CS60064 Spring 2022

Computational Geometry

Instructors

Bhargab B. Bhattacharya (BBB)

Partha Bhowmick (PB)

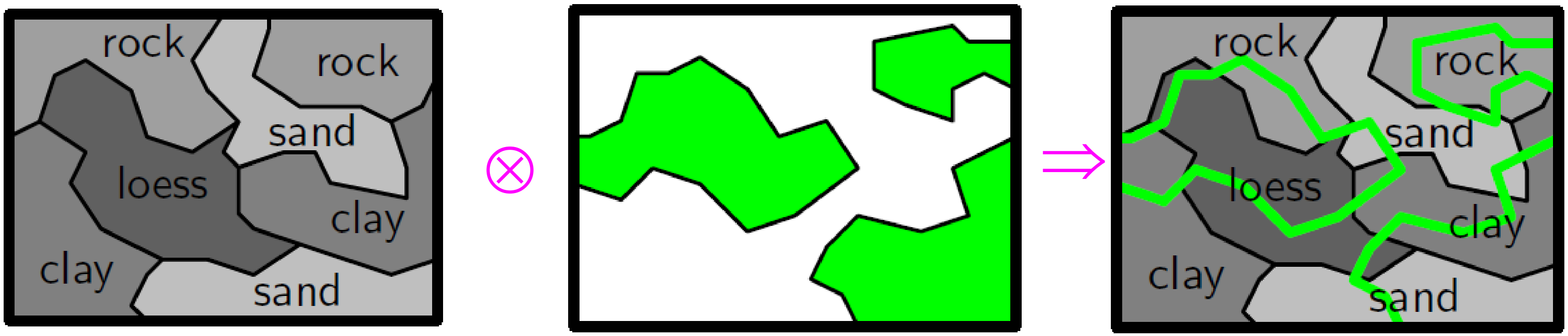
Lecture #15

09 February 2022

Indian Institute of Technology Kharagpur

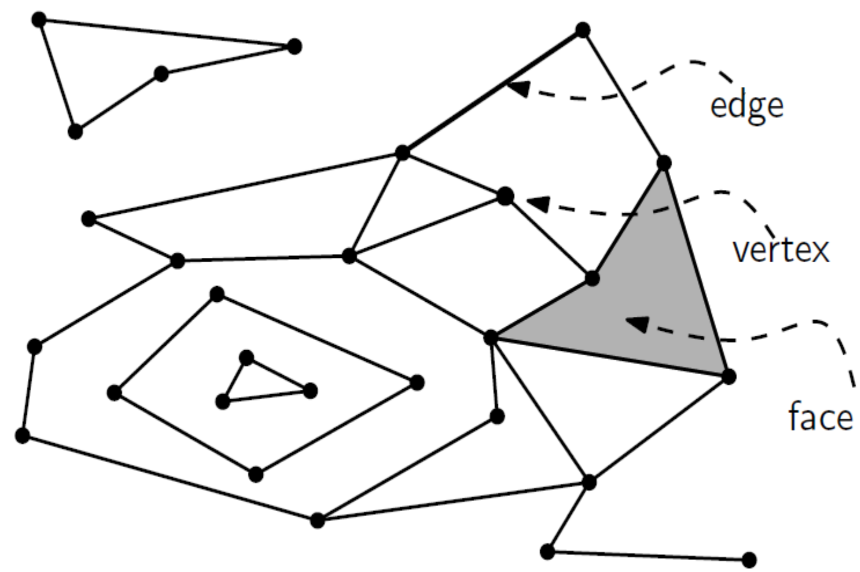
Computer Science and Engineering

Map Overlay



To solve map overlay questions, we need to represent subdivisions

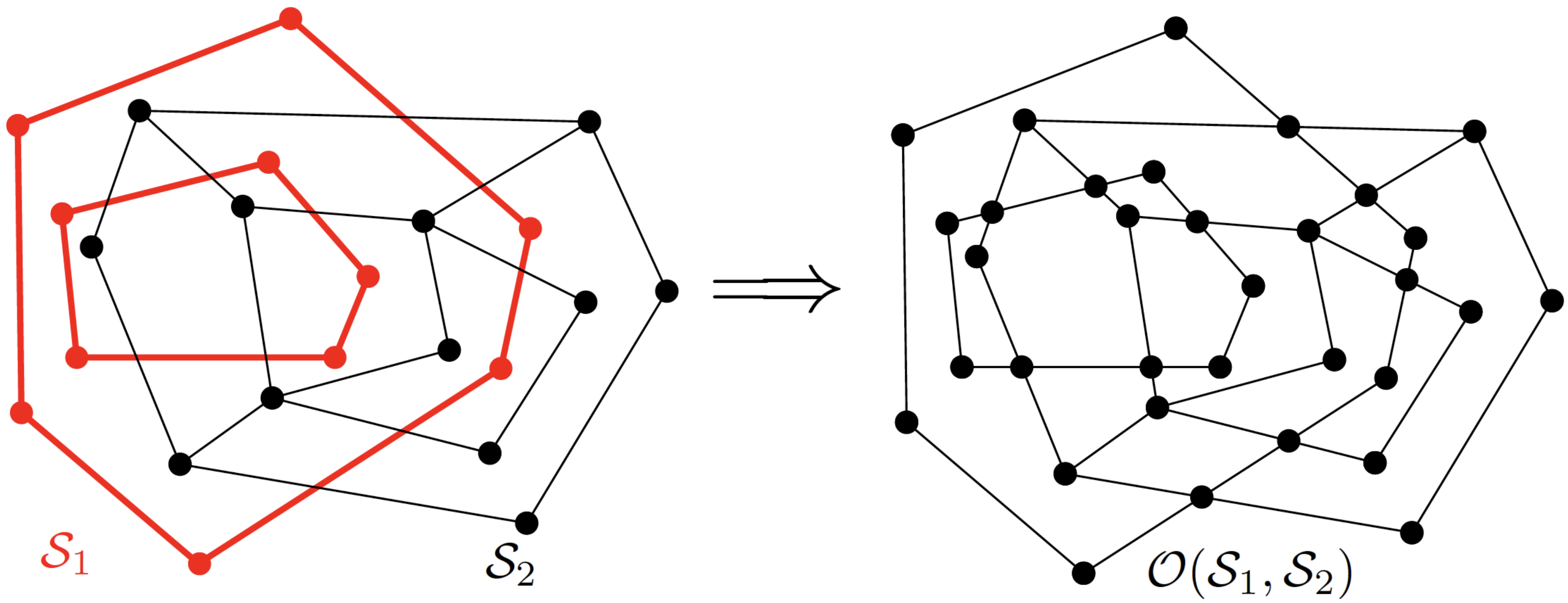
A planar subdivision is a structure induced by a set of line segments in the plane that can only intersect at common endpoints. It consists of vertices, edges, and faces



Representation: DCEL

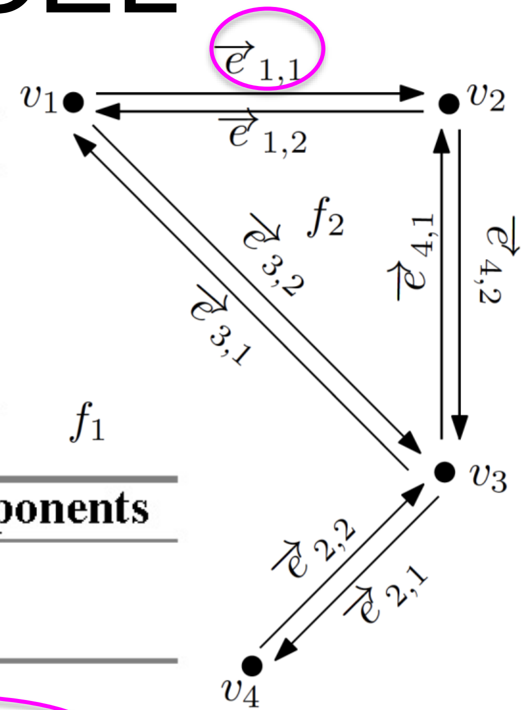
Computing the Overlay

- Input: DCEL for S_1 and DCEL for S_2
- Output: DCEL for the overlay of S_1 and S_2



Ref: 4A Textbook

Example DCEL



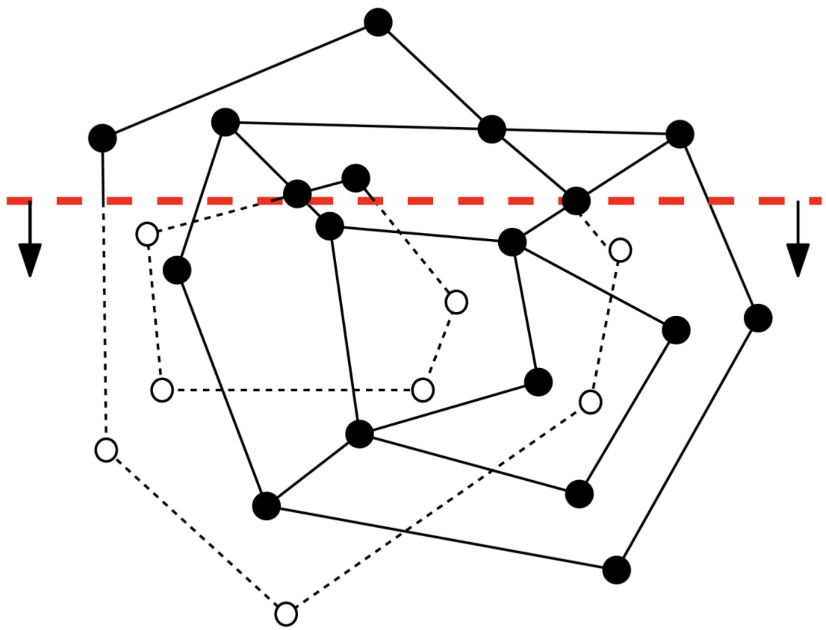
Vertex	Coordinates	IncidentEdge
v_1	$(0,4)$	$\vec{e}_{1,1}$
v_2	$(2,4)$	$\vec{e}_{4,2}$
v_3	$(2,2)$	$\vec{e}_{2,1}$
v_4	$(1,1)$	$\vec{e}_{2,2}$

Face	OuterComponent	InnerComponents
f_1	**nil**	$\vec{e}_{1,1}$
f_2	$\vec{e}_{4,1}$	**nil**

Half-edge	Origin	Twin	IncidentFace	Next	Prev
$\vec{e}_{1,1}$	v_1	$\vec{e}_{1,2}$	f_1	$\vec{e}_{4,2}$	$\vec{e}_{3,1}$
$\vec{e}_{1,2}$	v_2	$\vec{e}_{1,1}$	f_2	$\vec{e}_{3,2}$	$\vec{e}_{4,1}$
$\vec{e}_{2,1}$	v_3	$\vec{e}_{2,2}$	f_1	$\vec{e}_{2,2}$	$\vec{e}_{4,2}$
$\vec{e}_{2,2}$	v_4	$\vec{e}_{2,1}$	f_1	$\vec{e}_{3,1}$	$\vec{e}_{2,1}$
$\vec{e}_{3,1}$	v_3	$\vec{e}_{3,2}$	f_1	$\vec{e}_{1,1}$	$\vec{e}_{2,2}$
$\vec{e}_{3,2}$	v_1	$\vec{e}_{3,1}$	f_2	$\vec{e}_{4,1}$	$\vec{e}_{1,2}$
$\vec{e}_{4,1}$	v_3	$\vec{e}_{4,2}$	f_2	$\vec{e}_{1,2}$	$\vec{e}_{3,2}$
$\vec{e}_{4,2}$	v_2	$\vec{e}_{4,1}$	f_1	$\vec{e}_{2,1}$	$\vec{e}_{1,1}$

Use the plane-sweep algorithm

- Plane-sweep as in line-segment intersection



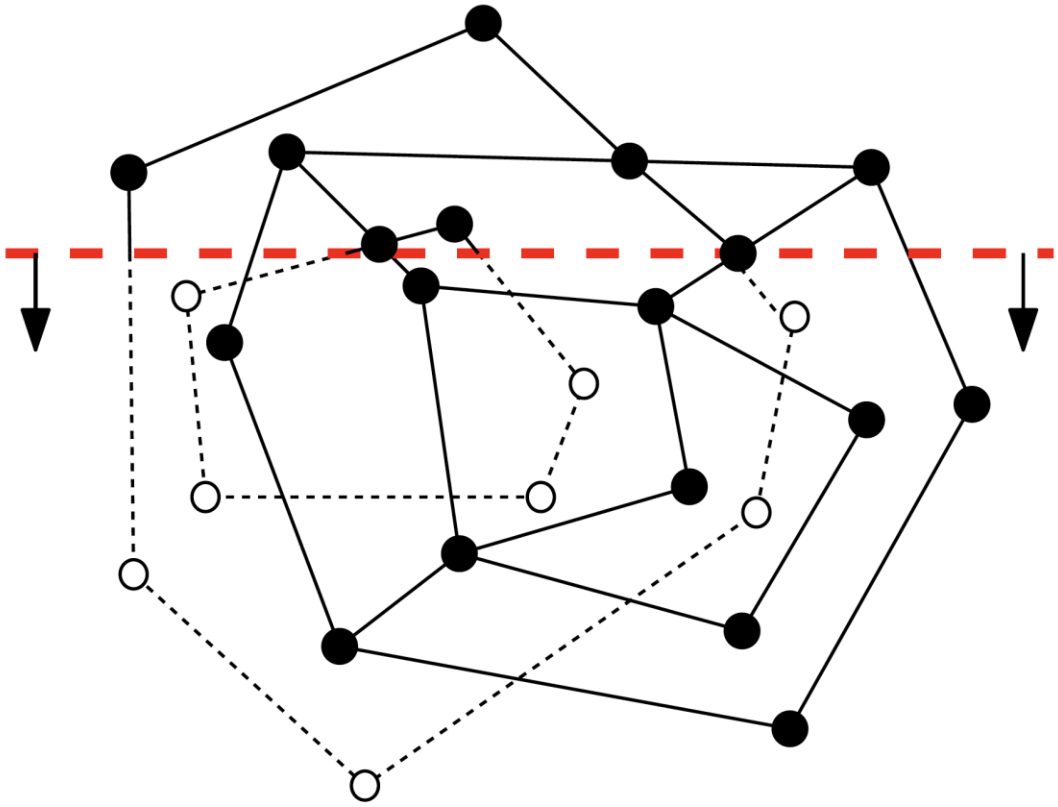
Status: the edges of S_1 and S_2 intersecting the sweep line in the left-to-right order;

Events happen:
at the vertices of S_1 and S_2;

at intersection points from S_1 and S_2

Event Management using DCEL

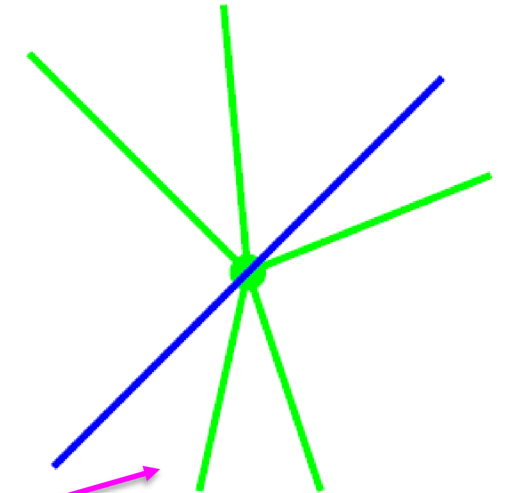
- For each intersection event, add a new vertex to the merged DCEL

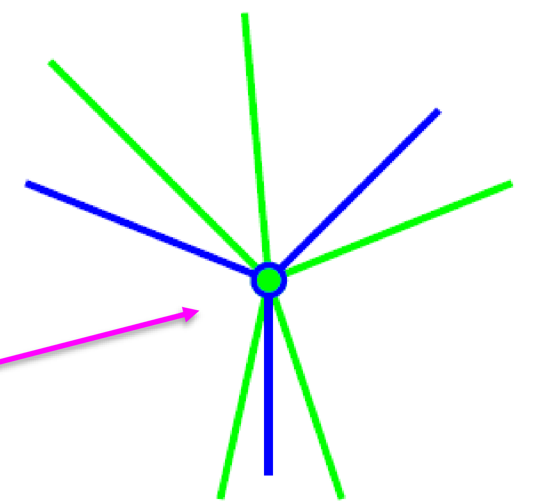


Overlay Events

Six types of events:

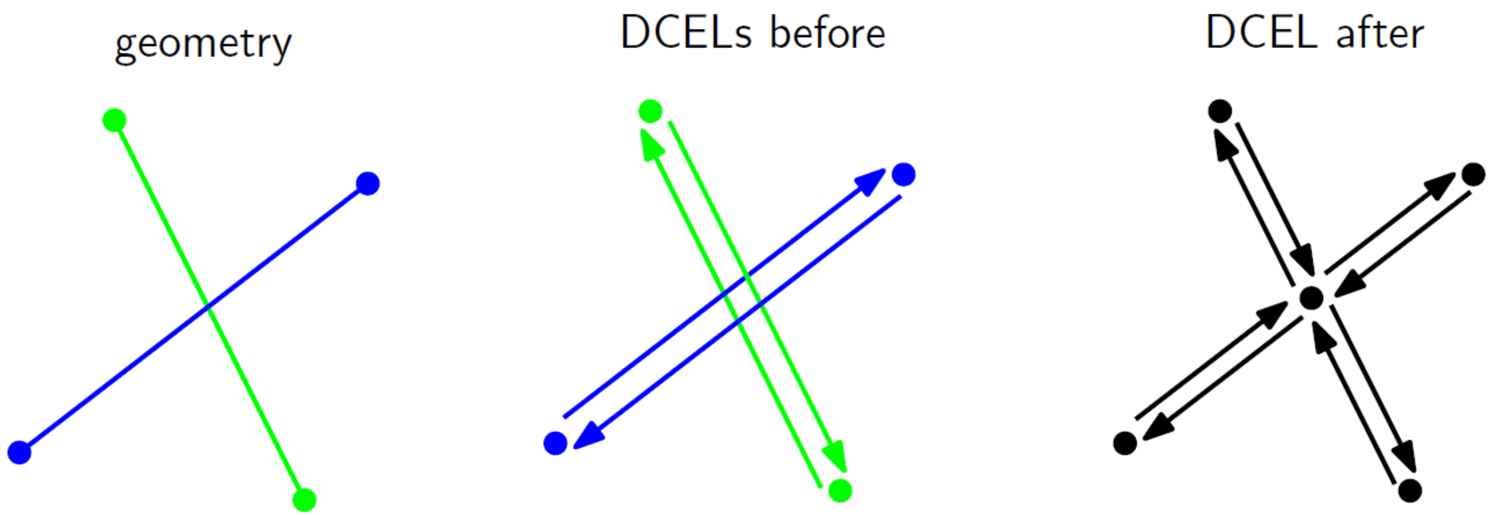
- A vertex of S_1
- A vertex of S_2
- An intersection point of one edge from S_1 and one edge from S_2
- An edge of S_1 goes through a vertex of S_2
- An edge of S_2 goes through a vertex of S_1
- A vertex of S_1 and a vertex of S_2 coincide





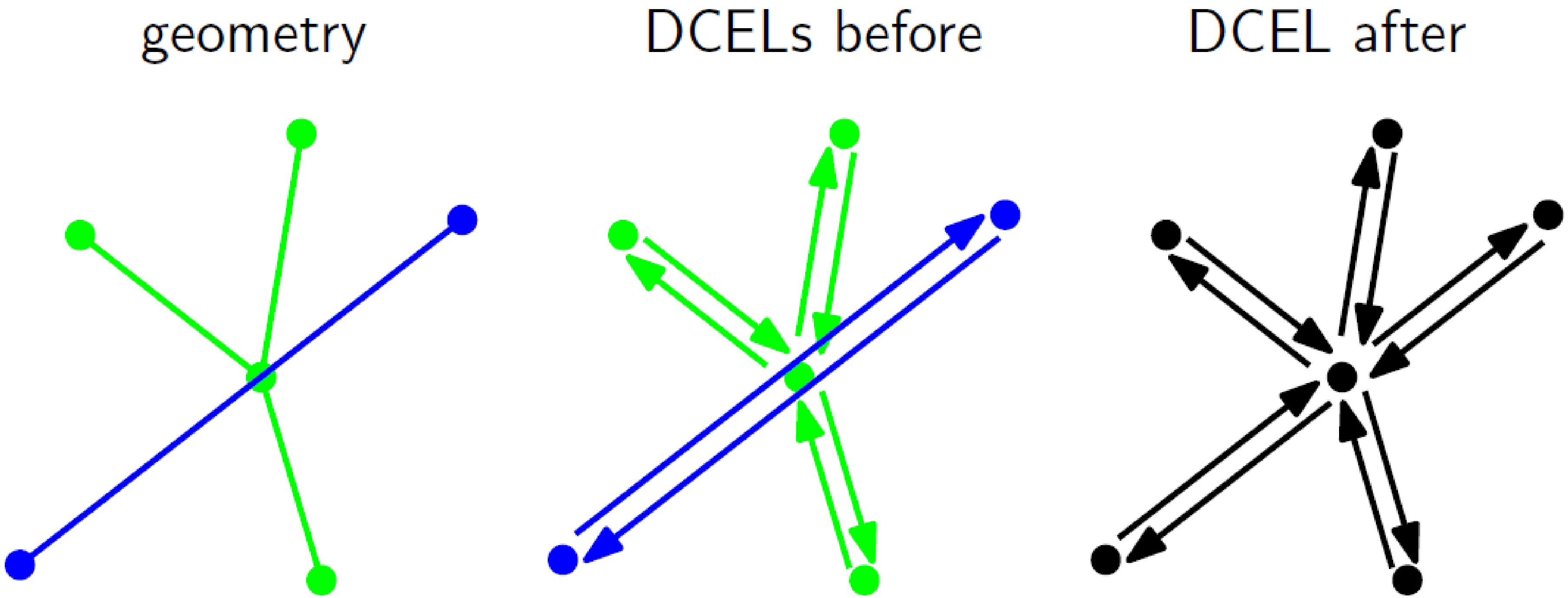
Overlay Events

Consider the event:
an intersection point of one edge from S_1 and one edge from S_2



Overlay Events

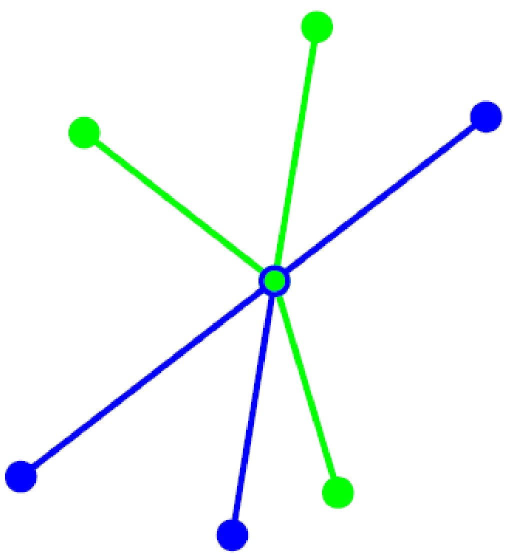
Consider the event:
an edge from S_1 goes through a vertex of S_2



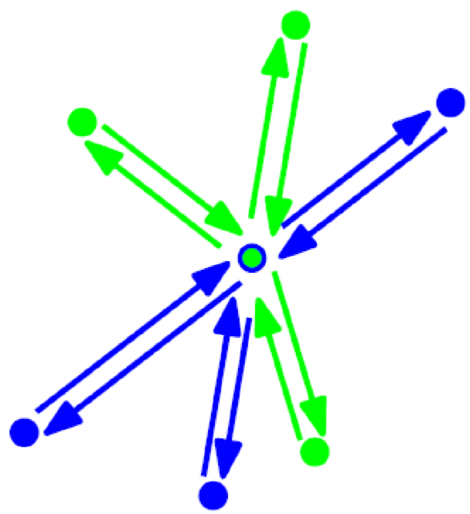
Overlay Events

Consider the event:
a vertex of S_1 and a vertex of S_2 coincide

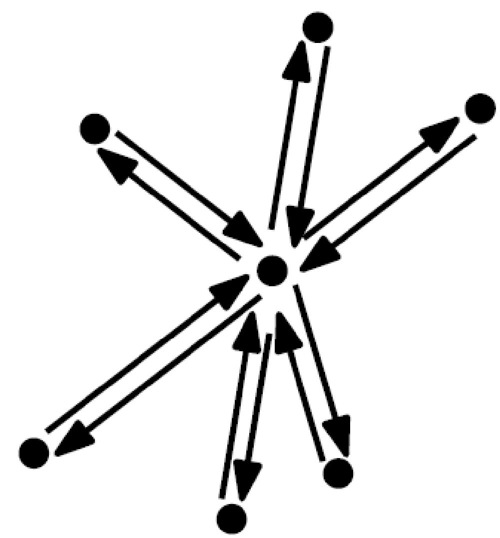
geometry



DCELs before



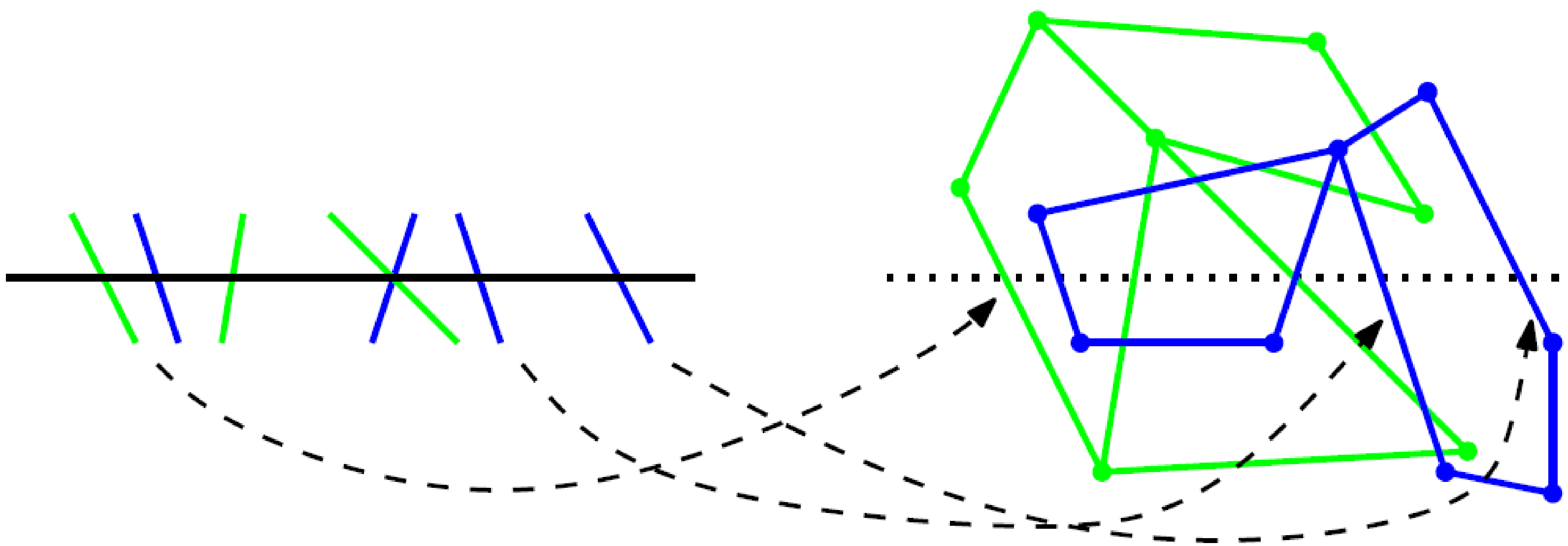
DCEL after



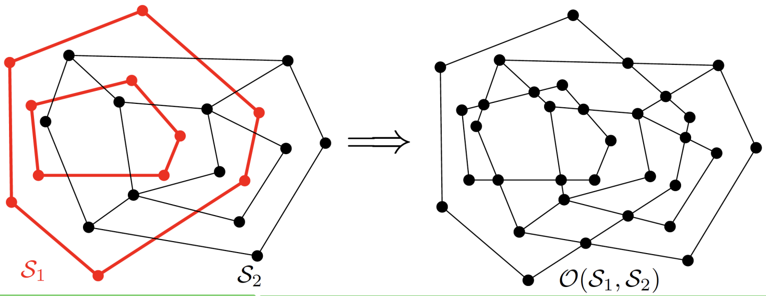
Overlay Data Structure

When we take an event from the event queue Q, we need quick access to the DCEL to make the necessary changes

We keep a pointer from each leaf in the status structure to one of the representing half-edges in the DCEL



Fixing the Incident Face for a Half-Edge in the Overlay Data Structure



A half-edge object stores:

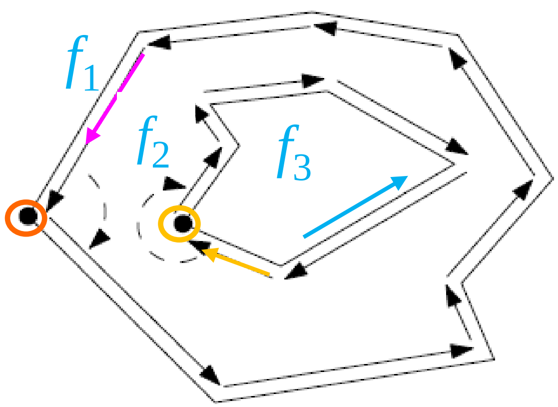
- √ **Origin** (vertex)
- √ **Twin** (half-edge)
- ? **IncidentFace** (face)
- **Next** (half-edge in cycle of the incident face)
- **Prev** (half-edge in cycle of the incident face)

faces = ? | = 1 + (#directed cycles)/2

Incident face for → f_2

Incident face for → f_3

How do we know whether a cycle is an *outer* boundary or the *inner* boundary of a face?

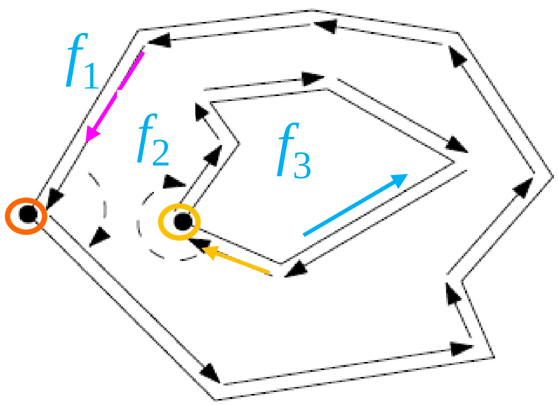


- Given a half-edge (e), determine the directed cycle following half-edges;

- Find the vertex with the least value of x (or y);

If the turning angle $< \pi$, then e is outer component of the incident face; otherwise (when $> \pi$) e is an inner component

Fixing the Incident Face for a Half-Edge in the Overlay Data Structure



A half-edge object stores:

- **Origin** (vertex)
- **Twin** (half-edge)
- ? **IncidentFace** (face)
- **Next** (half-edge in cycle of the incident face)
- **Prev** (half-edge in cycle of the incident face)

How do we know whether a cycle is an *outer* boundary or the *inner* boundary of a face?

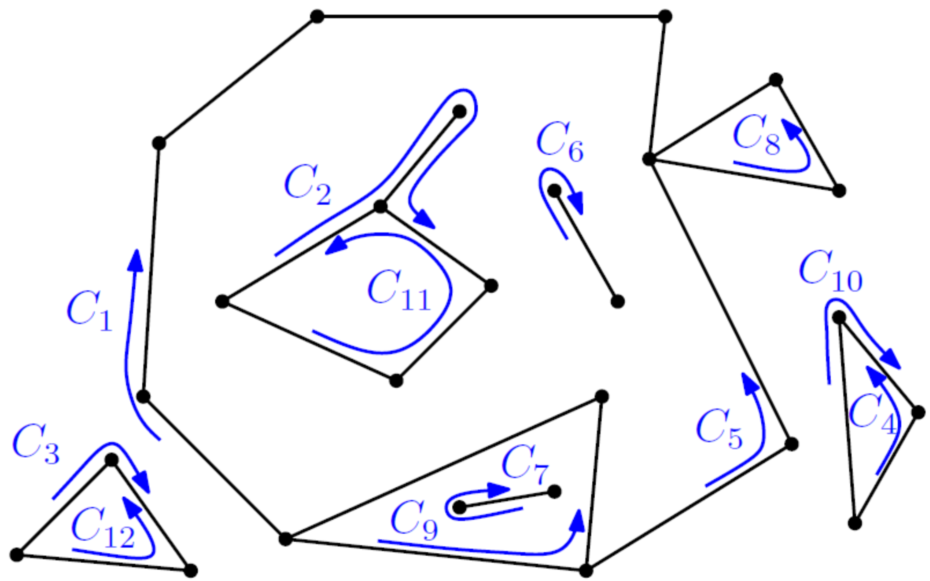
How do we know whether the incident face for the half-edge is same as the incident face of the half-edge ? Both are f_2

For each cycle, note the leftmost and rightmost x-values. Move a vertical sweep-line from L $\rightarrow$ R; Use orientation test to decide whether an inner vertex lies within the cycle; if so merge the label of the face defined by the inner boundary; finish processing this face when the rightmost vertex of this cycle is crossed

Overlay Face Updating

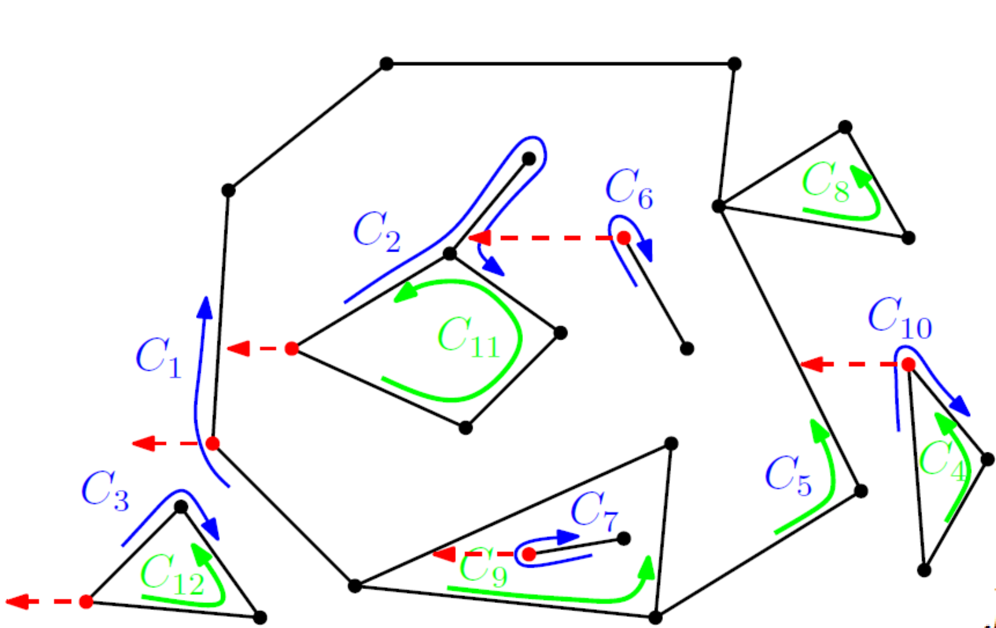
A face object stores:

- **OuterComponent** (half-edge of outer cycle)
- **InnerComponents** (half-edges for the inner cycles)



- Determine all cycles of half-edges, and whether they are inner or outer boundaries of the incident face
- Make a face object for each outer boundary, plus one for the unbounded face, and set the **OuterComponent** variable of each face. Set the **IncidentFace** variable for every half-edge in an outer boundary cycle

Overlay Face Updating



$C_1 \to$

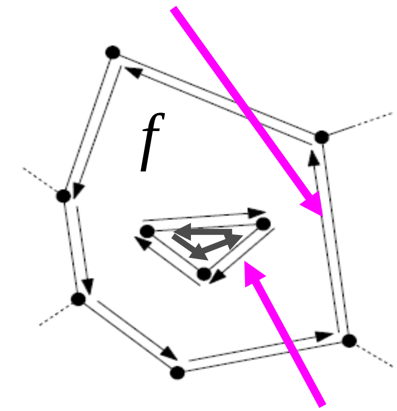
$C_3 \to$

$C_2 \to C_5$

$C_7 \to C_9$

$C_6 \to C_2$

$C_{10} \to C_1$



f_5: outer e_5; inner e_2, e_6

Determine the leftmost vertex of each inner boundary cycle;
Determine the edge horizontally left of it, take the downward half-edge; the **InnerComponents** of the corresponding face is set;
Set **IncidentFace** for half-edges accordingly;

Ref: 4A Textbook

Analysis

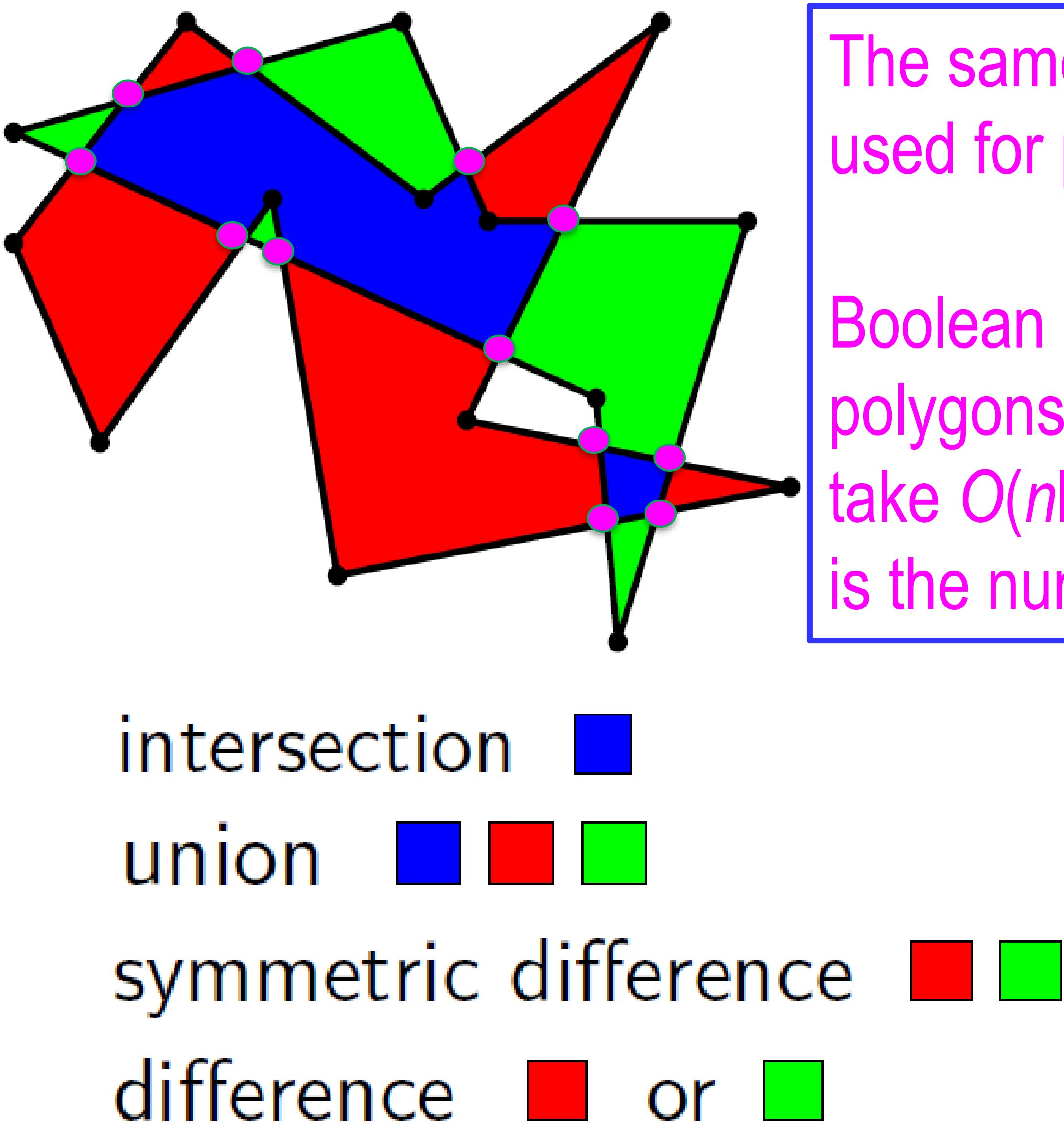
n: sum of the complexities of input DCELs

Every event takes $O(\log n)$ or $O(m + \log n)$ time to handle, where m is the sum of the degrees of any vertex from S_1 and/or S_2 involved

The sum of the degrees of all vertices is exactly twice the number of edges

Theorem: Given two planar subdivisions S_1 and S_2, their overlay can be computed in $O(n \log n + k \log n)$ time, where k is the number of vertices of the overlay

Boolean Operations on Polygons



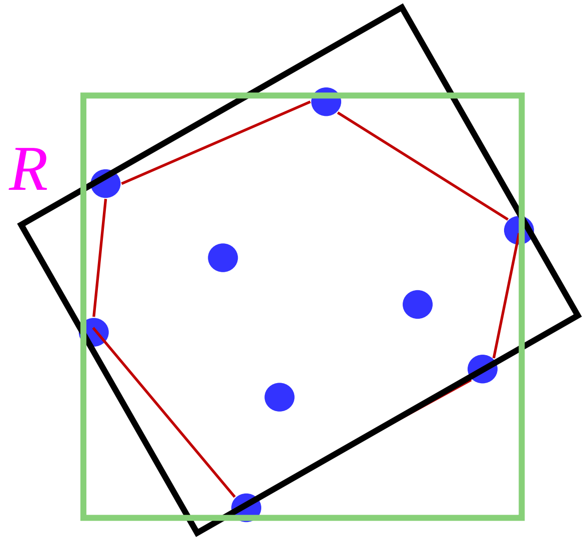
The same overlay algorithm can be used for polygon operations;

Boolean operations on two polygons with a total of *n* vertices take $O(n\log n + k\log n)$ time, where *k* is the number of intersection points

Ref: 4A Textbook

Problem of the Day

Problem: Given a set *P* of *n* points on the 2D plane, find the minimum-area rectangle *R* that encloses *P*



Claim: R must touch at least one of the edges of the convex hull of *P*

CS60064 Spring 2022

Computational Geometry

Instructors

Bhargab B. Bhattacharya (BBB)
Partha Bhowmick (PB)

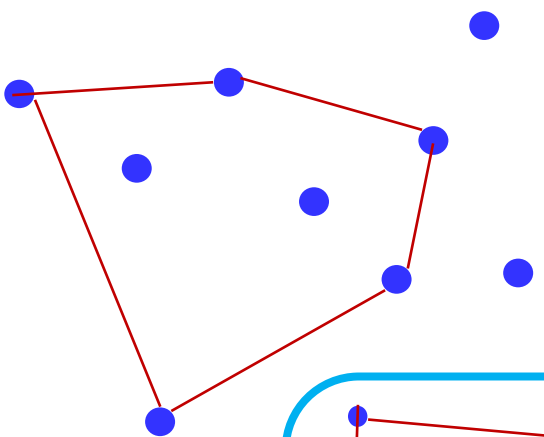
Lecture #16 & Lecture #17

11 February 2022

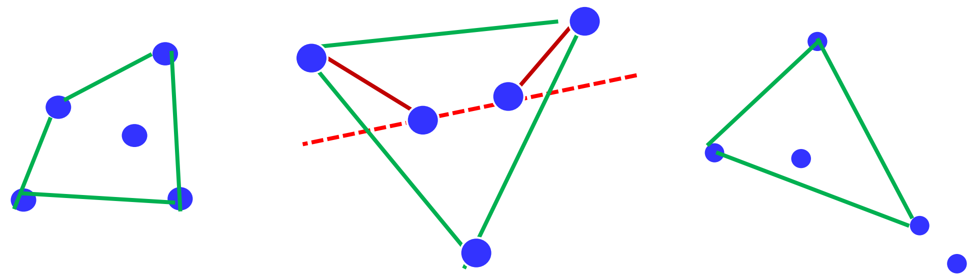
Indian Institute of Technology Kharagpur
Computer Science and Engineering

Problem of the Day

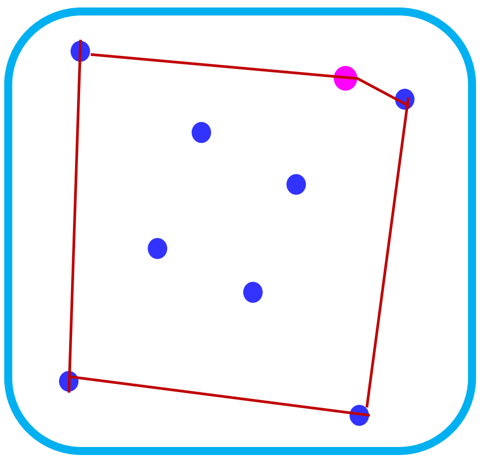
Problem: Given nine random points on the plane in general positions, show that there always exists a convex pentagon!



1933: Budapest, Hungary
Esther Klein gave a puzzle to George Szekeres and Paul Erdős



$f(5) > 8$;



$f(3) = 3$; $f(4) = 5$; $f(5) = 9$;

2006: Szekeres and Peters: $f(6) = 17$

The value of $f(N)$ is still unknown for all $N > 6$

1935 Erdős-Szekeres: $f(n) \leq 4^{n - o(n)}$

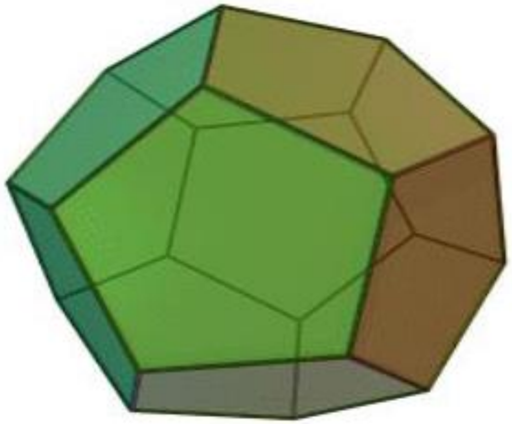
1960 Erdős-Szekeres Conjecture: $f(n) \geq 2^{(n-2)} + 1$

2016: Andrew Suk: $f(n) = 2^{n + 6n^{(2/3)} \log n}$

$f(n)$: the smallest number of points, an arrangement of which always contains a convex n-gon

Erdős-Szekeres Problem of Convex Polygons

Agenda

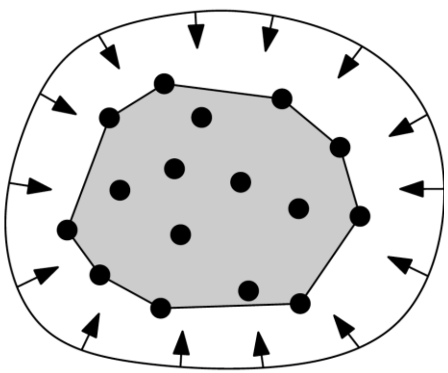


Convex Hulls: Management of Corners

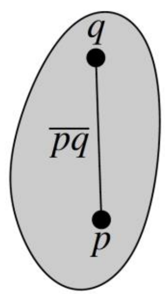
At any street corner the feeling of absurdity can strike any man in the face.

-- *Albert Camus,* The Myth of Sisyphus (1942)

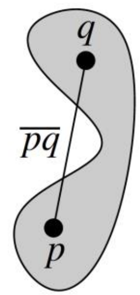
Convex hull is like sorting to computational geometry, a first step to apply to unstructured data.
The *convex hull* $C(S)$ of a set of points S is the smallest convex polygon containing S.



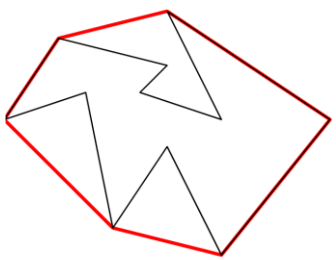
Convex hull of a point-set



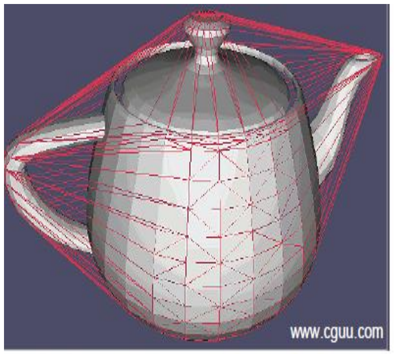
convex



not convex



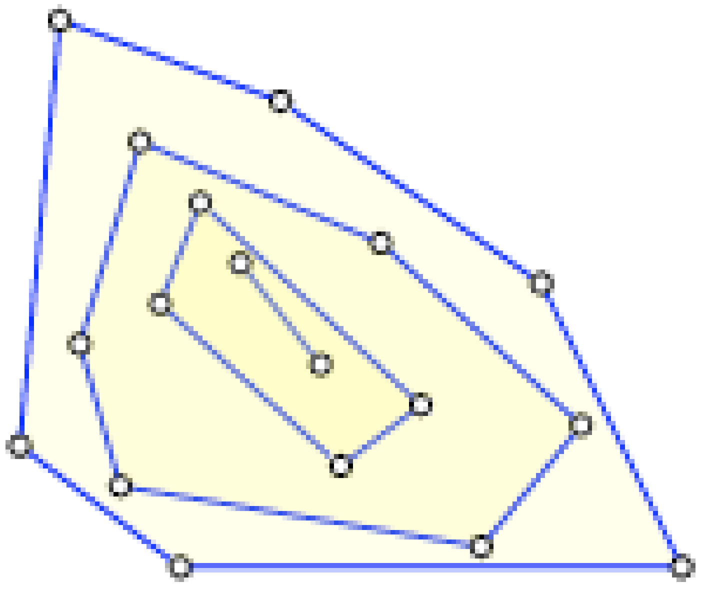
Convex hull of a polygon



3D hull of an object

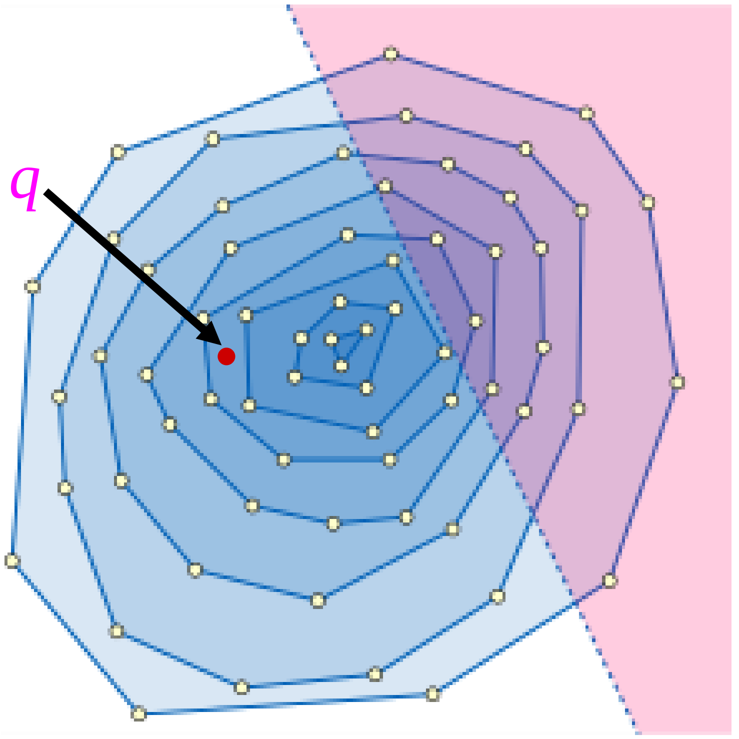
- A set $C \subseteq \mathbf{R}^2$ is *convex* if for every two points $p,q \in C$, the line segment $\overline{pq}$ is fully contained in C

Convex Layering: Onion Peeling



Data depth statistics

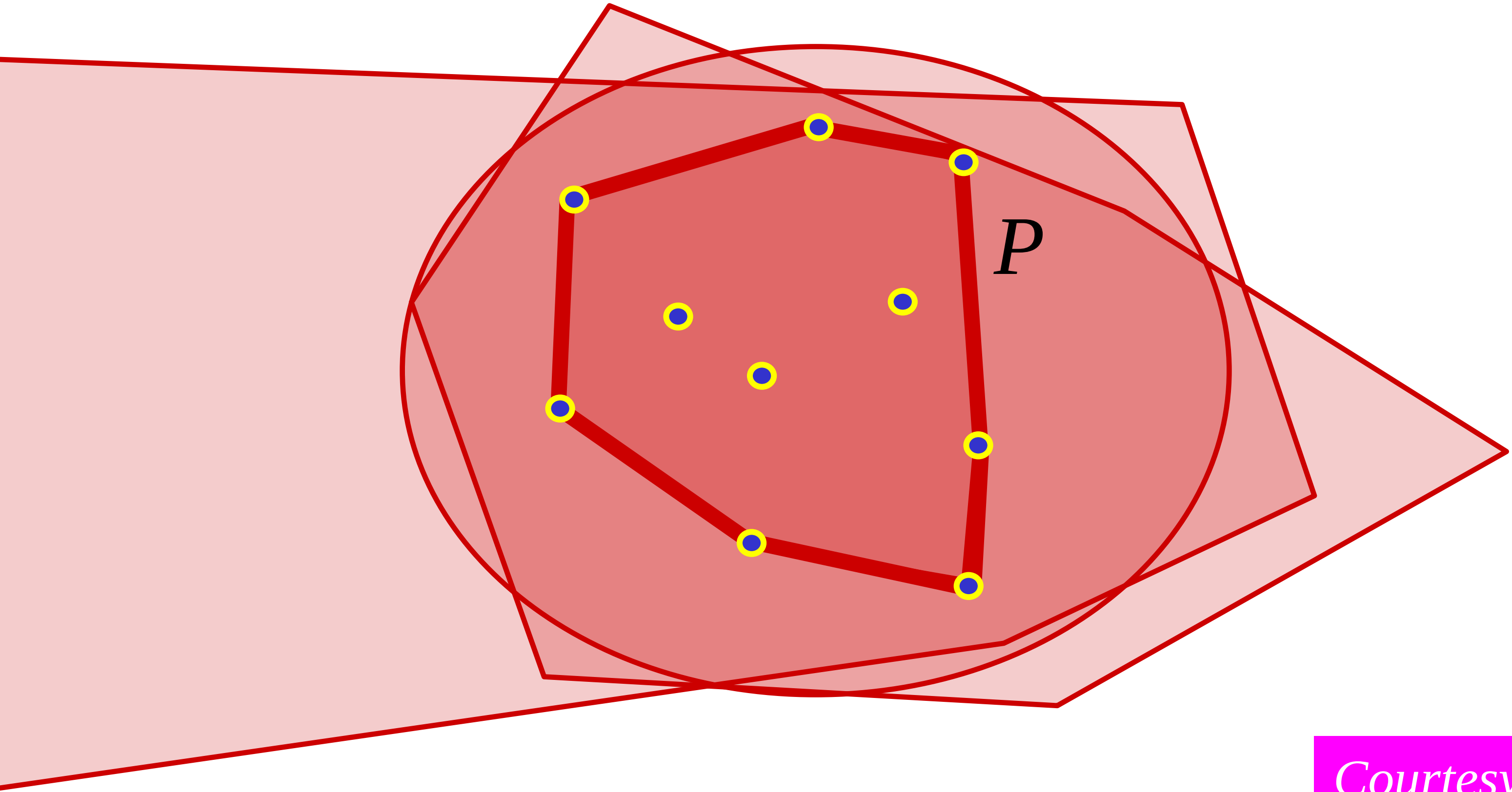
Concentric hulls



data-depth(q) = 5

Convex hull as intersection of convex supersets

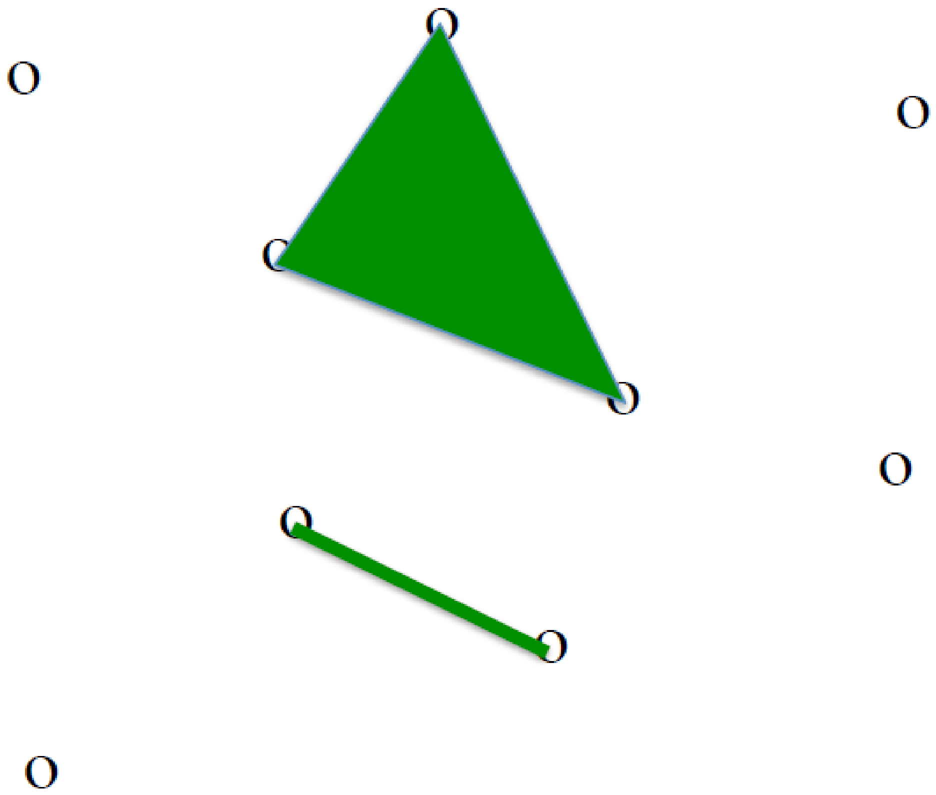
- The convex hull $CH(P)$ of a point set $P \subseteq \mathbf{R}^2$ is the smallest convex set $C \supseteq P$. In other words $\mathrm{CH(P)} = \bigcap_{\substack{C \supseteq P \\ C \text{ convex}}} \mathrm{C}$



Courtesy: Carola Wenk

Convex hull as union of convex subsets

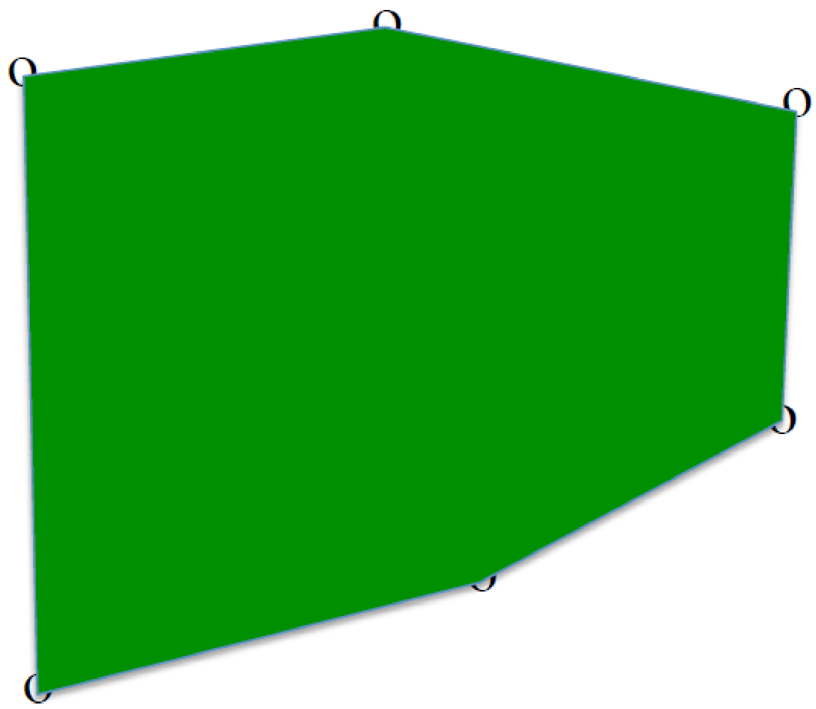
Let S be a set of n points on the plane



convex combinations

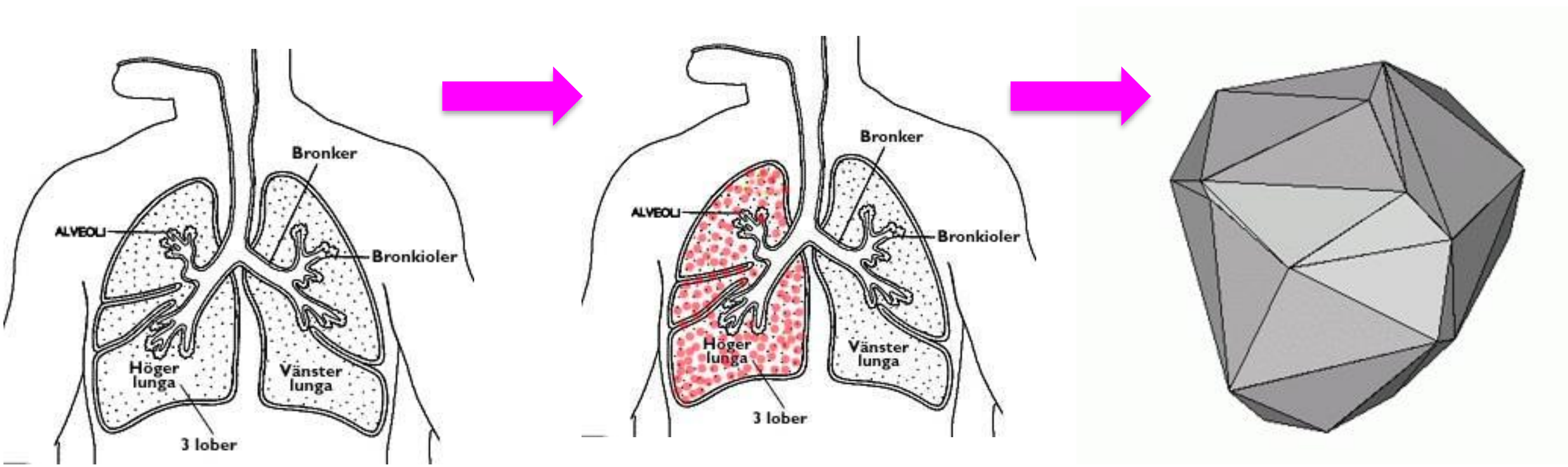
Convex hull as union of convex subsets

Let S be a set of n points on the plane



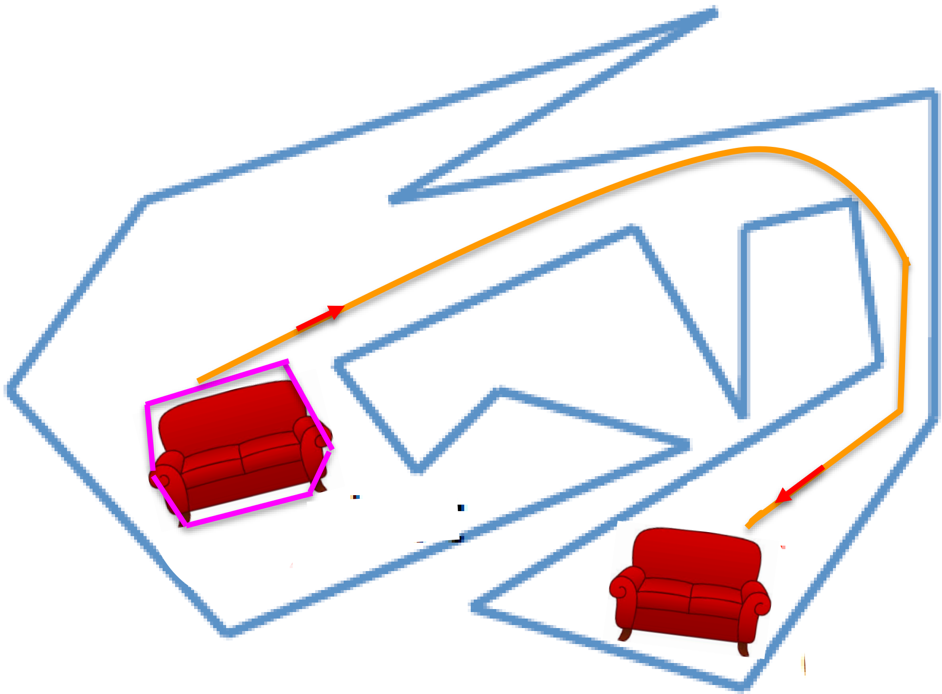
convex hull =
union of all
convex combinations

Convex Hull in Biomedical Applications



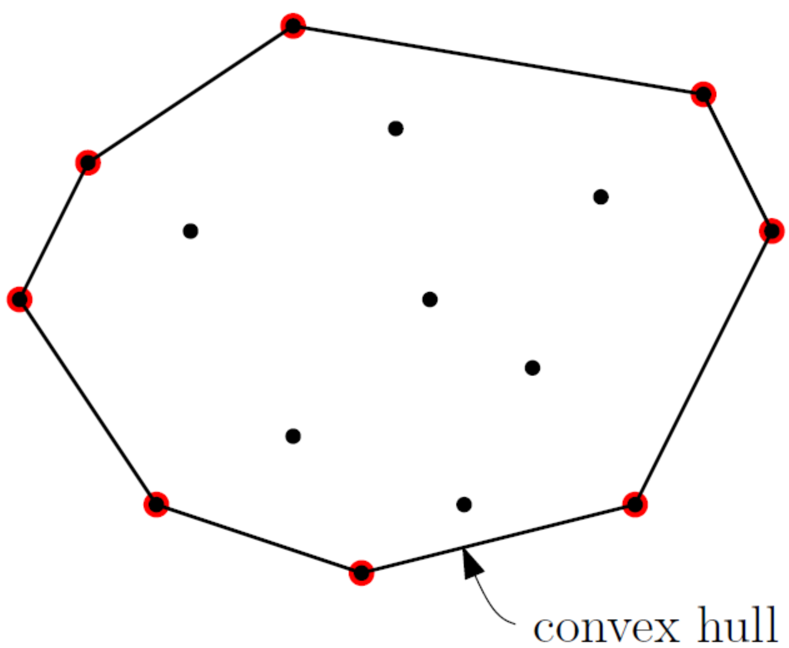
Patients are asked to inhale gaseous nano-bots $\Rightarrow$ these fluorescent sensors provide their 3D coordinates via imaging $\Rightarrow$ compute 3D convex hull $\Rightarrow$
volume of hull $\propto$ lung size!

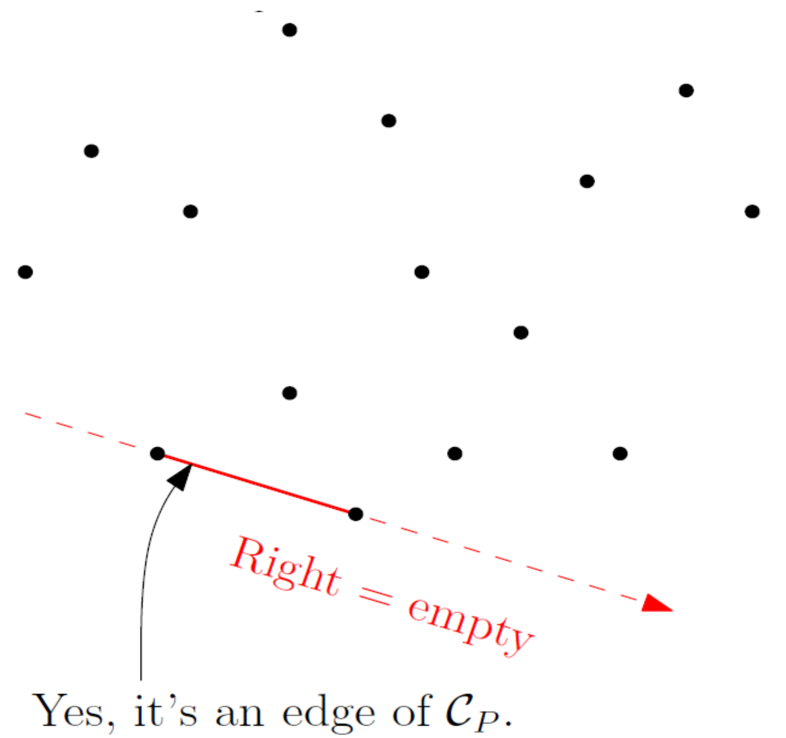
Moving a Sofa Through a Corridor



Sufficiency: If the convex hull of the sofa avoids collision with obstacles, so does it $\Rightarrow$ robot movement avoiding obstacles

Convex Hulls





Convex Hull

Let S be a set of n points on the plane

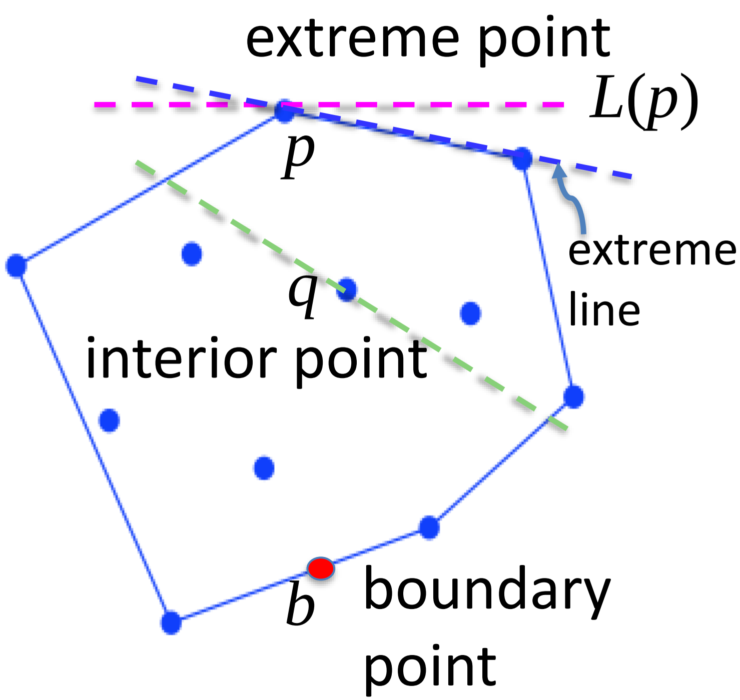
A point p is an extreme point if $\exists$ a line $L(p)$ through p such that all the remaining points strictly lie on one side of $L(p)$;

An extreme line passes through two points such that the remaining points lie on one side

A point q is an interior point if any line through q splits the point set

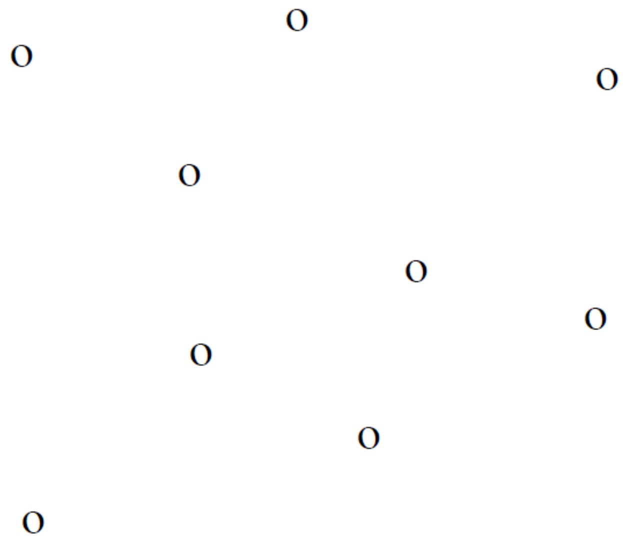
A point b is a boundary point if it is not an extreme or an interior point

The vertices of Convex_Hull(S) comprises only the extreme points of S; its edges coincide with the extreme lines only



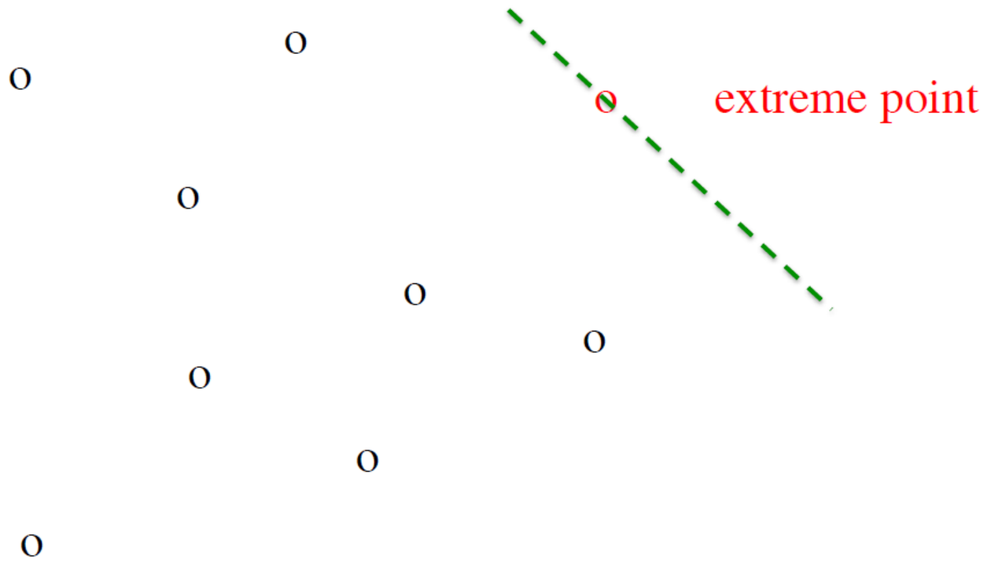
Convex Hull

Let S be a set of n points on the plane



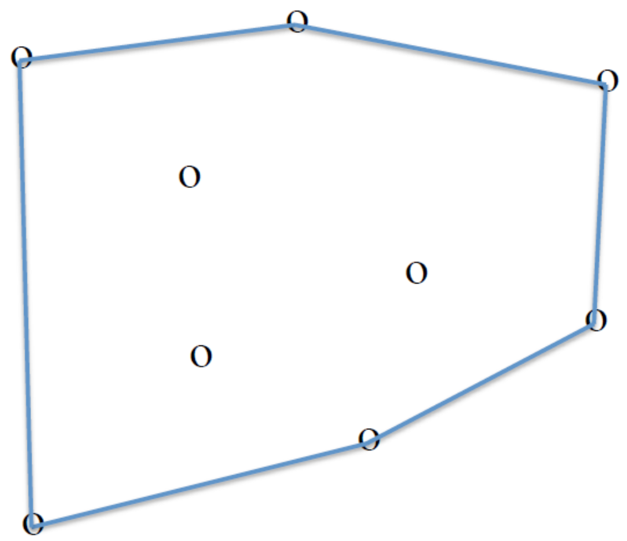
Convex Hull

Let S be a set of n points on the plane



Convex Hull

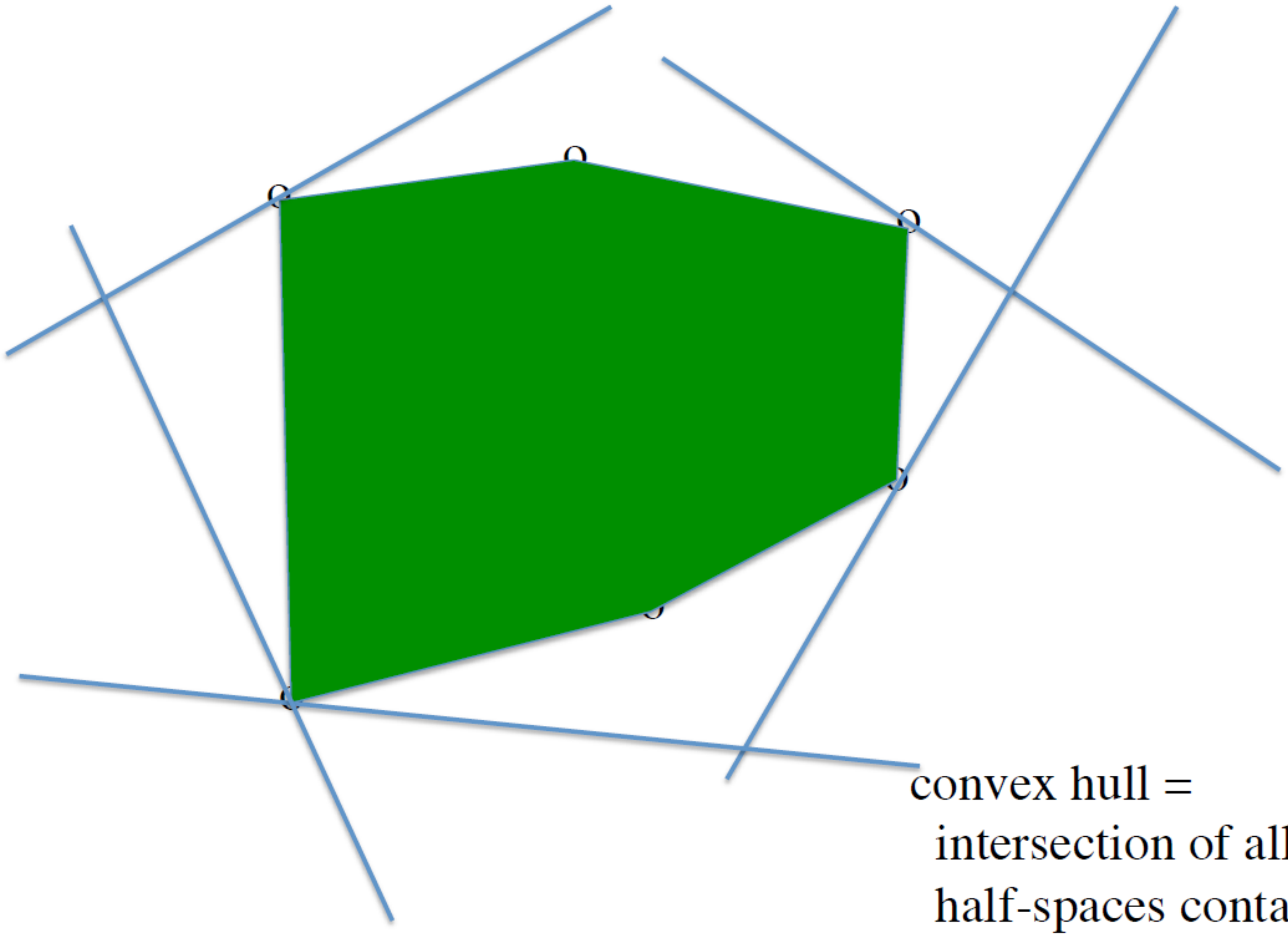
Let S be a set of n points on the plane



convex-hull: polygon whose vertices are extreme points

Convex Hull

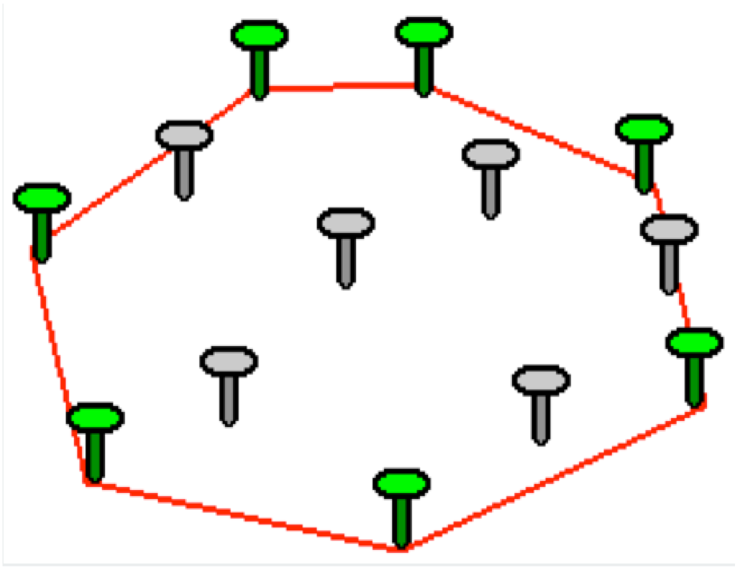
Let S be a set of n points on the plane



convex hull =
intersection of all
half-spaces containing S

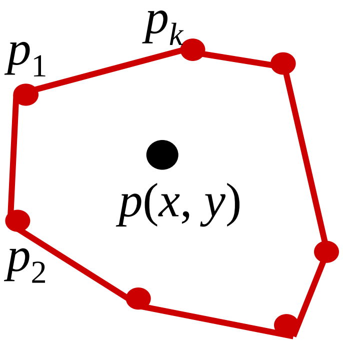
Convex Hull: Mechanical Analogy

Hammer nails on the points perpendicular to plane;
Stretch elastic rubber band to surround them tightly



- Shortest (perimeter) fence surrounding the points
- Smallest (area) convex polygon enclosing the points

Properties of Convex Hull



1. Convex combination of points p, q is any point that can be expressed as $(1 - \alpha)\, p + \alpha q$, where $0 \le \alpha \le 1$

2. $CH(S)$ is union of all convex combinations of S.

3. S convex iff for all $x, y \in S$, $\overline{xy} \in S$.

4. $CH(S)$ is intersection of all convex sets containing S.

5. $CH(S)$ is intersection of all halfspaces containing S.

6. $CH(S)$ is smallest convex set containing S.

7. In R^2, $CH(S)$ is smallest area (perimeter) convex polygon containing S.

8. In R^2, $CH(S)$ is union of all triangles formed by triples of S.

Convex Hull: Easy Try

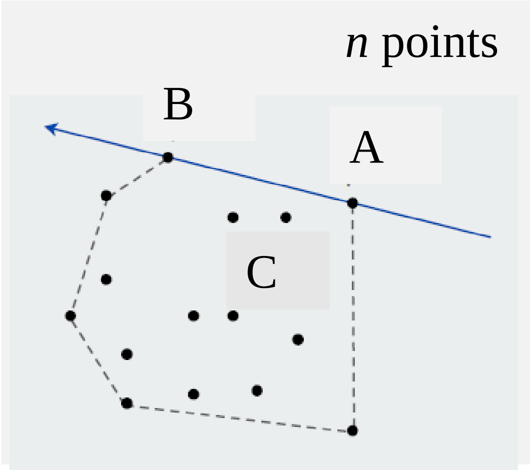
Orientation Test: $D = \begin{Vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{Vmatrix}$

Observation 1:

Edges of convex hull of *P* connect pairs of points in *P*

Observation 2:

- AB is an edge of the convex hull iff *Orient* (*A*, *B*, *C*) have the same sign for all other points C
  - This gives us a simple algorithm



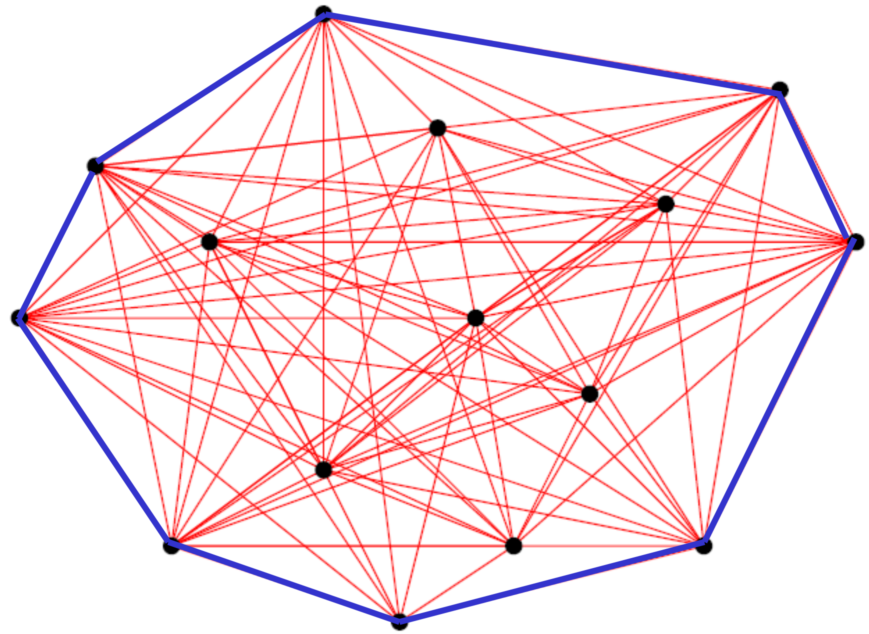
- For each A and B:
  if *Orient* (*A*, *B*, *C*) > 0 for all $C \neq A, B$:
  - Record the edge $A \to B$
- Walk along the recorded edges to recover the convex hull

Complexity: $O(n^3)$

Convex Hull: Easy Try

- AB is an edge of the convex hull iff *Orient* (*A*, *B*, *C*) have the same sign for all other points C
  - This gives us a simple algorithm

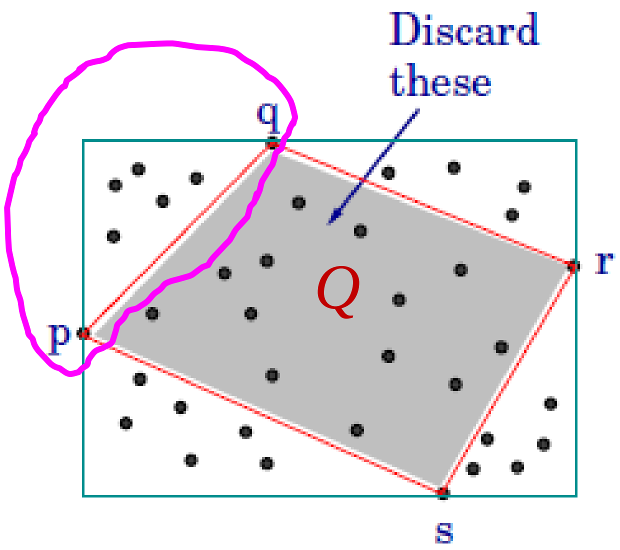
Retain only those edges that pass the unilateral Orientation Test



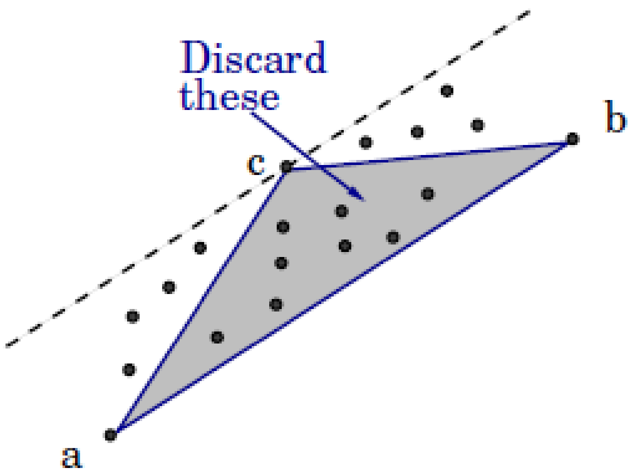
$|P| = n \Rightarrow O(n^2)$ pairs $\Rightarrow O(n^3)$ time

Improved Technique: Quick Hull Algorithm

Find four extremal points (leftmost, rightmost, topmost, bottom-most) to define *Q*



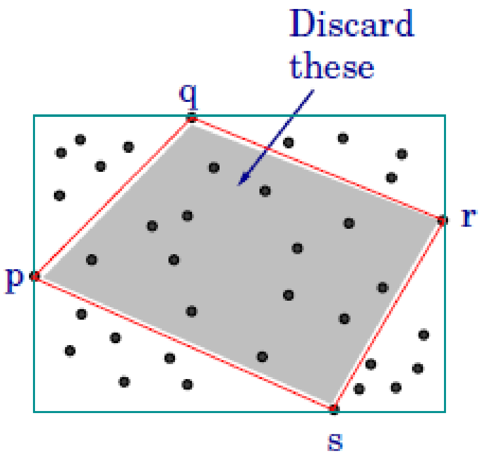
Initialization



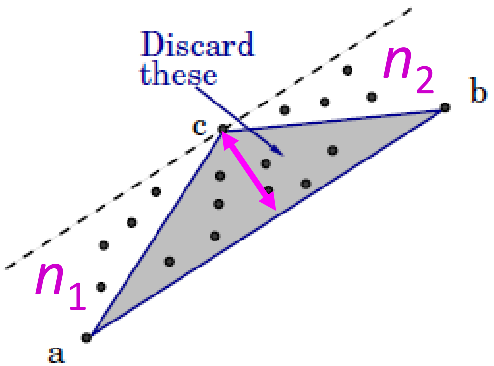
Recursive Elimination

1. Form initial quadrilateral Q, with left, right, top, bottom; they will be hull vertices. discard points inside Q
2. Recursively, a convex polygon, with some points “outside” each edge
3. For an edge *ab*, find the farthest outside point *c*; c will be a hull vertex; discard points inside triangle *abc*
4. Split remaining points into “outside” points for *ac* and *bc*
5. Edge *ab* on CH when no point outside

Analysis: Quick Hull Algorithm



Initialization



Recursive Elimination

1. Initial quadrilateral phase takes $O(n)$ time

2. $T(n)$: time to solve the problem for an edge with n points outside

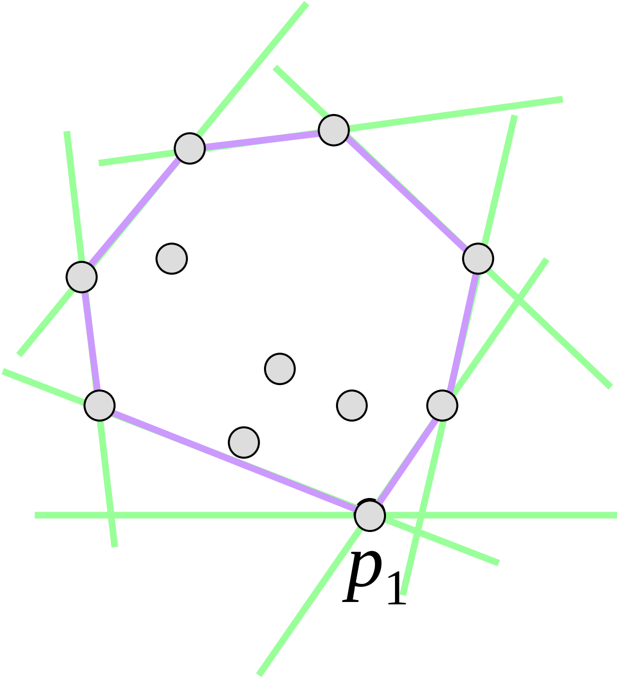
3. Let n_1, n_2 be sizes of subproblems. Then,

$$T(n) = \left\{ \begin{array}{ll} 1 & \textbf{if } n = 1 \\ n + T(n_1) + T(n_2) & \textbf{where } n_1 + n_2 \leq n \end{array} \right\}$$

4. Analogous to QuickSort; Likewise, this has expected running time complexity $O(n \log n)$, but worst-case time $O(n^2)$

Gift Wrapping: Jarvis March

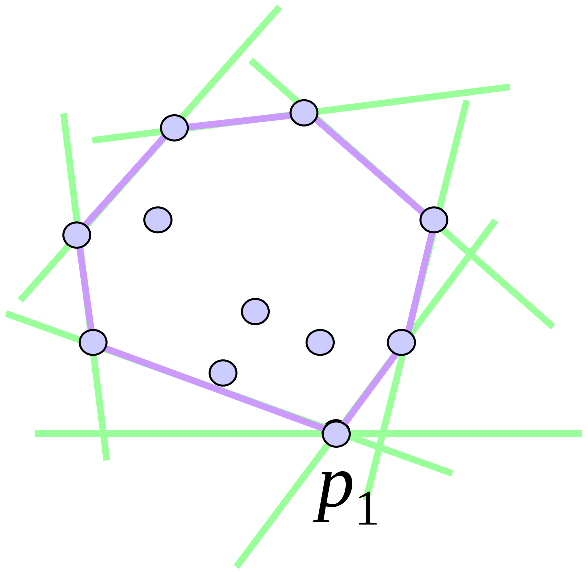
- Main Idea
  - Find a point p_1 on the convex hull (e.g. the lowest point)
  - Rotate counter-clockwise a line through p_1 until it touches one of the other points (start from a horizontal orientation)
  - Repeat until you reach p_1 again
  - Analogous to selection sort



Gift Wrapping: Jarvis March

Implementation

- Compute angle between current line and all remaining points;
- Pick the vertex with smallest angle
- Repeat until you return to the initial point



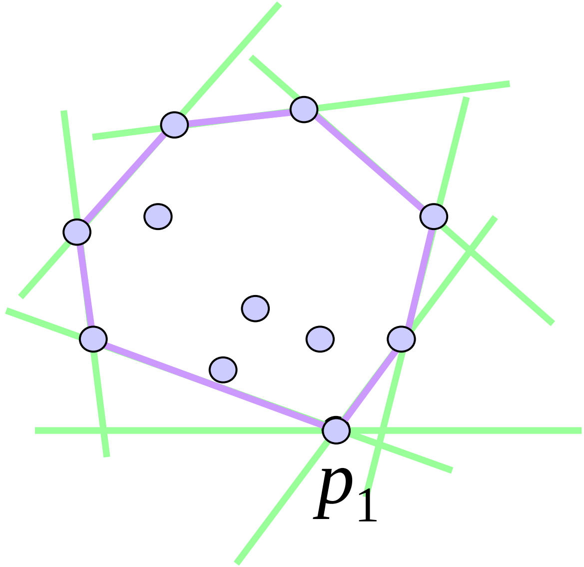
Time complexity $O(n.h)$, where n is the input size and h is the number of vertices on the convex hull (output sensitive); $3 \leq h \leq n$; Worst case $O(n^2)$.

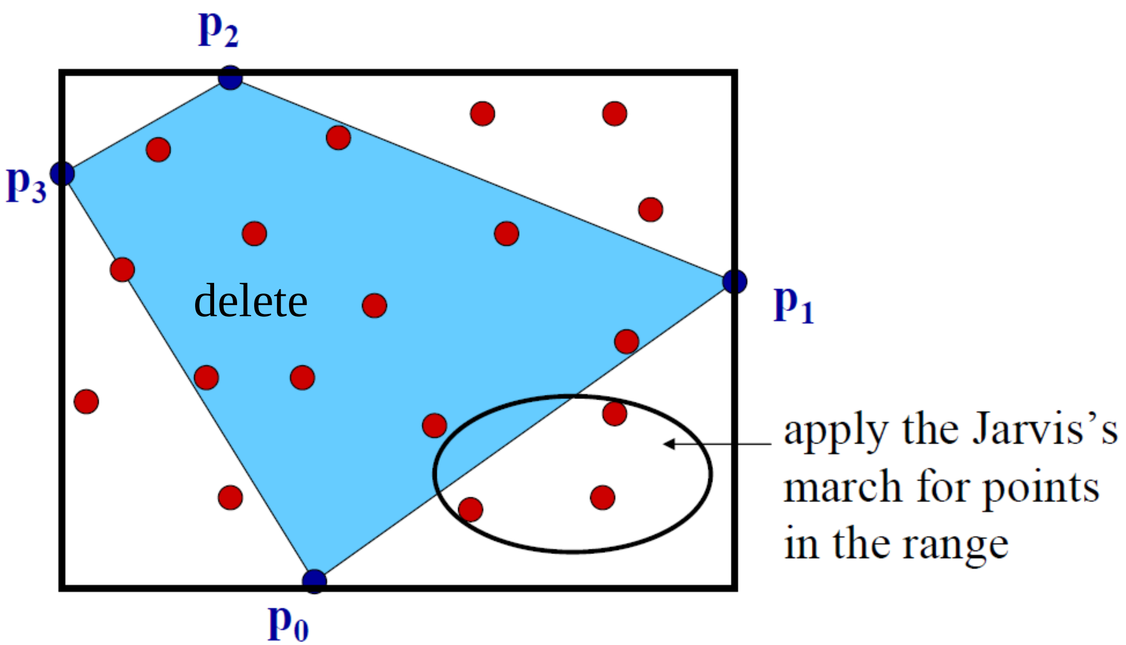
Space: $O(n)$

Gift Wrapping: Jarvis March

Implementation

- Compute angle between current line and all remaining points;
- Pick the vertex with smallest angle
- Repeat until you return to the initial point

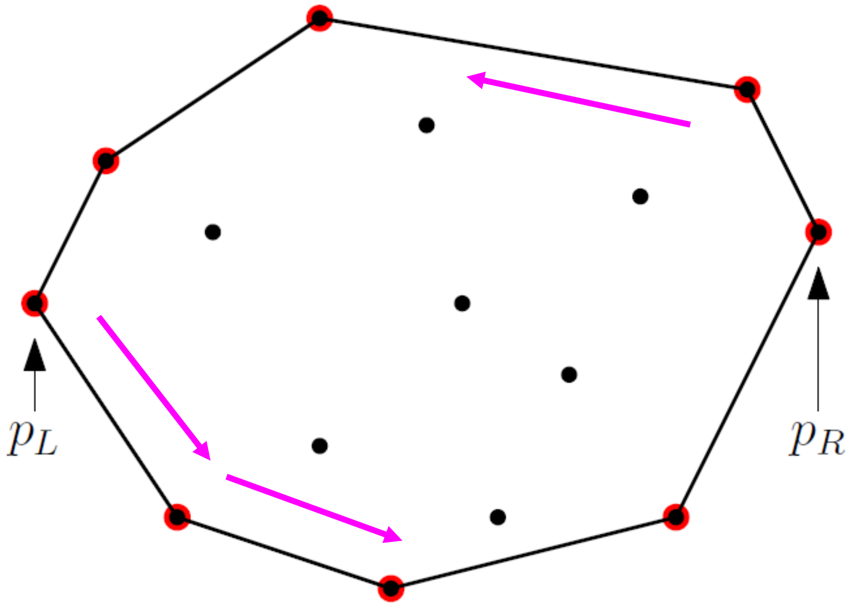




Some pre-processing idea that is useful for practical implementation

Faster Algorithm: Graham Scan

Ron Graham



Based on the basic idea of "orientation":

When we travel along the boundary of convex polygon in CCW (CW), we always take left (right) turn!

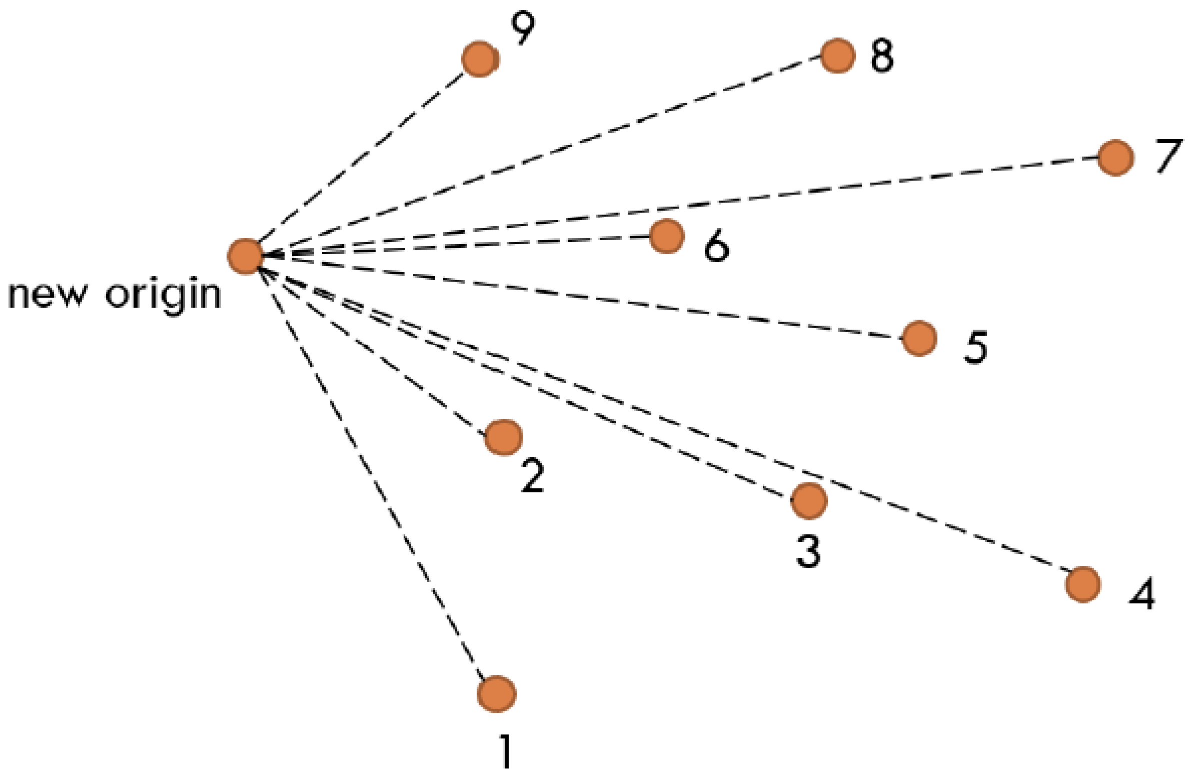
Faster Algorithm: Graham Scan

- We know that the leftmost given point has to be in the convex hull
- Fix the origin at the leftmost point (assume unique)
- All other points have positive *x* coordinates
- Sort the points in increasing order of *y/x,* i.,e., angular sorting
- Incrementally construct the convex hull using a stack and Orientation test

Complexity: $O(n \log n)$

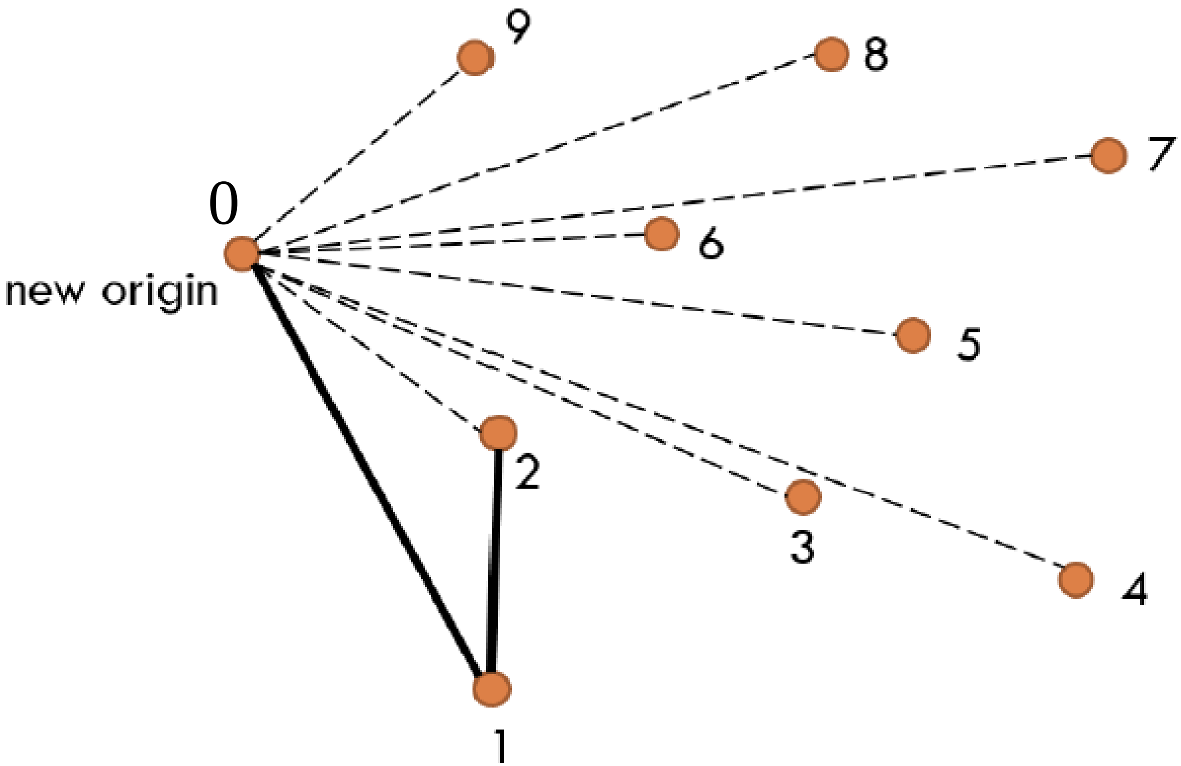
Graham Scan

Points are numbered in increasing order of *y/x* (angular sorting)



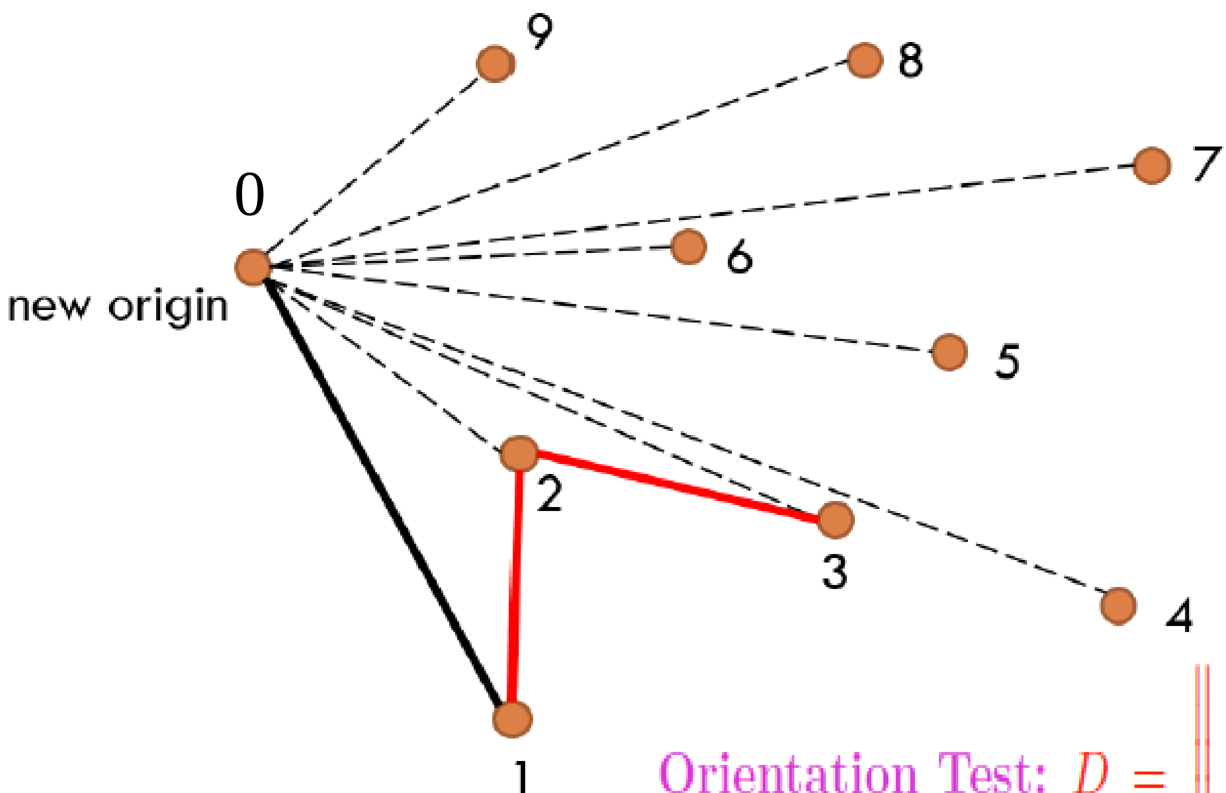
Graham Scan

start from the origin and add first two points (0, 1) on the running hull; this edge will always be on the hull



Graham Scan

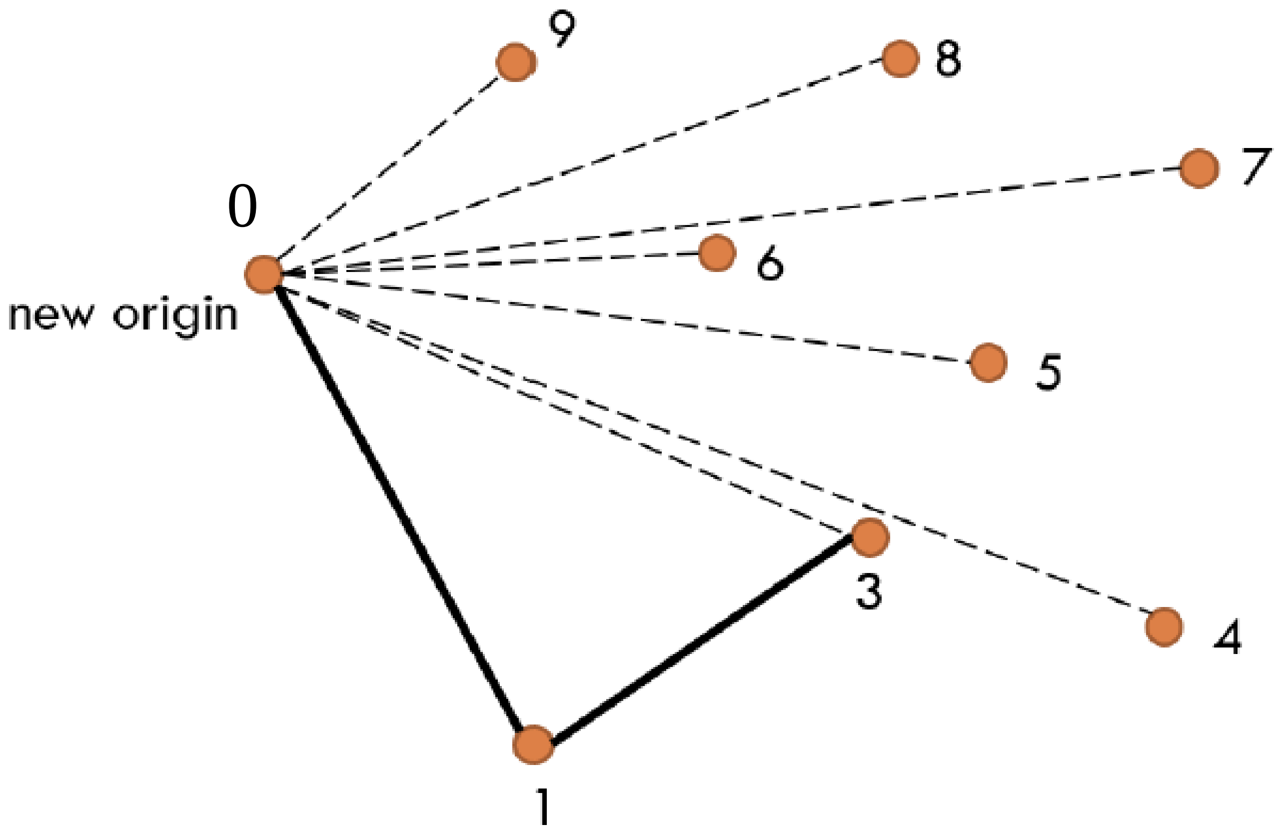
Adding point 3 causes a consecutive triple 1-2-3, with a right-turn at the concave corner 2; right-turns not allowed in CCW traversal!
this can be checked by Orientation Test; corner 2 cannot be on the hull; remove 2, add direct edge (1, 3)



Orientation Test: $D = \begin{Vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{Vmatrix}$

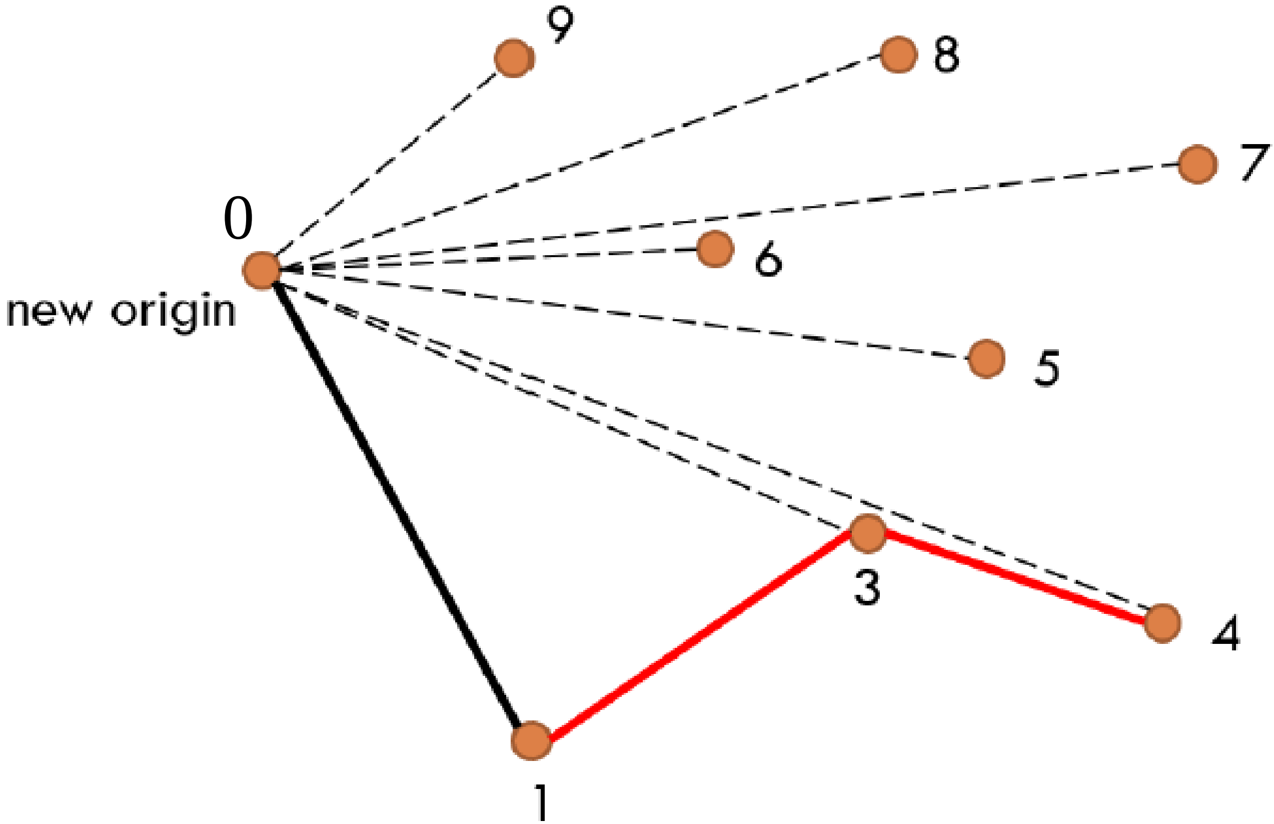
Graham Scan

… add direct edge between (1, 3)



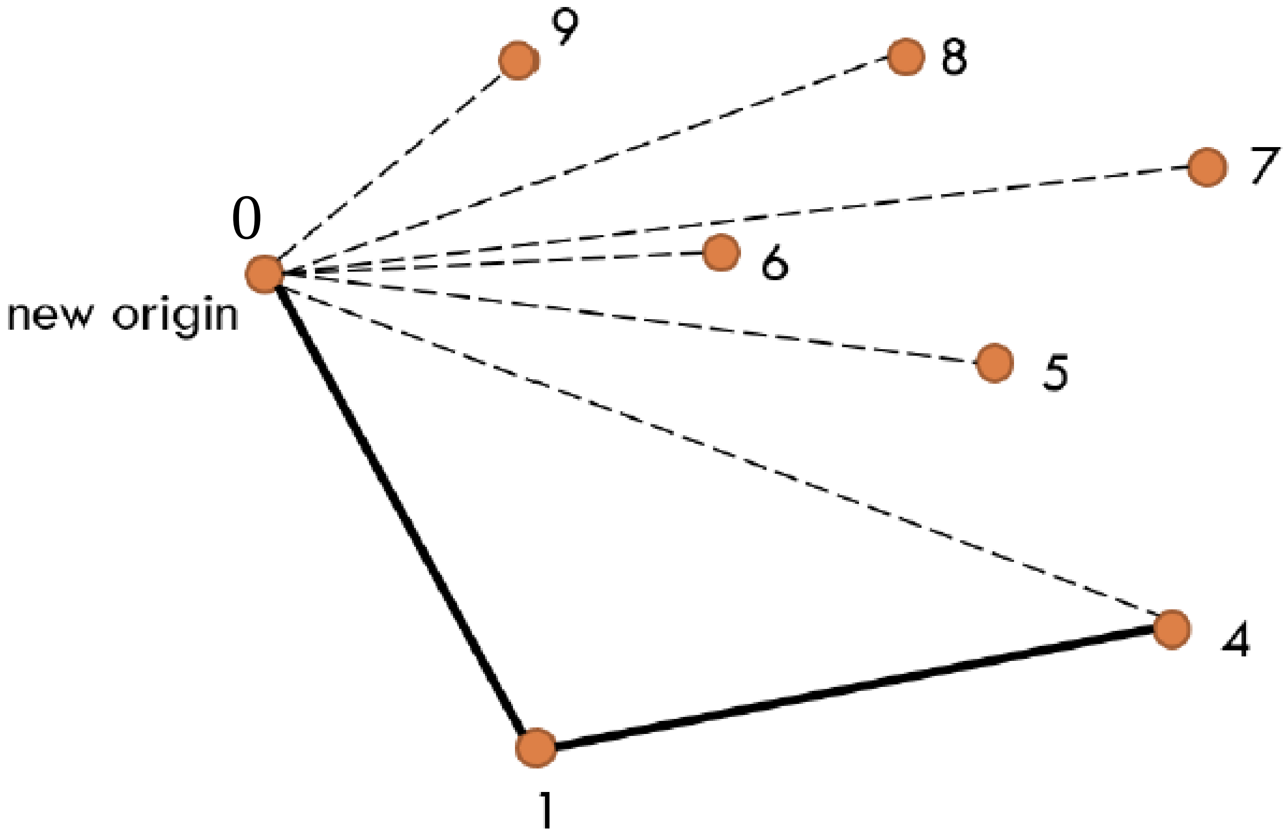
Graham Scan

Adding point 4 to the chain causes a right-turn, remove 3



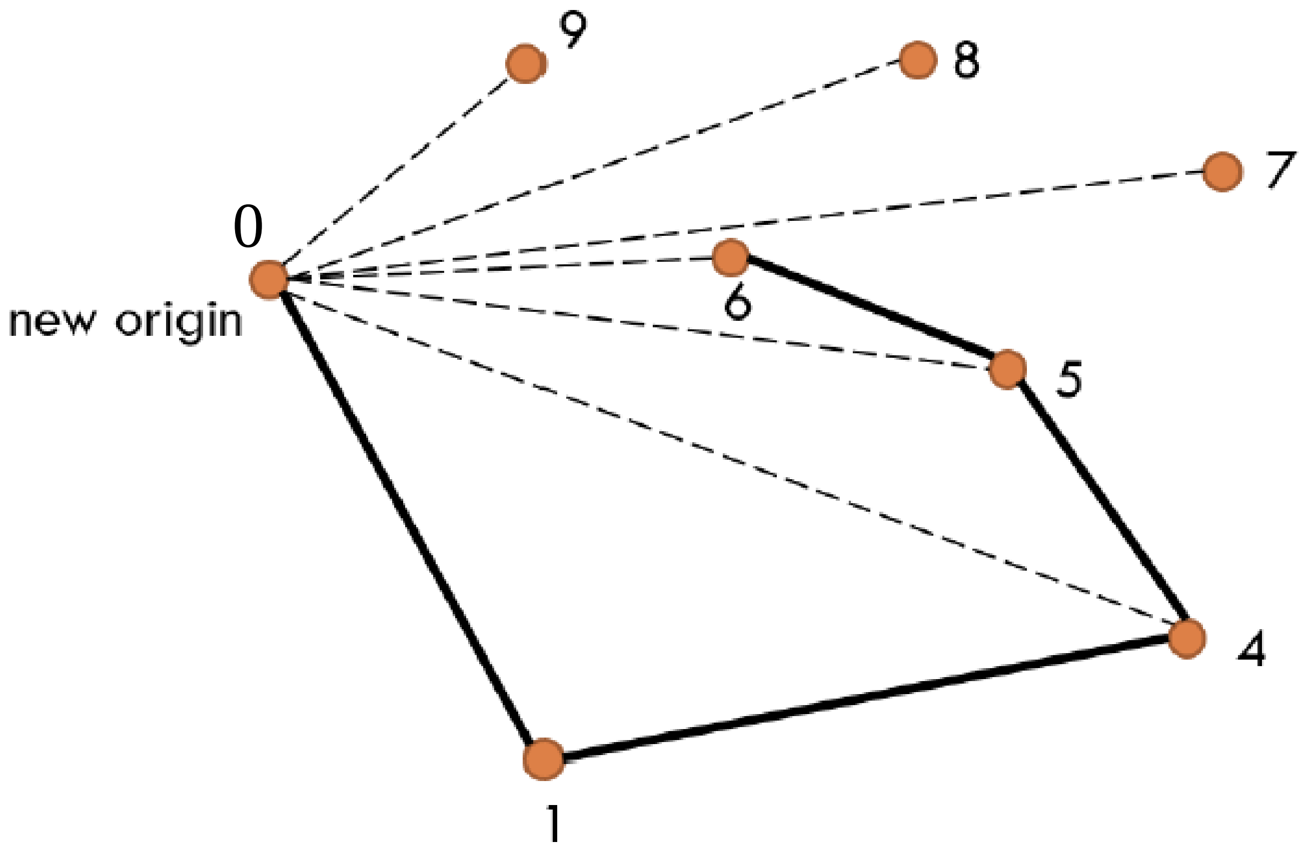
Graham Scan

Remove 3 and add direct edge (1, 4)



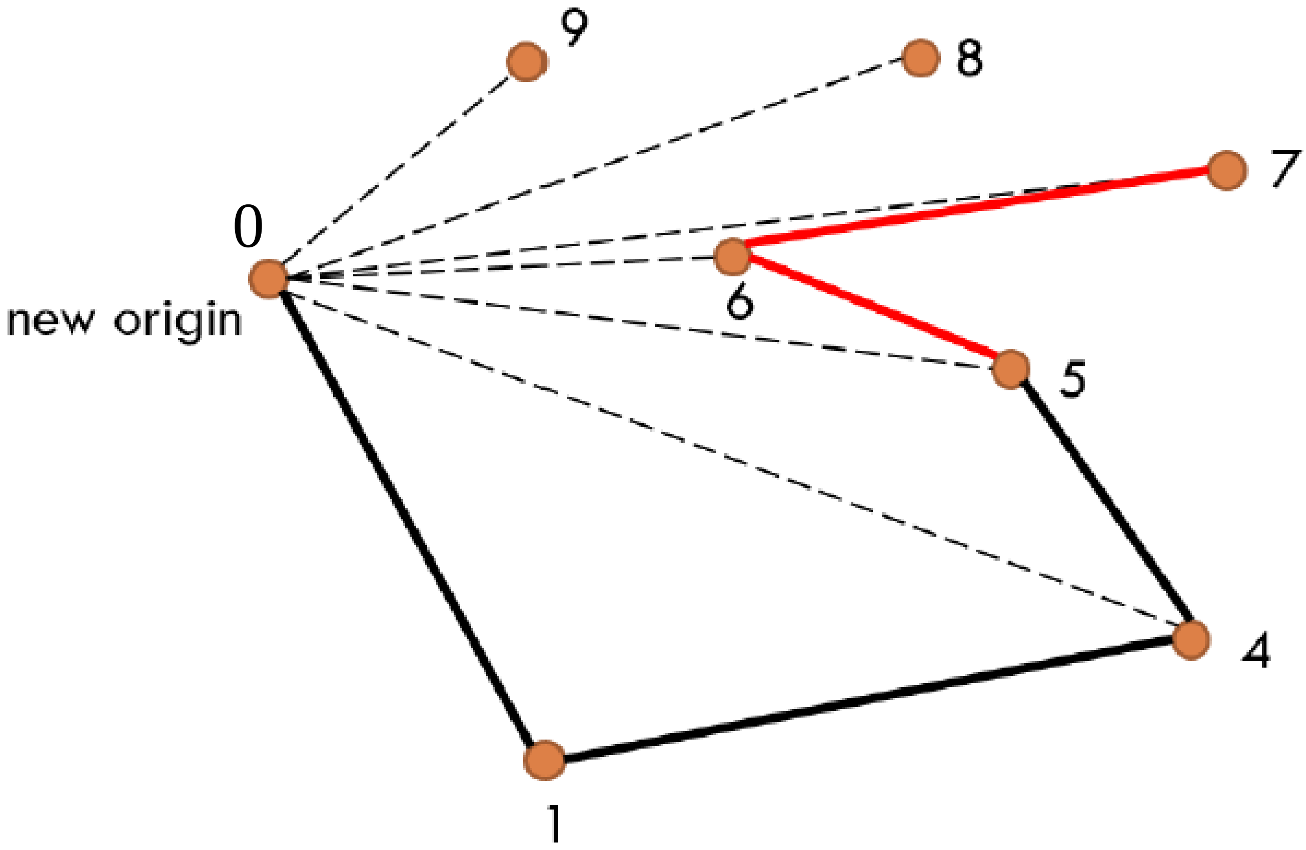
Graham Scan

Continue adding points as long as we see left-turns, i.e., convex corners …



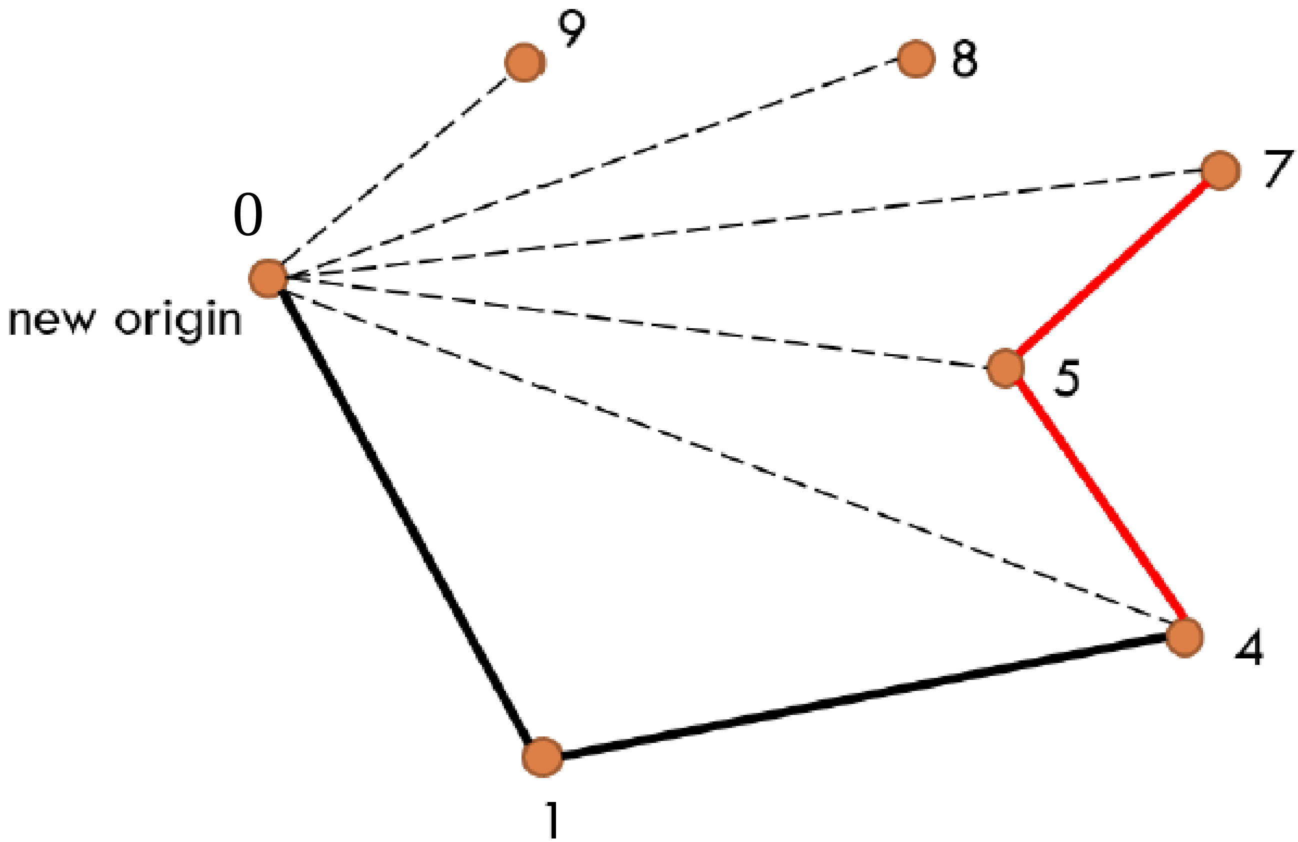
Graham Scan

Right-turn again, concave corner ..remove 6, add (5, 7)



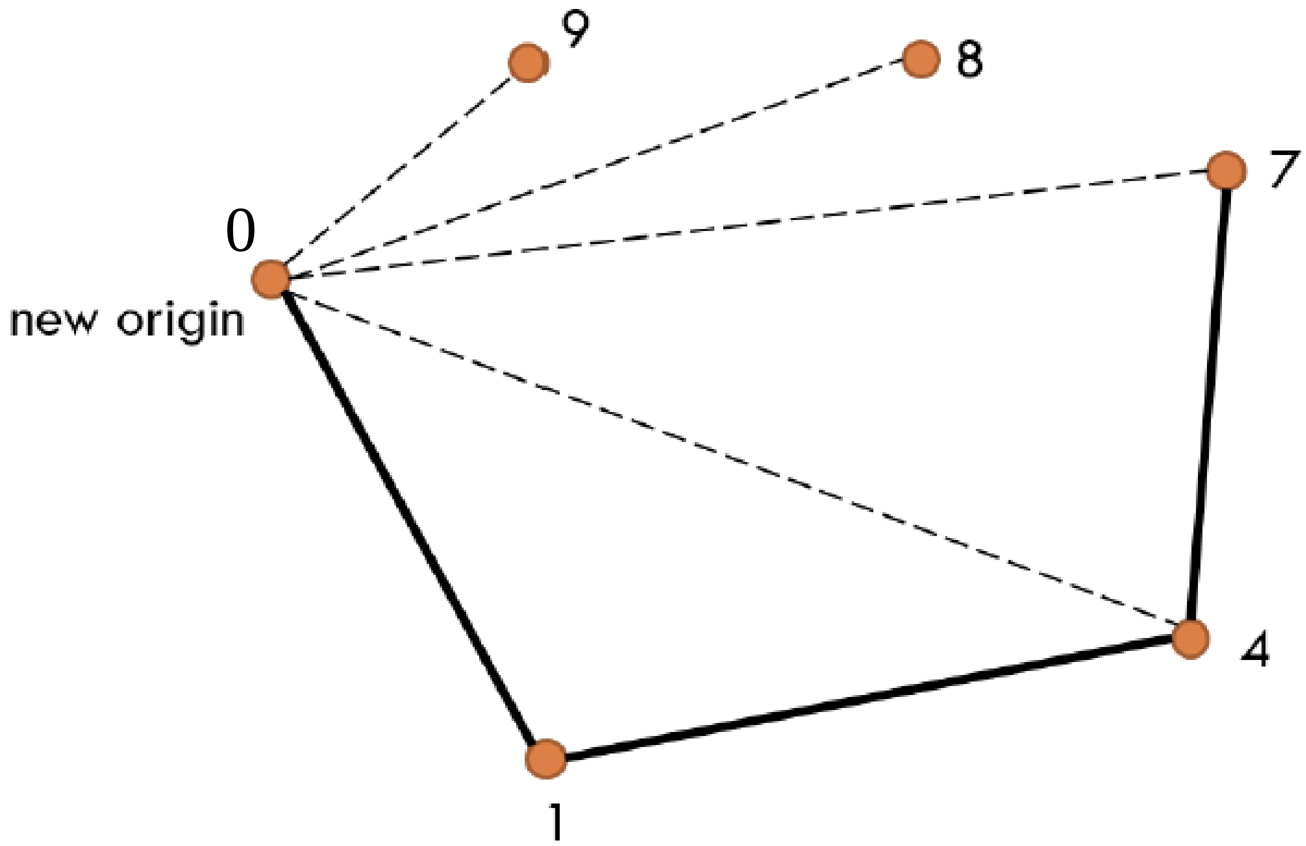
Graham Scan

.. after adding (5, 7), bad corner still remains at 5; remove 5, add edge (4, 7)



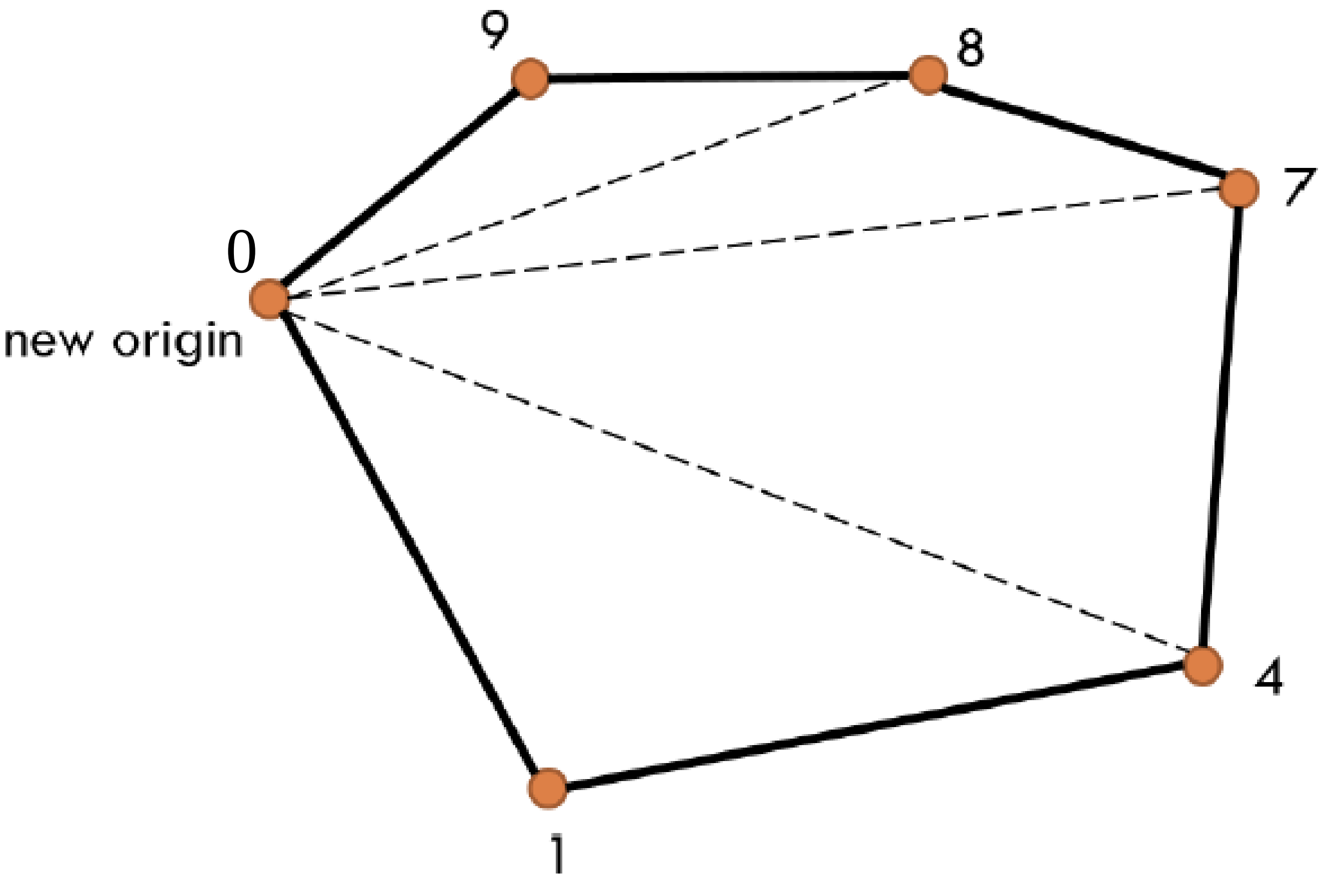
Graham Scan

… remove 5, add edge (4, 7)



Graham Scan

Continue adding edges … all convex .. Done!



Main Idea: Graham Scan

We will construct a *convex chain* of the given points

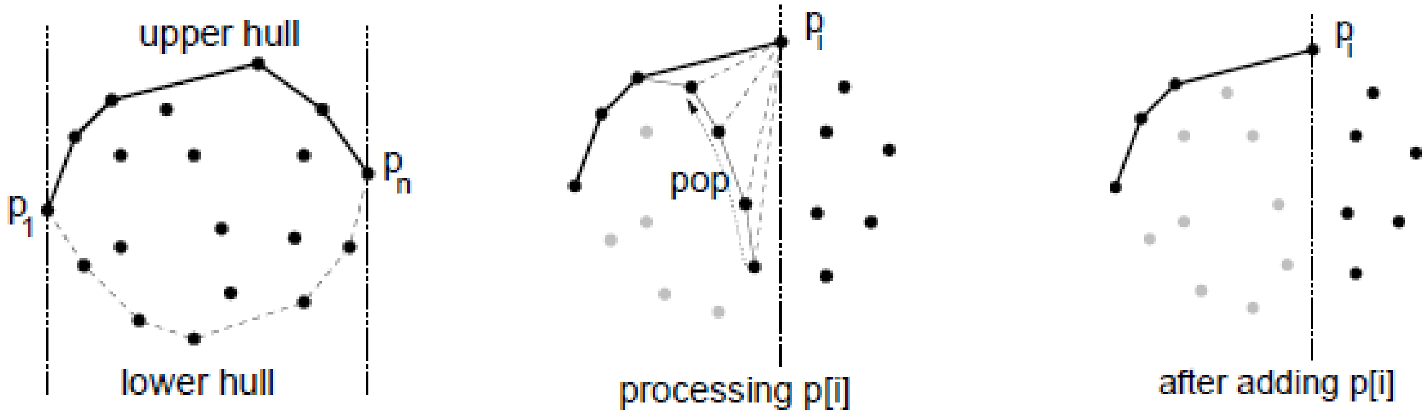
- For each *i*, do the following:
- Append point *i* to the current chain
- If the new point causes a concave corner, remove it from the chain
- Repeat until the new chain becomes convex

Algorithm: Graham Scan

- Set the leftmost point as the origin, and angularly sort the rest of the points in increasing order (of y/x)
- Initialize stack S: *push* p_i in stack, for $i = 0, 1$;
- For $i = 2, \ldots, n - 1$,
  - Let A be the second topmost element of S, B be the topmost element of S, and C be the i^{th} point
  - If Orient(A,B,C) < 0, pop S and go back
  - Push C to S
- Return S; Points in S form the convex hull

Complexity: $O(n \log n)$ for sorting, $O(n)$ for the rest; Each point is pushed onto the stack once, and popped out of it at most once! Space: $O(n)$

Sort-Hull Algorithm: Same time complexity

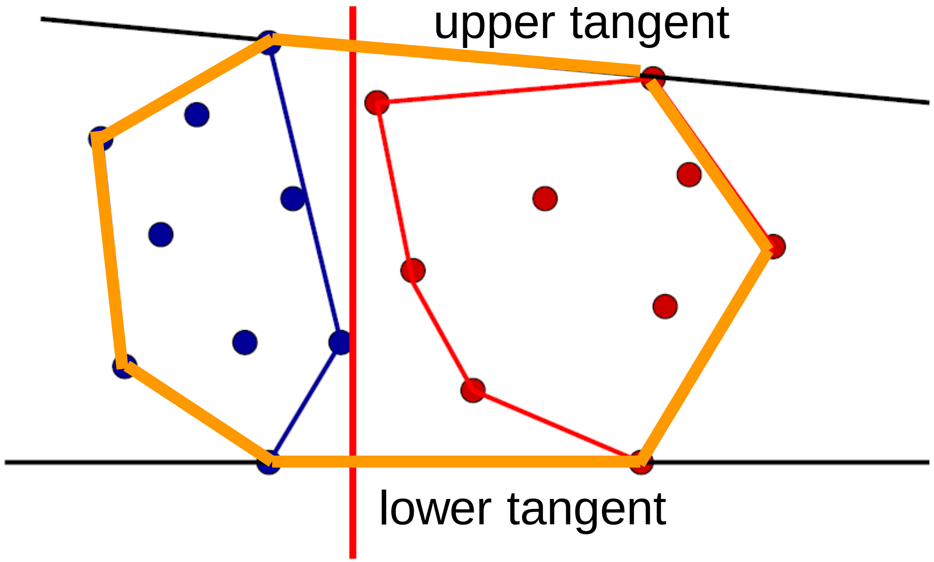


sweep-line →

Sort the vertices w.r.t. *x*-coordinates and use a vertical sweep-line; construct *convex chains* for the upper hull and lower hull in two passes

- For each *i*, do the following:
- Append point *i* to the current chain
- If the new point causes a concave corner, remove it from the chain
- Repeat until the new chain becomes convex
- Use stack as before for implementation; time complexity: $O(n \log n)$

Divide and Conquer



x-sort all points $\Rightarrow O(n\log n)$

Recurrence for time complexity:

$T(n) = 1$ if $n \leq 3$

$= c.n + 2T(n/2)$, otherwise

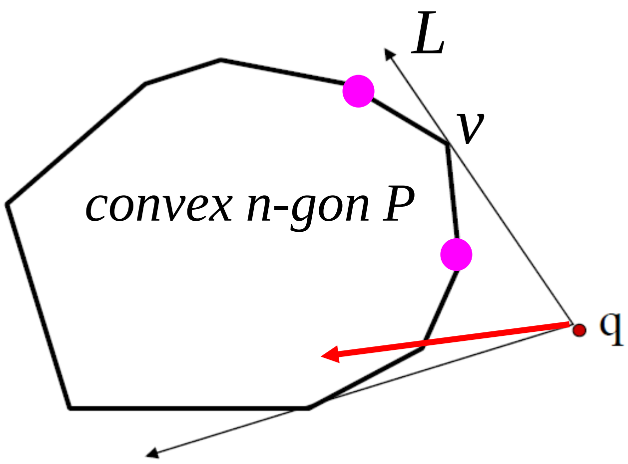
$\Rightarrow T(n) = O(n\log n)$

Assuming computation of upper and lower tangents and unification of the two can be done in $O(n)$ time at every step

Main idea:

- Divide a set of points into two subsets by a vertical line at the median x-coordinate of the points;
- Compute the convex hull for each subset recursively;
- Merge the two convex hulls into one convex polygon;
  - Find two external tangent lines to merge them
- Analogous to merge sort

Divide and Conquer



A step required in the divide-and-conquer algorithm:

Given a polygon P and a line L joining a point q and a vertex v of P, determine whether L is a tangent to P at v

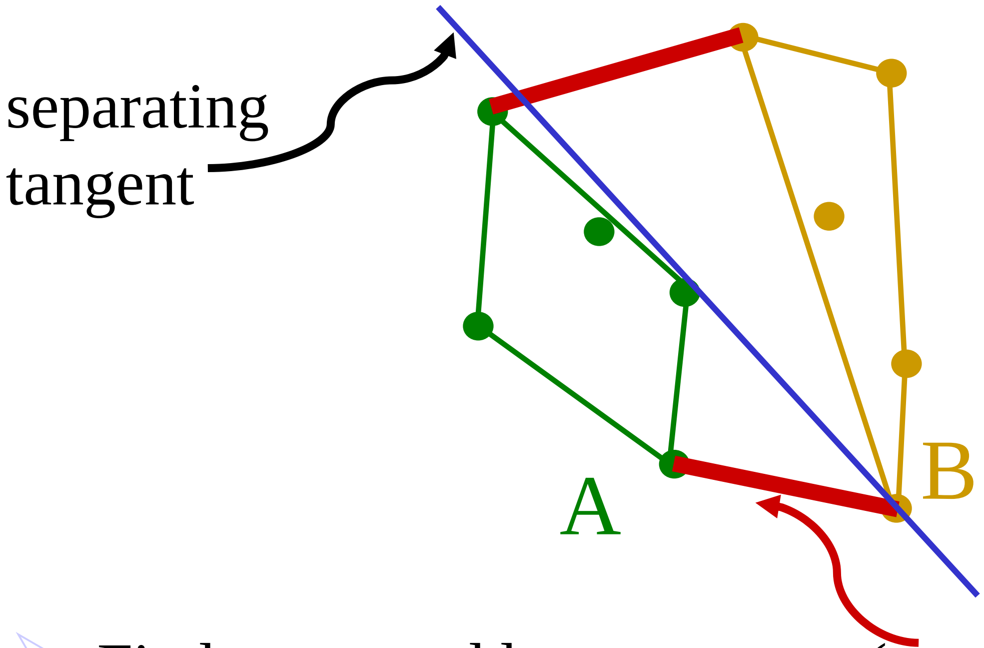
$\Rightarrow$ $O(1)$ by Orientation Test

Note:

Two tangents must be distinct for non-trivial cases

The two tangents can also be discriminated using Orientation Test

Merging



- Find upper and lower tangent (**outer tangents**)
- CH(A∪B) can be computed from CH(A) and CH(B) in O(n) time

Finding the outer tangents

Finding the lower tangent:

a = rightmost point of A

b = leftmost point of B

while $L = ab$ not lower tangent to both convex hulls of A and B do{

while L not lower tangent to convex hull of A do{

$a = a - 1$

}

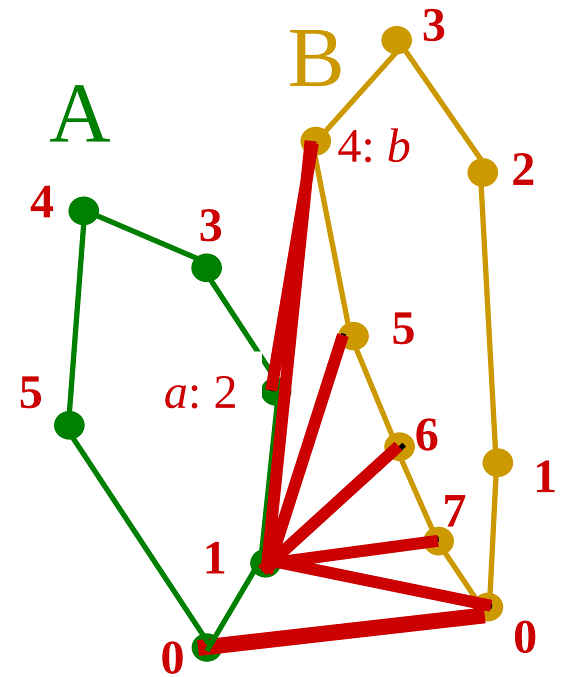
while L not lower tangent to convex hull of B do{

$b = b + 1$

}

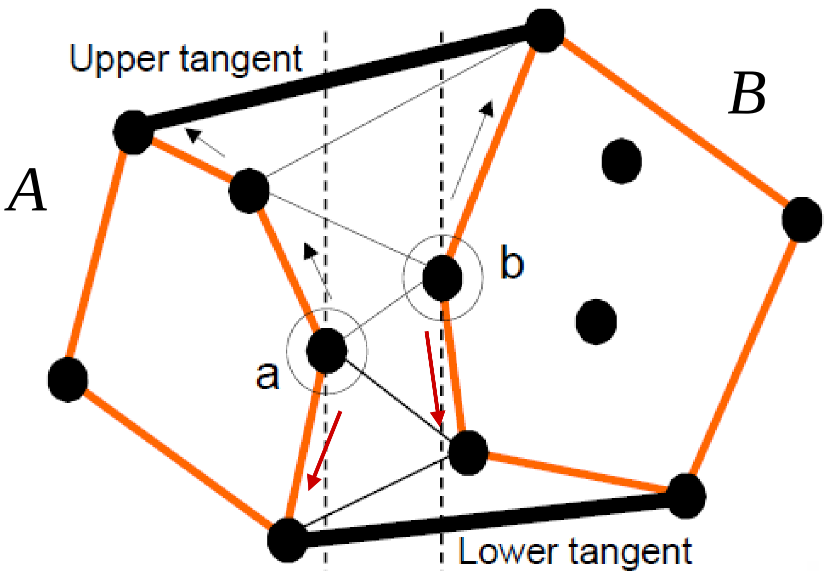
}

check with orientation test $O(1)$



Tangent finding and merging: $O(n)$

Divide and Conquer



x-sort all points $\Rightarrow O(n\log n)$

Recurrence for time complexity:

$T(n) = 1$ if $n \leq 3$

$= c.n + 2T(n/2)$, otherwise

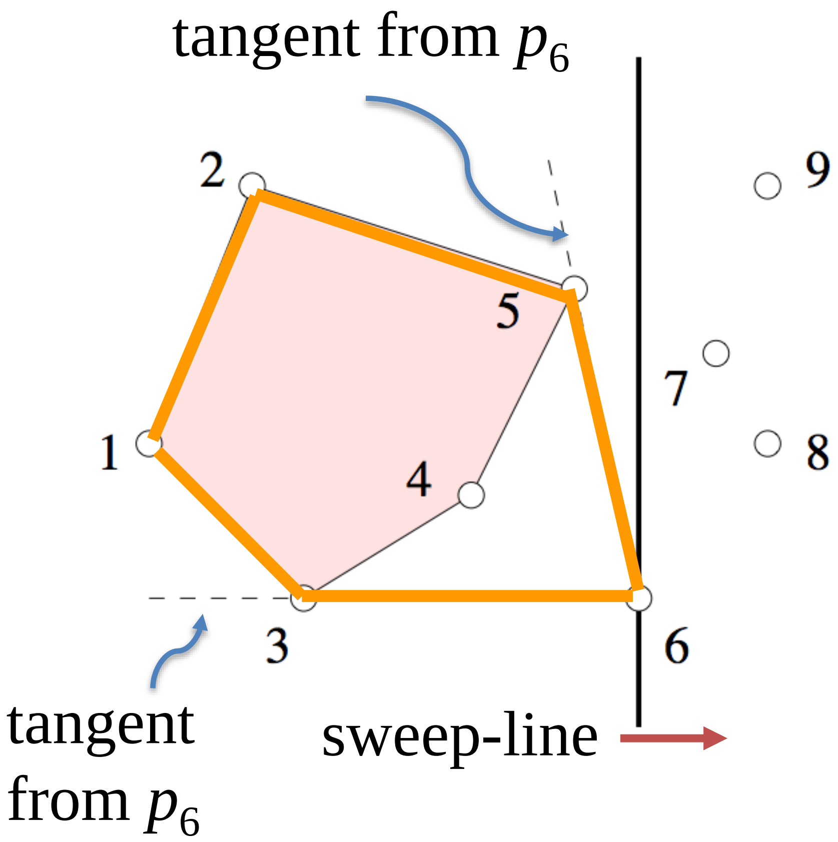
$\Rightarrow T(n) = O(n\log n)$

Note that the computation of upper and lower tangents and unification of the two can be done in $O(n)$ time at every step

Main idea:

- Divide a set of points into two subsets by a vertical line at the median x-coordinate of the points;
- Compute the convex hull for each subset recursively;
- Merge the two convex hulls into one convex polygon;
  - Find two external tangent lines to merge them
- Analogous to merge sort

Incremental Convex Hull (also online algorithm)



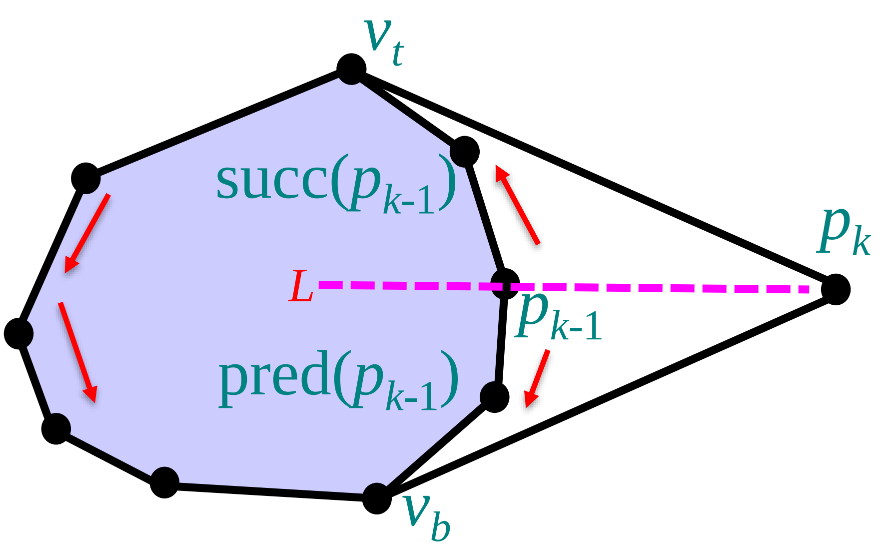
x-sort n points and move a vertical sweep-line halting at points $\Rightarrow O(n\log n)$

We have CH on k – 1 points and now processing the k-th point (e.g. p_6); initially draw Δ123

Draw tangents on CH(k-1) from point k; delete all interior points (e.g. 4) that lie in the pocket

Update the convex hull

Analysis: Incremental Convex Hull



x-sort *n* vertices and move a vertical sweep-line halting at points $\Rightarrow O(n\log n)$

We have CH on $k-1$ points and now processing the *k*-th vertex (e.g. p_k);

Draw tangents on CH(k-1) from point p_k; delete all interior points that lie in the pocket

Update the convex hull

Analysis:

Draw a line L passing through p_k and p_{k-1}; If L is a tangent, record it and find the other tangent by following either pred(p_{k-1}) or succ(p_{k-1}); otherwise traverse along the direction of both pred(p_{k-1}) or succ(p_{k-1}) until you find the two tangents;

While performing tests for tangency ($O(1)$ per vertex) and delete the points which do not satisfy tests;

Total number of checks for tangency and deletions from pockets over all iterations is at most (n -3); Overall time complexity $\Rightarrow O(n\log n) + O(n) \Rightarrow O(n\log n)$

Lower bound on convex hull time complexity

Algorithm CH_Sort(S):

/* Sorts a set of numbers using a convex hull algorithm

Converts numbers to points, runs CH, converts back to sorted sequence */

Input: Set of numbers $S \subseteq \mathbf{R}$

Output: A list L of numbers in S sorted in increasing order

$P = \emptyset$

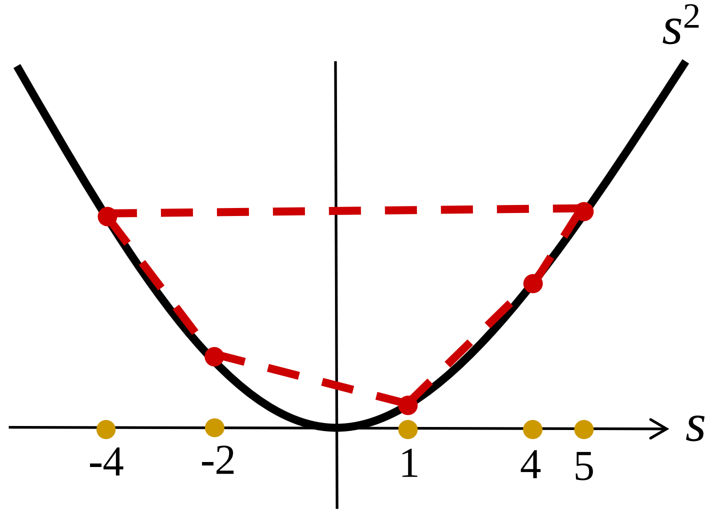
for each $s \in S$ insert (s, s^2) into P

$L' = \mathrm{CH}(P)$ // compute convex hull

Find point $p' \in P$ with minimum x-coordinate

for each $p=(p_x,p_y) \in L'$, starting with p', add p_x into L

return L



Goal: sorting of numbers

Map a number $s \rightarrow (s, s^2)$ in 2D

Construct the convex hull

Traverse L → R along the lower chain ⇒ sorted sequence

Thus, sorting ⇒ CH; since all other operations need linear time, any convex hull algorithm has to take $\Omega(n \log n)$ time in the worst case;
More refined bound: $\Omega(n \log h)$, where h: #hull vertices

Convex Hull: Summary So Far

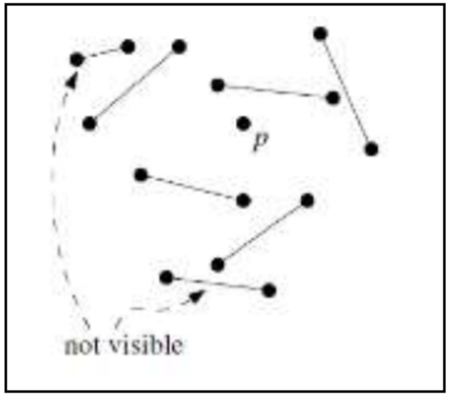
- Brute force algorithm: $O(n^3)$
- Quick-Hull: $O(n^2)$
- Jarvis' march (gift wrapping): $O(nh)$
- Incremental insertion: $O(n \log n)$
- Divide-and-conquer: $O(n \log n)$
- Graham's scan: $O(n \log n)$
- Lower bound: $\Omega(n \log n)$

Can you improve it further (output-sensitive), e.g. $O(n \log h)$, where h: #hull vertices ?

Homework Set - 03

Homework Set - 03 (Total Marks = 20)

1. Let S be a set of n disjoint line segments in the plane, and let p be a point not on any of the line segments of S. We wish to determine all line segments of S that p can see, that is, all line segments of S that contain some point q so that the open segment pq does not intersect any line segment of S. Give an $O(n \log n)$ time algorithm for solving this problem. See the example shown on the right. [5]



2. Let S be a subdivision of complexity n, represented using DCEL data structure, and let P be a set of m query points. Give an $O((n + m) \log(n + m))$ time algorithm that computes for every point in P in which face of S it is contained. [5]

3. Write a formal proof for the following claim: Any polygon with h holes and a total of n vertices (including those defining the polygon and holes), can always be guarded by $\lfloor (n + 2h)/3 \rfloor$ vertex guards. Note that a hole may be surrounded by other holes, and thus it may not be always visible from the boundary of the polygon. [5]

4. Consider an implementation of Hertel-Melhorn (HM) Algorithm for convex-partitioning of a simple polygon P with n vertices. Assume that a triangulation of P is given. Suggest a data structure and the required procedure so that HM-Algorithm can be implemented in $O(n)$-time. [5]

Submit solutions via Moodle. Due: 24 February 2022, 23:55; Credit: 10%

Announcement of Online Test - 01

Friday, 25 February, 2022; 11:05 am - 12:50 pm

Coverage: Polygons, point-inclusion queries, orientation test, robustness issues, polygonization, diagonals, triangulation, convex partition, art-gallery problems, DCEL, intersections, map overlay, convex hull, related algorithms, data structures, and complexities (material covered until 18 February 2022);

Questions will be made available through Moodle and answers should also be submitted via Moodle. The submission server will open at 10:55 am and close at 1:05 pm, 25 February 2021.

Credit: 25%