2D Geometry

Pierre Alliez
Inria Sophia Antipolis

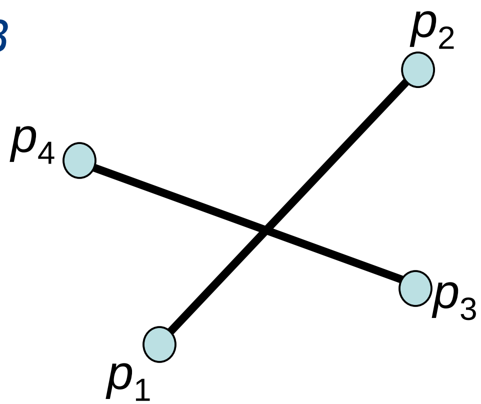
Outline

- Sample problems
- Polygons
- Graphs
- Convex hull
- Voronoi diagram
- Delaunay triangulation

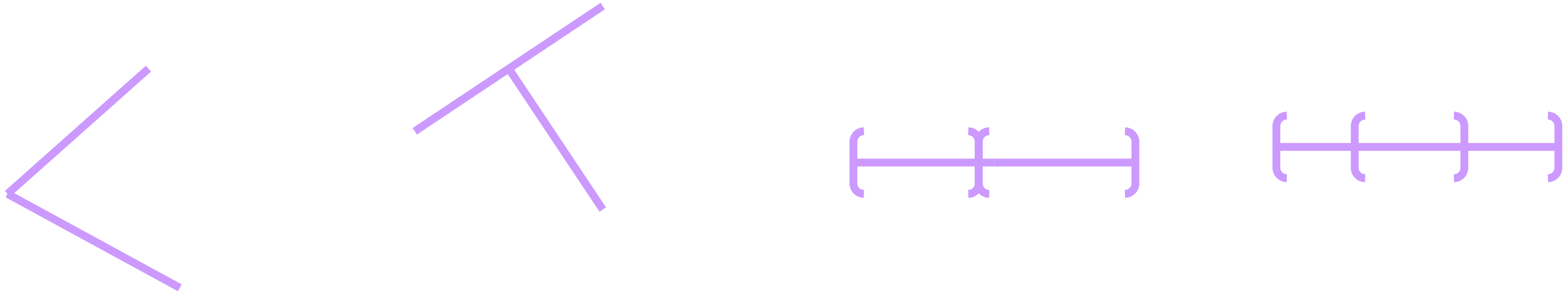
Sample Problems

Line Segment Intersection

- **Theorem:** Segments (p_1,p_2) and (p_3,p_4) intersect in their interior iff p_1 and p_2 are on different sides of the line p_3p_4 and p_3 and p_4 are on different sides of the line p_1p_2.

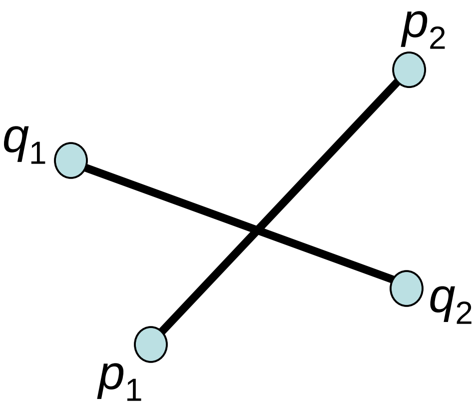


- This can be checked by computing the orientations of *four* triangles. Which ?
- Special cases:



Computing the Intersection

$$p(t) = p_1 + (p_2 - p_1)t \qquad 0 \le t \le 1$$
$$q(s) = q_1 + (q_2 - q_1)s \qquad 0 \le s \le 1$$



Question: What is the meaning of other values of *s* and *t* ?

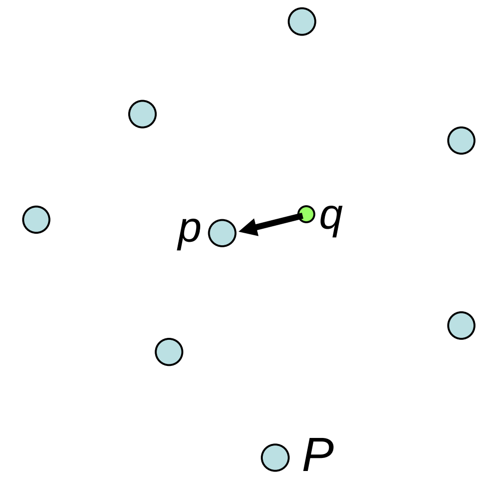
Solve (2D) linear vector equation for *t* and *s*:

$$p(t) = q(s)$$

check that $t \in [0,1]$ and $s \in [0,1]$

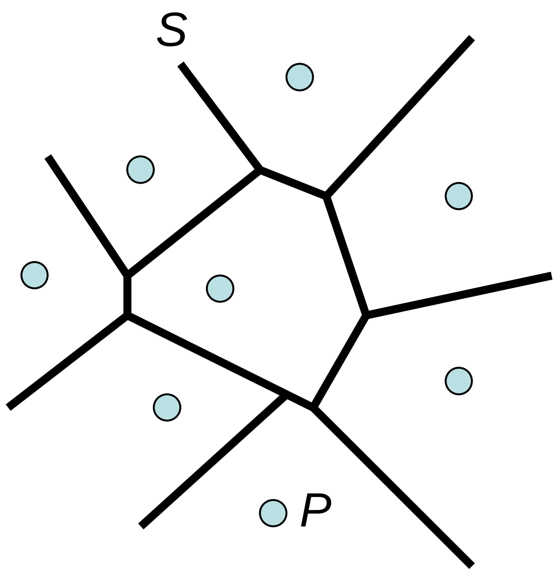
Nearest Neighbor

- Problem definition:
  - Input: a set of points (*sites*) P in the plane and a query point q.
  - Output: The point $p \in P$ closest to q among all points in P.
- Rules of the game:
  - One point set, multiple queries
  - Applications:
    - Store Locator
    - Cellphones



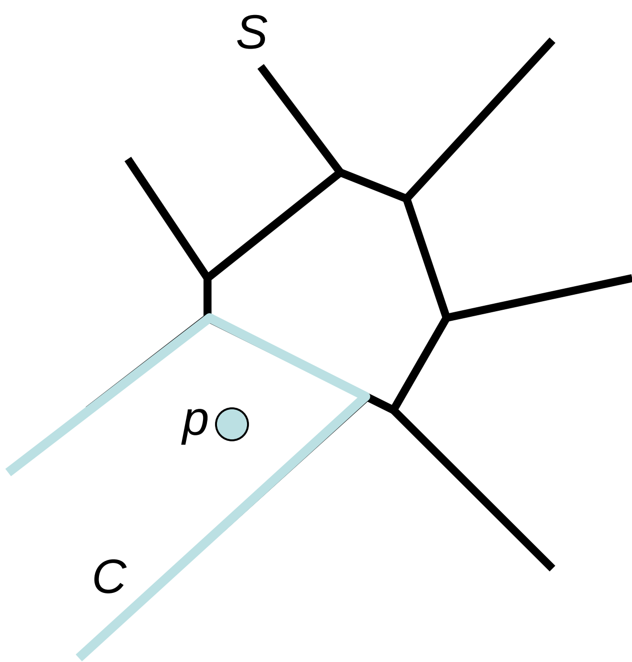
The Voronoi Diagram

- Problem definition:
  - Input: a set of points (*sites*) P in the plane.
  - Output: A planar subdivision S into cells per site. The cell corresponding to $p \in P$ contains all the points to which p is the closest.



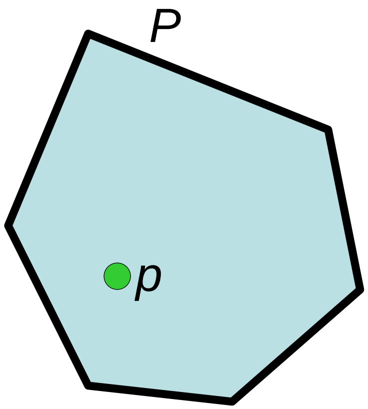
Point Location

- Problem definition:
  - Input: A partition S of the plane into cells and a query point p.
  - Output: The cell $C \in S$ containing p.
- Rules of the game:
  - One partition, multiple queries
- Applications:
  - Nearest neighbor
  - State locator



Point in Polygon

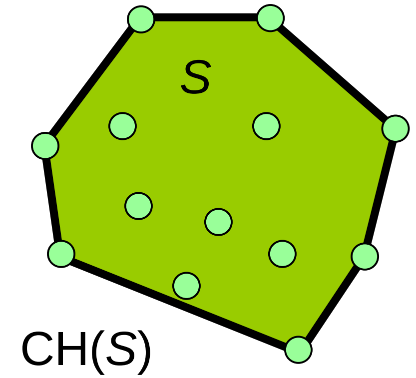
- Problem definition:
  - Input: a polygon *P* in the plane and a query point *p*.
  - Output: *true* if $p \in P$, else *false*.



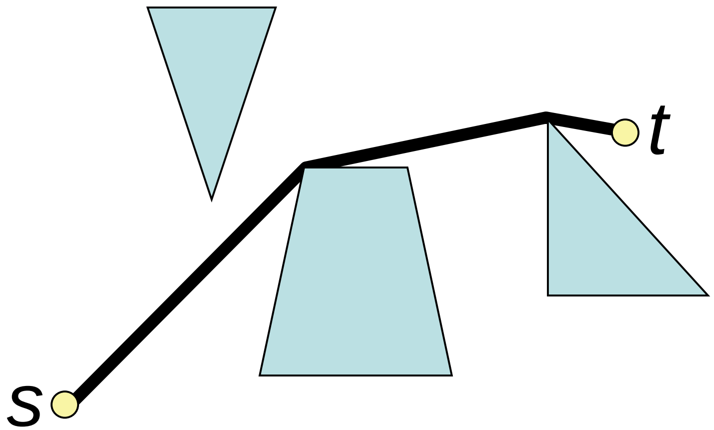
- Rules of the game:
  - One polygon, multiple queries

Convex Hull

- Problem definition:
  - Input: a set of points *S* in the plane.
  - Output: Minimal convex polygon containing *S*.



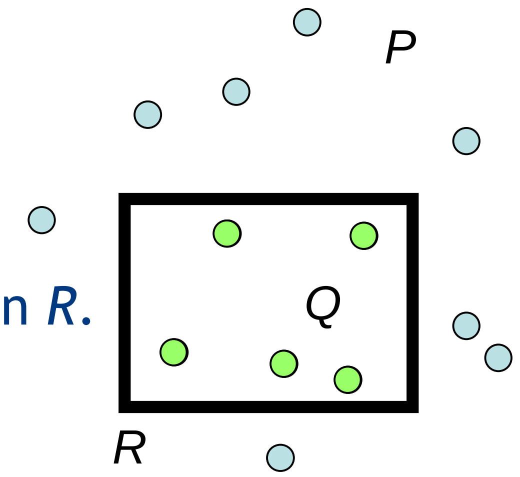
Shortest Path



- Problem definition:
  - Input: Obstacles locations and *query* endpoints *s* and *t*.
  - Output: the shortest path between *s* and *t* that avoids all obstacles.
- Rules of the game: One obstacle set, multiple queries.
- Application: Robotics.

Range Searching and Counting

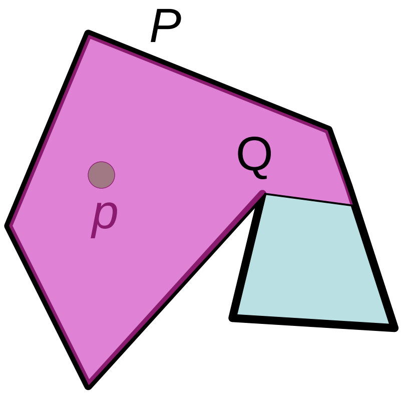
- Problem definition:
  - Input: A set of points *P* in the plane and a query rectangle *R*
  - Output: (report) The subset $Q \subseteq P$ contained in *R*.
    (count) The size of *Q*.



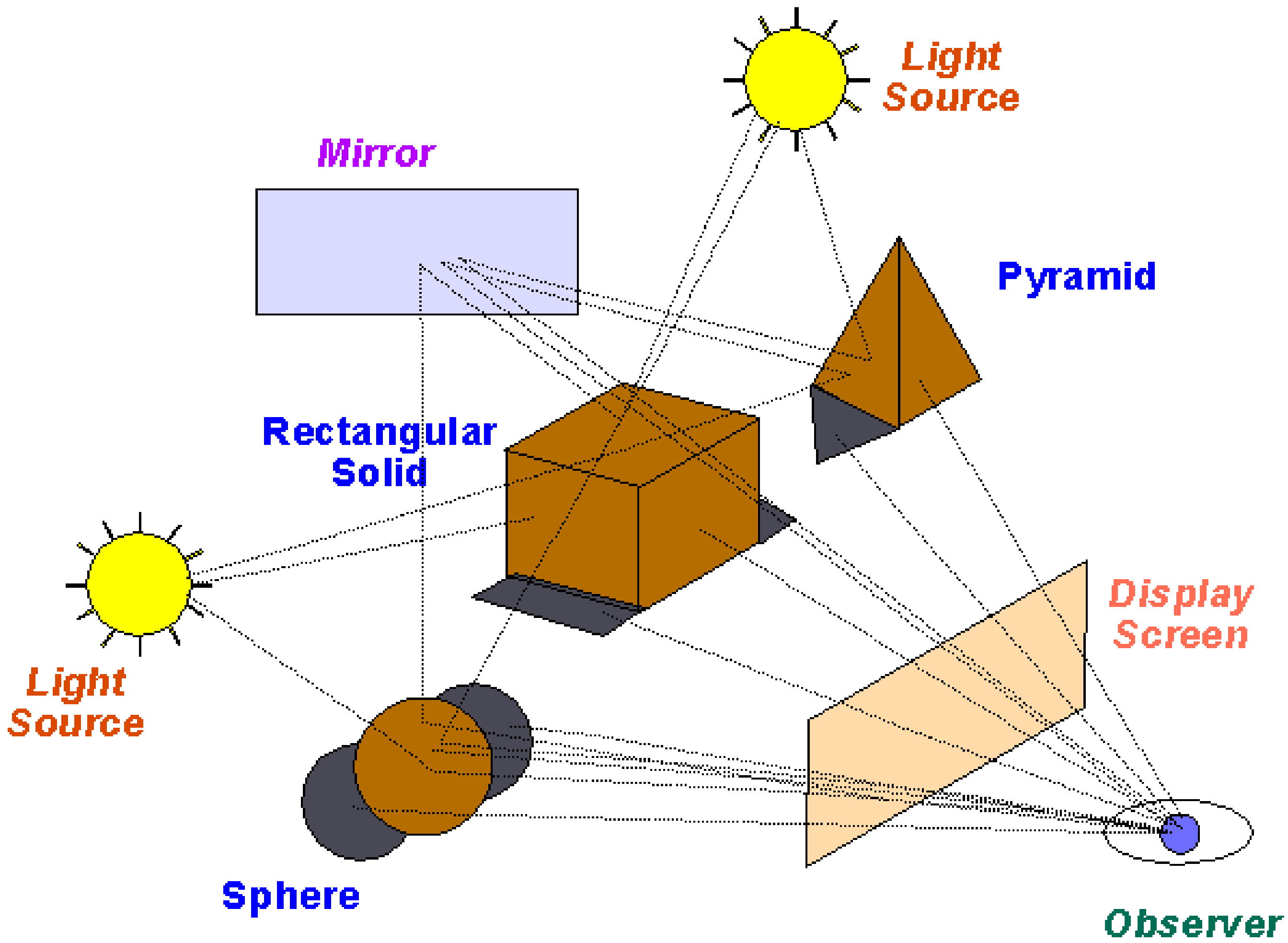
- Rules of the game:
  - One point set, multiple queries.
- Application: Urban planning.

Visibility

- Problem definition:
  - Input: a polygon *P* in the plane and a query point *p*.
  - Output: Polygon Q $\subseteq$ P, visible to p.
- Rules of the game:
  - One polygon, multiple queries
- Applications: Security

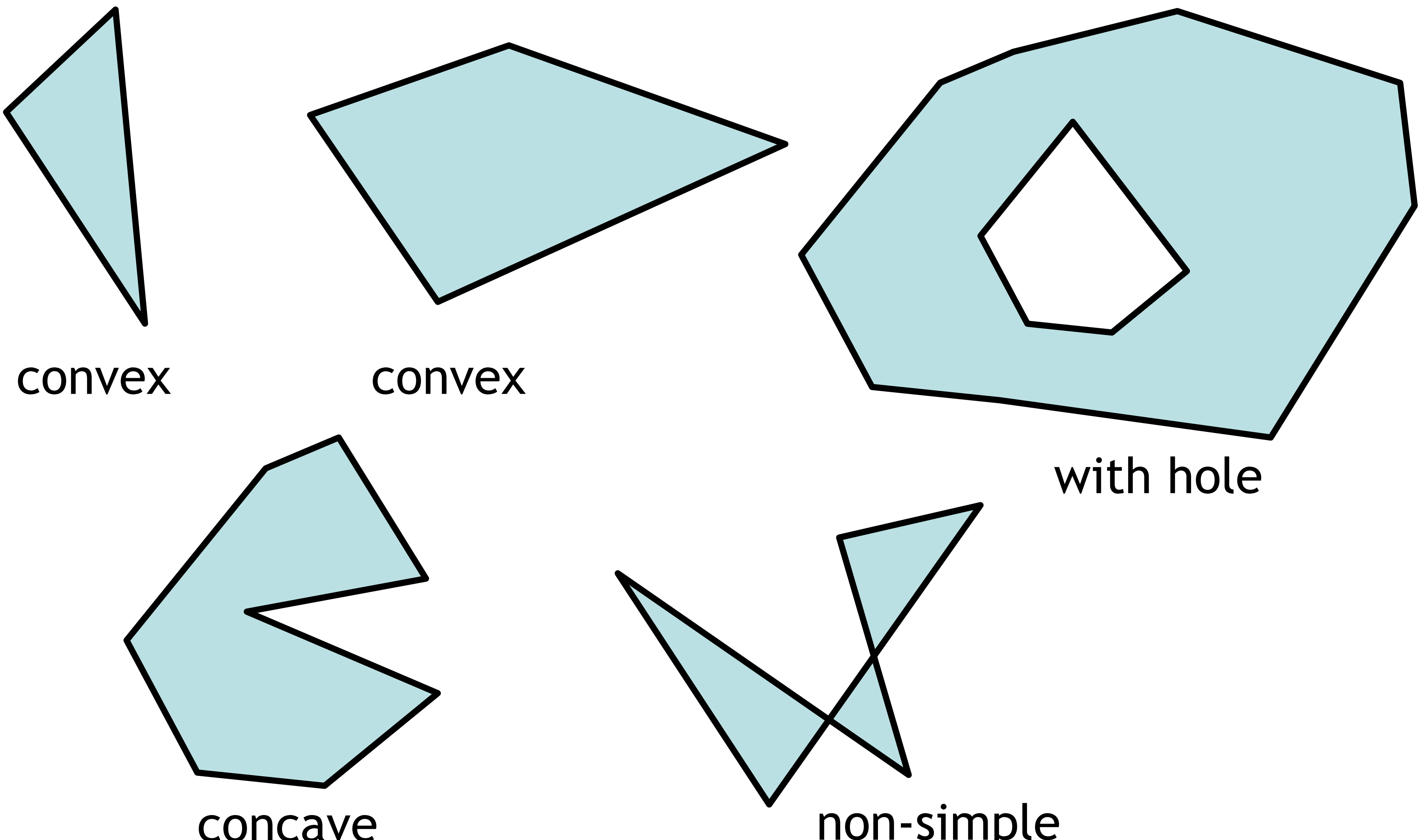


Ray Tracing


Light
Source
Mirror
Pyramid
Rectangular
Solid
Light
Source
Display
Screen
Sphere
Observer

Polygons

Polygon Zoo



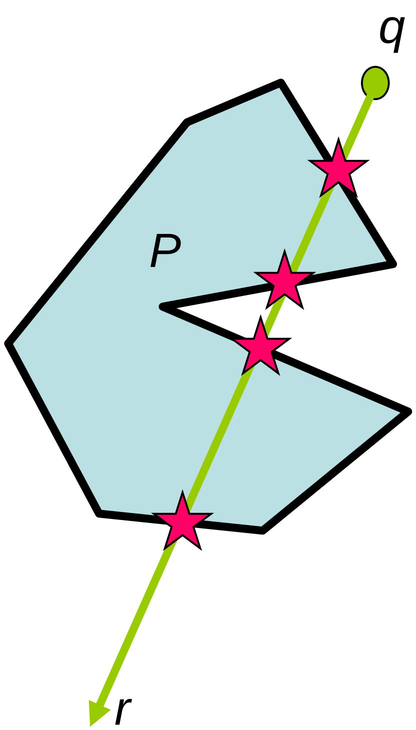
Point in Polygon

- Given a polygon P with n sides, and a point q, decide whether $q \in P$.

- Solution A: Count how many times a ray r originating at q intersects P. Then $q \in P$ iff this number is odd.

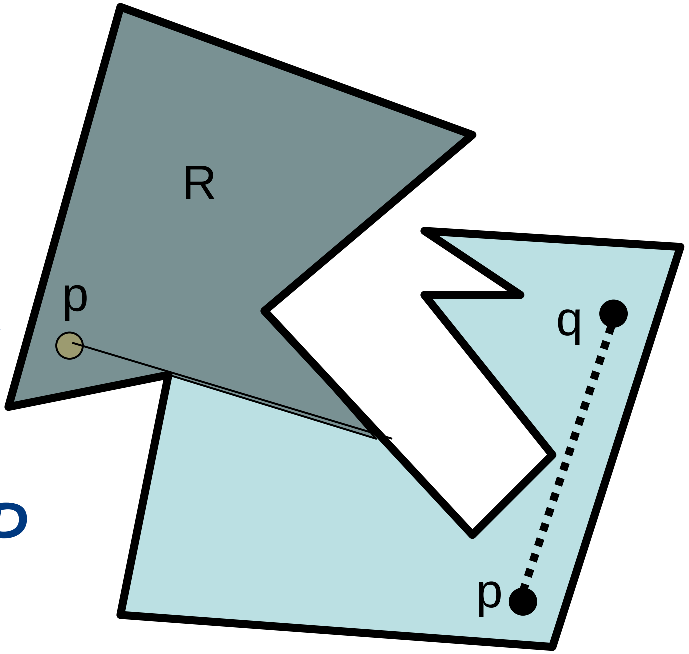
- Complexity: $O(n)$

- **Question:** Are there special cases ?



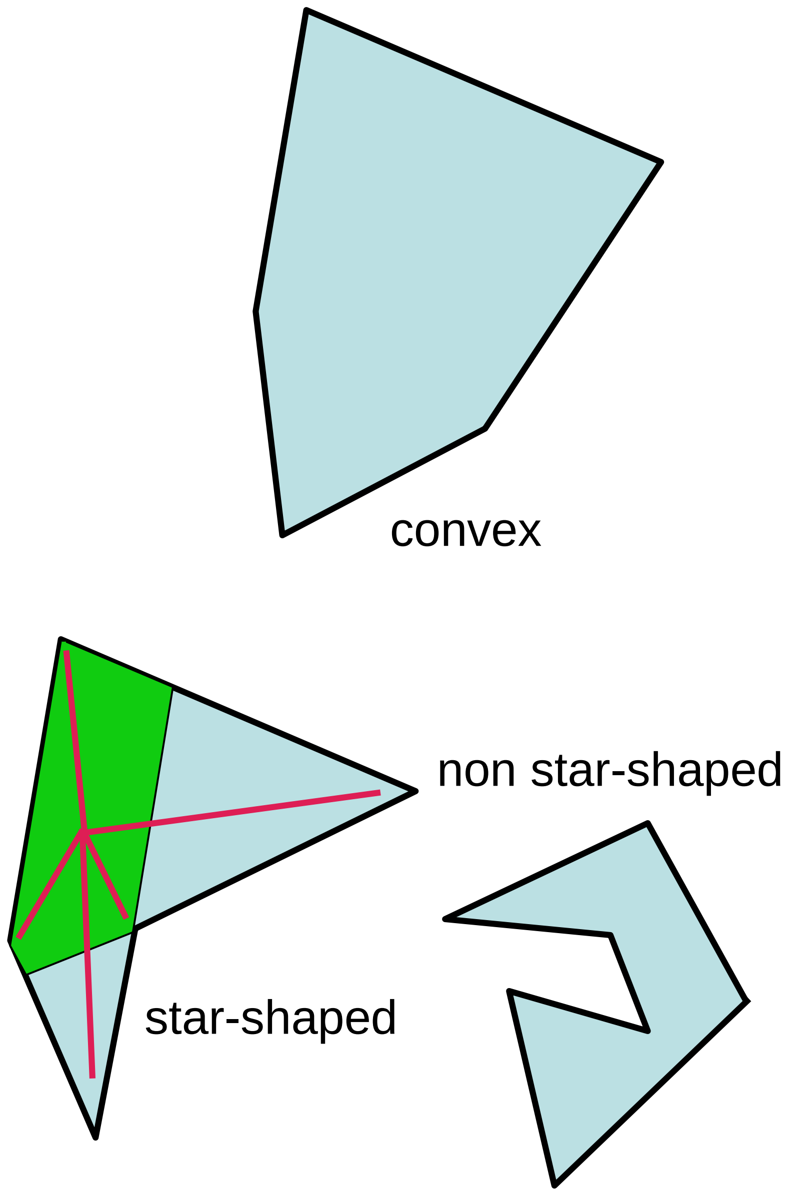
Art Gallery Problem

- Given a simple polygon P, say that two points p and q can *see* each other if the open segment pq lies entirely within P.
- A point *guards* a region $R \subseteq P$ if p sees all $q \in R$.
- Given a polygon P, what is the minimal number of guards required to guard P, and what are their locations ?



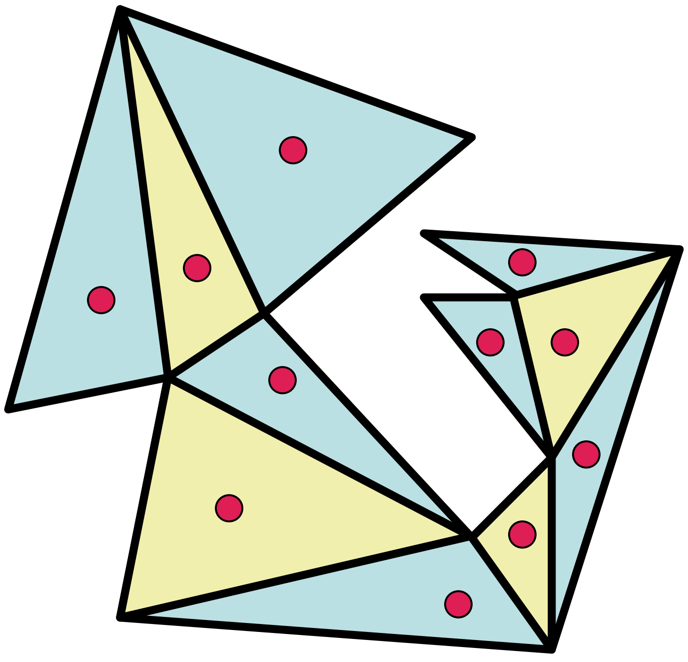
Observations

- The *entire* interior of a convex polygon is visible from *any* interior point.
- A *star-shaped* polygon requires only one guard located in its *kernel*.



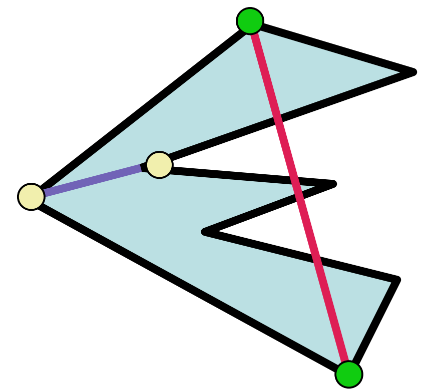
Art Gallery Problem – Easy Upper Bound

- *n*-2 guards suffice:
  - Subdivide the polygon into *n*-2 triangles (triangulation)
  - Place one guard in each triangle.
- **Theorem**: Any simple planar polygon with *n* vertices has a triangulation of size *n*-2.



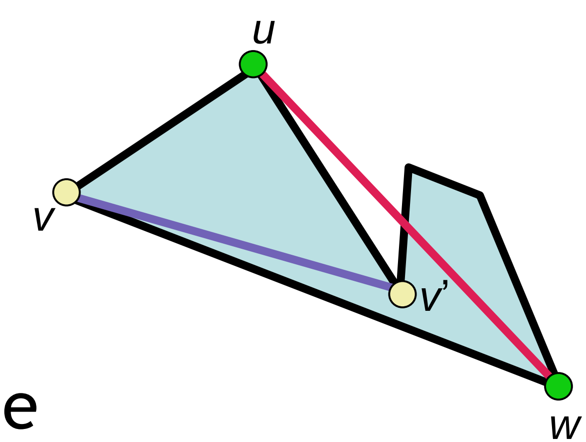
Diagonals in Polygons

- A *diagonal* of a polygon *P* is a line segment connecting two vertices which lies entirely within *P*.



- **Theorem:** Any polygon with n>3 vertices has a diagonal, which may be found in O(*n*) time.

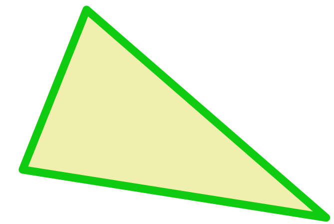
- **Proof:** Find the leftmost vertex *v*. Connect its two neighbors *u* and *w*. If this is not a diagonal there are other vertices inside the triangle *uvw*. Connect *v* with the vertex *v*’ furthest from the segment *uw*.



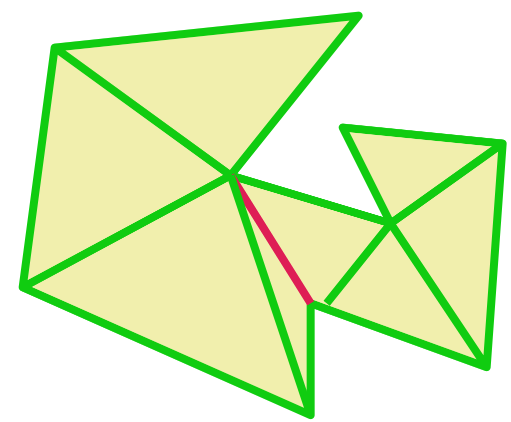
- **Question:** Why not connect *v* with the second leftmost vertex?

O(n^2) Polygon Triangulation

- **Theorem:** Every simple polygon with n vertices has a triangulation consisting of n-3 diagonals and n-2 triangles.

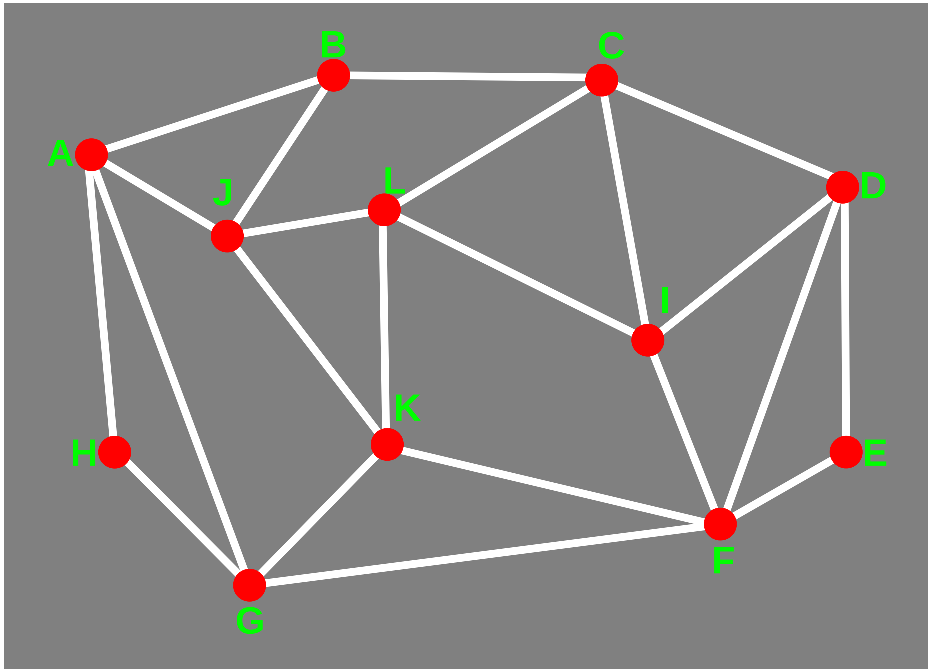


- **Proof:** By induction on n:
  - Basis: A triangle (n=3) has a triangulation (itself) with no diagonals and one triangle.
  - Induction: for a n+1 vertex polygon, construct a diagonal dividing the polygon into two polygons with n_1 and n_2 vertices such that $n_1+n_2-2=n$. Triangulate the two parts of the polygon. There are now $n_1-3+n_2-3+1=n-3$ diagonals and $n_1-2+n_2-2=n-2$ triangles.



Graphs

Graph Definitions

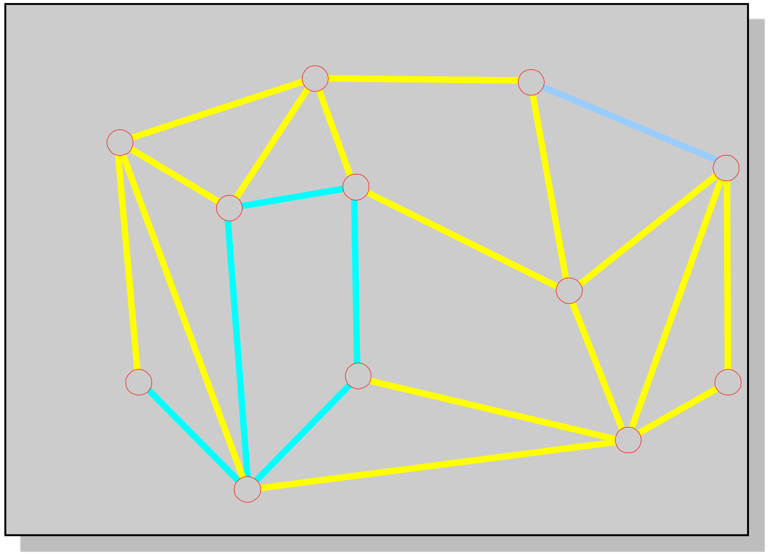


G = <**V**,**E**>
V = vertices =
{A,B,C,D,E,F,G,H,I,J,K,L}
E = edges =
{(A,B),(B,C),(C,D),(D,E),(E,F),(F,G),
(G,H),(H,A),(A,J),(A,G),(B,J),(K,F),
(C,L),(C,I),(D,I),(D,F),(F,I),(G,K),
(J,L),(J,K),(K,L),(L,I)}

Vertex *degree* (valence) = number of edges incident on vertex.
deg(J) = 4, deg(H) = 2
k-regular graph = graph whose vertices *all* have degree *k*

A *face* of a graph is a cycle of vertices/edges which cannot be shortened.
F = faces =
{(A,H,G),(A,J,K,G),(B,A,J),(B,C,L,J),(C,I,J),(C,D,I),
(D,E,F),(D,I,F),(L,I,F,K),(L,J,K),(K,F,G),(A,B,C,D,E,F,G,H)}

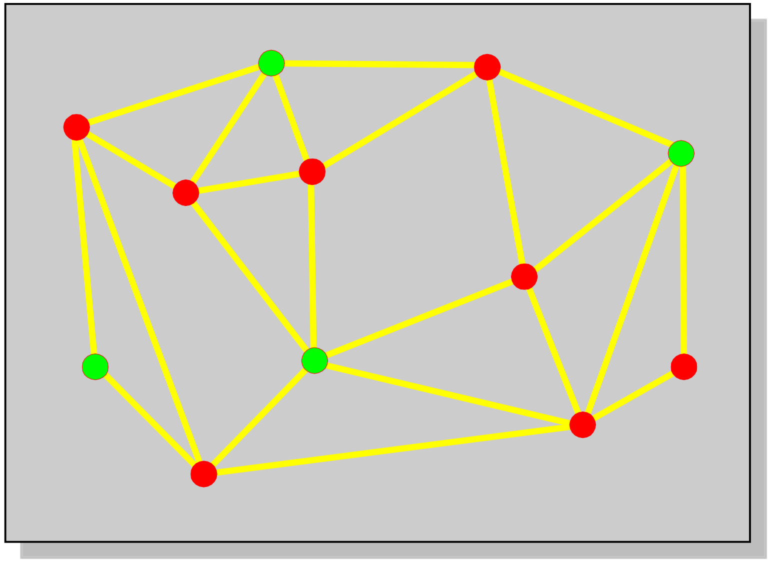
Connectivity



A graph is *connected* if there is a path of edges connecting every two vertices.

A graph is *k-connected* if between every two vertices there are *k* edge-disjoint paths.

A graph **G'**=<**V'**,**E'**> is a *subgraph* of a graph **G**=<**V**,**E**> if **V**' is a subset of **V** and **E'** is the subset of **E** incident on **V'**.

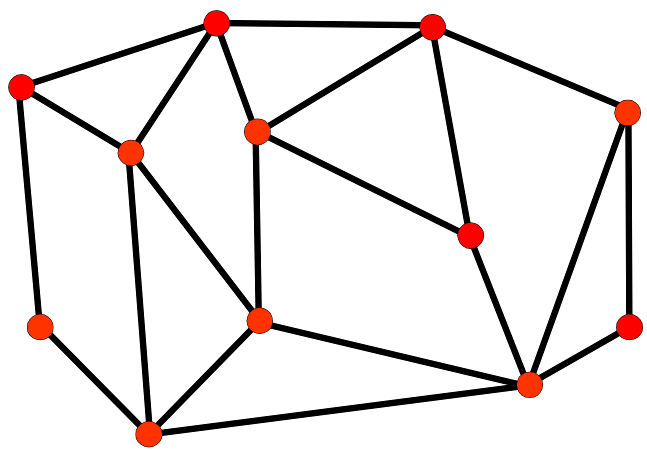


A *connected component* of a graph is a maximal connected subgraph.

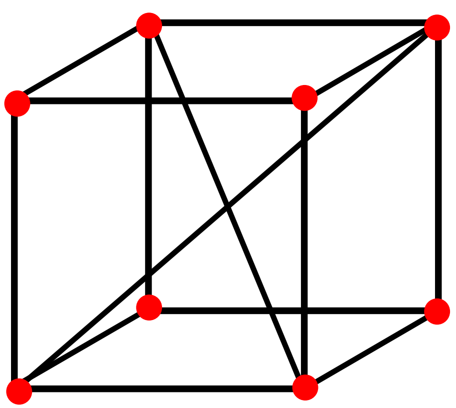
A subset **V'** of **V** is an *independent* set in **G** if the subgraph it induces does not contain any edges of **E**.

Graph Embedding

A graph is *embedded* in R^d if each vertex is assigned a position in R^d.



Embedding in R^2



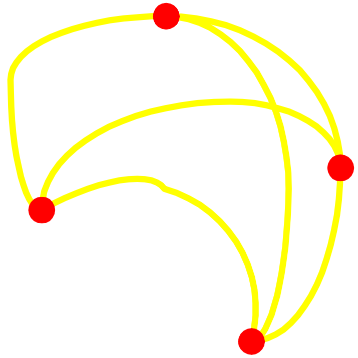
Embedding in R^3

Planar Graphs

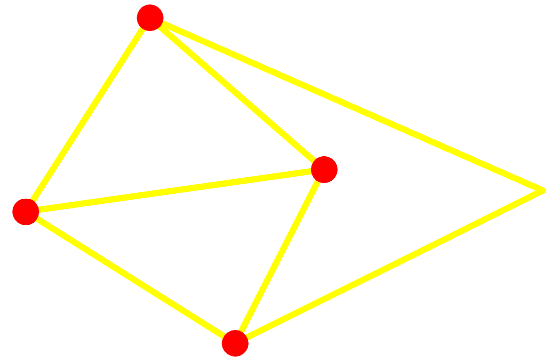
A planar graph is a graph whose vertices and edges can be embedded in R2 such that its edges do not intersect.

Every planar graph can be drawn as a straight-line plane graph.

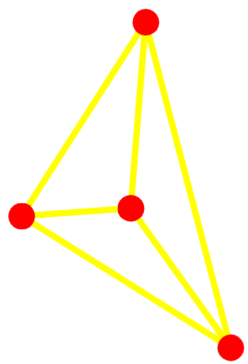
Planar Graph



Planar Graph

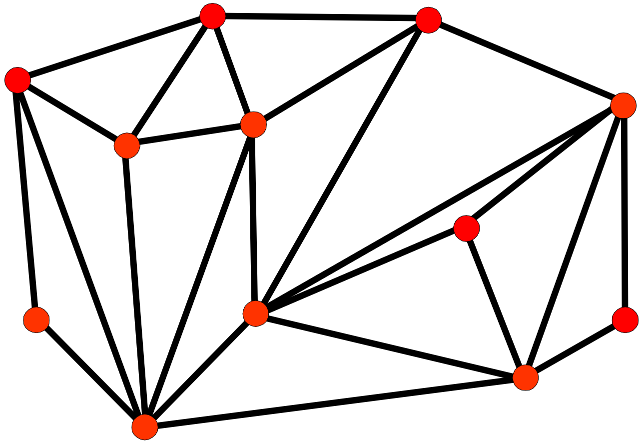


Straight Line Plane Graph



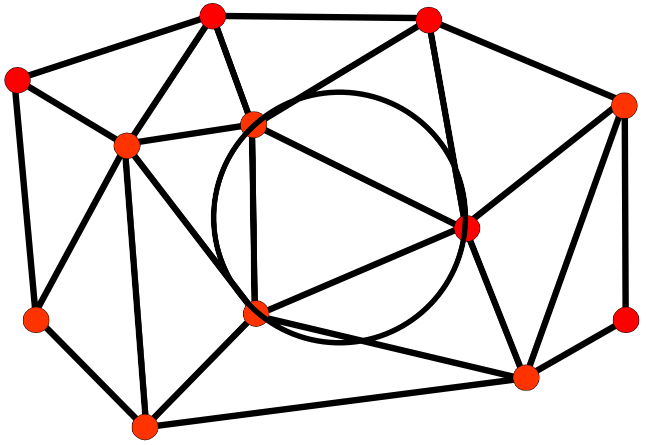
Triangulation

A *triangulation* is a straight line plane graph whose faces are all triangles.

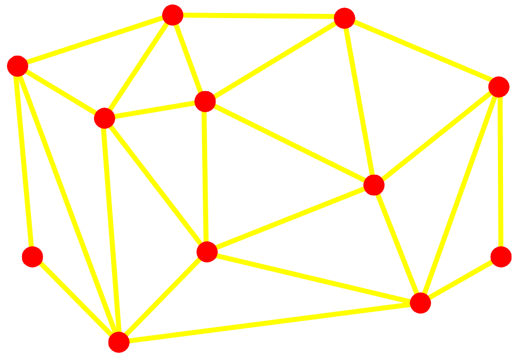


A *Delaunay triangulation* of a set of points is the unique set of triangles such that such that the circumcircle of any triangle does not contain any other point.

The Delaunay triangulation avoids long and skinny triangles.



Topology



v =12
f = 14
e = 25
c = 1
b = 1

Euler Formula

For a planar graph:

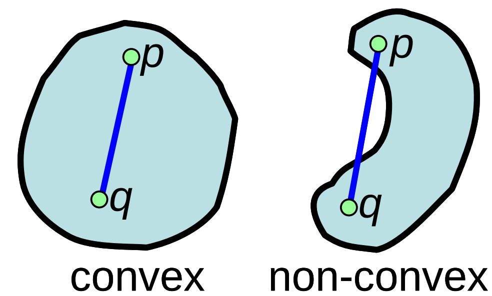
$v+f-e = 2c-b$

v = # vertices   c = # conn. comp.
f = # faces
e = # edges   b = # boundaries

Convex Hull

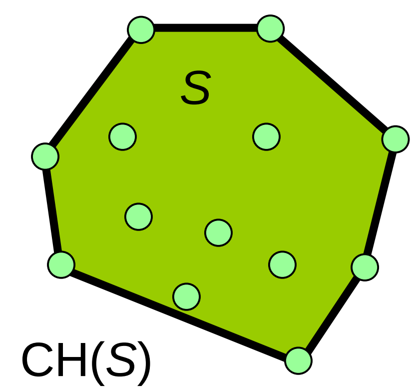
Convexity and Convex Hull

- A set S is *convex* if any pair of points $p,q \in S$ satisfy $pq \subseteq S$.



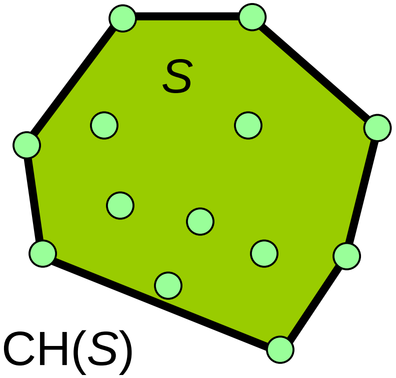
- The *convex hull* of a set S is:
  - The minimal convex set that contains S, i.e. any convex set C such that $S \subseteq C$ satisfies $CH(S) \subseteq C$.
  - The intersection of all convex sets that contain S.
  - The set of all convex combinations of $p_i \in S$, i.e. all points of the form:

$$\sum_{i=1}^{n} \alpha_i p_i \,, \qquad \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i = 1$$



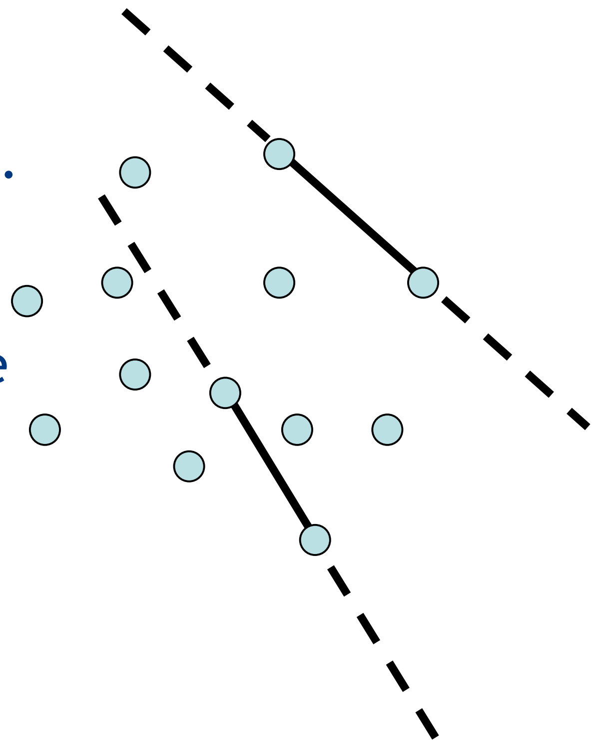
Convex Hulls – Some Facts

- The convex hull of a set is unique (up to colinearities).
- The boundary of the convex hull of a point set is a polygon on a subset of the points.



Convex Hull – Naive Algorithm

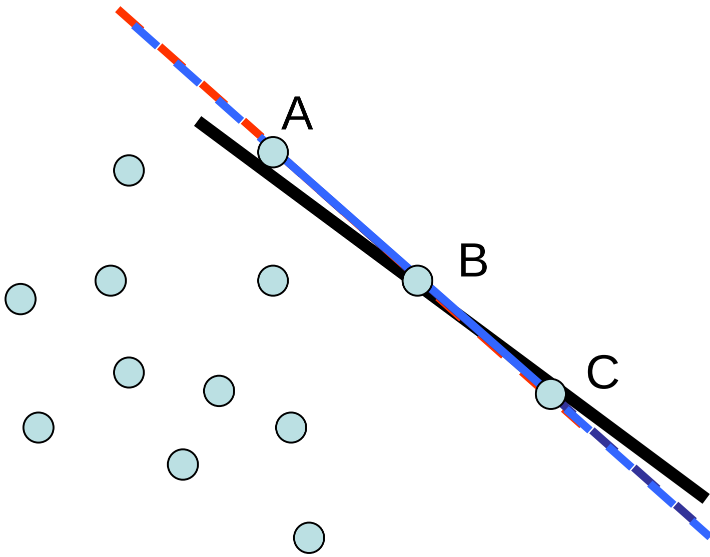
- Description:
  - For each pair of points construct its connecting segment and *supporting line*.
  - Find all the segments whose supporting lines divide the plane into two halves, such that one half plane contains *all* the other points.
  - Construct the convex hull out of these segments.



- Time complexity:
  - All pairs: $O\left(\binom{n}{2}\right) = O\left(\frac{n(n-1)}{2}\right) = O(n^2)$
  - Check all points for each pair: O(n)
  - Total: O(n^3)

Possible Pitfalls

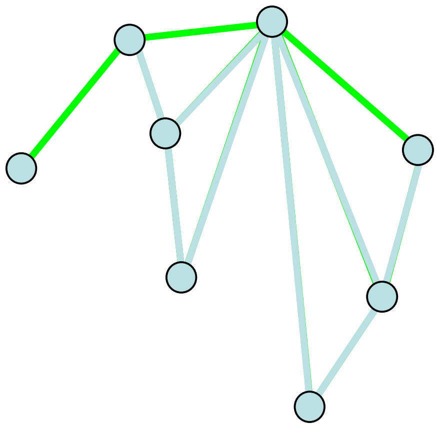
- Degenerate cases – e.g. 3 collinear points. Might harm the correctness of the algorithm. Segments AB, BC and AC will *all* be included in the convex hull.



- Numerical problems – We might conclude that *none* of the three segments belongs to the convex hull.

Convex Hull – Graham's Scan

- Algorithm:
  - Sort the points according to their *x* coordinates.
  - Construct the upper boundary by scanning the points in the sorted order and performing only "right turns".
  - Construct the lower boundary (with "left turns").
  - Concatenate the two boundaries.
- Time Complexity: O(*n*log*n*)
- May be implemented using a stack
- **Question:** How do we check for "right turn" ?



The Algorithm

- Sort the points in increasing order of *x*-coord:

  $p_1, \ldots, p_n$.

- Push(S,p_1); Push(S,p_2);
- For i = 3 to n do
  - While Size(S) $\geq$ 2 and Orient(p_i,top(S),second(S)) $\leq$ 0 do Pop(S);
  - Push(S,p_i);
- Print(S);

Graham's Scan – Time Complexity

- Sorting – O(n log n)
- If D_i is number of points popped on processing p_i,

$$\text{time} = \sum_{i=1}^{n} (D_i + 1) = n + \sum_{i=1}^{n} D_i$$

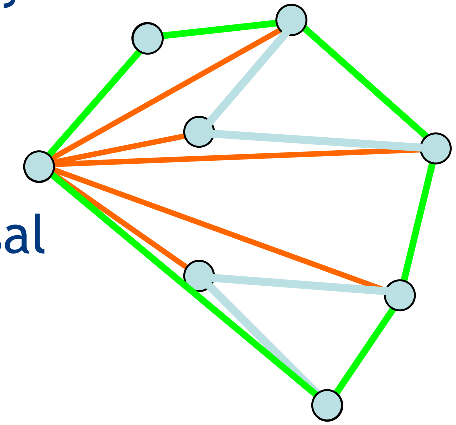
- Each point is pushed on the stack only once.
- Once a point is popped – it cannot be popped again.

$$\sum_{i=1}^{n} D_i \le n$$

- Hence

Graham's Scan– a Variant

- Algorithm:
  - Find one point, p_0, which must be on the convex hull.
  - Sort the other points by the *angle* of the rays to them from p_0.
  - **Question:** Is it necessary to compute the actual angles ?
  - Construct the convex hull using one traversal of the points.



- Time Complexity: O(n log n)

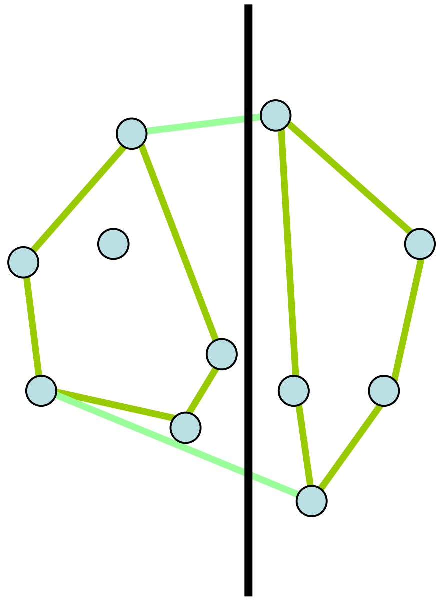
- **Question:** What are the pros and cons of this algorithm relative to the previous ?

Convex Hull - Divide and Conquer

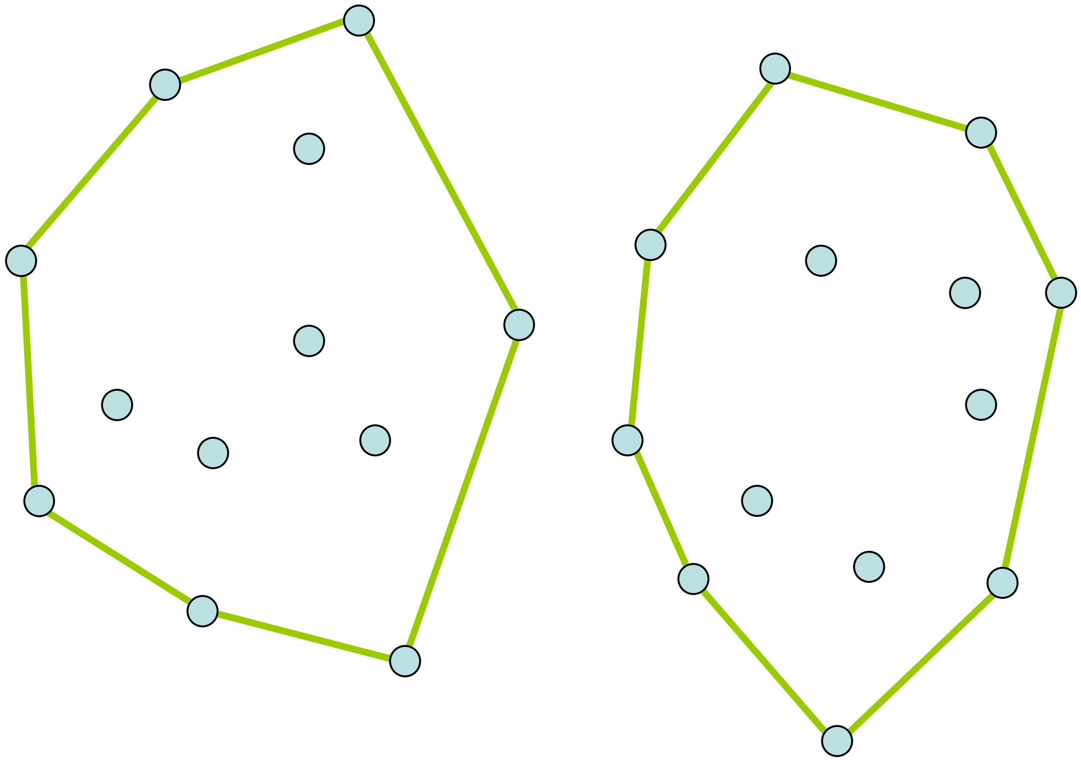
- Algorithm:
  - Find a point with a median *x* coordinate (time: O(*n*))
  - Compute the convex hull of each half (recursive execution)
  - Combine the two convex hulls by finding common *tangents*.

    This can be done in O(*n*).

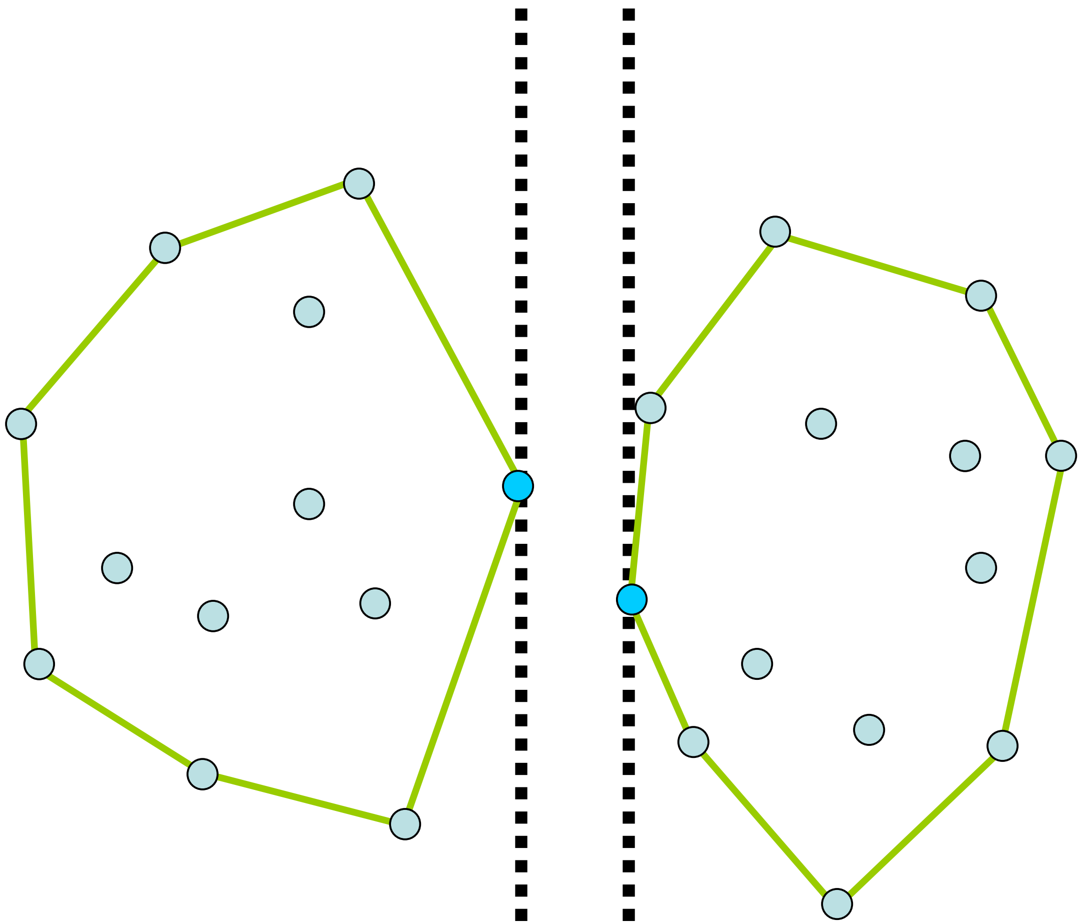
- Complexity: O(*n*log*n*)



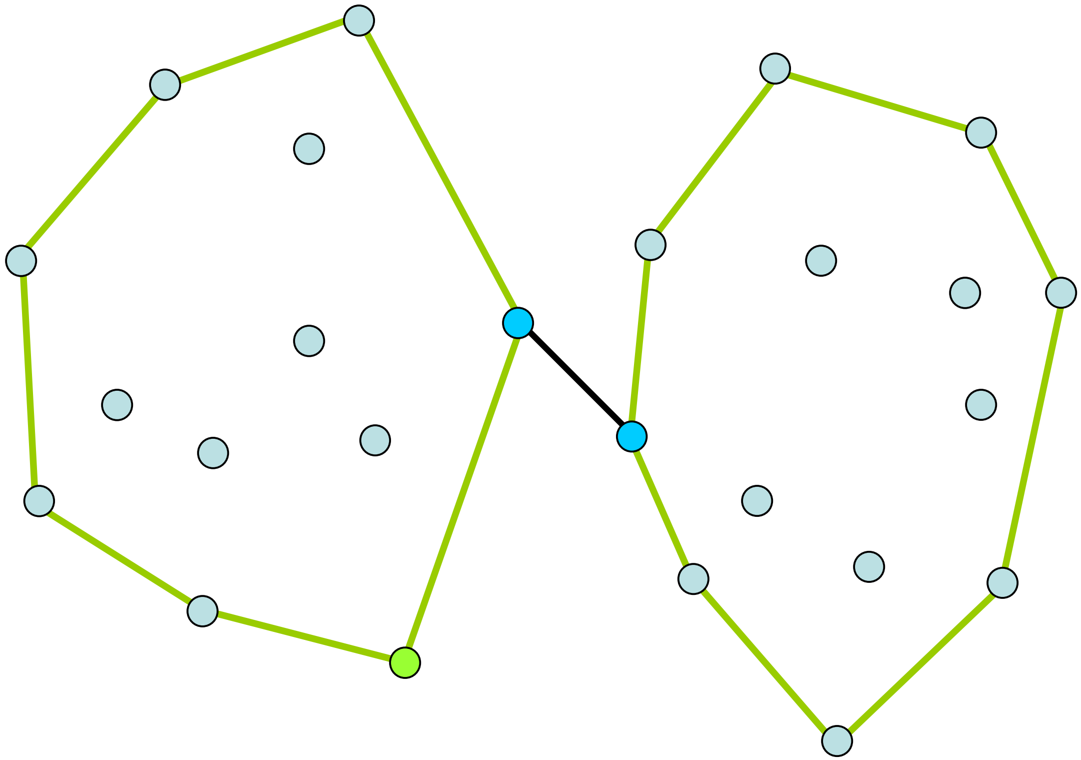
Finding Common Tangents



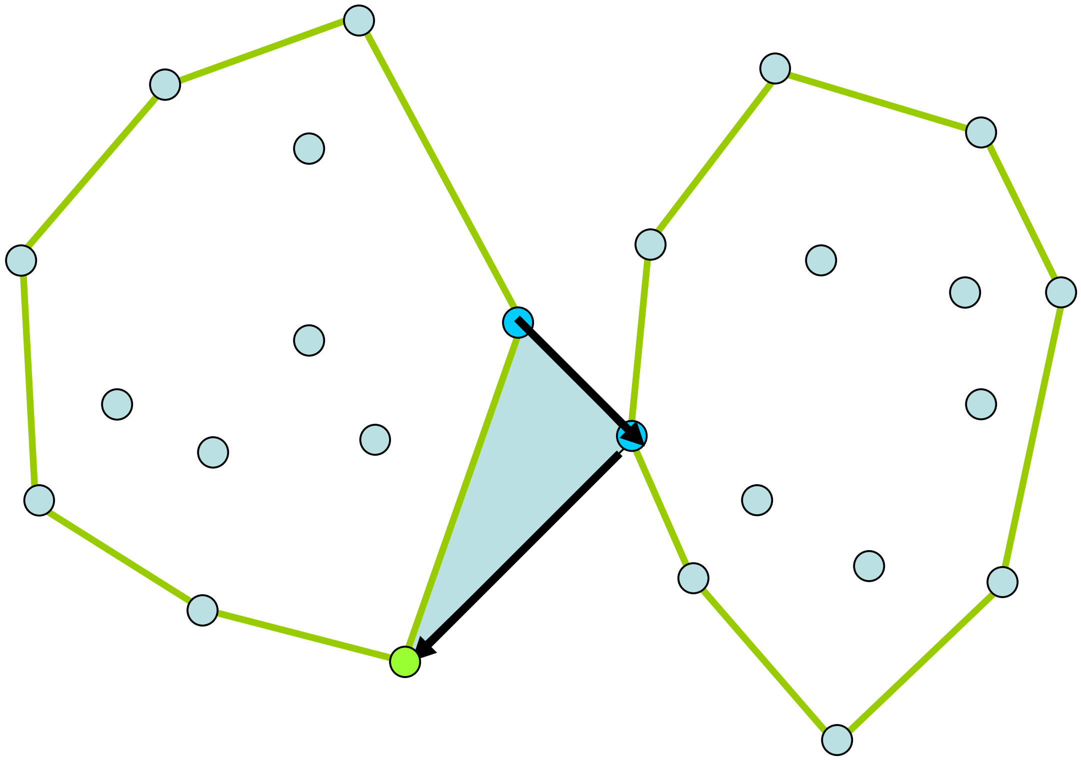
Finding Common Tangents



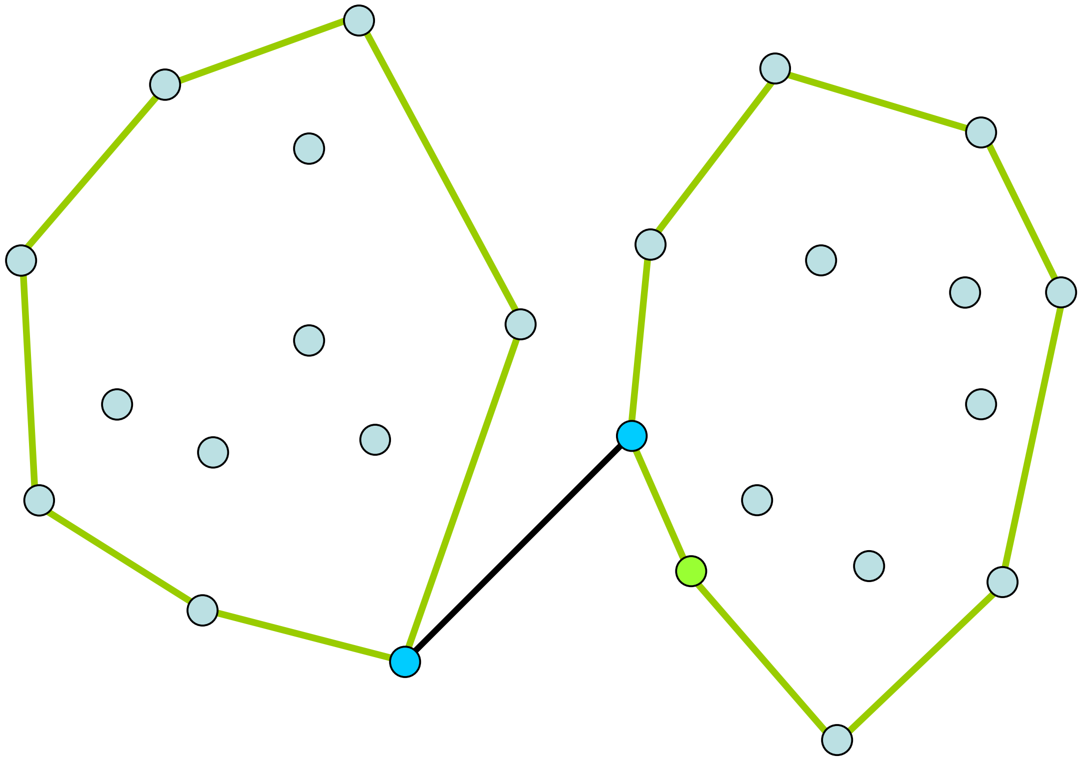
Finding Common Tangents



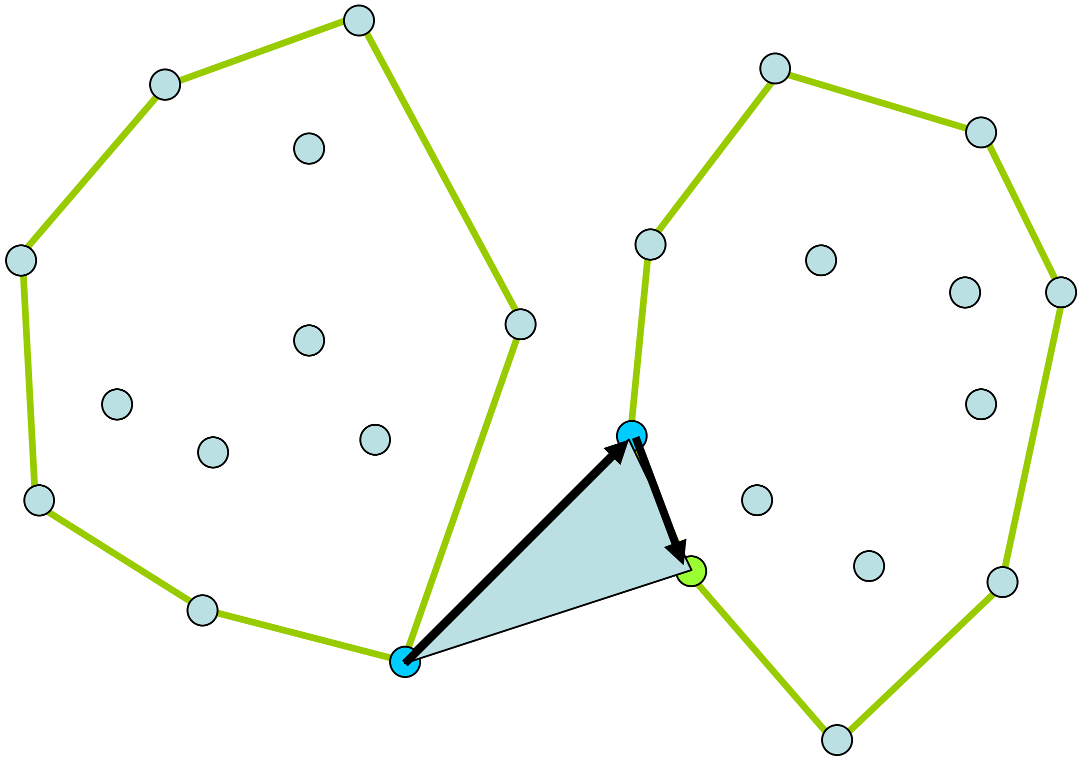
Finding Common Tangents



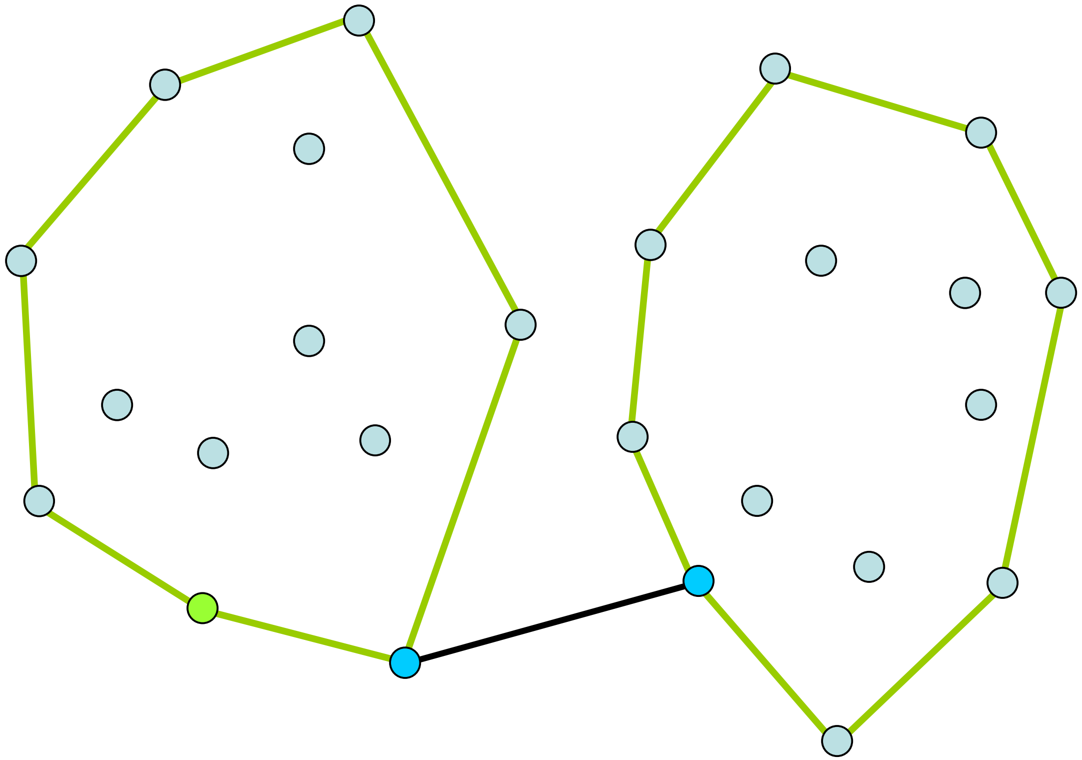
Finding Common Tangents



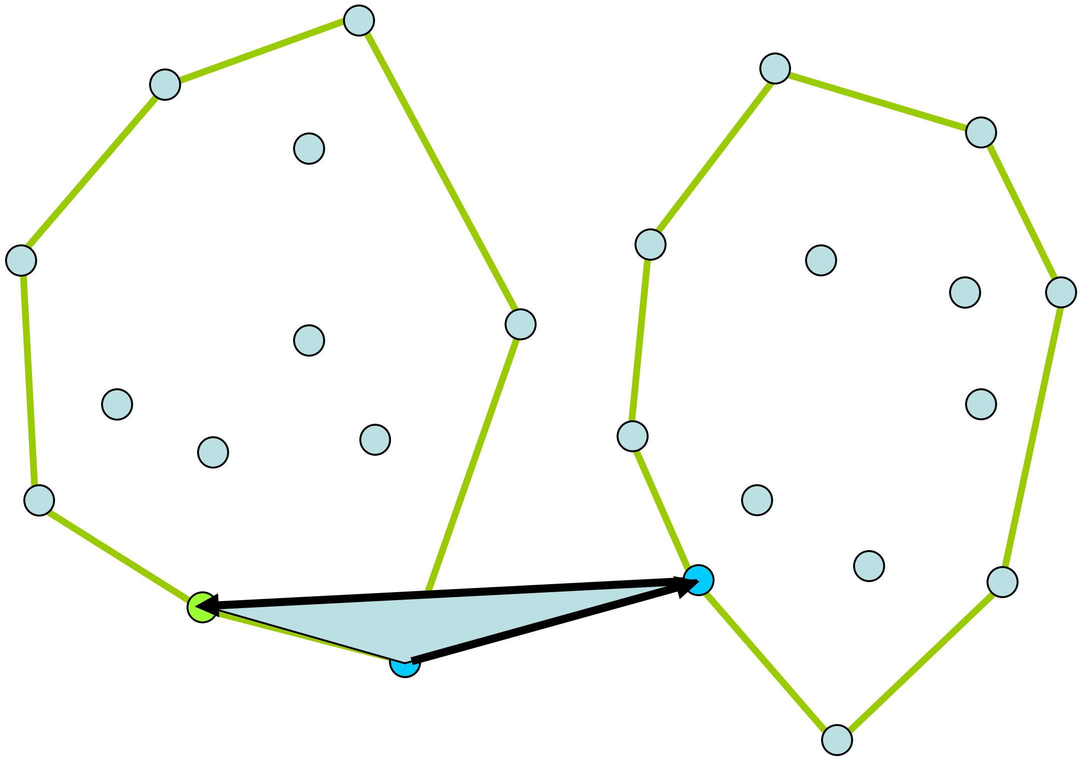
Finding Common Tangents



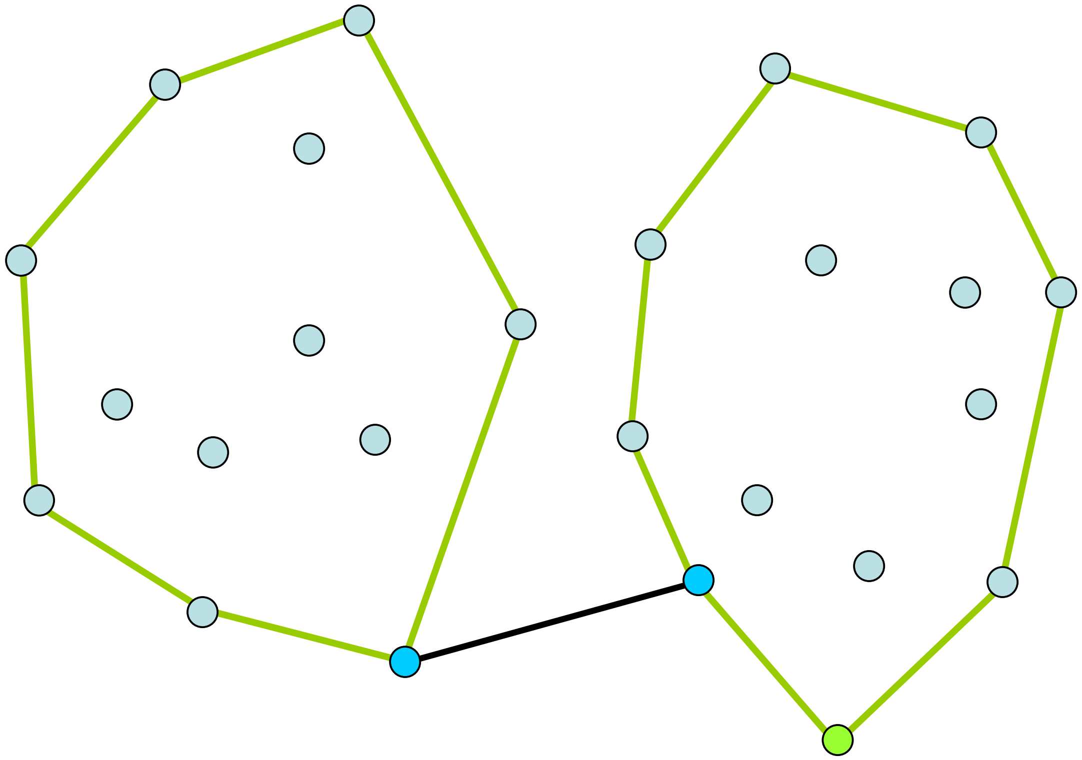
Finding Common Tangents



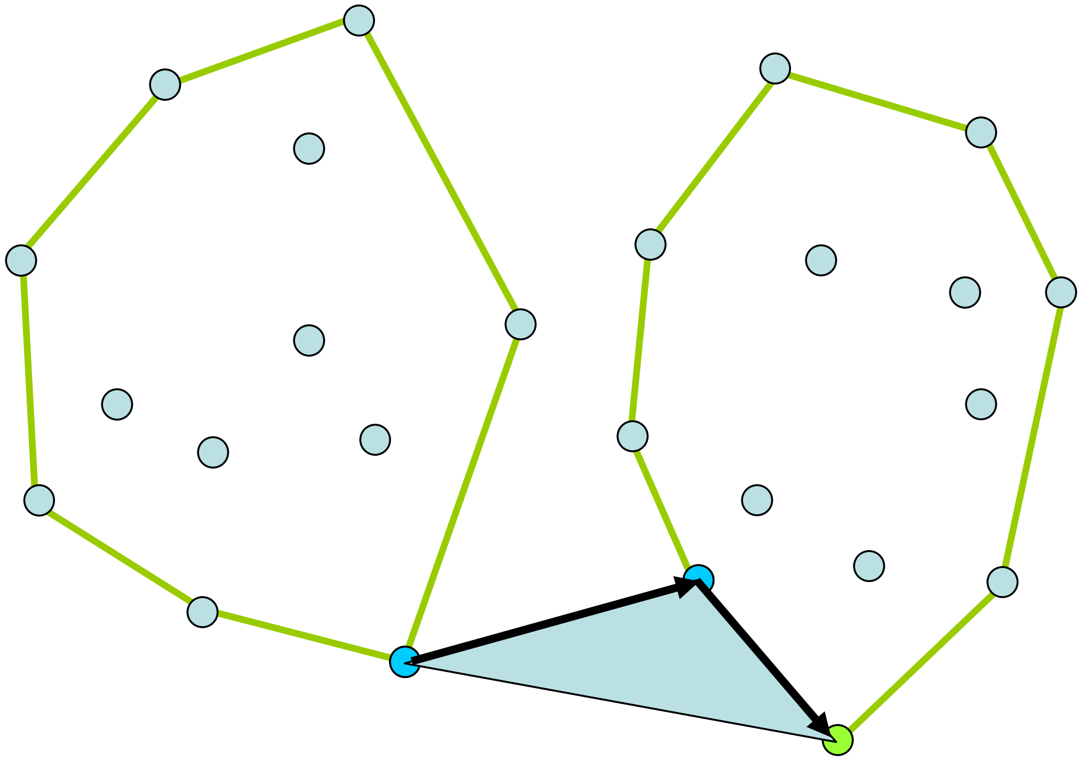
Finding Common Tangents



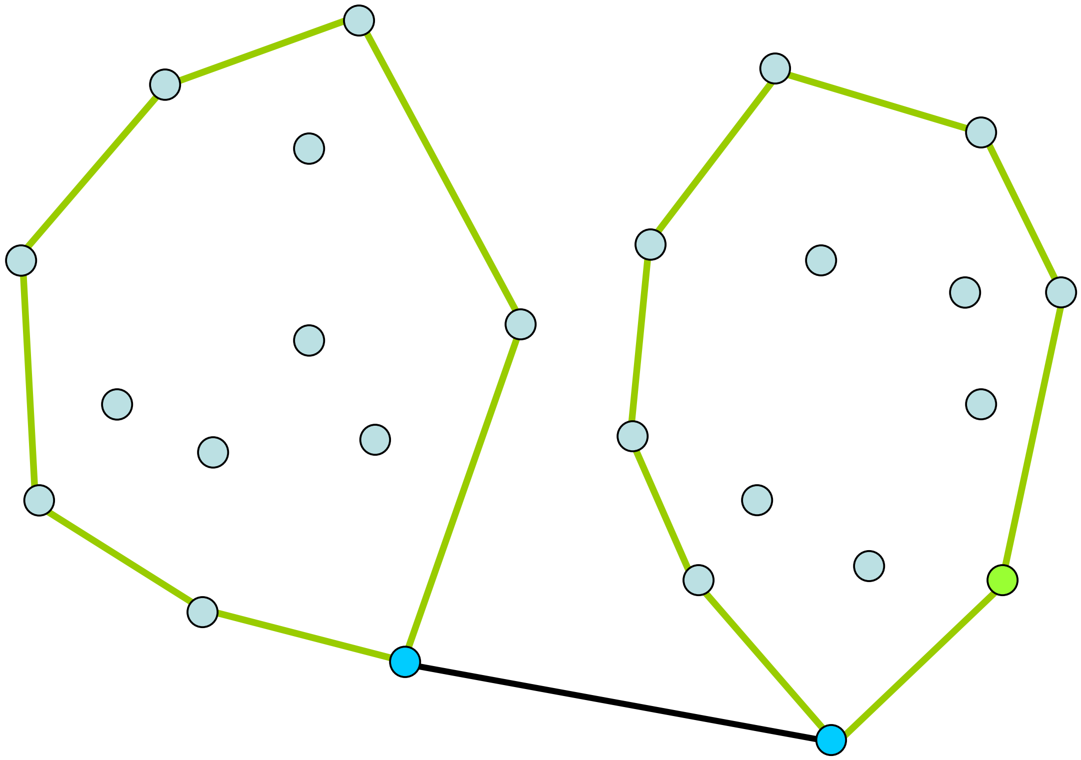
Finding Common Tangents



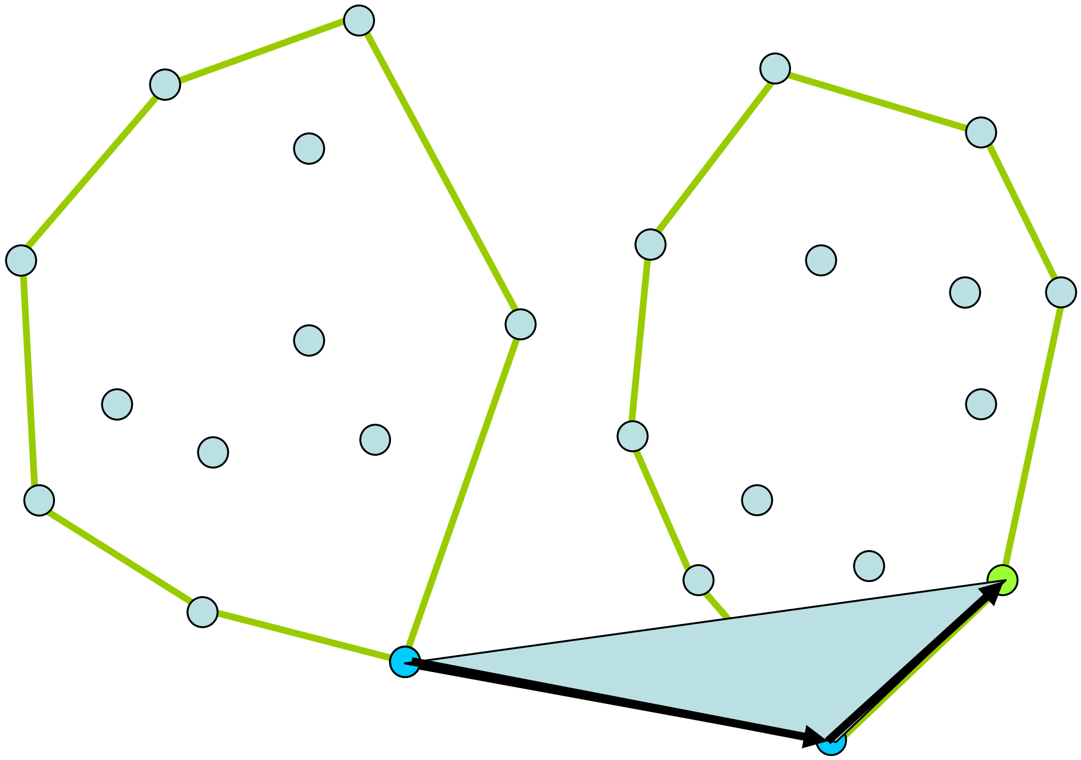
Finding Common Tangents



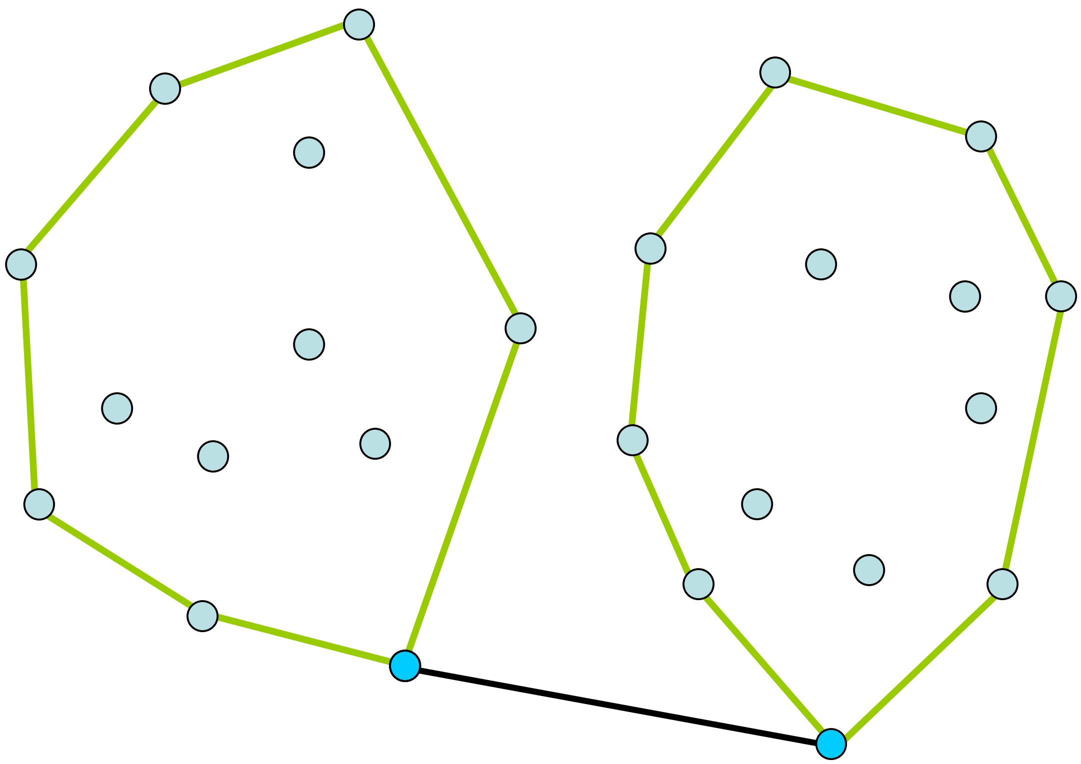
Finding Common Tangents



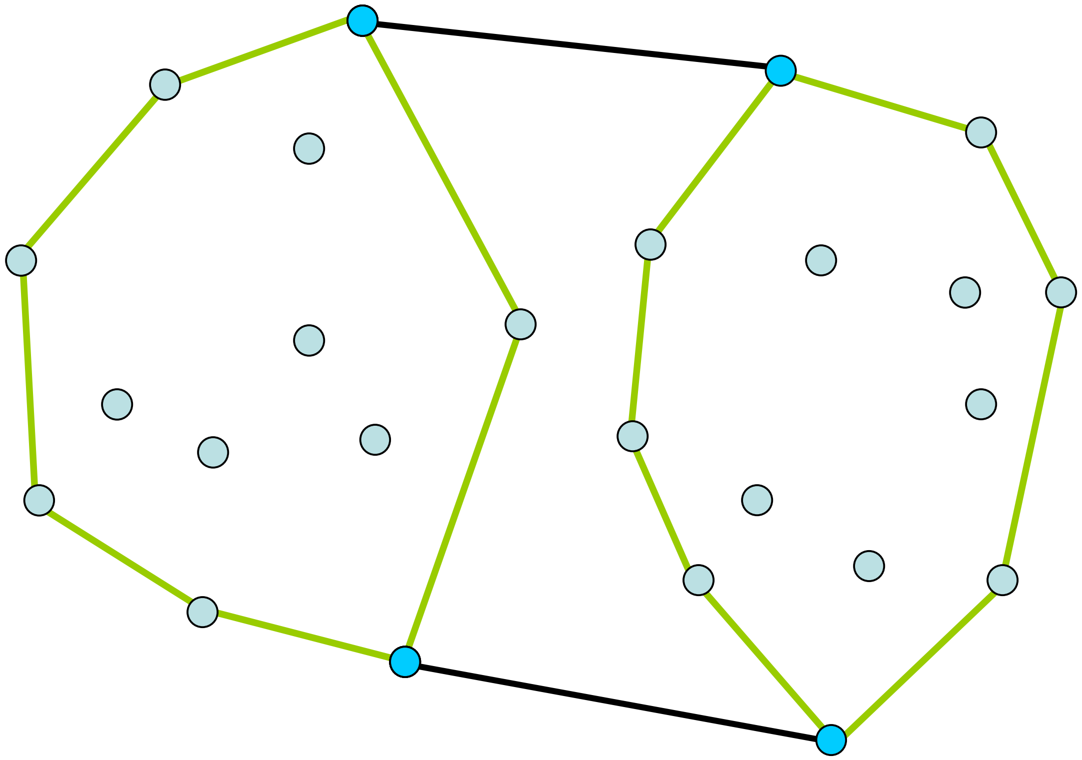
Finding Common Tangents



Finding Common Tangents



Finding Common Tangents



Finding Common Tangents

To find lower tangent:

- Find a - the rightmost point of H_A
- Find b – the leftmost point of H_B

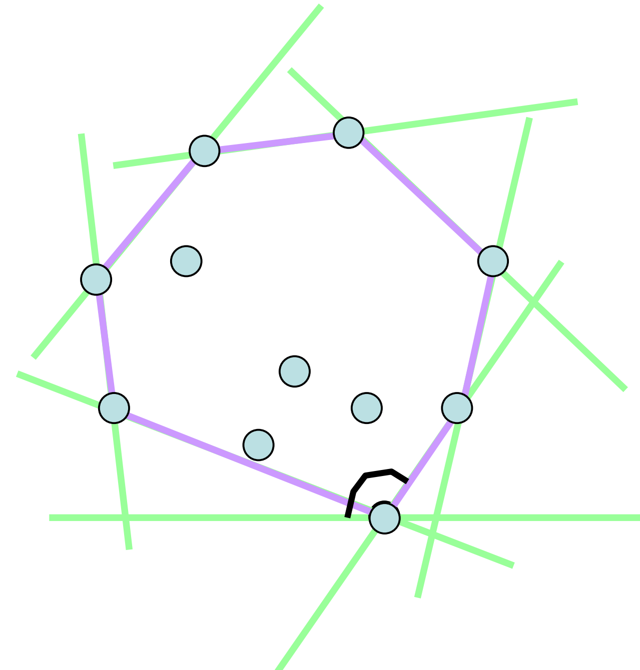
$\Big\}$ O(n)

- While ab is not a lower tangent for H_A and H_B, do:
  - If ab is not a lower tangent to H_A do $a = a$-1
  - If ab is not a lower tangent to H_B do $b = b$-1

Output-Sensitive Convex Hull Gift Wrapping

- Algorithm:
  - Find a point p_1 on the convex hull (e.g. the lowest point).
  - Rotate counterclockwise a line through p_1 until it touches one of the other points (start from a horizontal orientation).

    Question: How is this done ?



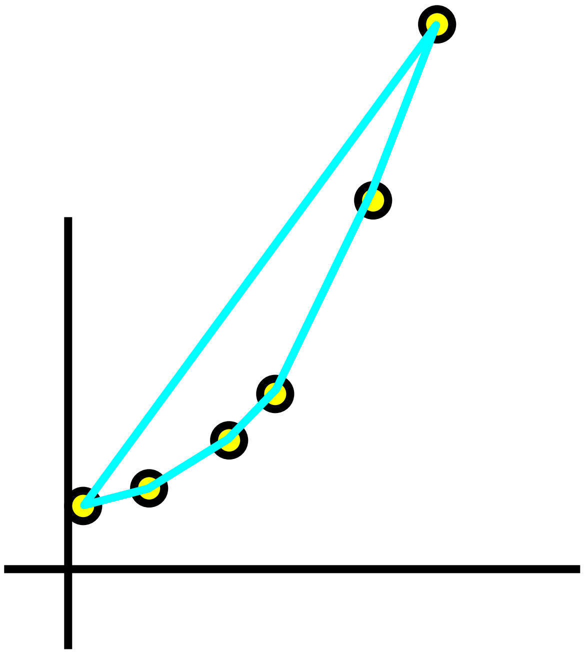
  - Repeat the last step for the new point.
  - Stop when p_1 is reached again.
- Time Complexity: O(nh), where n is the input size and h is the output (hull) size.

General Position

- When designing a geometric algorithm, we first make some simplifying assumptions, e.g.
  - No 3 collinear points.
  - No two points with the same *x* coordinate.
  - etc.
- Later, we consider the general case:
  - How should the algorithm react to degenerate cases ?
  - Will the correctness be preserved ?
  - Will the runtime remain the same ?

Lower Bound for Convex Hull

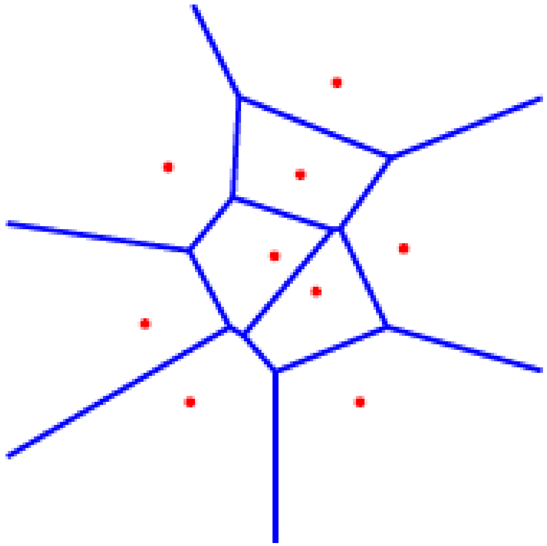
- A reduction from sorting to convex hull is:
  - Given n real values x_i, generate n 2D points on the graph of a convex function, e.g. (x_i, x_i^2).
  - Compute the (ordered) convex hull of the points.
  - The order of the convex hull points is the numerical order of the x_i.
- So $CH = \Omega(n \lg n)$



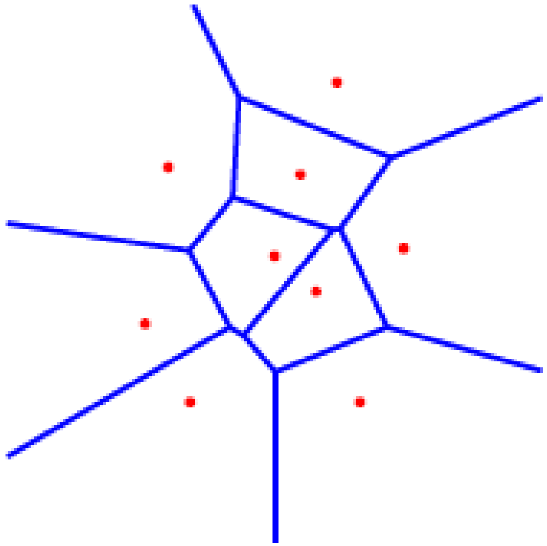
Voronoi Diagram and Delaunay triangulation

Let $\mathcal{E} = \{\mathbf{p_1}, \ldots, \mathbf{p_n}\}$ be a set of points (so-called sites) in $\mathbb{R}^d$. We associate to each site $\mathbf{p_i}$ its Voronoi region $V(\mathbf{p_i})$ such that:

$$V(\mathbf{p_i}) = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x} - \mathbf{p_i}\| \leq \|\mathbf{x} - \mathbf{p_j}\|, \forall j \leq n\}.$$

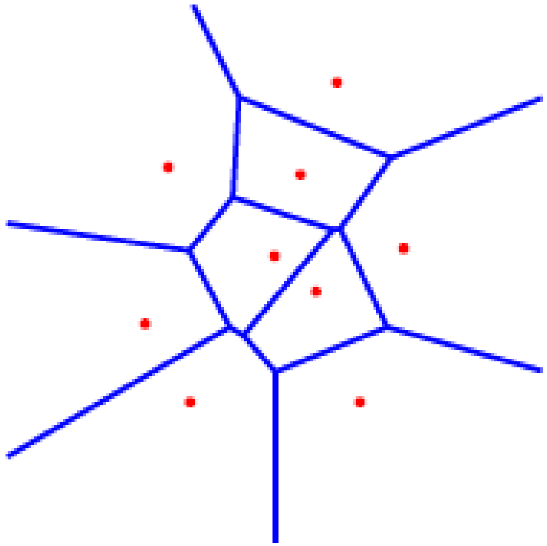


- The collection of the non-empty Voronoi regions and their faces, together with their incidence relations, constitute a cell complex called the **Voronoi diagram** of E.
- The locus of points which are equidistant to two sites pi and pj is called a **bisector**, all bisectors being affine subspaces of IR^d (lines in 2D).



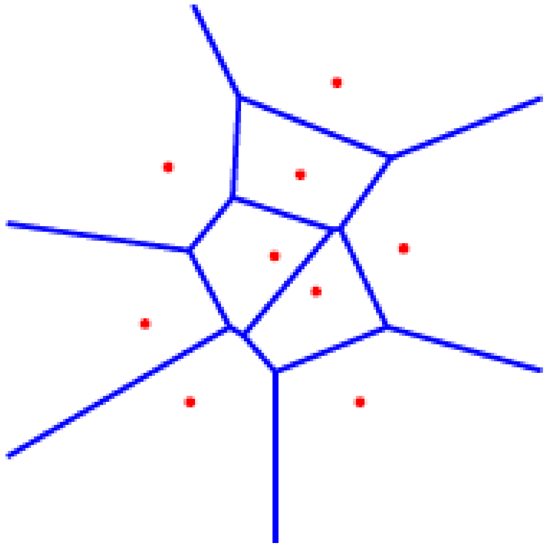
demo

- A Voronoi cell of a site *pi* defined as the intersection of closed half-spaces bounded by bisectors. Implies: All Voronoi cells are **convex**.



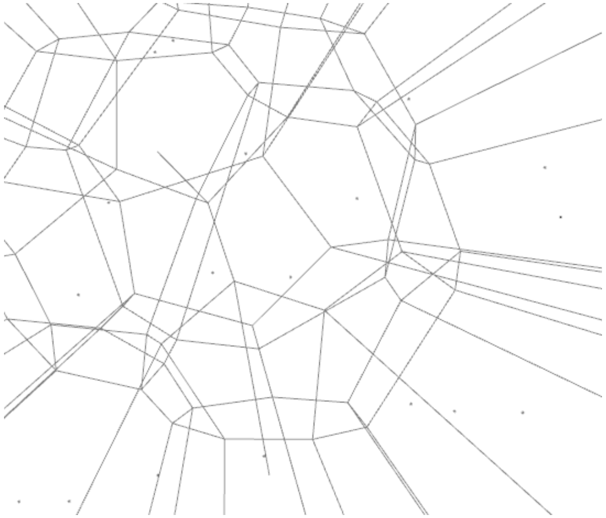
demo

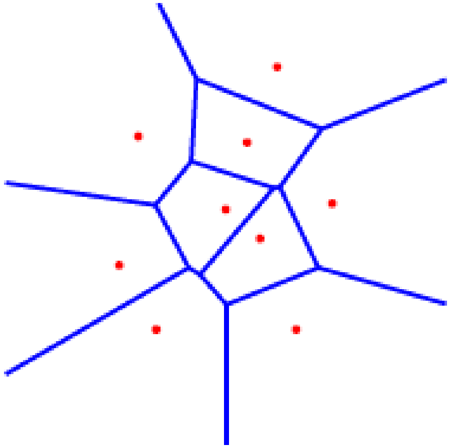
- Voronoi cells may be **unbounded** with unbounded bisectors. Happens when a site pi is on the boundary of the convex hull of E.



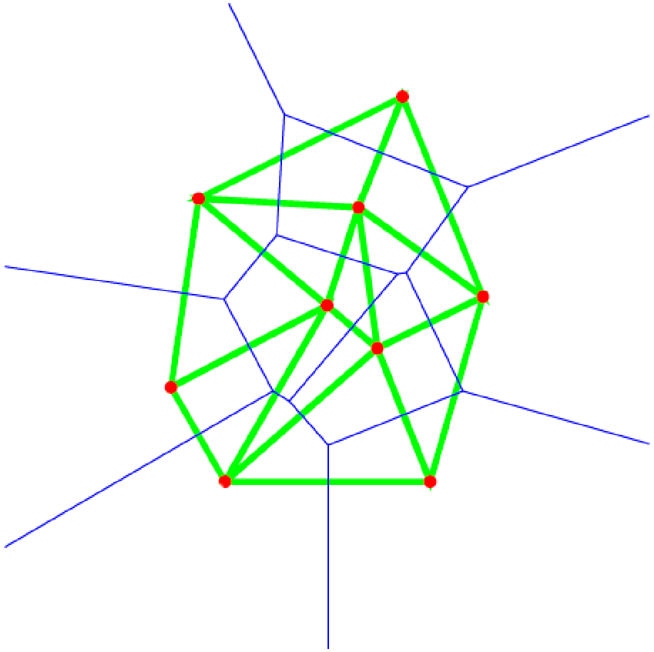
demo

- **Voronoi cells** have **faces** of different dimensions.
- In 2D, a face of dimension k is the intersection of 3 - k Voronoi cells. A **Voronoi vertex** is generically equidistant from three points, and a **Voronoi edge** is equidistant from two points.



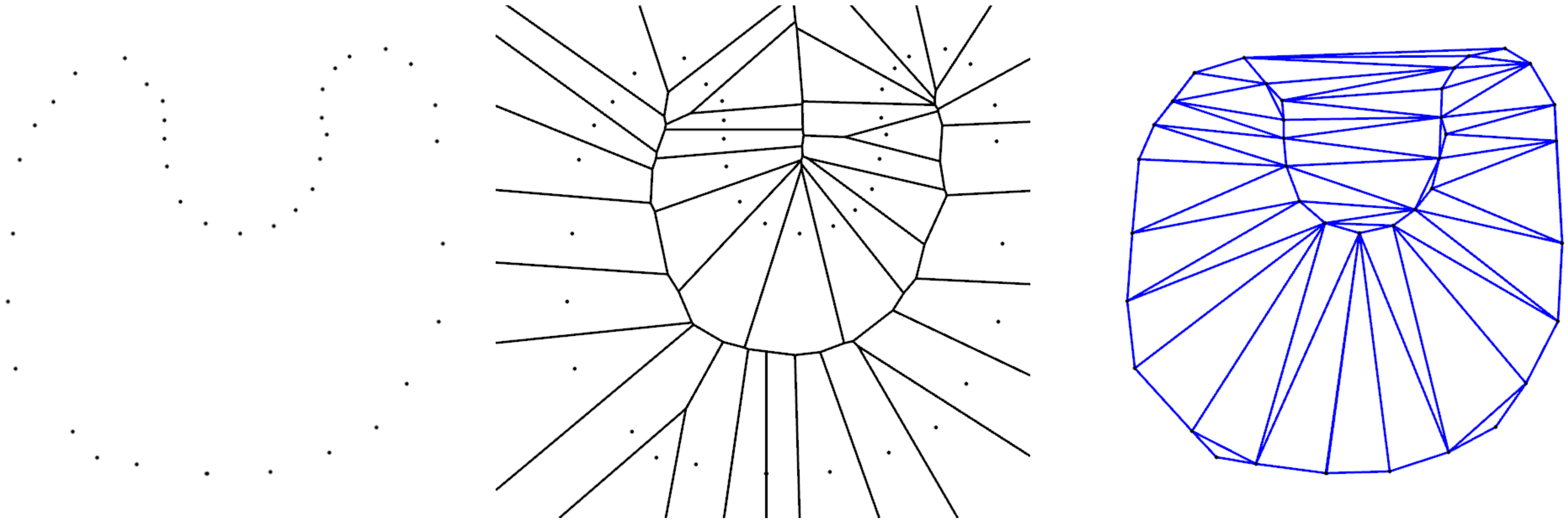


- Dual structure of the Voronoi diagram.
- The Delaunay triangulation of a set of sites E is a simplicial complex such that k+1 points in E form a Delaunay simplex if their Voronoi cells have nonempty intersection

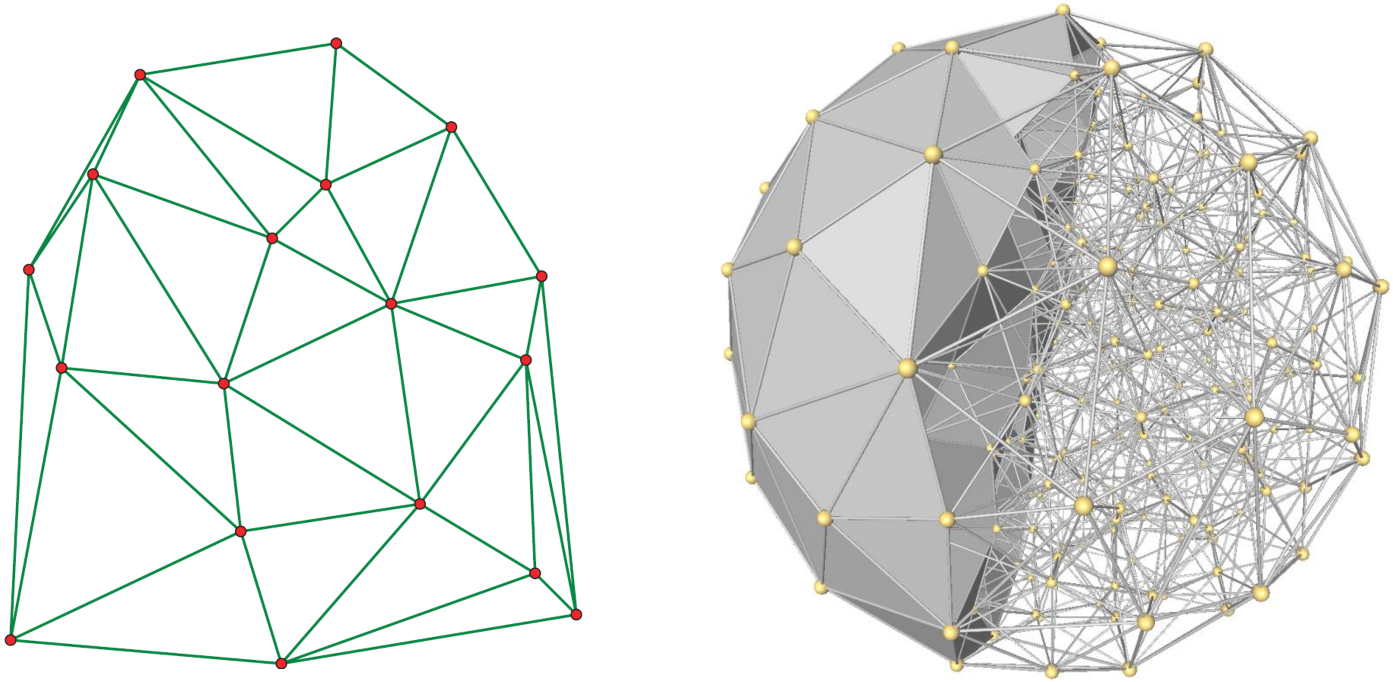


demo

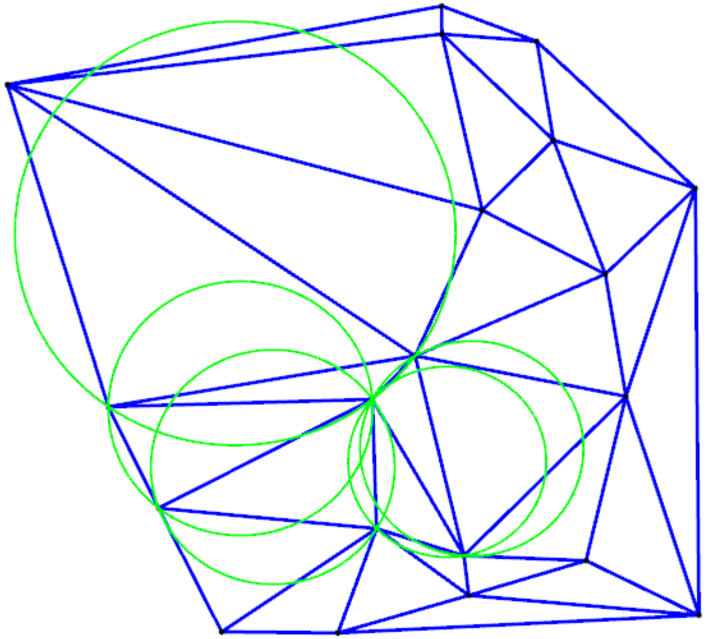
- The Delaunay triangulation of a point set E covers the convex hull of E.



- canonical triangulation associated to any point set

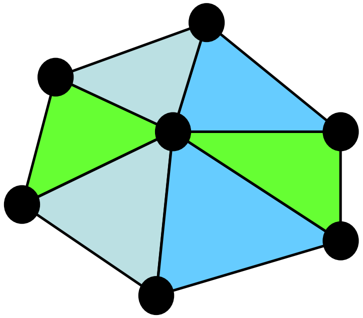


- **Empty circle:** A triangulation T of a point set E such that any d-simplex of T has a circumsphere that does not enclose any point of E is a Delaunay triangulation of E. Conversely, any k-simplex with vertices in E that can be circumscribed by a hypersphere that does not enclose any point of E is a face of the Delaunay triangulation of E.

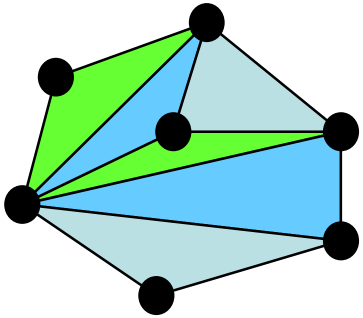


demo

- **In 2D: « quality » triangulation**
  - Smallest triangle angle: The Delaunay triangulation of a point set E is the triangulation of E which **maximizes the smallest angle**.
  - Even stronger: The triangulation of E whose **angular vector** is **maximal** for the lexicographic order is the Delaunay triangulation of E.



good

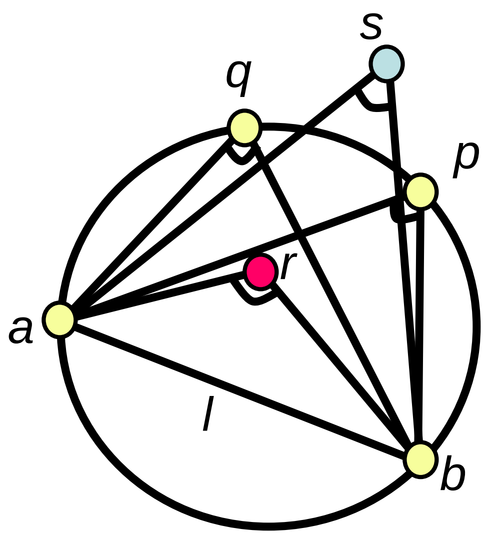


bad

- **Thales' Theorem**: Let C be a circle, and l a line intersecting C at points a and b. Let p, q, r and s be points lying on the same side of l, where p and q are on C, r inside C and s outside C. Then:

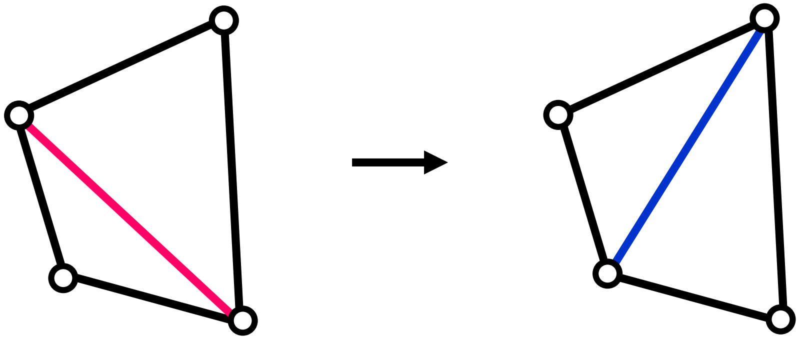
$$\angle arb = 2\angle apb$$

$$\angle aqb > \angle asb$$



Improving a triangulation:

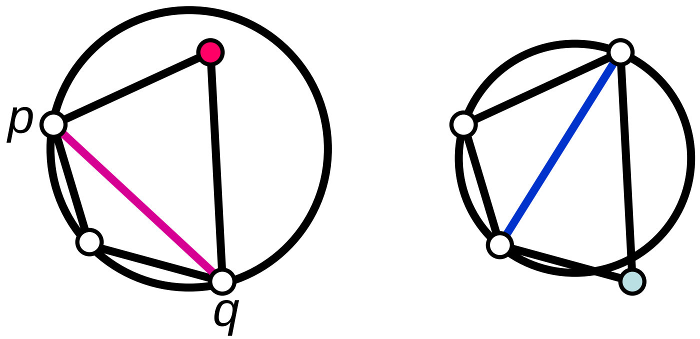
- In any convex quadrangle, an *edge flip* is possible. If this flip *improves* the triangulation locally, it also improves the global triangulation.



- If an edge flip improves the triangulation, the first edge is called **illegal**.

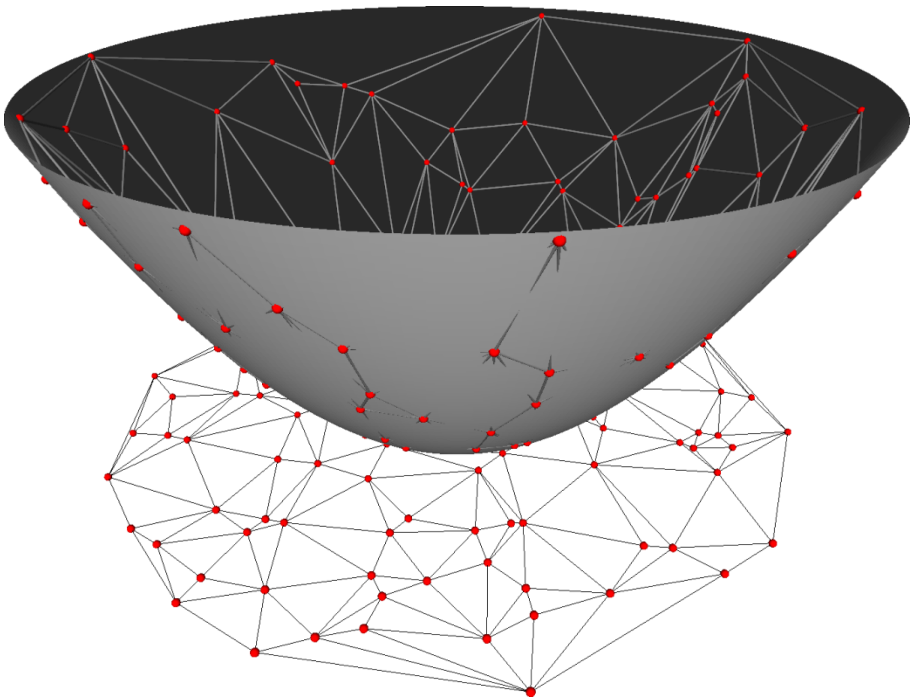
Illegal edges:

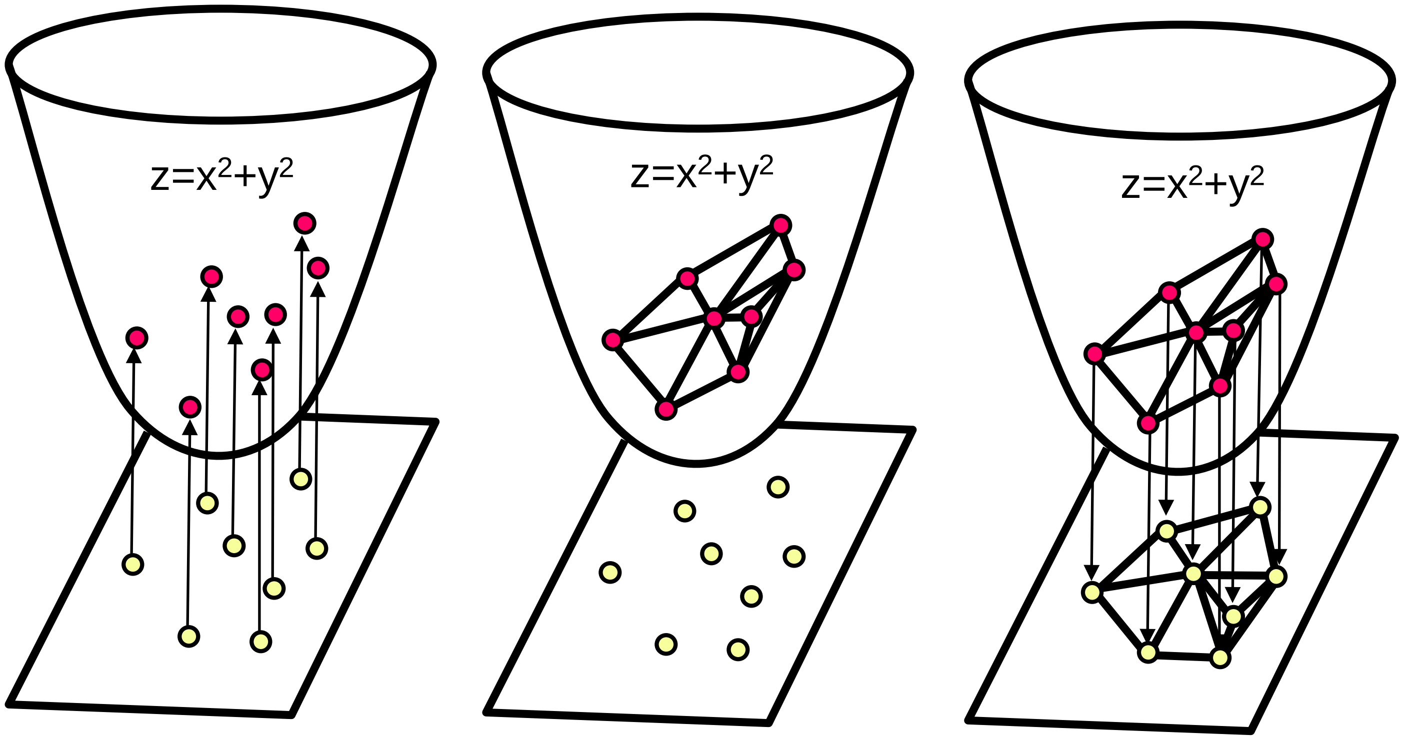
- **Lemma:** An edge *pq* is illegal iff one of its opposite vertices is inside the circle defined by the other three vertices.
- **Proof:** By Thales' theorem.



- **Theorem:** A Delaunay triangulation does not contain illegal edges.
- **Corollary:** A triangle is Delaunay iff the circle through its vertices is empty of other sites (the *empty-circle* condition).
- **Corollary:** The Delaunay triangulation is not unique if more than three sites are co-circular.

- **Duality on the paraboloid:** Delaunay triangulation obtained by projecting the lower part of the convex hull.



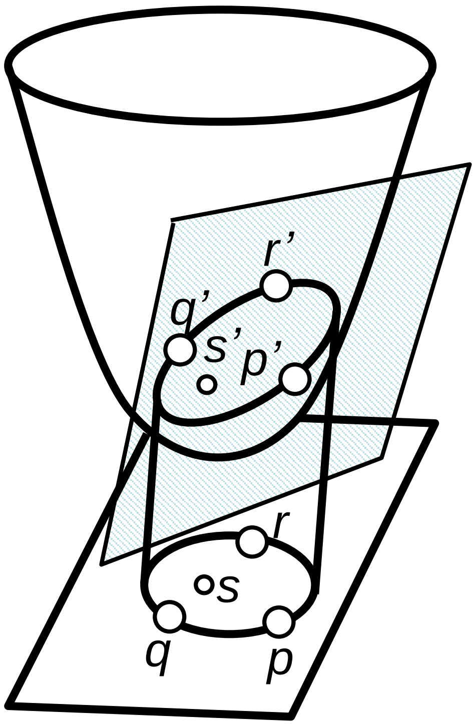


Project the 2D point set onto the 3D paraboloid → Compute the 3D lower convex hull → Project the 3D facets back to the plane.

- The intersection of a plane with the paraboloid is an ellipse whose projection to the plane is a circle.
- *s* lies within the circumcircle of *p*, *q*, *r* iff *s*' lies on the lower side of the plane passing through *p*', *q*', *r*'.

$\downarrow$

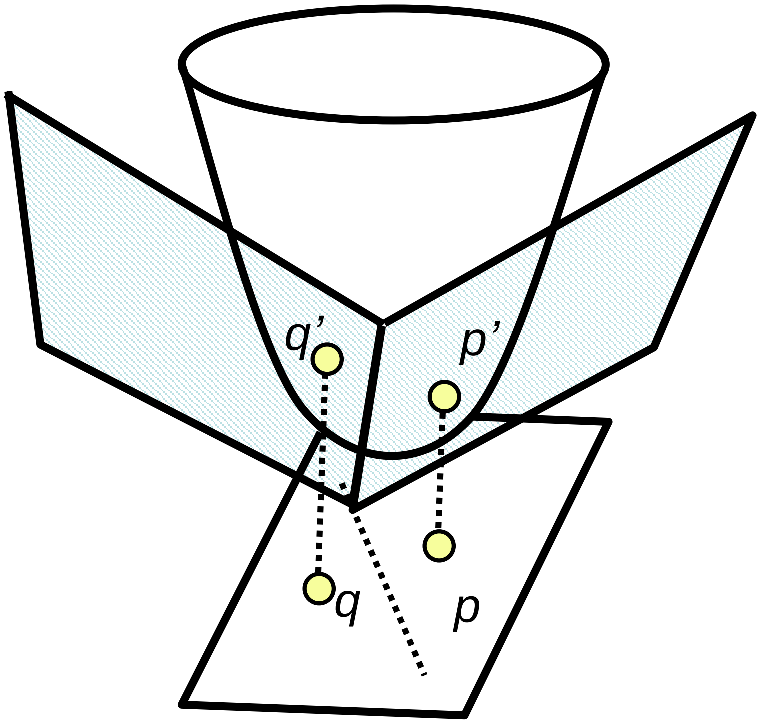
- *p*, *q*, *r* $\in$ *S* form a Delaunay triangle iff *p*', *q*', *r*' form a face of the convex hull of *S*'.

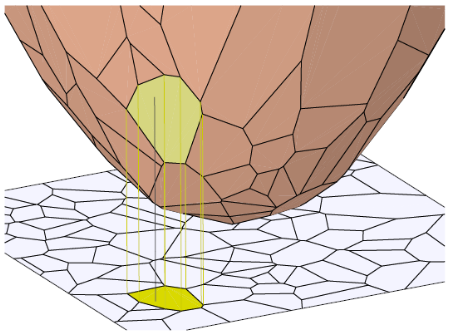


- Given a set S of points in the plane, associate with each point $p=(a,b)\in S$ the plane tangent to the paraboloid at p:

$$z = 2ax+2by-(a2+b2).$$

- VD(S) is the projection to the (x,y) plane of the 1-skeleton of the convex polyhedron formed from the intersection of the halfspaces above these planes.





An naïve O(n^4) Construction Algorithm

- **Repeat until impossible:**
  - Select a triple of sites.
  - If the circle through them is empty of other sites, keep the triangle whose vertices are the triple.

Theorem: If *a*,*b*,*c*,*d* form a CCW convex polygon, then *d* lies in the circle determined by *a*, *b* and *c* iff:

$$\det\begin{pmatrix} a_x & a_y & a_x^2+a_y^2 & 1 \\ b_x & b_y & b_x^2+b_y^2 & 1 \\ c_x & c_y & c_x^2+c_y^2 & 1 \\ d_x & d_y & d_x^2+d_y^2 & 1 \end{pmatrix} > 0$$

Proof: We prove that equality holds if the points are co-circular. There exists a center *q* and radius *r* such that:

$$(a_x - q_x)^2 + (a_y - q_y)^2 = r^2$$

and similarly for *b*, *c*, *d*:

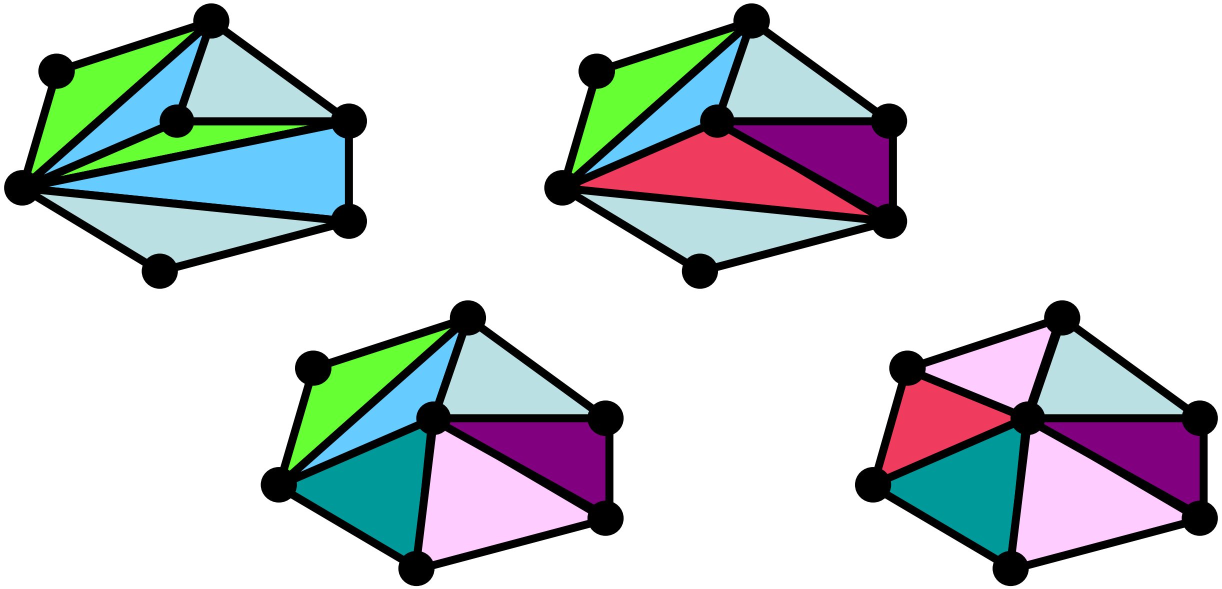
$$\begin{pmatrix} a_x^2+a_y^2 \\ b_x^2+b_y^2 \\ c_x^2+c_y^2 \\ d_x^2+d_y^2 \end{pmatrix} - 2q_x \begin{pmatrix} a_x \\ b_x \\ c_x \\ d_x \end{pmatrix} - 2q_y \begin{pmatrix} a_y \\ b_y \\ c_y \\ d_y \end{pmatrix} + (q_x^2 + q_y^2 - r^2) \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = 0$$

So these four vectors are linearly dependent, hence their det vanishes.

Corollary: *d*$\in$circle(*a*,*b*,*c*) iff *b*$\in$circle(*c*,*d*,*a*) iff *c*$\notin$circle(*d*,*a*,*b*) iff *a*$\notin$circle(*b*,*c*,*d*)

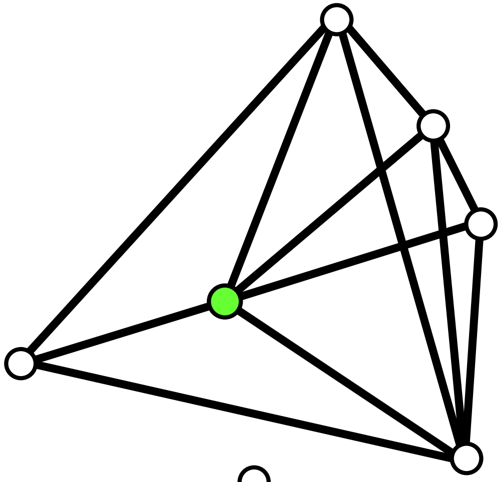
Another naive construction:

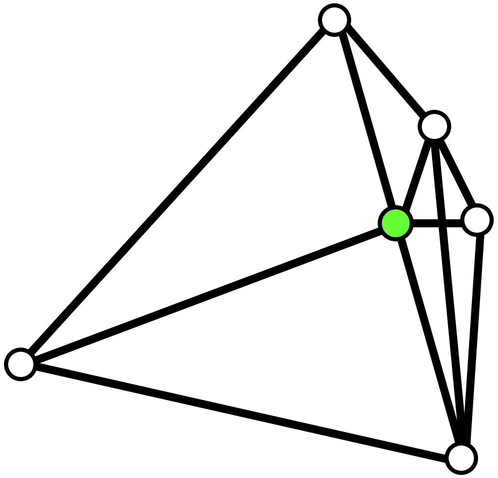
- Start with an arbitrary triangulation. Flip any illegal edge until no more exist.
- Requires proof that there are no local minima.
- Could take a long time to terminate.



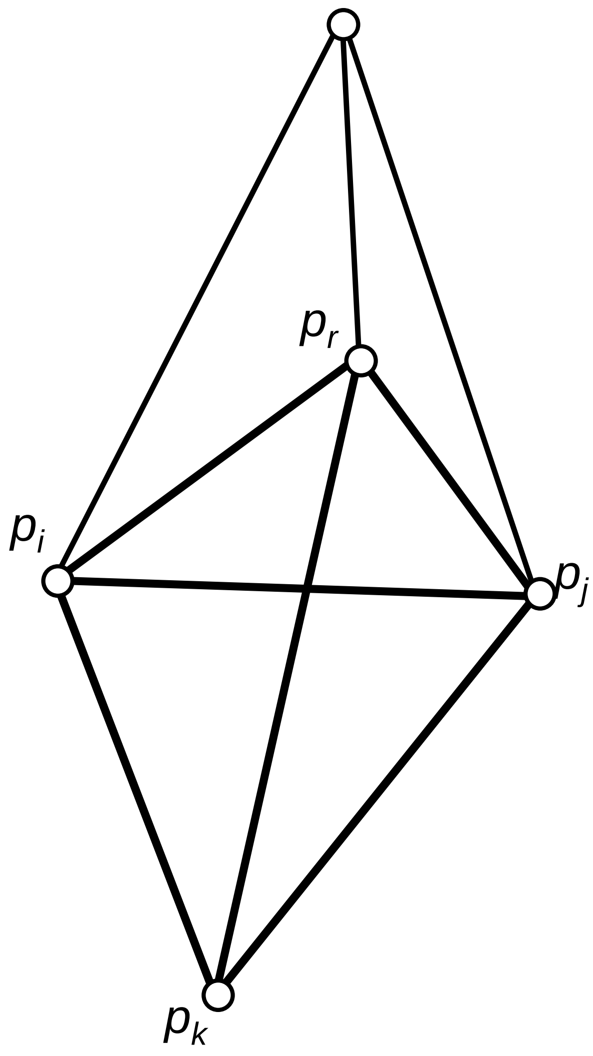
Incremental algorithm:

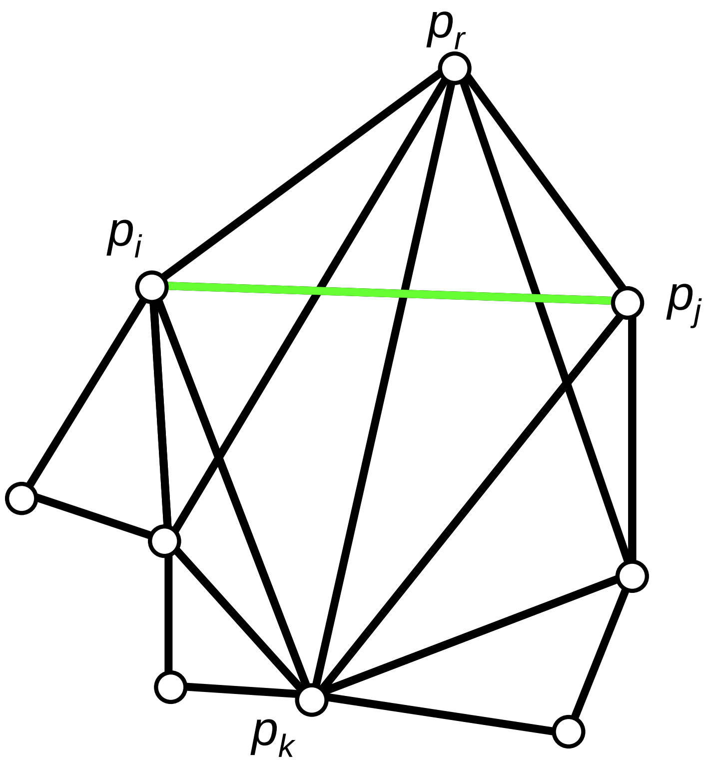
- Form bounding triangle which encloses all the sites.
- Add the sites one after another in random order and update triangulation.
- **If the site is inside an existing triangle:**
  - Connect site to triangle vertices.
  - Check if a 'flip' can be performed on one of the triangle edges. If so – check recursively the neighboring edges.
- **If the site is on an existing edge:**
  - Replace edge with four new edges.
  - Check if a 'flip' can be performed on one of the opposite edges. If so – check recursively the neighboring edges.





- A new vertex p_r is added, causing the creation of edges.
- The legality of the edge $p_i p_j$ (with opposite vertex) p_k is checked.
- If $p_i p_j$ is illegal, perform a flip, and recursively check edges $p_i p_k$ and $p_j p_k$, the new edges opposite p_r.
- Notice that the recursive call for $p_i p_k$ cannot eliminate the edge $p_r\, p_k$.
- **Note:** All edge flips replace edges opposite the new vertex by edges incident to it!

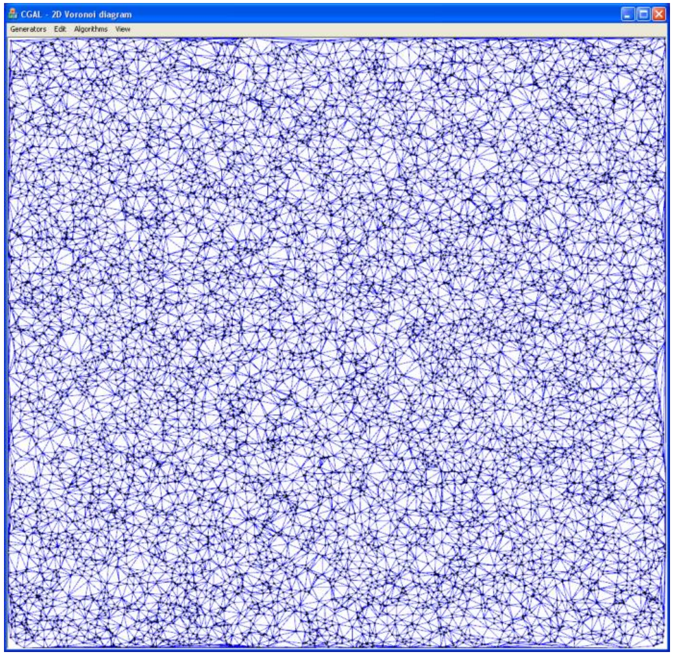



p_r
p_i
p_j
p_k

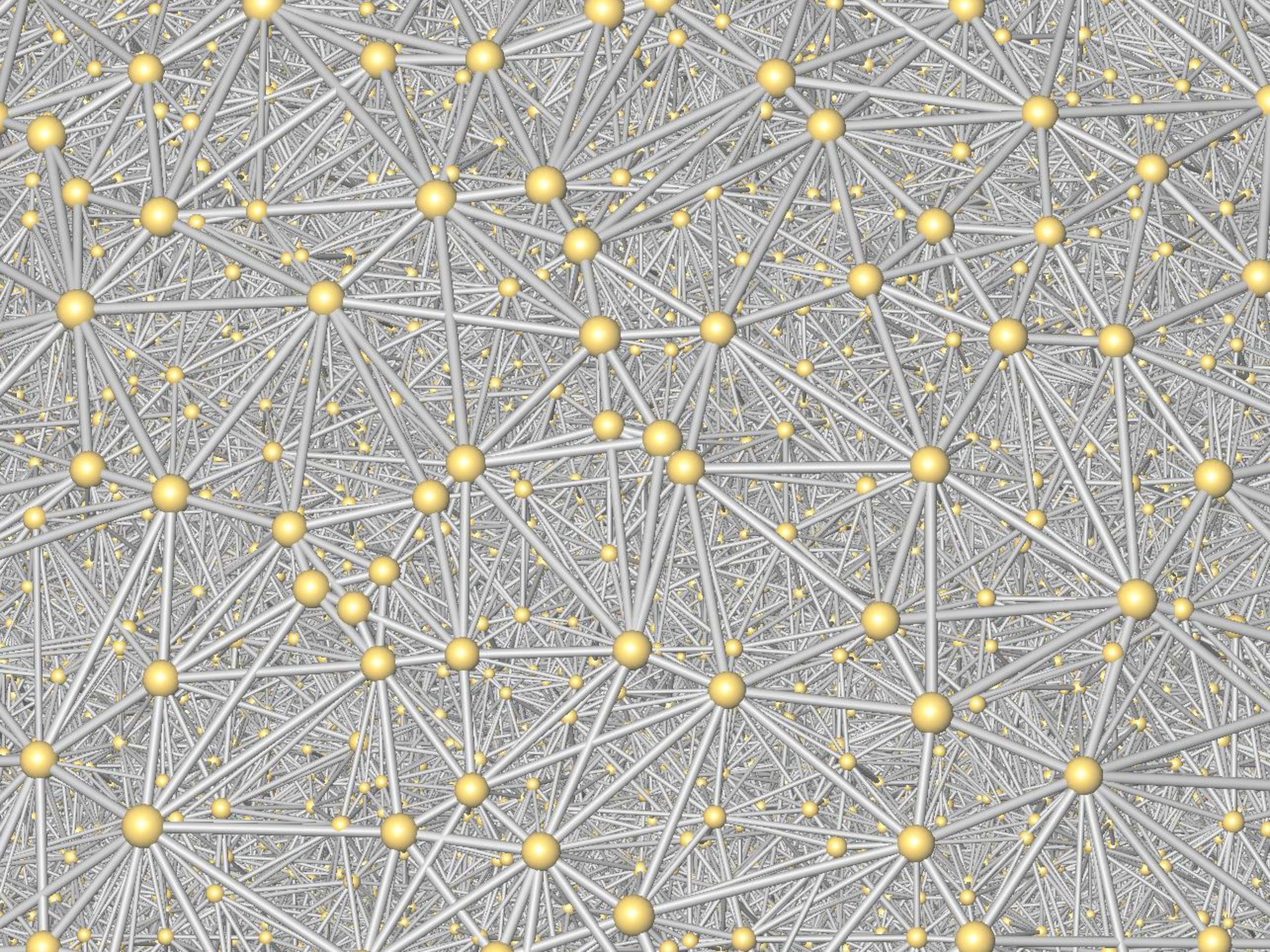
- **Theorem:** The expected number of edges flips made in the course of the algorithm (some of which also disappear later) is at most $6n$.
- **Proof:** During insertion of vertex p_i, k_i new edges are created: 3 new initial edges, and k_i-3 due to flips.

  Backward analysis: E[k_i] = the expected degree of p_i after the insertion is complete = 6 (Euler).

- **Point location for every point:** O(log n) time.
- **Flips:** Θ(n) expected time in total (for all steps).
- **Total expected time:** O(n log n).
- **Space:** Θ(n).

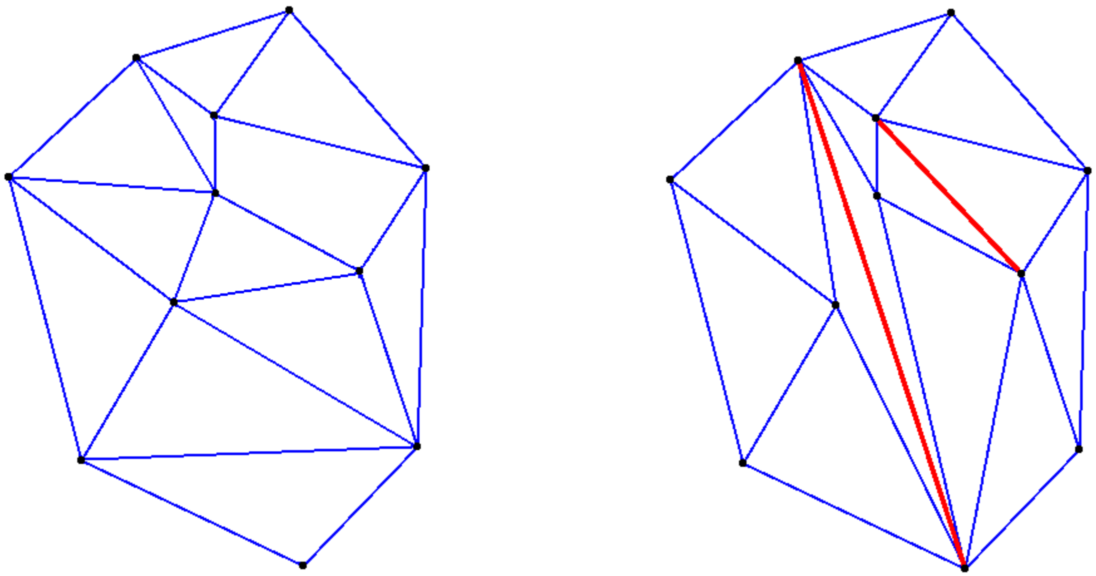


demo

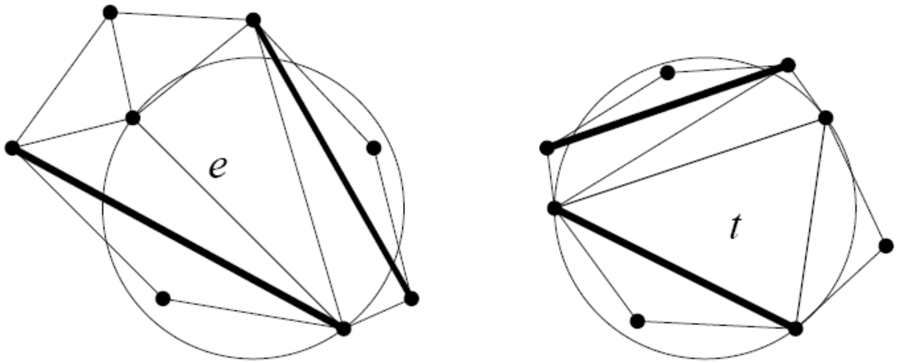


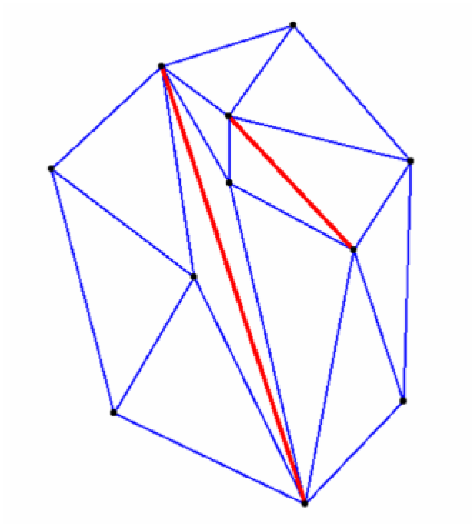
Constrained Delaunay triangulation

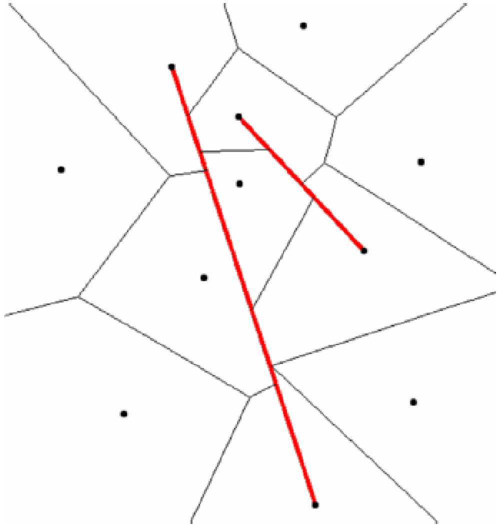
- **Definition 1** : Let (P, S) be a PSLG. The constrained triangulation T(P, S) is constrained Delaunay iff the circumcircle of any triangle t of T encloses no vertex visible from a point in the relative interior of t.

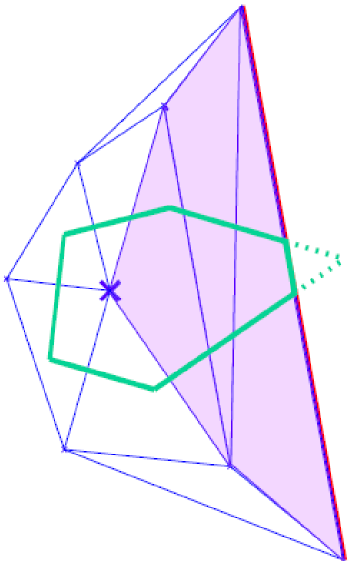


- **Definition 2** : Let (P, S) be a PSLG. The constrained triangulation T(P, S) is constrained Delaunay iff any edge e of T is either a segment of S or is constrained Delaunay.
- Simplex e constrained Delaunay with respect to the PSLG (P, S) iff: int(e) $\cap$ S = 0
- There exists a circumcircle of e that encloses no vertex visible from a point in the relative interior of e.









constrained

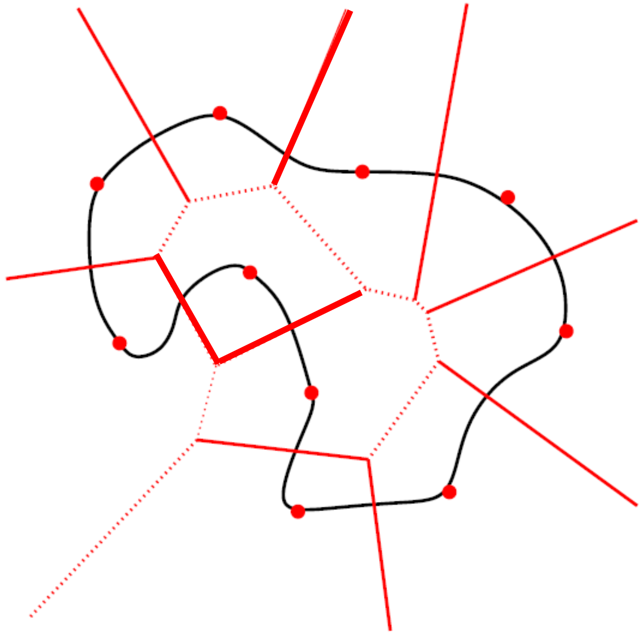
Bounded Voronoi diagram

“blind” triangles

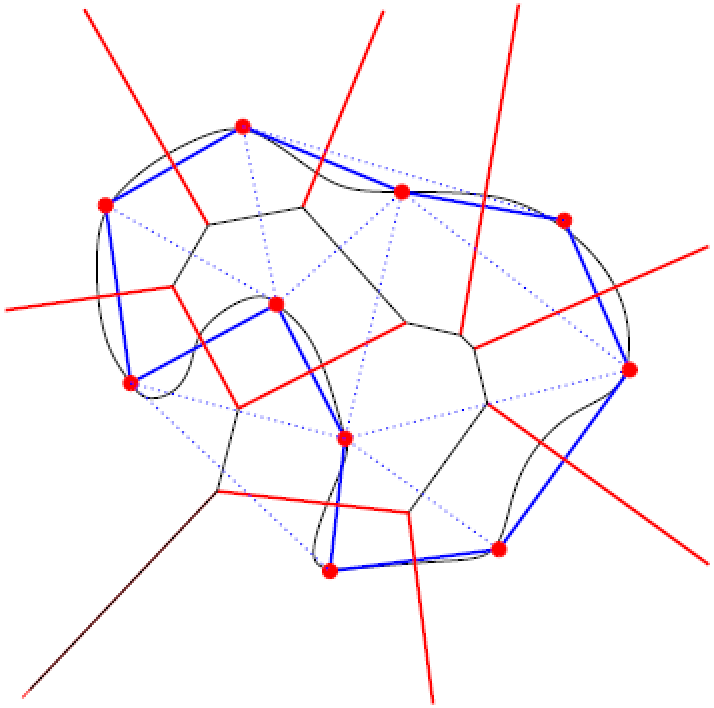
- Any PSLG (P, S) has a constrained Delaunay triangulation. If (P, S) has no degeneracy , this triangulation is unique.

Delaunay Filtering

- The Voronoi diagram **restricted** to a curve S, $\text{Vor}_{|S}(E)$, is the set of edges of Vor(E) that intersect S.



- The restricted Delaunay triangulation restricted to a curve S is the set of **edges** of the Delaunay triangulation whose dual edges **intersect** S.



(2D)

- The restricted Delaunay triangulation restricted to a surface S is the set of **triangles** of the Delaunay triangulation whose dual edges **intersect** S.