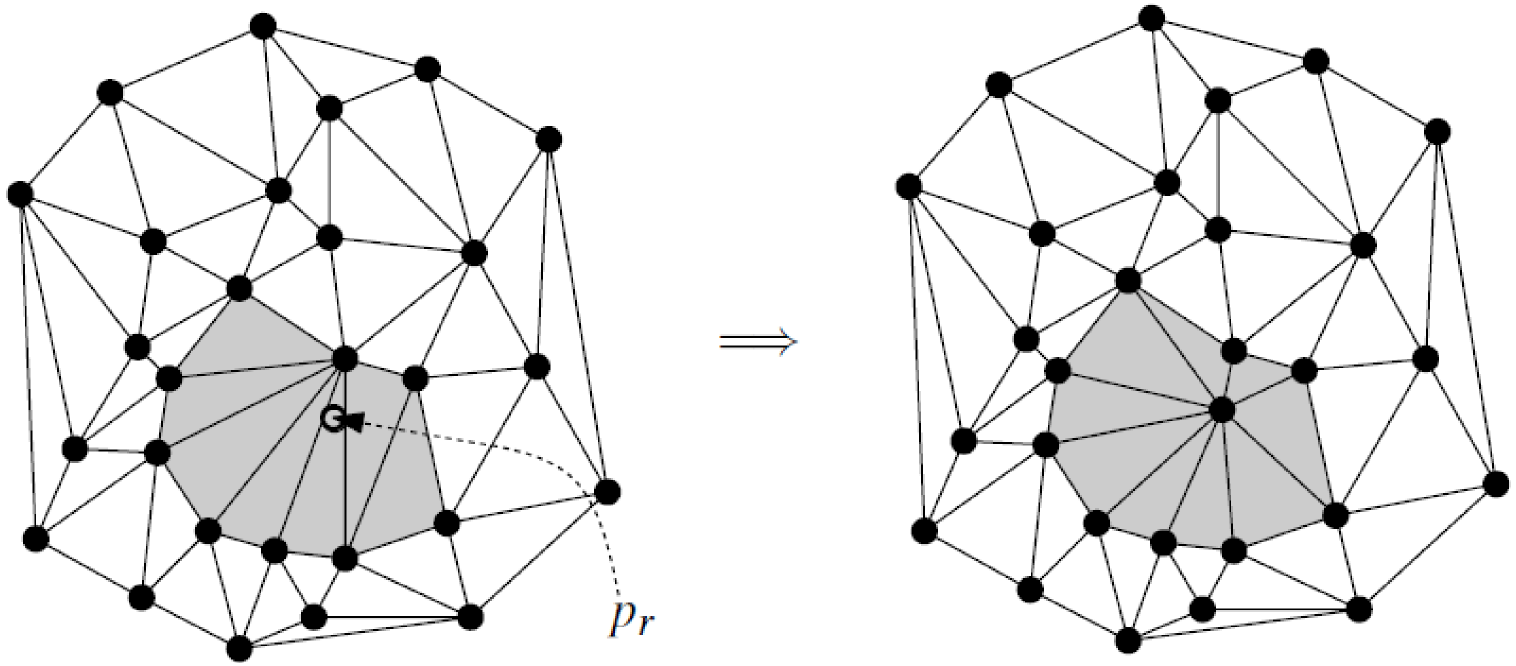
Legalize Edge

$\text{LegalizeEdge}(p_r, \overline{p_i p_j}, \mathcal{T})$

1. (∗ The point being inserted is p_r, and $\overline{p_i p_j}$ is the edge of $\mathcal{T}$ that may need to be flipped. ∗)
2. **if** $\overline{p_i p_j}$ is illegal
3. **then** Let $p_i p_j p_k$ be the triangle adjacent to $p_r p_i p_j$ along $\overline{p_i p_j}$.
4. (∗ Flip $\overline{p_i p_j}$: ∗) Replace $\overline{p_i p_j}$ with $\overline{p_r p_k}$.
5. $\text{LegalizeEdge}(p_r, \overline{p_i p_k}, \mathcal{T})$
6. $\text{LegalizeEdge}(p_r, \overline{p_k p_j}, \mathcal{T})$



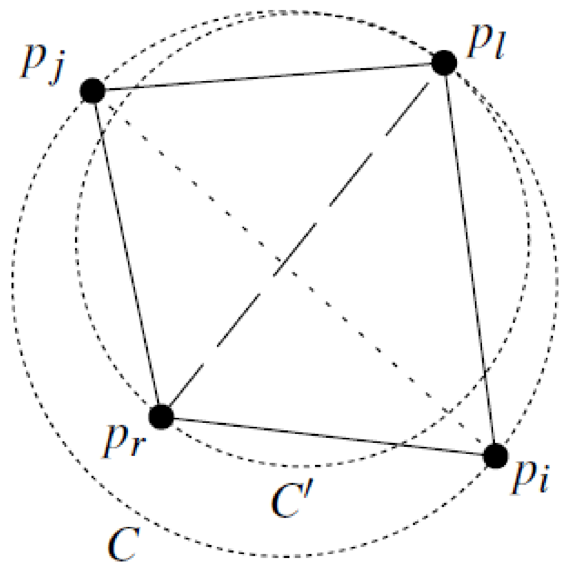
Correctness



All edges created are incident to p_r.

Correctness: Are new edges legal?

Correctness



Correctness:

For any new edge there is an empty circle through endpoints.

New edges are legal.

Incremental Algorithm

Initializing triangulation: treat p_{-1} and p_{-2} symbolically.
No actual coordinates.
Modify tests for point location and illegal edges to work as if far away.

Point location: search data structure.
Point visits triangles of previous triangulations that contain it.

Search Data Structure