

CMSC 754 - Computational Geometry

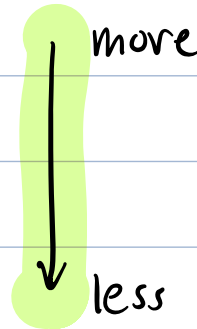
Lecture 1: Introduction

What is Computational Geometry?

- Subfield of algorithm theory involving discrete geometric structures

- points, lines + line segments, polygons, spatial subdivisions

in 2-dimensional
3-dimensional
low dimensional
high dimensional

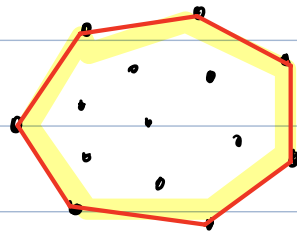


Features:

- Worst-case asymptotic complexity
deterministic + randomized
- Rigorous - provably correct + efficient (in theory)
- Discrete inputs/outputs
- Combinatorial-based analysis
- "Simple" geometry - flat, Euclidean
- low dimensionality

Topics:

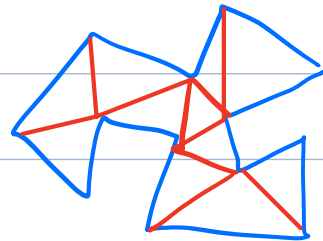
- Convex hulls →



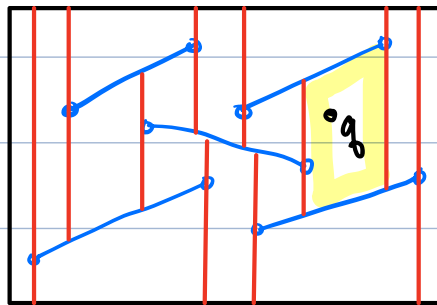
- Intersections



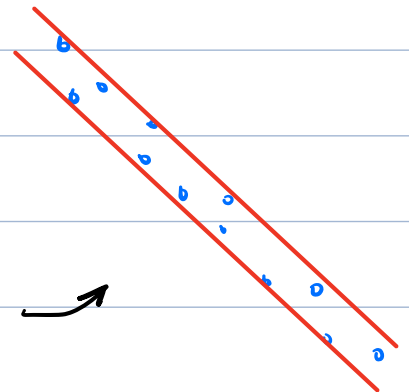
- Triangulations
+ spatial subdivisions



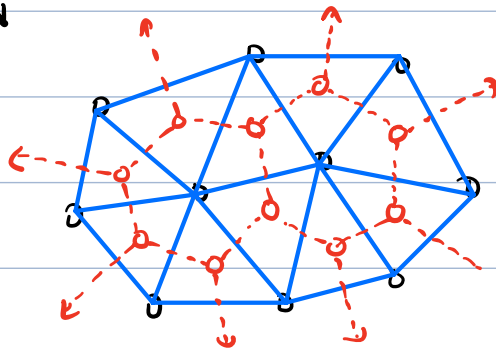
- Point location ↗



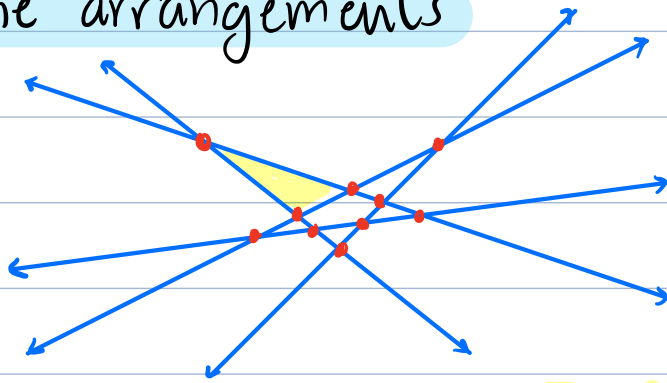
- Linear programming + duality →



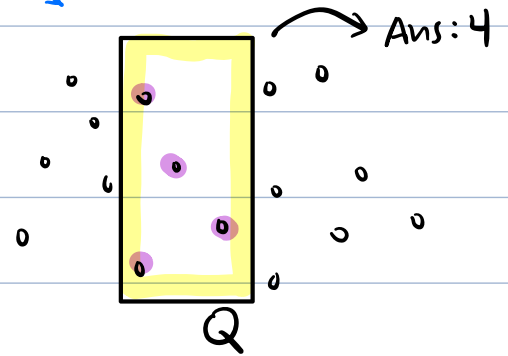
- Voronoi diagrams + Delaunay triangulations



- Line / hyperplane arrangements

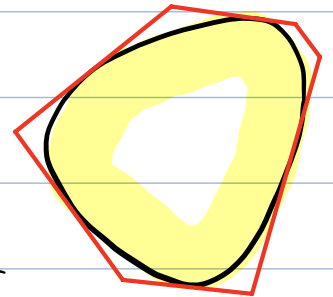


- Search + Data Structures



- Approximation

- ϵ -nets
- ϵ -kernels + coresets



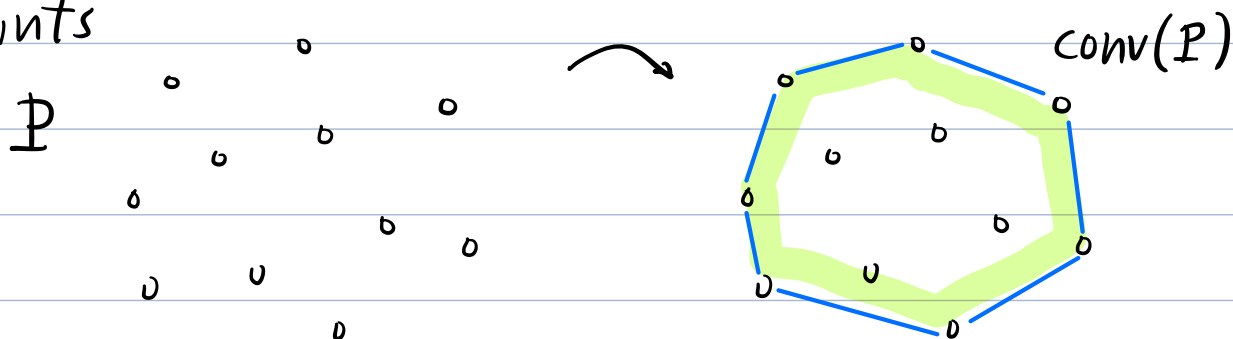
- More? High dimensional geometry
Computational topology

CMSC 754 - Computational Geometry

Lecture 2: Convex Hulls in the Plane

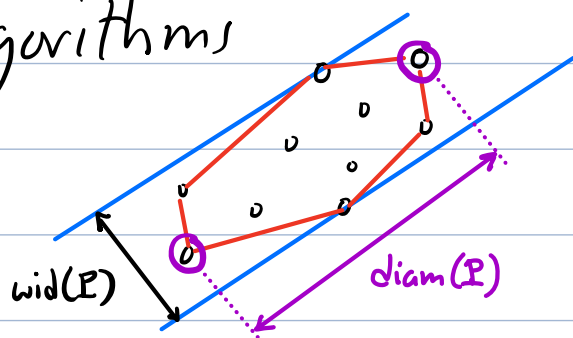
Convex Hull: (Intuitive definition)

Given a point set P in $\mathbb{R}^2$, imagine snapping a rubber band around the points



Uses:

- Shape approximation (intersection test)
- first step in other algorithms
 - diameter
 - width



Geometric Basics:

Affine Geometry: The geometry of "flats"

Objects:

- Scalars (Real numbers) $3, -17, \pi, \sqrt{2}$
- Points (Position) p, q
- (Free) Vectors (Direction + magnitude)
↳ not anchored to origin ↗ ↘

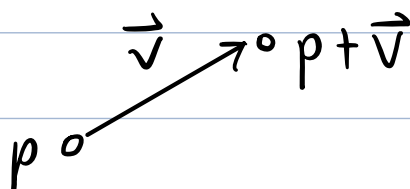
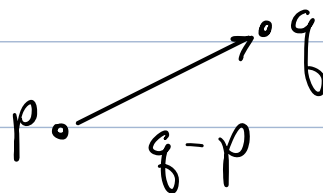
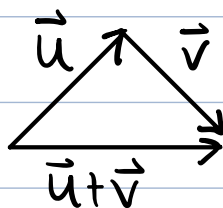
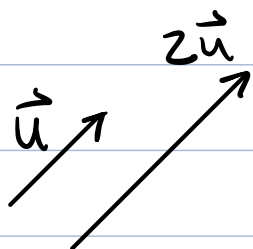
Operations:

$S \cdot V \rightarrow V$ - scalar-vector mult

$V + V \rightarrow V$ - vector addition (+ subtract)

$P - P \rightarrow V$ - point-point subtract

$P + V \rightarrow P$ - point-vector addition



Note: $P + P$ not allowed

Affine Combination:

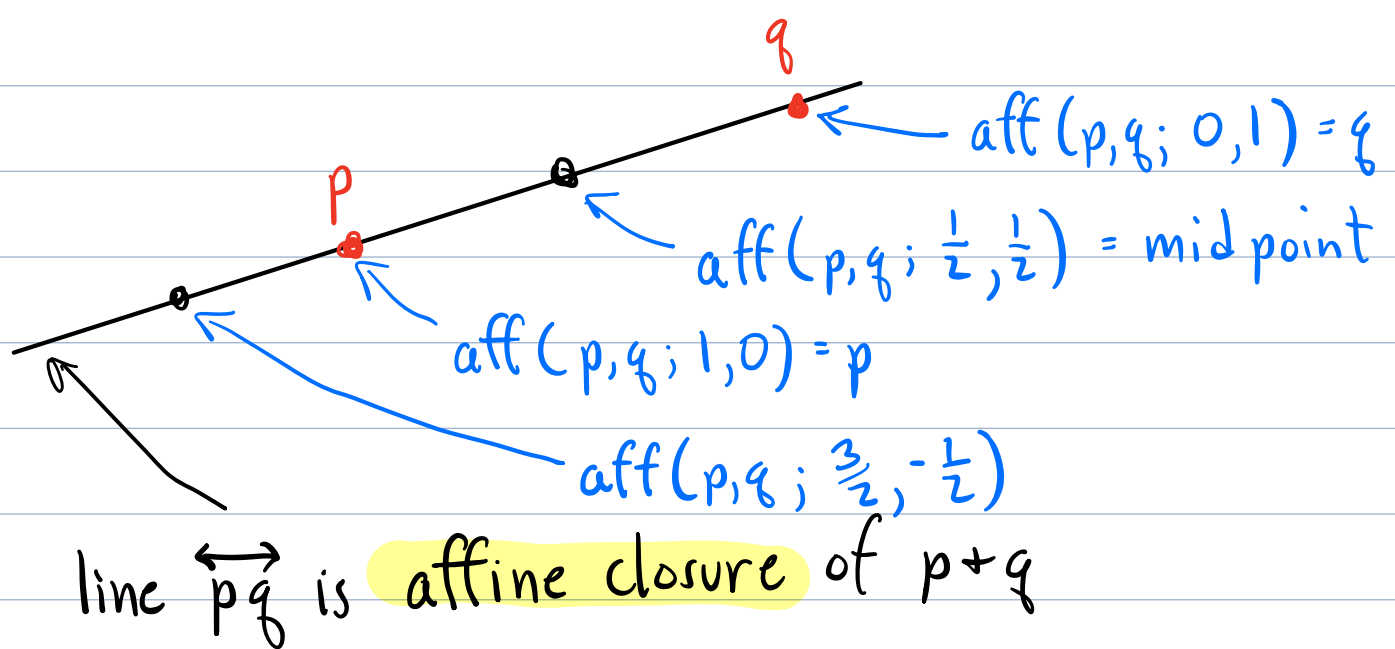
Given points p, q and scalars α, β where $\alpha + \beta = 1$, define:

$$\text{aff}(p, q; \alpha, \beta) = p + \beta(q - p)$$

Equivalent to:

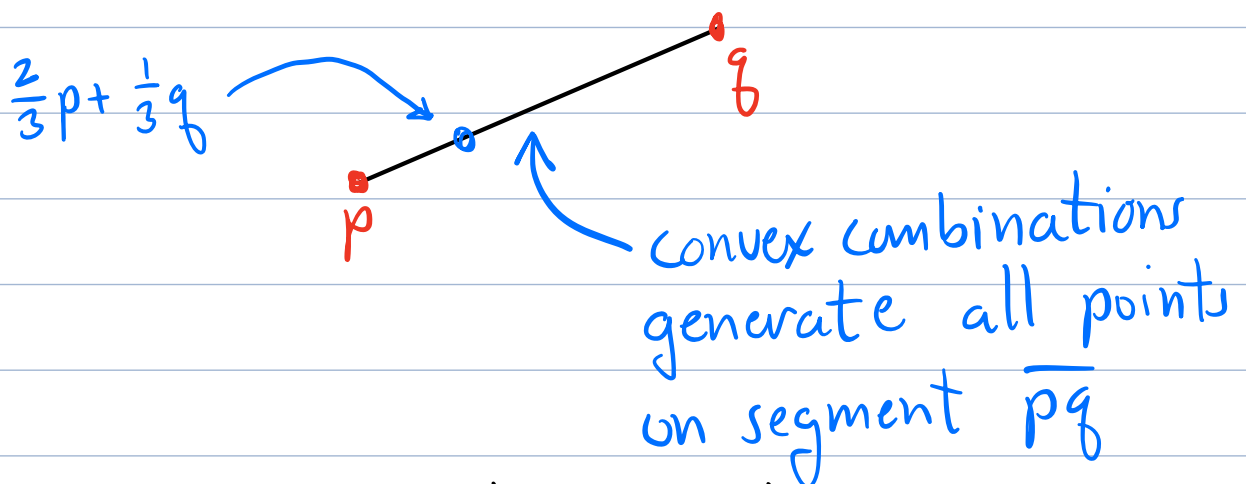
$$\begin{aligned} & p + \beta q - \beta p \\ &= (1 - \beta)p + \beta q \\ &= \alpha p + \beta q \end{aligned}$$

Weighted sum
of $p + q$

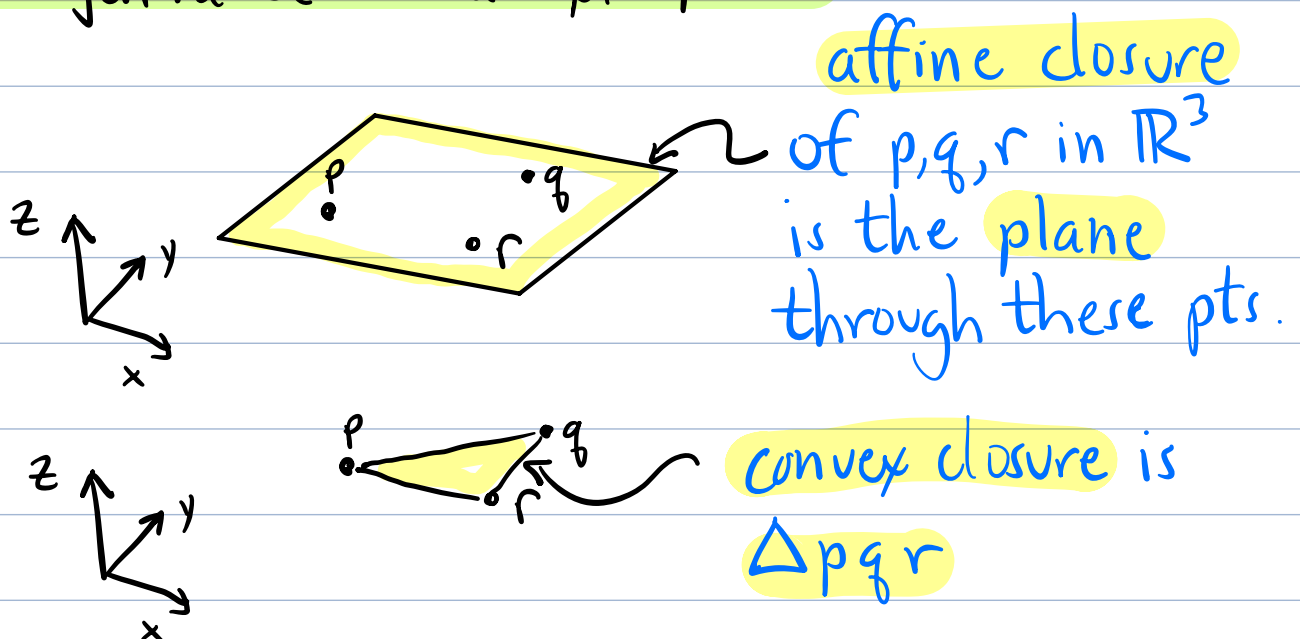


Convex Combination:

Special case when $0 \leq \alpha, \beta \leq 1$



These generalize to multiple points:



But... No way to define angles or lengths!

Euclidean Geometry: Affine + Inner product

Inner product - Linear operator that maps
2 vectors to a scalar

$$\langle \cdot, \cdot \rangle : V \times V \rightarrow \mathbb{R}$$

Dot product (most common inner product)

$$\vec{u} = (u_1, \dots, u_d) \quad \vec{v} = (v_1, \dots, v_d)$$

$$\vec{u} \cdot \vec{v} = u_1 v_1 + \dots + u_d v_d = \sum_{i=1}^d u_i v_i$$

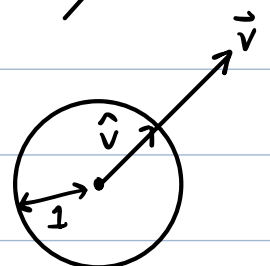
Inner product allows us to define:

Vector length: $\|\vec{v}\| = \sqrt{\vec{v} \cdot \vec{v}}$



Normalization (to unit length):

$$\hat{v} = \vec{v} / \|\vec{v}\|$$



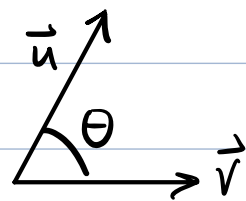
Distance between points:

$$\text{dist}(p, q) = \|q - p\|$$

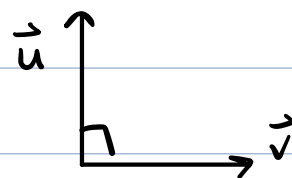


Angles between vectors:

$$\text{ang}(\vec{u}, \vec{v}) = \cos^{-1}(\hat{u}, \hat{v})$$



Perpendicularity: $\vec{u} \perp \vec{v} \Leftrightarrow \vec{u} \cdot \vec{v} = 0$

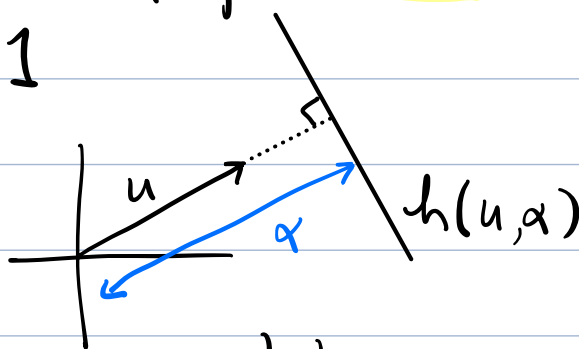


Lines, Hyperplanes, Halfspaces:

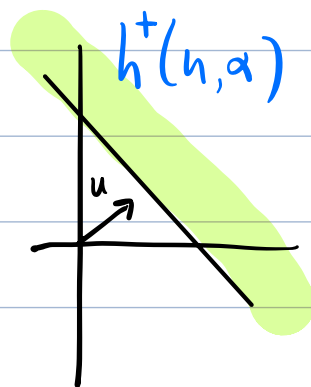
Given nonzero vector u + scalar α ,

$h(u, \alpha) = \{ p \in \mathbb{R}^d \mid p \cdot u = \alpha \}$ is hyperplane

If $\|u\| = 1$

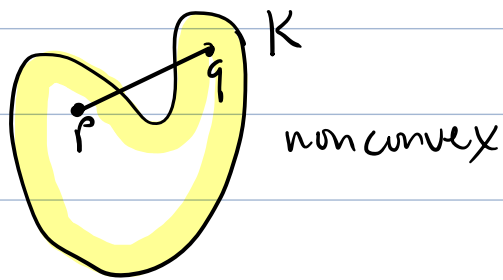
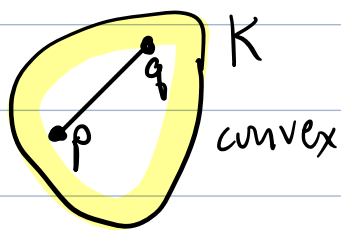


$$h^+(u, \alpha) = \{ p \in \mathbb{R}^d \mid p \cdot u \geq \alpha \}$$



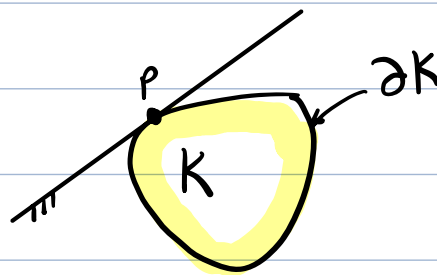
Convexity:

A set $K \subseteq \mathbb{R}^d$ is convex if $\forall p, q \in K$ the line segment $\overline{pq}$ (equiv. any conv. combination of p + q) lies within K



Support Hyperplane:

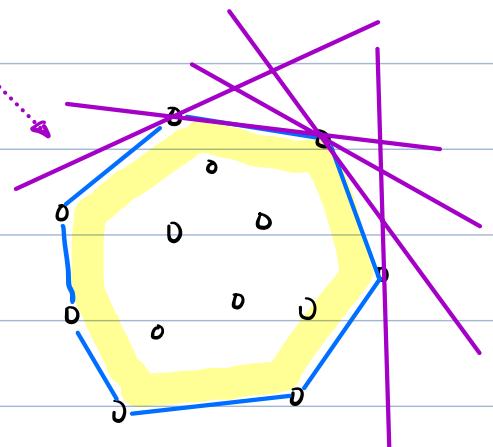
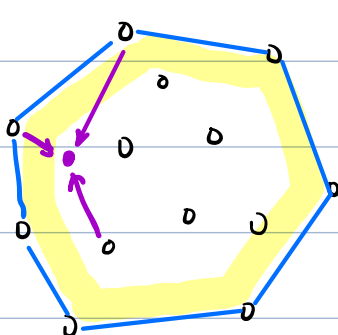
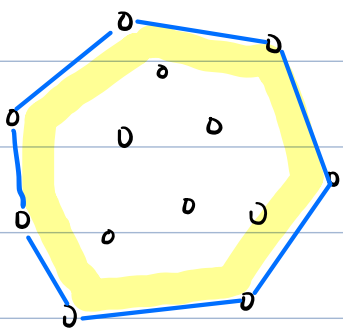
Given convex K and any point $p \in \partial K$,
 $\exists$ hyperplane passing through p with K
 lying all on one side.



Convex Hull:

Given a set P of points in $\mathbb{R}^2$, the convex
 hull, $\text{conv}(P)$, is the **smallest convex set**
containing P .

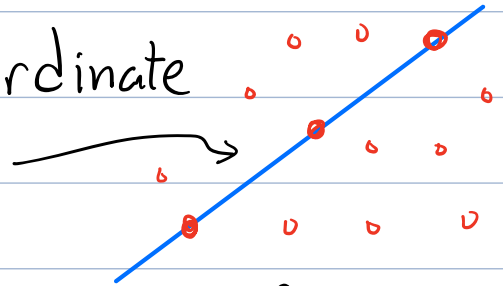
- The **set of all convex combs in P**
- The **intersection of all halfspaces**
containing P



General Position:

Geometric algorithms are complicated by rare (?) degenerate cases:

- points having same coordinate
- ≥ 3 collinear points
- ≥ 4 cocircular points

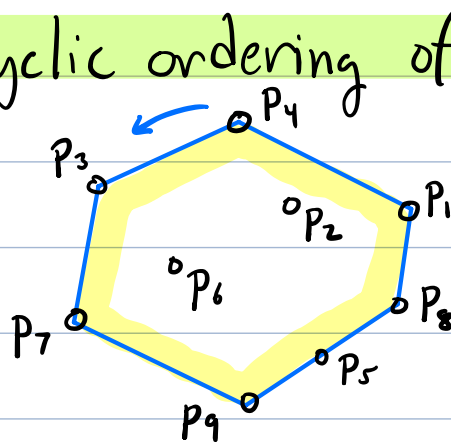


To simplify algorithm presentation we often assume these do not arise in the input.

Called general-position assumption

(Planar) Convex Hull Problem: Given a set of n pts $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^2$ ($p_i = (x_i, y_i)$) compute $\text{conv}(P)$.

Output: Cyclic ordering of vertices on the hull



possible output: (indices)

$\langle 4, 3, 7, 9, 8, 1 \rangle$

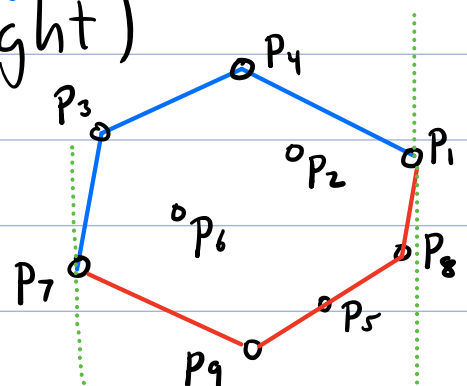
Note: p_5 not output

(Can assume this away by "general position")

Alternative output: (left to right)

Upper-hull + Lower-hull

$\langle 7, 3, 4, 1 \rangle + \langle 7, 9, 8, 1 \rangle$

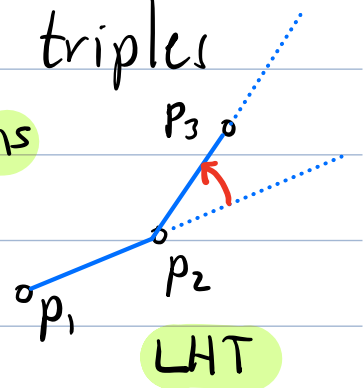
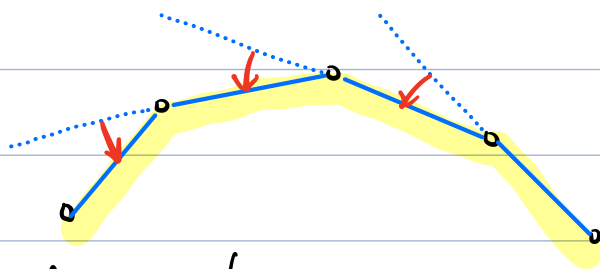


Graham's Scan: $O(n \log n)$ solution

- Compute upper + lower hulls separately
- Upper-hull:
 - Sort pts by x-coords
 - Add each to upper hull
 - Remove pts no longer on hull
- Lower-hull: (symmetrical) ← How?

Observations:

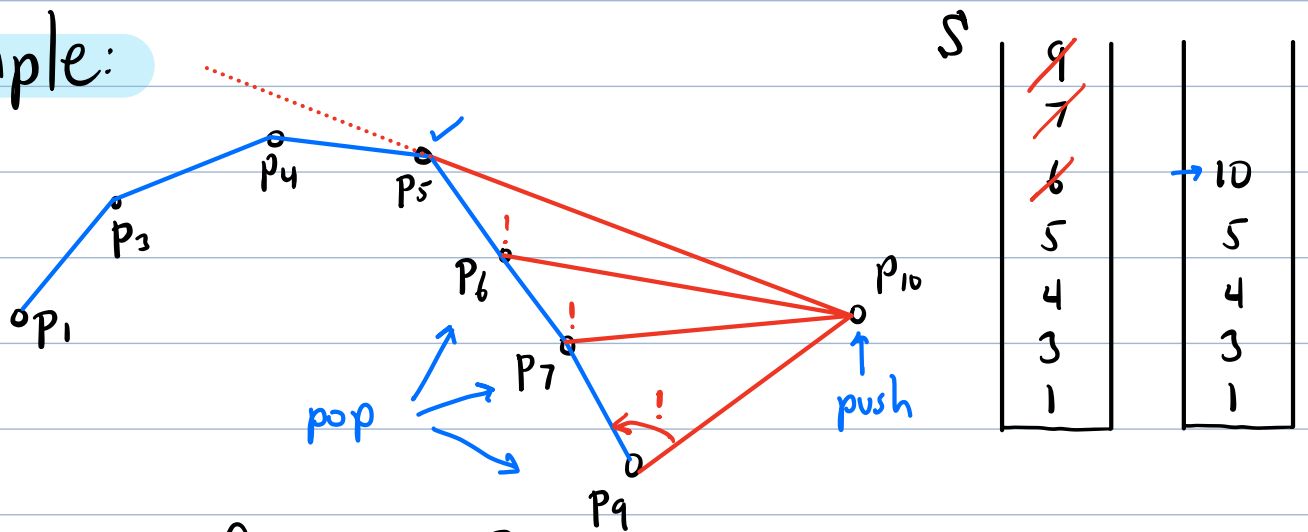
- The rightmost pt always on hull
- Reading right to left, consecutive triples on the hull form left-hand turns



Incremental Approach:

- Store vertices (indices) of upper hull on stack
- For each new point p_i (left to right)
 - While $\langle p_i, S[\text{top}], S[\text{top}-1] \rangle$ do not form LHT — pop $\$$
- Push p_i

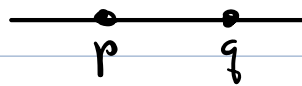
Example:



How to test for LHT?

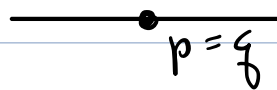
Orientation: In $\mathbb{R}^1$, for any p, q we have either:

$p < q$



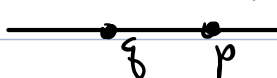
$$q - p > 0 \quad +$$

$p = q$



$$q - p = 0 \quad 0$$

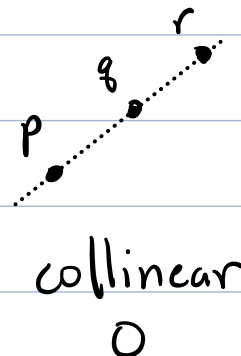
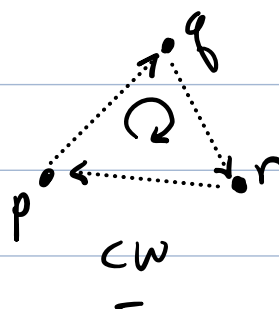
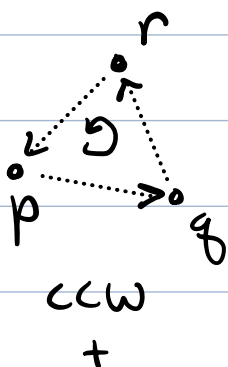
$p > q$



$$q - p < 0 \quad -$$

How to define "order" in $\mathbb{R}^2$??

Ans: order = orientation is a property of a seq. of 3 points



How to compute?

$\text{orient}(p, q, r) = \text{sign det}$

$$\begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix}$$

Note: Order of points is critical!

Orientation can generally be defined on any sequence of $d+1$ points in $\mathbb{R}^d$

Graham's Scan: (Upper Hull only)

- Sort pts by increasing x-coords $\langle p_1, \dots, p_n \rangle$
- Push p_1 + p_2 onto S
- for $i \leftarrow 3$ to n
 - while ($|S| \geq 2$ and $\text{orient}(p_i, S[t], S[t-1]) \leq 0$) pop S
t = "top"
 - push p_i

Correctness:

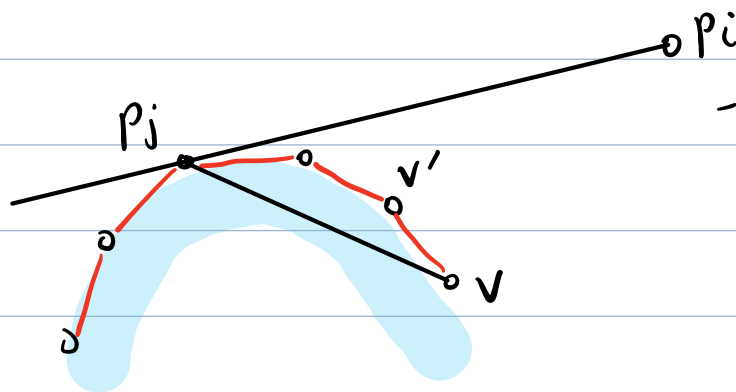
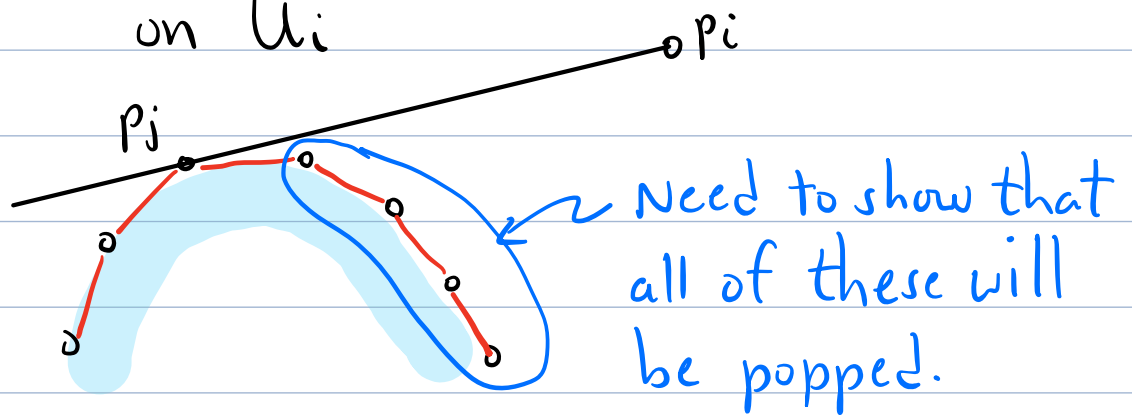
Lemma: After processing p_i , S contains upper hull of $\langle p \dots p_i \rangle$

Proof: By induction on i .

Basis: $i=1,2$ - Trivial

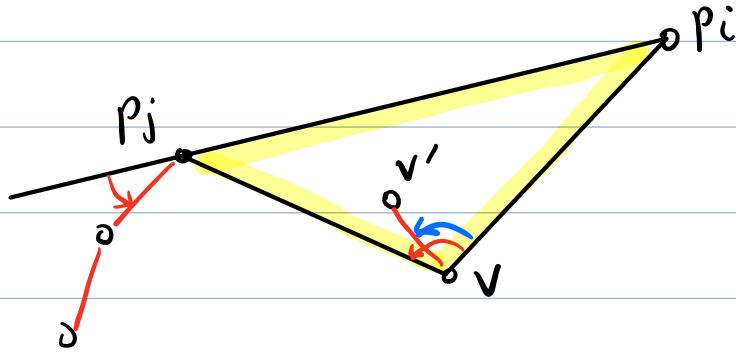
Step: For $i \geq 3$, let U_i denote the vertices on upper hull up to p_i .

- By induction U_{i-1} is correct up to p_{i-1}
- We'll show it's correct after adding p_i
- Let p_j denote vertex before p_i on U_i



- Let v be current vertex & let v' be predecessor

- By convexity:
 - all pts lie below $\overline{p_i p_j}$
 - all pts after p_j lie above $\overline{p_j v}$



$\Rightarrow v'$ lies in $\Delta p_j v p_i$

$\Rightarrow \angle p_i v v' \leq \angle p_i v p_j \leq \pi$

$\Rightarrow \text{orient}(p_i, v, v') \leq 0$

$\Rightarrow v$ is popped off stack

On arriving at p_j , orientation flips
 so popping stops at p_j
 + finally p_i pushed □

Running time:

- $O(n \log n)$ to sort
- for $3 \leq i \leq n$, let $d_i = \text{num. of pops}$ when inserting p_i

- Time for scan is $\sim$

$$\sum_{i=3}^n (d_i + 1) \leq n + \sum_{i=3}^n d_i$$

↑ ↖
for pops for push of p_i

- Note that $\sum d_i \leq n \rightarrow \text{Why?}$

- Total time: $O(n \log n + 2n) = O(n \log n)$

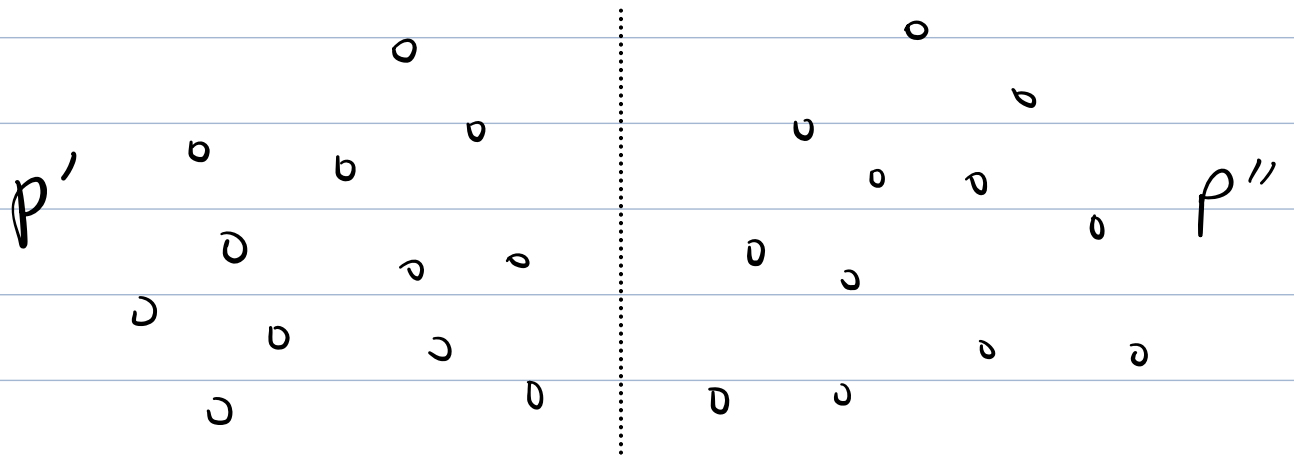
Divide + Conquer Algorithm:

Given point set P :

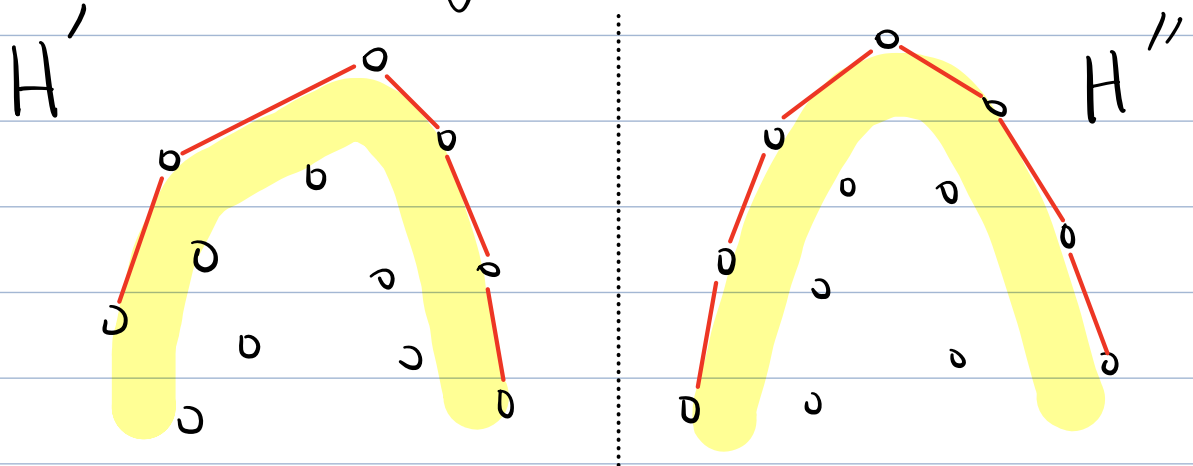
if $|P| \leq 3$ then compute hull by
brute force ($O(1)$)

else

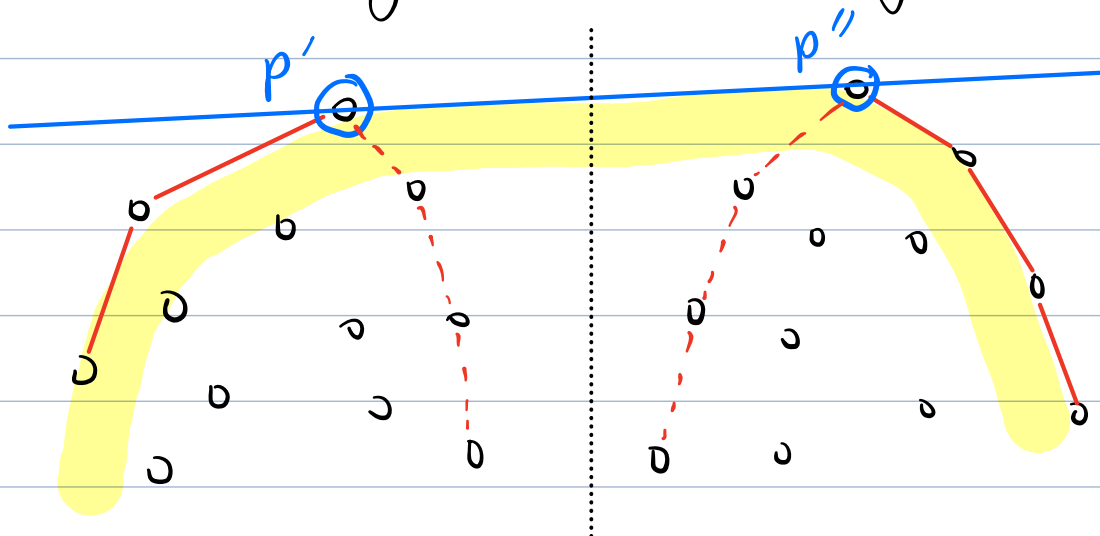
- Partition P by vertical line into P' , P'' of sizes $\sim n/2$



- Recursively compute hull of each



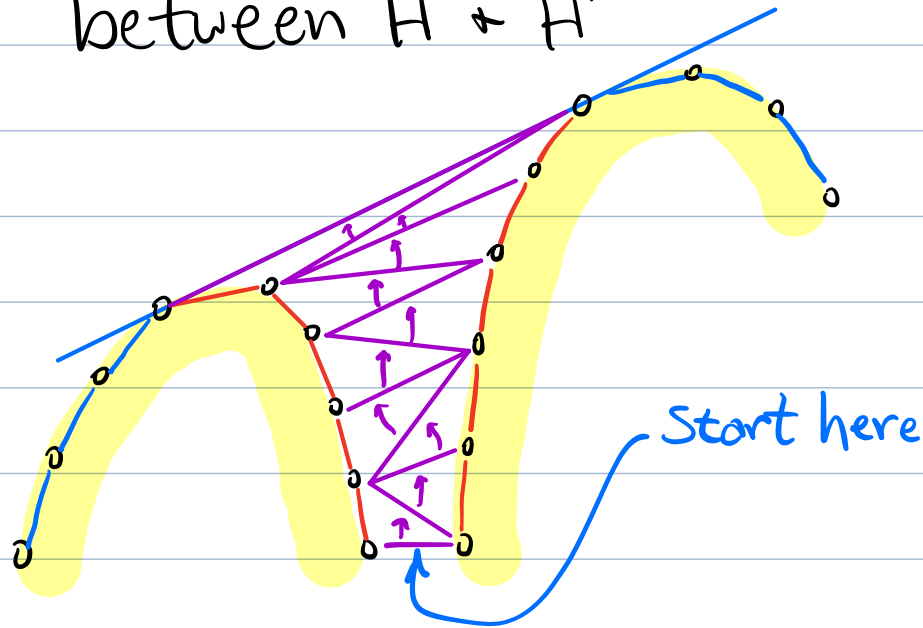
- Compute pts $p' \in H'$ + $p'' \in H''$ defining upper tangent



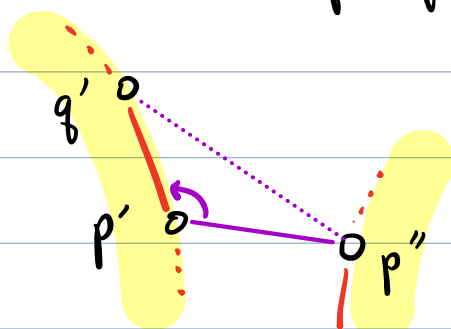
- Merge partial hulls together

How to compute upper tangent?

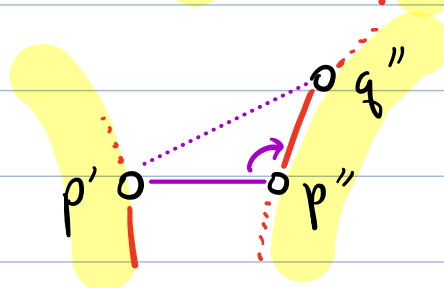
- Start with a chord joining closest points (w.r.t. x) of $H' + H''$
- "Walk" this chord up the ladder between $H' + H''$



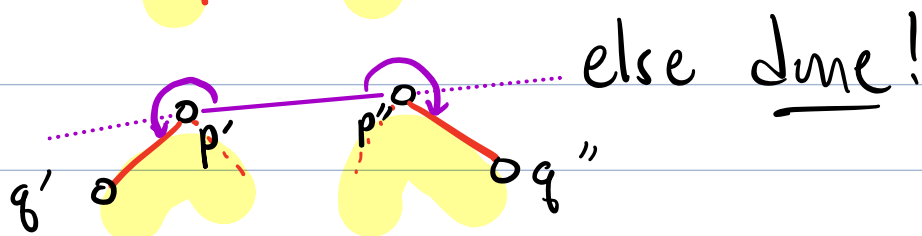
- How? Let p', p'' be current vertices
Let q', q'' be vertices above



if $\text{orient}(p'', p', q') \leq 0$
 $p' \leftarrow q'$



if $\text{orient}(p', p'', q'') \geq 0$
 $p'' \leftarrow q''$



else done!

Correctness? (Exercise)

Running time?

- $O(n)$ time to find upper tangent by walking
 - Each step takes $O(1)$ time
 - Eliminates one vertex

- Gives recurrence:

$$T(n) = 2T(n/2) + n$$

Recursively compute
two hulls, each
from $n/2$ pts

Split,
tangent,
merge

- Same as Mergesort. By
Master Theorem

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

Jarvis March: An $O(nh)$ algorithm

Idea: Compute any one vertex of hull $\rightarrow v_1$
for $i = 2, 3, \dots$

compute next vertex v_i on hull

if $(v_i = v_1)$ return $\langle v_1, \dots, v_{i-1} \rangle$

v_1 ? Point of P with min y -coordinate

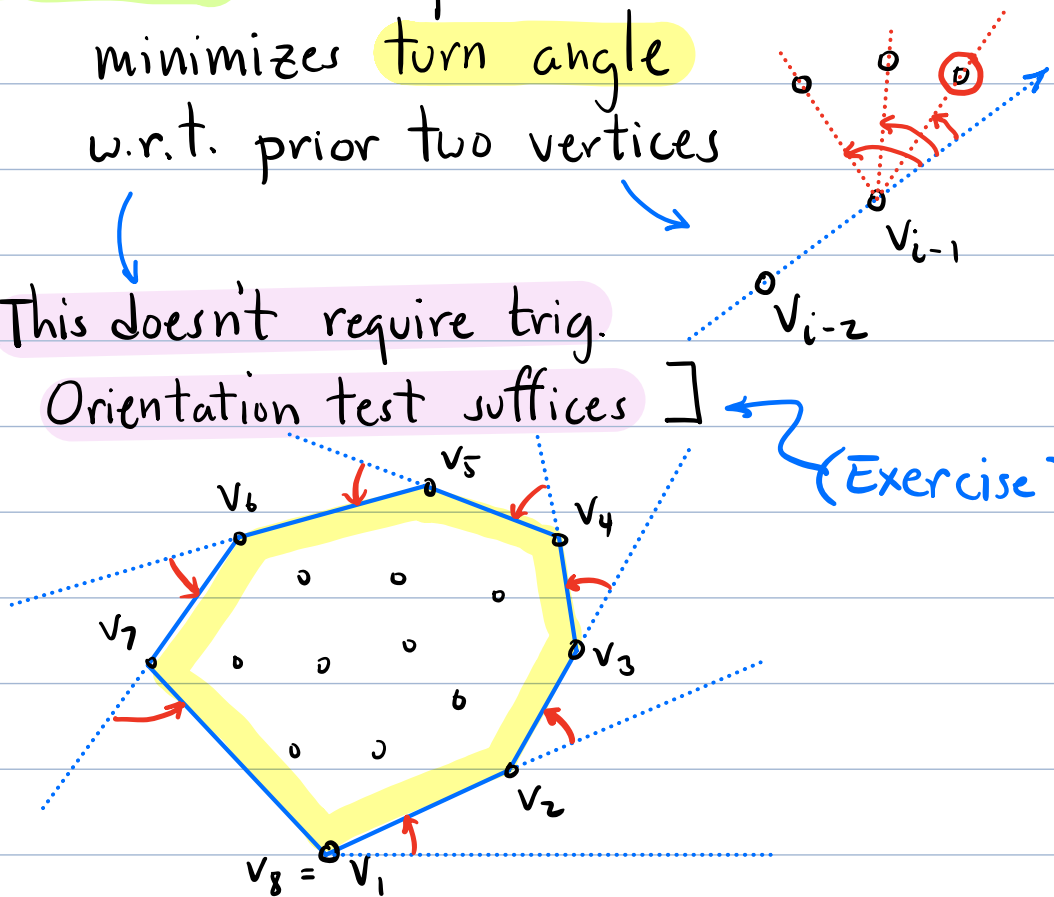
next vertex? The point of P that

minimizes turn angle
w.r.t. prior two vertices

[This doesn't require trig.

Orientation test suffices]

(Exercise)



Correctness: Easy

Running time: Compute $v_1 - O(n)$

Compute $v_i - O(n) \leftarrow$ Repeat h times

Total: $O((h+1)n) = O(h \cdot n)$

Summary:

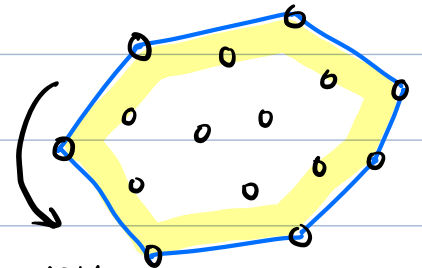
- Affine geometry + Orientations
 - Convex hulls in $\mathbb{R}^2$
 - Graham Scan
 - Divide + Conquer
 - Jarvis March
- $O(n \log n)$
- $O(h \cdot n)$
 $h = \text{size of hull}$

CMSC 754 - Computational Geometry

Lecture 3: Convex Hulls - Chan's Algorithm

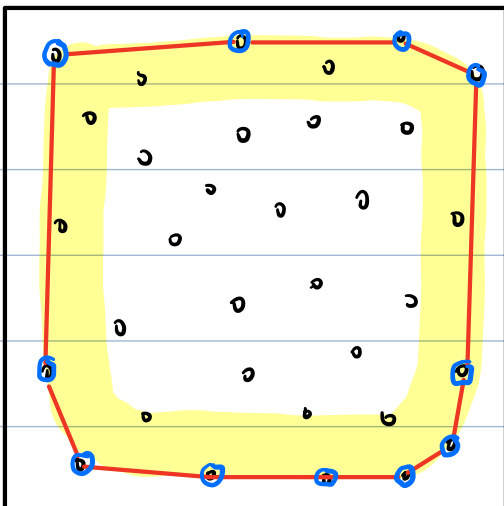
Recap:

- Given a pt. set $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^2$ compute $\text{conv}(P)$ - smallest convex set containing P .
- Output: Cyclic sequence of hull vertices
- Algorithms: Graham's Scan $O(n \log n)$, Divide & Conquer $O(n \log n)$, Jarvis March $O(n \cdot h)$



This Lecture:

- Can we beat $O(n \log n)$ time?
 - We'll give an $\Omega(n \log h)$ lower bound
- Can we achieve $O(n \log h)$?
 - Chan's algorithm



- Good when number of hull vertices h is very small

Theorem: Given n pts unif. distributed in a unit square $E[h] = \log n$

(Chan runs in $O(n \log \log n)$)

Lower bound for convex hulls:

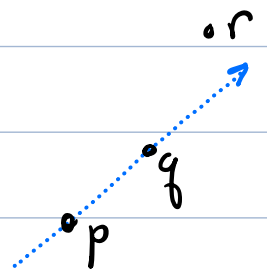
Conv: Given a set P of n pts in $\mathbb{R}^2$, compute the vertices of $\text{conv}(P)$ in **cyclic order**.

Def: An algorithm is **comparison-based** if its decisions are based on the sign of a fixed-degree polynomial function of inputs. (**Algebraic decision tree model**)

Almost all geometric primitives satisfy:

E.g. **if ($\langle p, q, r \rangle$ form a left-hand turn)**

$$\equiv \text{if} \left(\det \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix} > 0 \right)$$



$$\equiv \text{if} (f(p_x, p_y, q_x, q_y, r_x, r_y) > 0)$$

where:

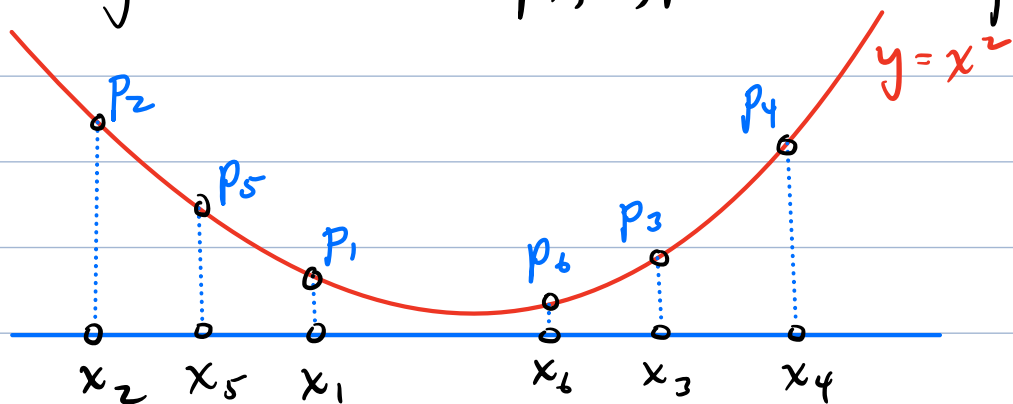
$$f(\dots) = (q_x r_y - q_y r_x) \\ - (p_x r_y - p_y r_x) \\ + (p_x q_y - p_y q_x)$$

A polynomial of
degree 2

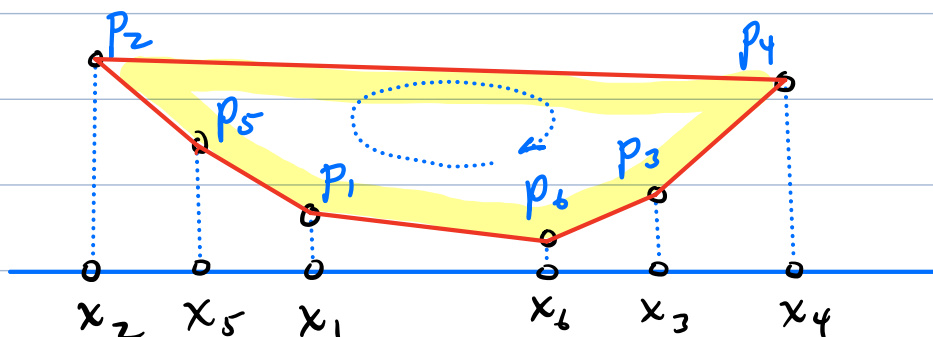
Theorem: Assuming a comparison-based algorithm, conv has a worst-case lower bound of $\Omega(n \log n)$

Proof: We will use the well-known fact that any comparison-based alg. for sorting reqs. $\Omega(n \log n)$ time in worst case.

We'll reduce sorting to conv. Given set $X = \{x_1, \dots, x_n\}$ to be sorted in $O(n)$ time we generate $P = \{p_1, \dots, p_n\}$ where $p_i = (x_i, x_i^2)$



If we compute $\text{conv}(P)$, the vertices appear in sorted order of X , up to reversal and adjusting starting point $\leftarrow O(n)$ time



Letting $T(n)$ denote the time to compute $\text{conv}(P)$, up to constant factors, we can sort X in time

$\overset{\text{compute } P \text{ from } X}{\uparrow}$

$n + T(n) + n$, which must be $\geq c \cdot n \log n$

$\uparrow$ reorient output

$$\Rightarrow T(n) \geq c \cdot n \log n - 2n \Rightarrow T(n) = \Omega(n \log n)$$

□

Obs: This exploits the fact that output is sorted cyclically. What if not?

Theorem: Assuming a comparison-based algorithm determining whether $\text{conv}(P)$ has h distinct vertices requires $\Omega(n \log h)$ time.

$\Rightarrow$ Just counting vertices reqs. $\log$ factor.

(See latex lecture notes for proof)

Output Sensitivity: Algorithm's running time depends on output size

$\rightarrow$ Is $O(n \log h)$ possible?

Yes!

Chan's Algorithm

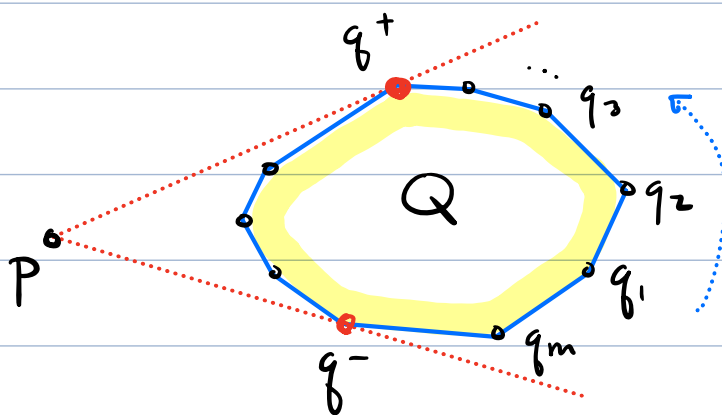
- combines - Graham scan $O(n \log n)$
+ Jarvis March $O(nh)$

Chan's Algorithm: An $O(n \log h)$ algorithm

- **Optimal** w.r.t. input size n + output size h
- Combines **two slow** algorithms (Graham + Jarvis) to make **faster** algorithm
- **Chicken + Egg:** Algorithm needs to know value of h - How is this possible?

Tangent Lemma:

Given a convex polygon Q given as a cyclic sequence of m vertices $\langle q_1, \dots, q_m \rangle$ and $p \notin Q$, can compute **tangent vertices** $q^- + q^+$ w.r.t. p in time $O(\log m)$



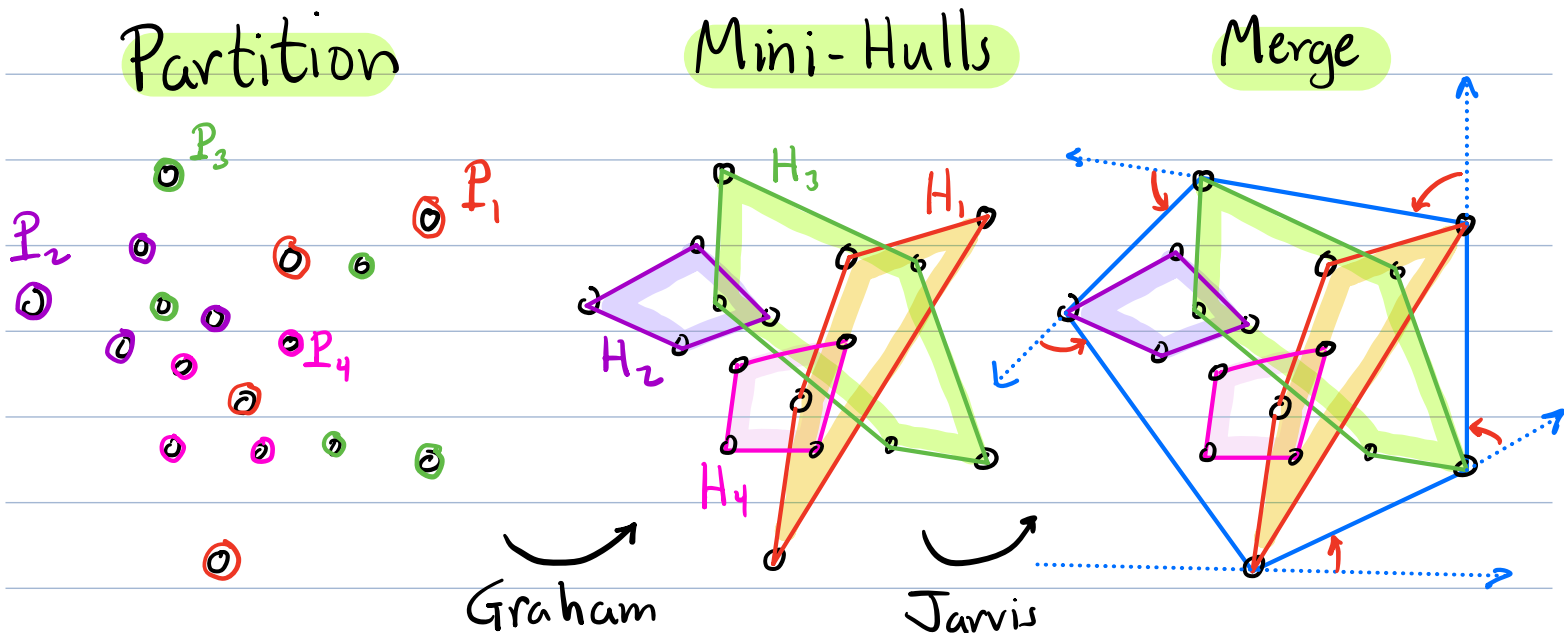
How? Exercise

Hint: Variant of **binary search**

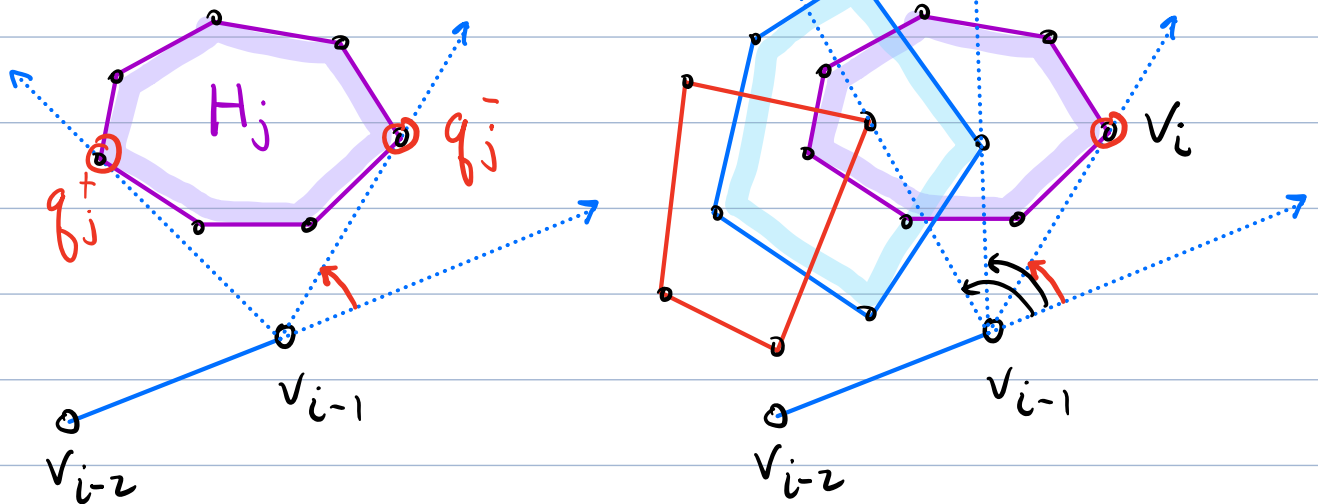
How to achieve $O(n \log h)$?

- Can't sort any set of size $\gg h$
- Guess the hull size - h^*
- Partition P into $\lceil n/h^* \rceil$ groups, each of size $\leq h^*$
 $\rightarrow P_1, \dots, P_k, k = O(n/h^*) \rightarrow O(n)$
- Run Graham on each group forming k "mini-hulls" $H_1, \dots, H_k \rightarrow O(k \cdot h^* \log h^*) = O(n \log h^*)$
- If we guess right ($h^* = h$) $\rightarrow O(n \log h)$
- Run Jarvis, but treat each mini-hull as a "fat point"
- use the utility function to compute turning angles

Example: Suppose $k=5$



Merging Mini-hulls:



- By the **Tangent Lemma**, compute tangents $q_j^- + q_j^+$ for each H_j in time $O(\log h^*)$
- Compute **all tangents** in time $O(k \cdot \log h^*)$
- $v_i \leftarrow$ tangent with smallest turning angle
- **Terminates after h iterations**

$\Rightarrow$ **Total merge time:** $O(h \cdot k \cdot \log h^*)$

$\rightarrow$ If we guess right ($h^* = h$) then

$$O(h^* \left(\frac{n}{h^*}\right) \log h^*) = O(n \log h^*) \\ = O(n \log h)$$

Summary: If we guess correctly ($h^* = h$) this computes $\text{conv}(P)$ in time $O(n \log h)$.

Conditional Hull (P, h^*):

Mini-hull Phase: $O(n \log h^*)$

Merge Phase: $O(n \frac{h}{h^*} \log h^*)$

If $h^* > h \Rightarrow$ Mini-hull phase is too slow

Note: Can tolerate a polynomial error. E.g. if $h \leq h^* \leq h^2$

$$\begin{aligned}\Rightarrow O(n \log h^*) &= O(n \log(h^2)) \\ &= O(2 \cdot n \log h) \\ &= O(n \log h) \text{ ok.}\end{aligned}$$

If $h^* < h \Rightarrow$ Merge phase too slow

- If Jarvis finds more than h^* hull pts - stop & return fail status

$\Rightarrow O(n \log h^*)$ time

Strategy:

Start small and increase until success

Arithmetic: $h^* = 3, 4, 5, \dots$ way too slow $\rightarrow O(n \cdot h \cdot \log h)$

Exponential: $h^* = 4, 8, 16, \dots, 2^i$ better $\rightarrow O(n \log^2 h)$

Double Exponential: $h^* = 4, 16, 256, \dots, 2^{2^i}$
best!

Note: $h_i^* = 2^{2^i}$ $h_i^* \leftarrow (h_{i-1}^*)^2$

Final Algorithm:

Chan Hull (P):

$h^* = 2$

repeat

$h^* \leftarrow (h^*)^2$

$\rightarrow h_i^* = 2^{2^i}$

$(\text{status}, \bar{V}) \leftarrow \text{conditionalHull}(P, h^*)$

until $(\text{status} == \text{success})$

return $\bar{V}$

Correctness: Already explained

Time:

- Running time per iteration $O(n \log h^*)$

- $h_i^* = 2^{2^i}$

- Stops when $h^* \geq h$

$2^{2^i} \geq h \Rightarrow i = \lceil \lg \lg h \rceil$ iterations

- Total time: [up to constants]

$$\sum_{i=1}^{\lg \lg h} n \cdot \lg(2^{2^i}) = n \sum_{i=1}^{\lg \lg h} 2^i$$

$$\leq 2n 2^{\lg \lg h} \quad [\text{Geom series}]$$

$$= 2n \lg h$$

$$= O(n \lg h)$$



Summary:

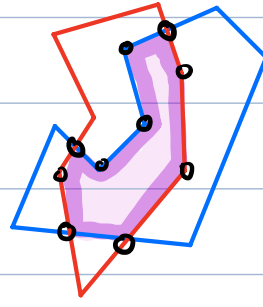
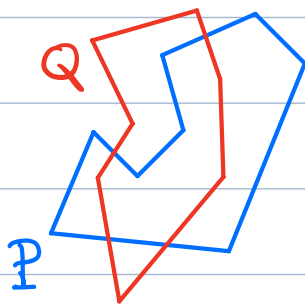
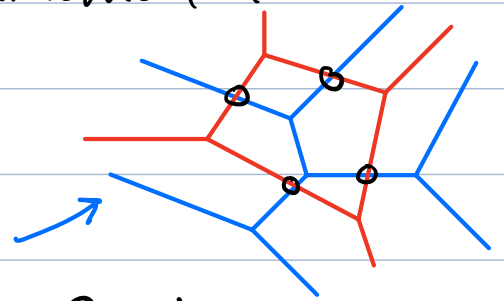
- Computing convex hulls
(in the algebraic decision tree model)
requires $\Omega(n \log n)$ + actually
 $\Omega(n \log h)$ time
- Chan's Algorithm
 - $O(n \log h)$ time
 - Combines Graham's scan
+ Jarvis March

CMSC 754 - Computational Geometry

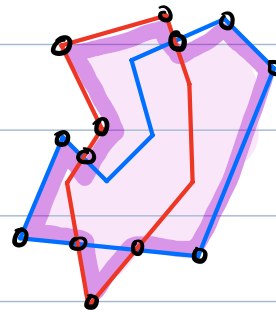
Lecture 4: Line Segment Intersection

Computing intersections is fundamental to geometric computation

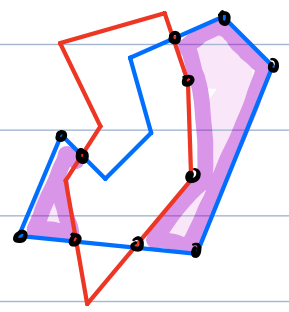
- collision detection
- subdivision overlay
- boolean operations - $\cap, \cup, \dots$



$P \cap Q$



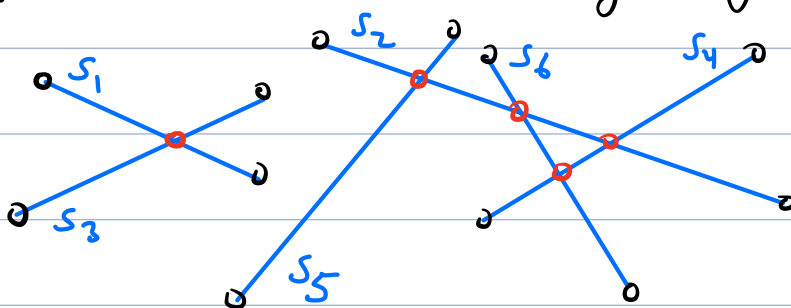
$P \cup Q$



$P \setminus Q$

Line Segment Intersection:

Given a set $S = \{s_1, \dots, s_n\}$ of line segments in $\mathbb{R}^2$ (where $s_i = \overline{p_i q_i}$), report all pairs of intersecting segments.



(s_1, s_3)

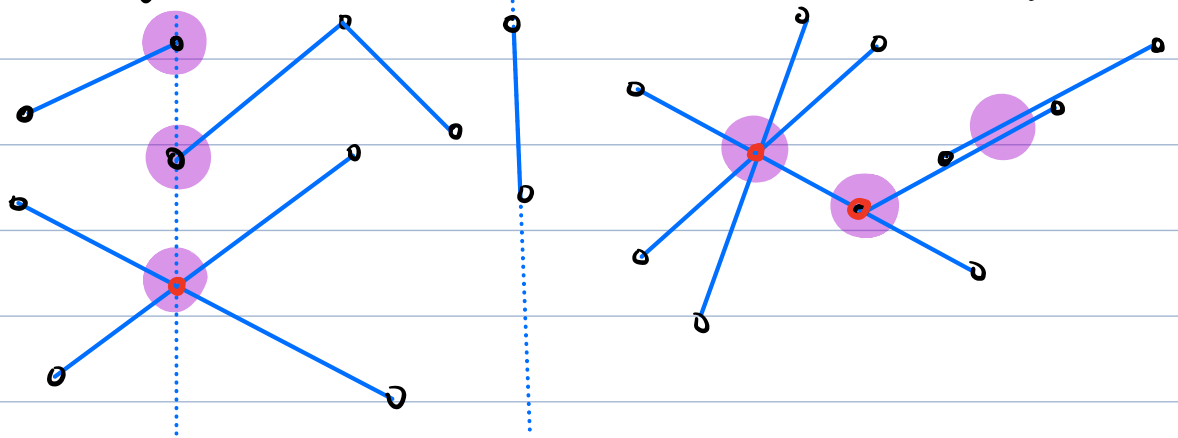
(s_2, s_5)

(s_2, s_6)

$\vdots$

General Position Assumptions:

- No duplicate x-coords
(for both endpoints + intersections)
- No segment endpt on another segment



Output Sensitivity:

Input size: n ($2n$ endpts, $4n$ coords)

Output size: m

$$0 \leq m \leq \binom{n}{2} = O(n^2)$$

Best possible: $O(m + n \log n)$

Follows from a lower bound
on element uniqueness

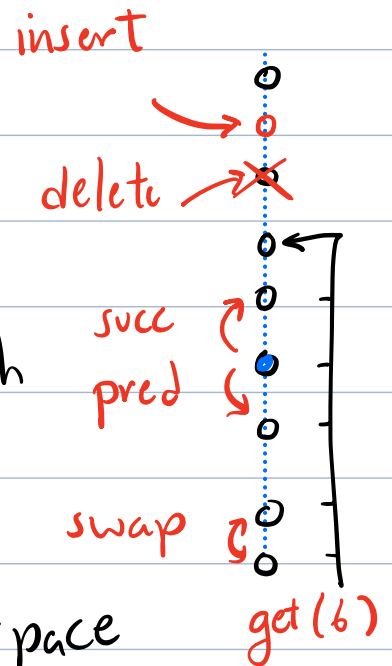
This lecture: $O((n+m) \log n)$

↳ Plane sweep

Utility Data Structures:

Ordered Dictionary: Supports:
insert, delete, find,
predecessor, successor, get-kth
swap adjacent

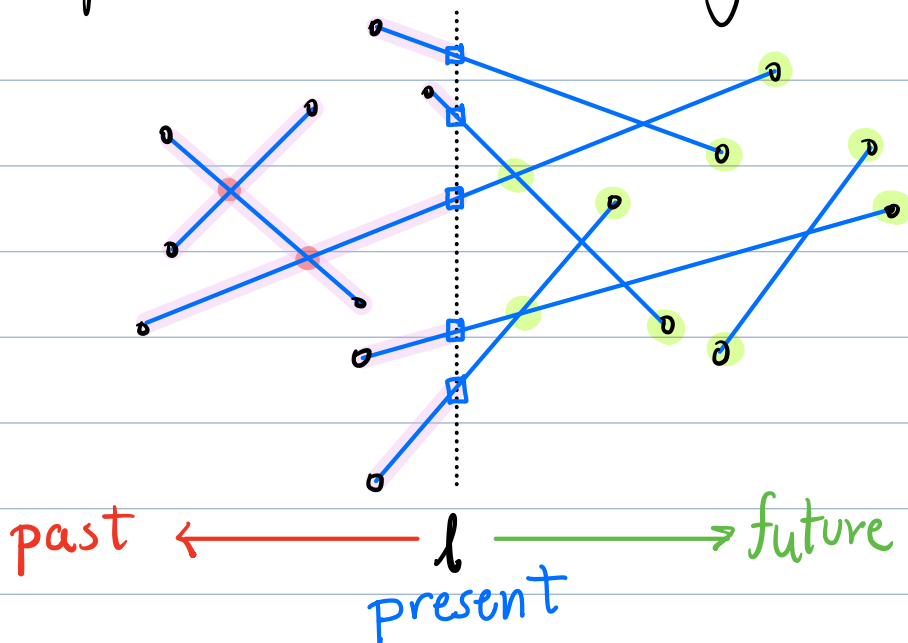
all in $O(\log n)$ time + $O(n)$ space



Priority Queue: Stores object σ + priority x
 $ref \leftarrow \text{enqueue}(\sigma, x)$
 $\sigma \leftarrow \text{extract_min}()$ - removes obj w.
 $\text{delete}(ref)$ min priority

Sweep-Line Algorithm:

Sweep a vertical line l from left to right
+ update solution as we go.

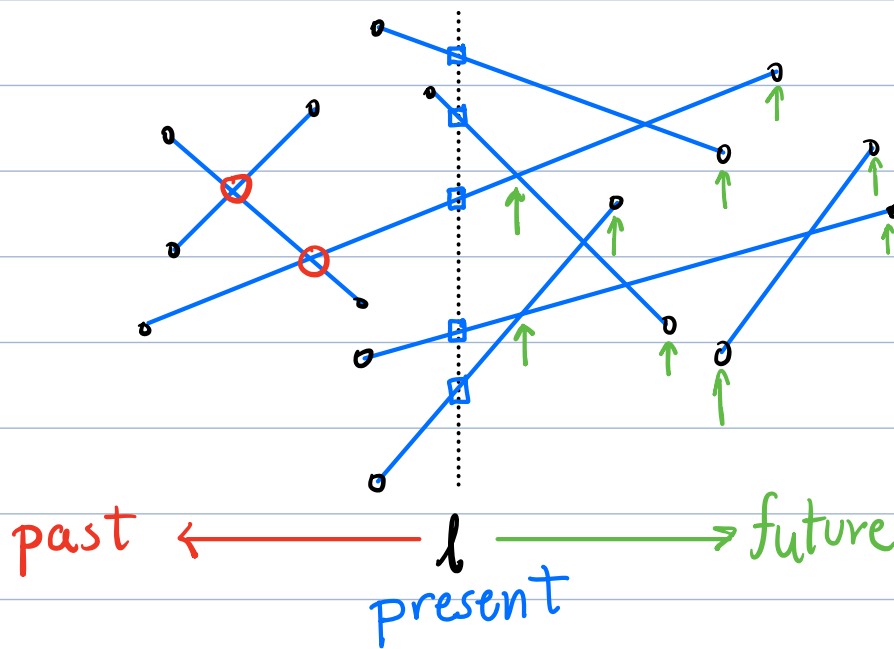


What we store: (Generic Plane Sweep)

(Past) Partial solution to left of l

(Present) Current status along l

(Future) (Known) Events to right of l



What we store: (For segment intersection)

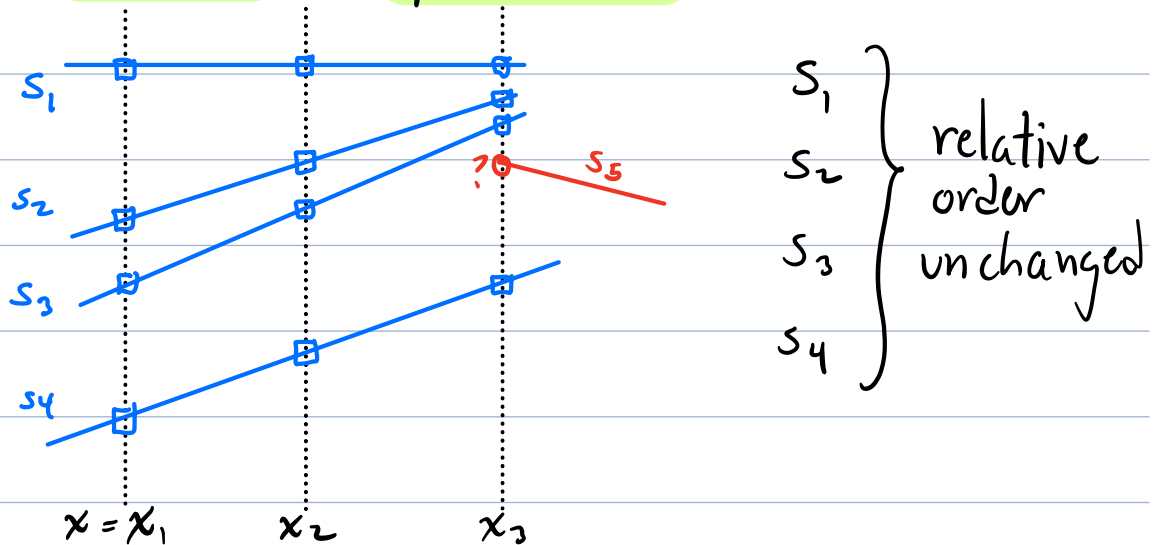
Past: List of intersecting pairs so far

Present: Ordered dictionary (top to bottom, say) of segments intersecting l
— sweep-line status

Future: Priority queue with future events:
- segment endpoints to right of l
- "imminent" intersections right of l

Sweep-Line Status:

- As l moves, all y -coordinates on sweep line change
- Much too slow to update all

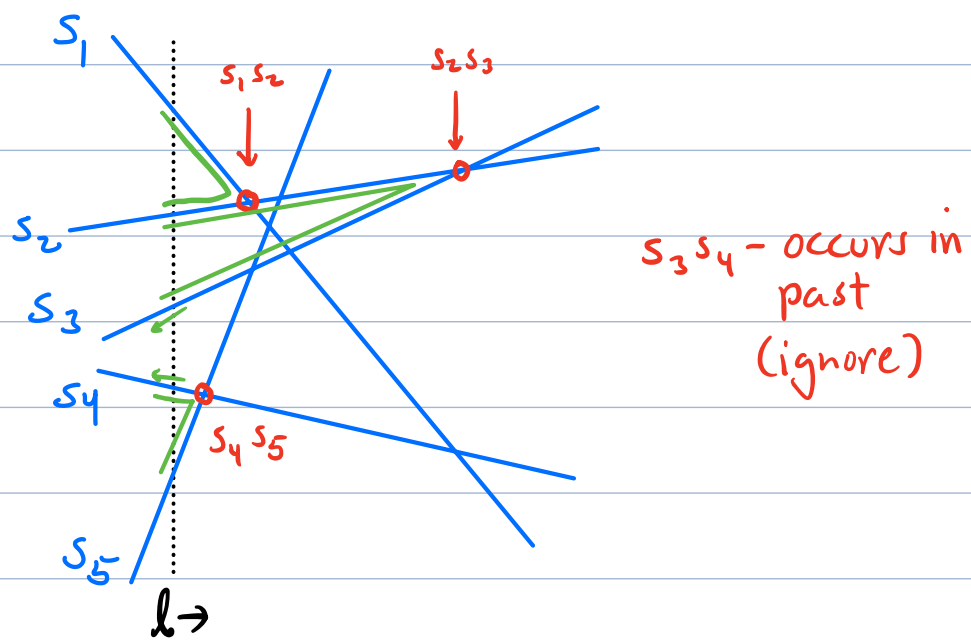


- **Dynamic comparator**: Rather than storing y coords in dictionary, store line equation: $y = ax + b$
- As x changes, **reevaluate** to compare y based on **current x value**

Future Events: (Stored in priority queue)

- All **segment endpoints** to right of sweep line
- **Imminent intersections**:

Intersections between pairs of lines that are **consecutive** on sweep line



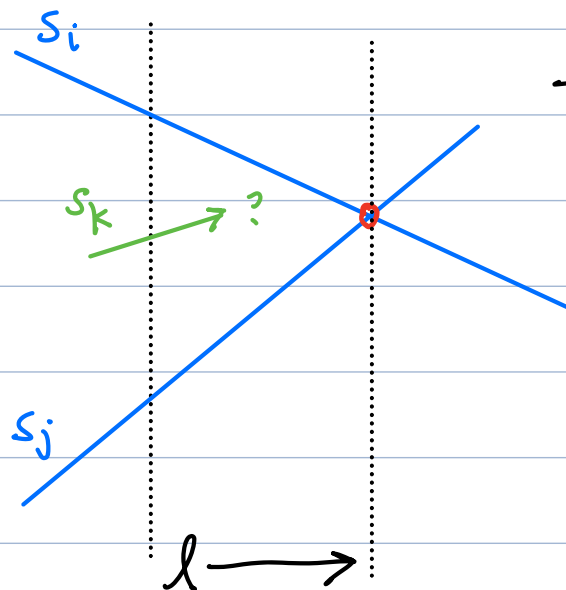
Why? - Consecutive pairs are easy to detect + update

- At most $n-1 = O(n)$ intersection events in priority queue
(+ $\leq 2n$ end pt events)

Lemma: If the next event is an intersection, these segments will be consecutive on the current sweep line.

Proof:

- Suppose not
- $s_i s_j$ is next event, but not consecutive

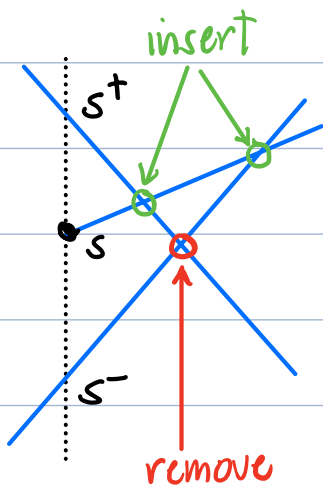


There must be an event involving s_k first

Final Sweep-Line Algorithm: $S = \{s_1, \dots, s_n\}$ $s_i = \overline{p_i q_i}$

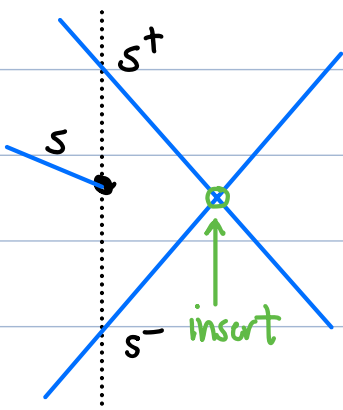
- Insert all seg. endpts into priority queue (sorted by x-coord)
- while (queue is non-empty) {
 - extract next event (min x)
 - cases:

Segment s left endpt:



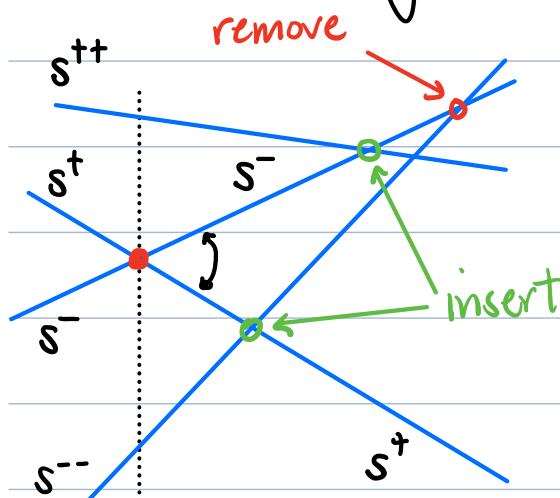
- Insert segment into sweep line (dictionary) based on y-coord
- Let $s^+ + s^-$ be segs just above and below
- If $s^+ s^-$ has intersection event, remove from priority queue
- Add to priority queue, intersection events for $ss^+ + ss^-$ (if appropriate)

Segment s right endpt:



- Let $s^+ + s^-$ be segments above + below
- Delete s from sweep-line
- Add to priority queue, intersect event for $s^+ s^-$ (if appropriate)

Segment s^+s^- intersection:



- Let s^{++} & s^{--} be segs above and below intersection

- Remove intersection events s^+s^{++} & s^-s^{--} (if exist)

- Swap s^+ & s^- on sweep line

- Add to prior. queue, intersect events for s^+s^{--} & s^-s^{++} (if appropriate)

Correctness: Easy, but be sure not to forget anything

Running Time: n = num. of segs. m = num. of intersects

Total events: $2n + m = O(n + m)$

Time per event: Extract min $\left\{ \begin{array}{l} O(1) \text{ dictionary ops} \\ O(1) \text{ queue ops} \end{array} \right\} O(\log n) \text{ total}$

Total time: $O((n + m) \log n)$

Space: $O(n)$ for data structures
 $O(m)$ for output

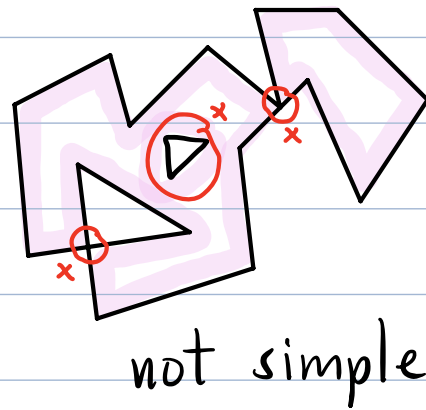
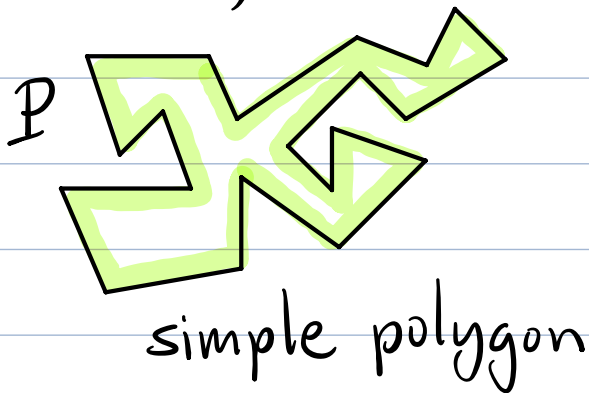
Summary:

- Plane-sweep algorithm to compute line segment intersections
- $O((n+m) \log n)$ time
 - $n = \text{no. of segs}$
 - $m = \text{no. of intersections}$
- Q: Can we do better?
 - $O(n \log n + m)$? (yes-later)
- Plane-sweep
 - Powerful technique
 - Keys:
 - Sweep-line status
 - Queue of future events
 - Update rules

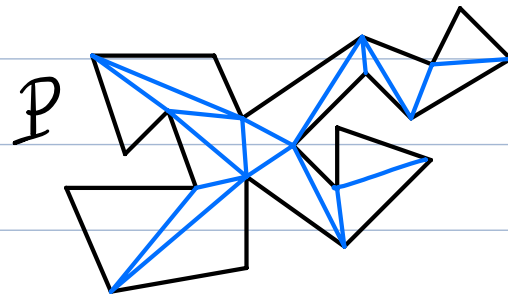
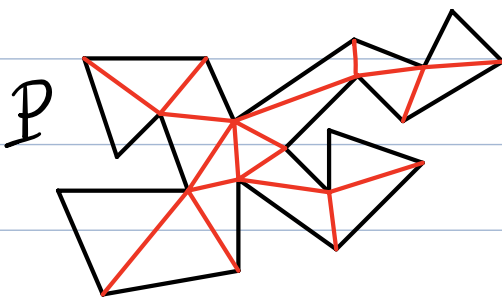
CMSC 754 - Computational Geometry

Lecture 5: Polygon Triangulation

Polygon Triangulation: Given a **simple polygon** P (that is, a simple, closed polygonal chain)...



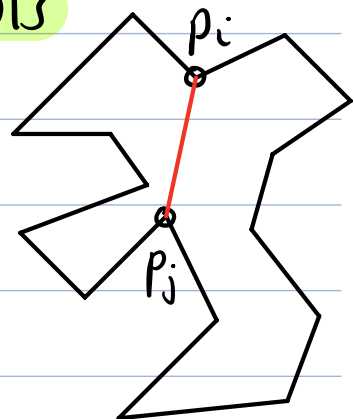
subdivide the interior of P into triangles
(vertices drawn from P 's vertices)



Notes: - P given as a **cyclic seq. of pts**

- Vertices p_i & p_j are **visible** if
open segment $\overline{p_i p_j} \subseteq \text{int}(P)$

- If p_i & p_j visible, segment $\overline{p_i p_j}$
called a **diagonal**



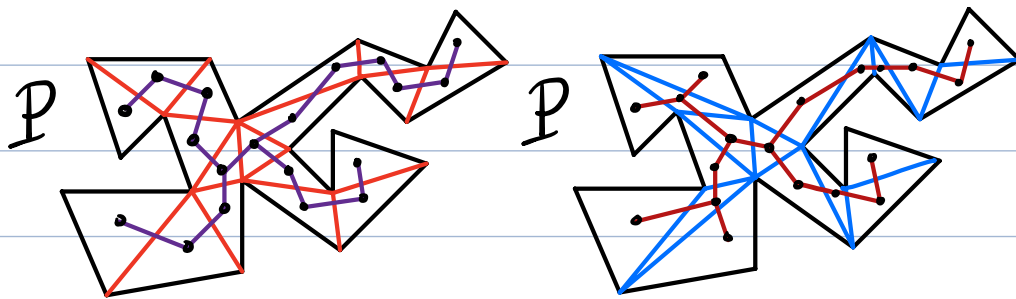
Lemma: Given any n -vertex simple polygon ($n \geq 3$)

- A triangulation exists
- Any triangulation has $n-3$ diagonals
- Any triangulation has $n-2$ triangles

Dual Graph: A triangulation defines a graph:

Vertices $\leftarrow$ triangles

Edges $\leftarrow$ adjacent (share common edge)



The dual graph of a polygon triangulation is **connected + acyclic** $\Rightarrow$ **tree**

History of Polygon Triangulation:

$O(n^2)$ - Easy (find a diagonal + recurse)

$O(n \log n)$ - We'll present this

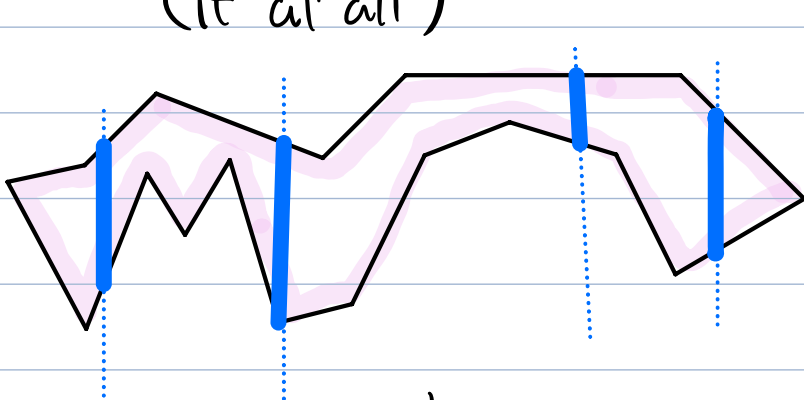
$O(n)$ - Chazelle 1991 (very complicated!)

Two steps:

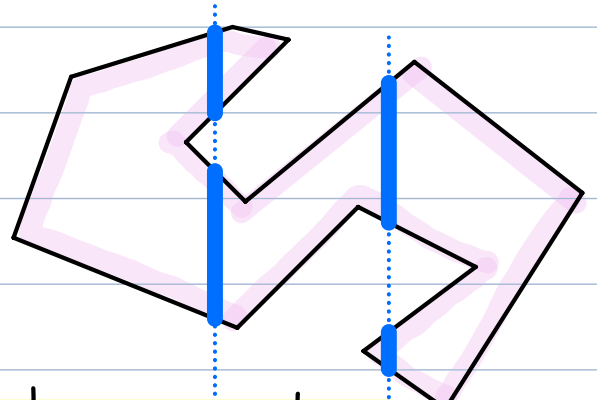
- ① Decompose the polygon into (simpler) polygons
- monotone polygons - $O(n \log n)$
- ② Triangulate each monotone polygon - $O(n)$

Output: Graph structure, called a doubly-connected edge list (DCEL) see text

Def: A polygon is x -monotone if any vertical intersects the polygon in a single segment (if at all)

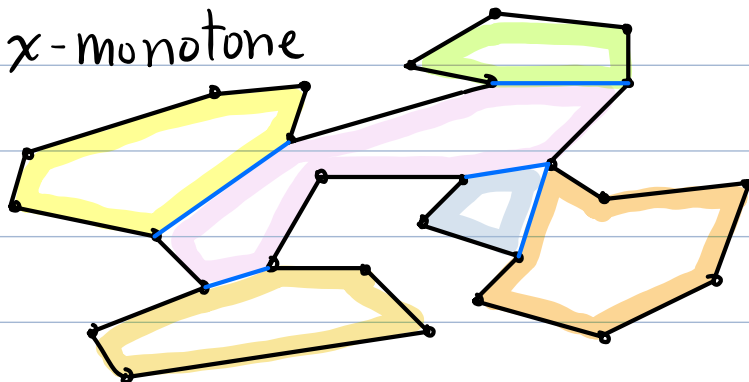


x -monotone



not x -monotone

Monotone Decomposition - Add (non-intersecting) diagonals so that connected components are all x -monotone

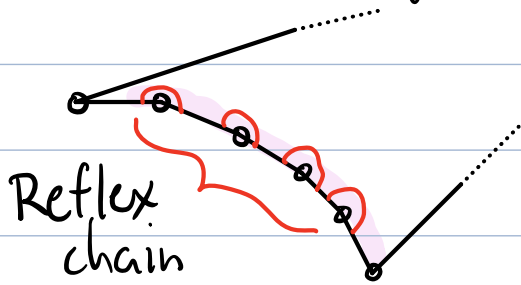


Triangulating a Monotone Polygon:

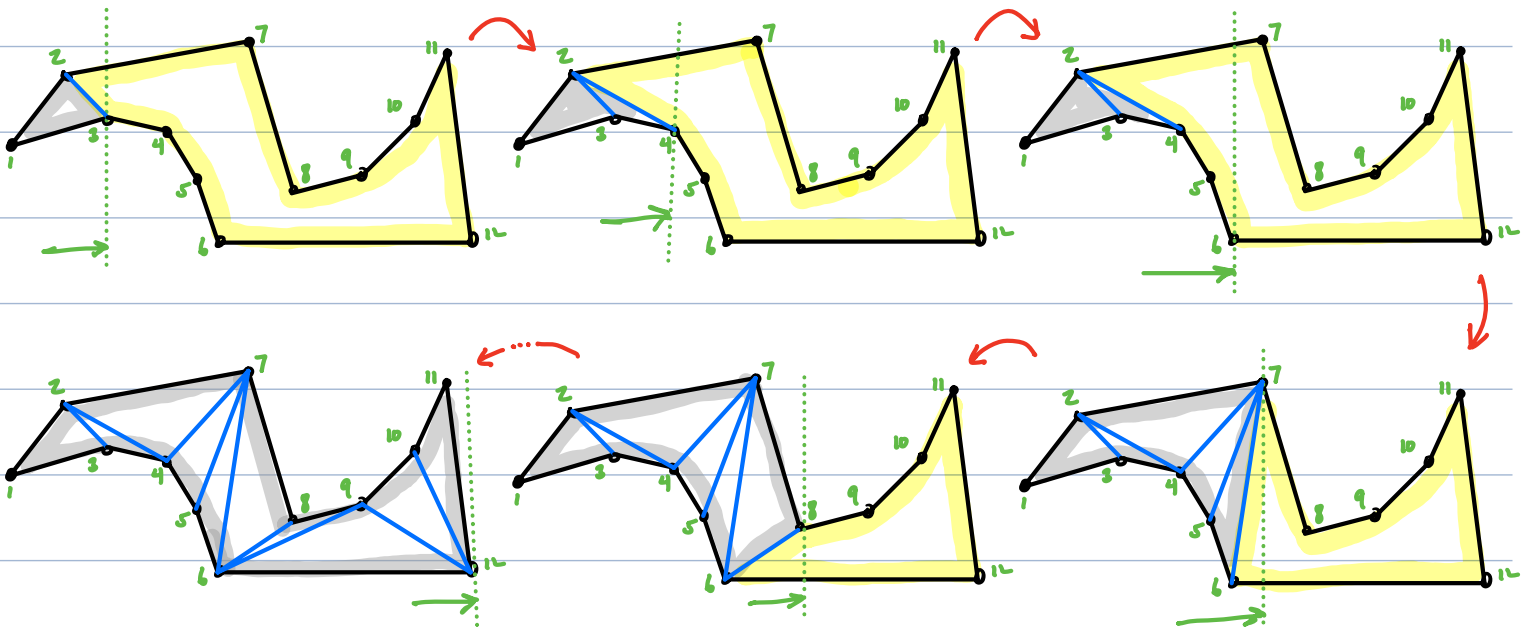
General position: No duplicate x-coords
(no vertical edges)

Reflex Vertex: Internal angle $\geq \pi$

Reflex Chain: Sequence of reflex vertices

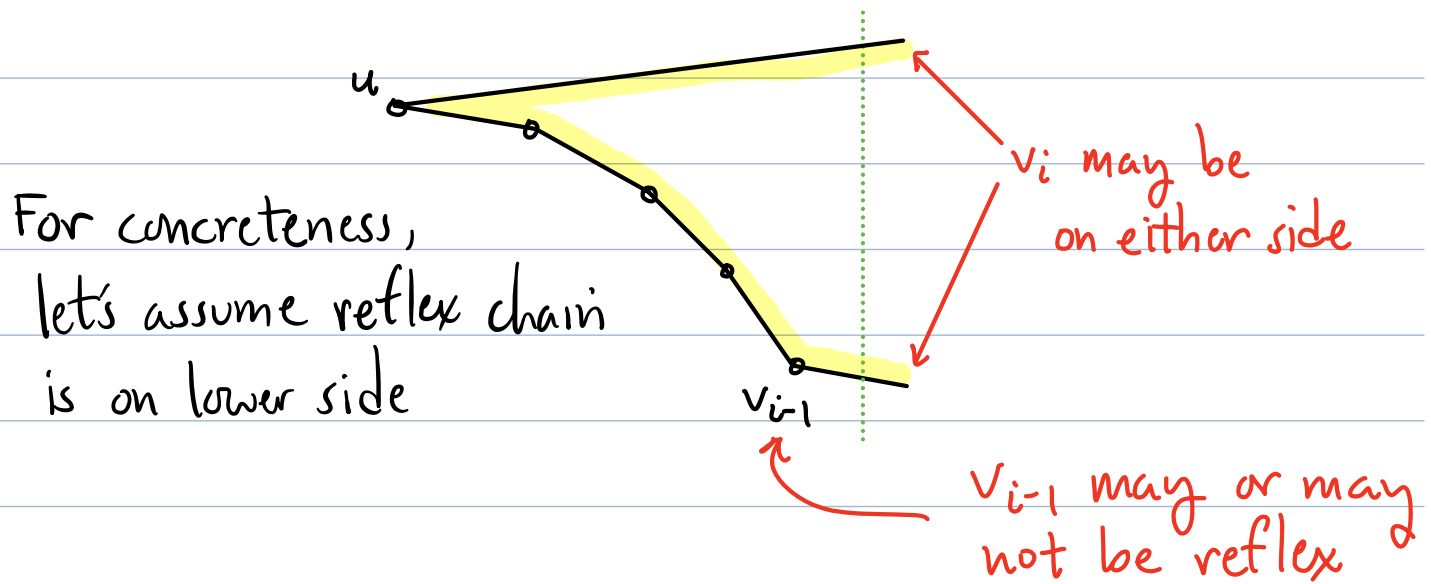


General approach: Sweep from left to right
+ triangulate as much as we can behind us.



What's the loop invariant?

Lemma: For $i \geq 2$, let v_i be the next vertex to process. The untriangulated region to left of v_i consists of two x -monotone chains starting from a common vertex u . One chain is a single edge, and the other is a reflex chain (of one or more edges).

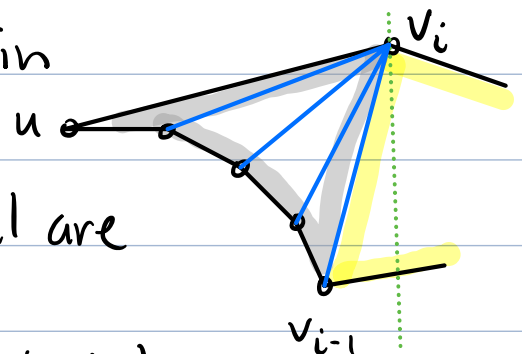


Case 1: (v_i lies on upper chain)

- add diagonals between v_i and all vertices of the chain

[By monotonicity, all are visible to v_i]

Now $u = v_{i-1}$. Reflex chain has just one edge.

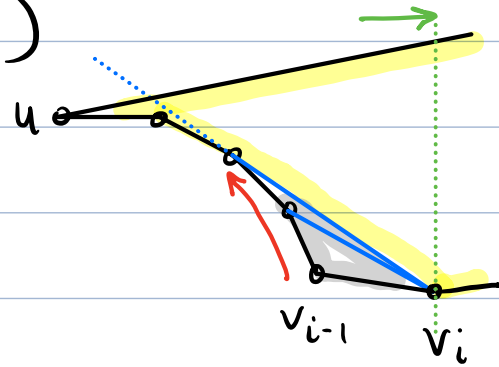


Case 2: (v_i lies on lower chain)

2a: (v_{i-1} is non-reflex)

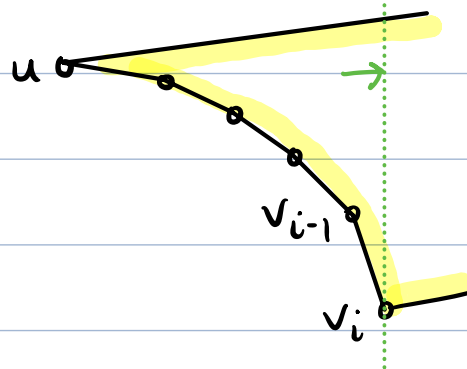
- connect v_i to all visible vertices on chain until hitting point of tangency. (Similar to Graham's scan)

[May go all the way back to u]



2b: (v_{i-1} is reflex)

- Add v_i to the chain



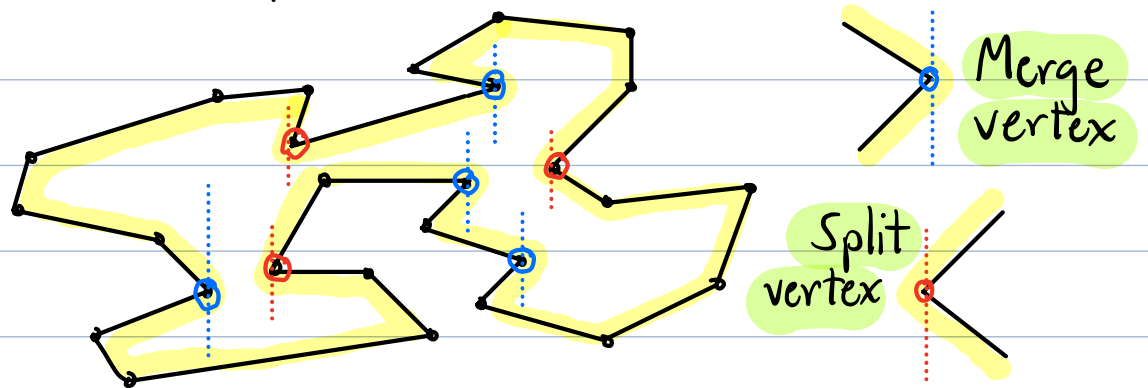
Correctness: Invariant holds after each iteration

Running time: $O(n)$ [As in Graham, once a vertex is removed from the chain, it never reappears]

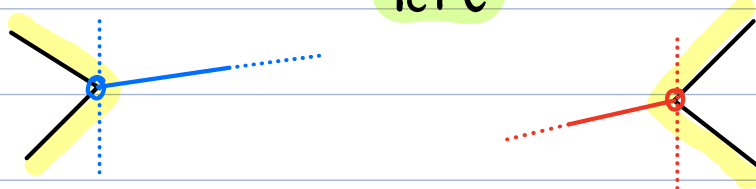
Monotone Subdivision:

Recall: Add diagonals to create x-monotone

Where? Scan reflex vertex: Reflex vertex where both edges on same side of vertical line.



Add a diagonal to right side of each merge
" " " " " " left split

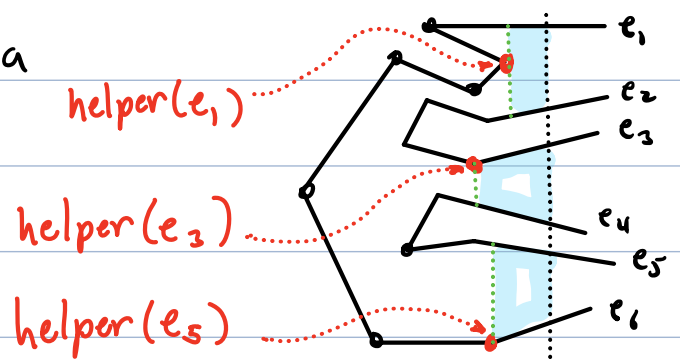


Plane-sweep Approach:

Need auxilliary info to help with diagonals
For each edge e_a of sweep line with $\text{int}(P)$ below:

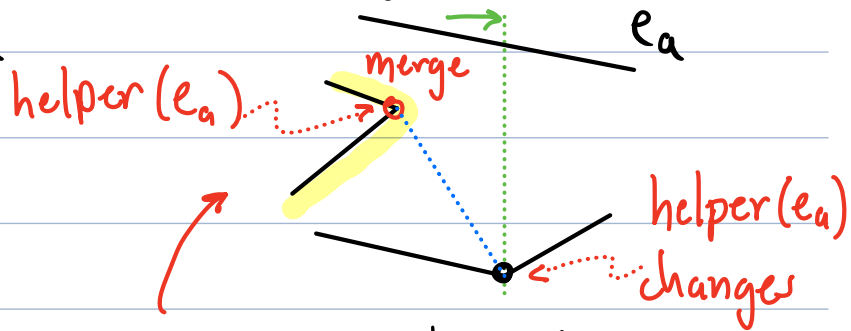
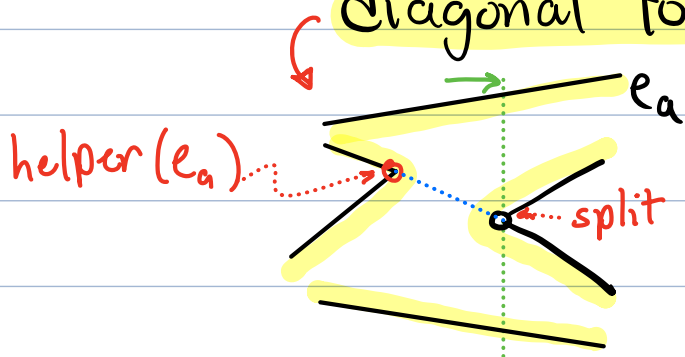
$\text{helper}(e_a)$ = rightmost vertically visible

vertex on or below e_a
to left of sweep line



Why is the helper helpful?

- When we see a **split vertex**, we **add diagonal to helper of edge above**



- When we see a **merge vertex**, it is the helper of edge above & we **connect it to next vertex where $helper(e_a)$ changes**

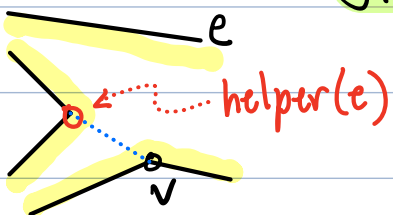
Events: Polygon vertices (sorted by x)

Sweep-line status: Edges intersecting the sweep line (ordered dictionary)

Event processing: There are many cases!

Utility:

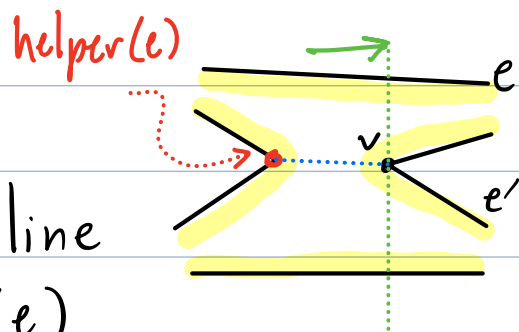
fix-up(v, e):



if ($helper(e)$ is a merge vertex)
add diagonal v to $helper(e)$

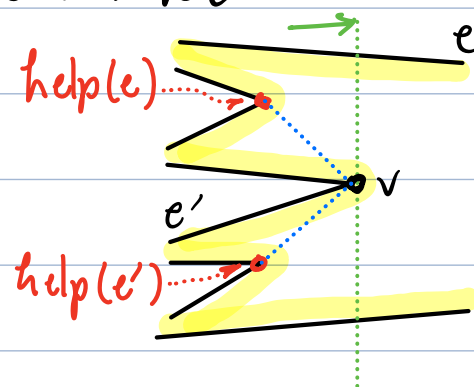
Split Vertex (v):

- $e \leftarrow$ edge above v in sweep line
- add diagonal v to $\text{helper}(e)$
- insert edges incident to v into sweep line
- letting e' be lower, set $\text{helper}(e') \leftarrow v$



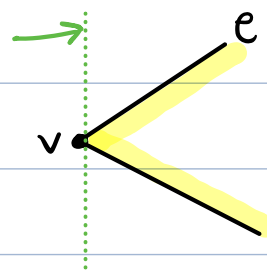
Merge Vertex (v):

- Consider two edges incident to v + let e' be lower one
- Delete both from sweep line
- Let e be edge above v
- $\text{fix-up}(v, e) + \text{fix-up}(v, e')$



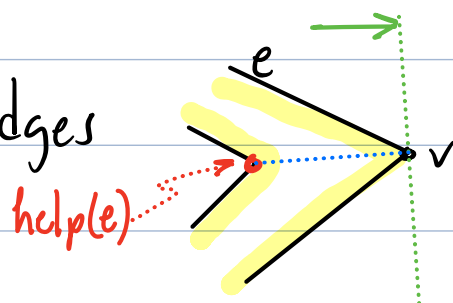
Start vertex (v):

- Insert v 's incident edges into sweep line
- Letting e be upper edge, $\text{helper}(e) \leftarrow v$



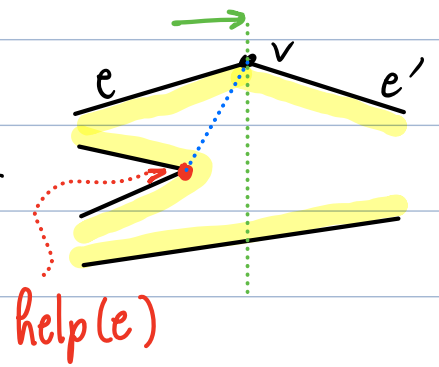
End vertex (v):

- Consider the two incident edges + let e be upper edge
- Delete both from sweep line
- $\text{fix-up}(v, e)$



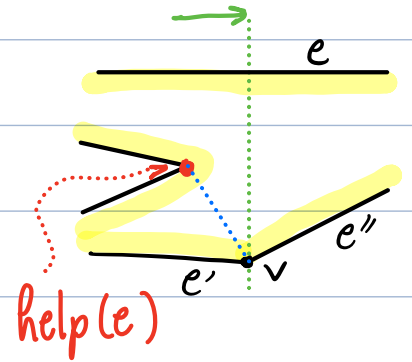
Upper-chain vertex (v):

- Let e be edge to left, e' to right
- $\text{fix-up}(v, e)$
- Replace e with e' in sweep line
- $\text{helper}(e') \leftarrow v$



Lower-chain vertex (v):

- Let e be edge above
- $\text{fix-up}(v, e)$
- Let e' be edge to left, e'' to right
- Replace e' with e'' in sweep line



Summary:

- Simple polygons
- Triangulation - Existence
 - Size
- Triangulating an x -monotone polygon $O(n)$
- Subdividing a polygon into x -monotone pieces - $O(n \log n)$
- Both based on plane sweep

CMSC 754 - Computational Geometry

Lecture 6: Halfplane Intersection & Duality

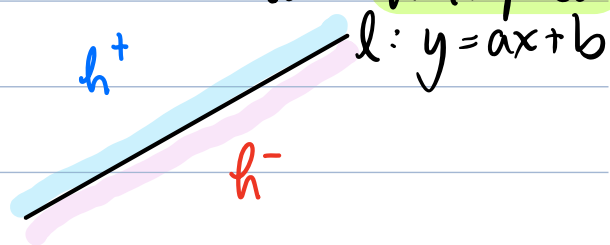
Halfplane Intersection:

Recall, each line in plane defines two halfspaces

$$l: y = ax + b$$

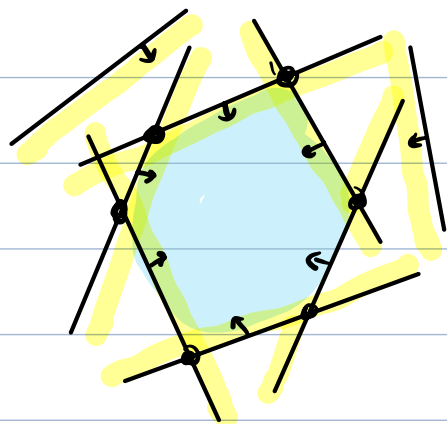
$$h^+: y \geq ax + b$$

$$h^-: y \leq ax + b$$

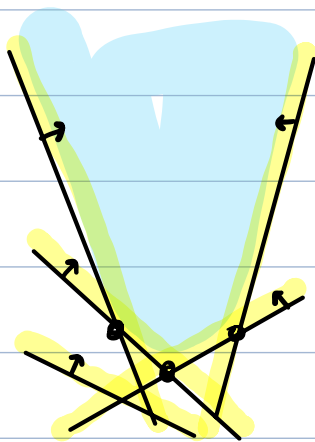


A halfspace is an (unbounded) convex set

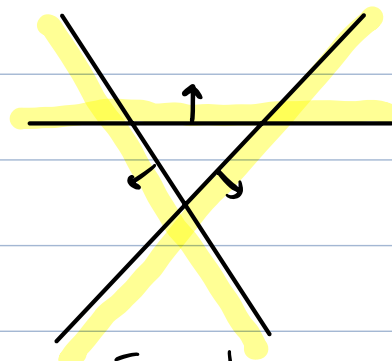
Given a set of halfspaces: $H = \{h_1, \dots, h_n\}$
their intersection $\bigcap h_i$ is a (possibly unbounded / possibly "empty") convex polygon



Bounded



Unbounded



Empty

Representing lines (and more):

$\mathbb{R}^2$ (Line)

$\mathbb{R}^d$ (Hyperplane)

Explicit:
 $y = f(x)$

$$y = ax + b$$

$$x_d = \sum_{i=1}^{d-1} a_i x_i + b$$

Implicit:

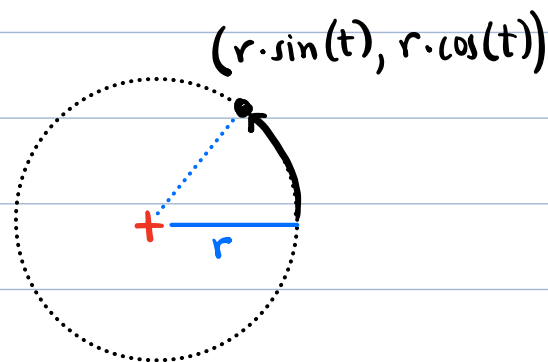
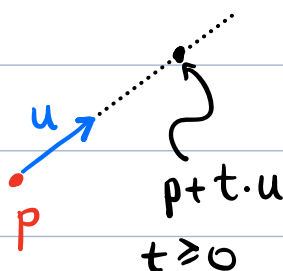
$$f(x, y) = ax + by + c$$

$$f(x, y) = 0$$

$$f(x_1, \dots, x_d) = \sum_{i=1}^d a_i x_i + b$$

Parametric:

$$(x(t), y(t))$$

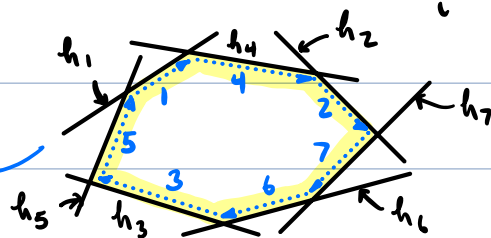


Halfplane Intersection:

Given halfplanes $H = \{h_1, \dots, h_n\}$ construct $H = \bigcap h_i$

Output: Sequence of edges

$\langle 5, 1, 4, 2, 7, 6, 3 \rangle$



Divide and Conquer Algorithm: $O(n \log n)$

Intersect(H) {

- if ($|H| = 1$) return h_1 [single halfspace]

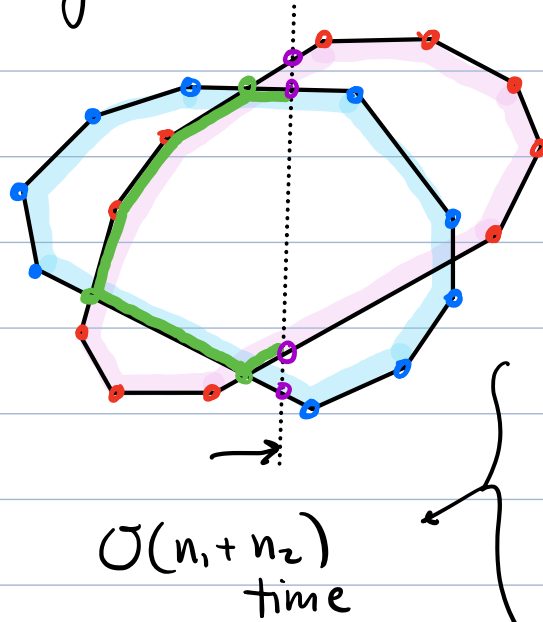
- else

partition $H \rightarrow \begin{cases} H_1 \\ H_2 \end{cases} \quad |H_i| \leq \frac{n}{2}$

$I_1 \leftarrow \text{Intersect}(H_1); I_2 \leftarrow \text{Intersect}(H_2)$

return merge(I_1, I_2) $\leftarrow$ How?

How to merge? Plane sweep



- At most 4 segments hit sweep line
- $\leq n_1 + n_2$ end pt events
 $n_i = |H_i|$
- $\leq 2(n_1 + n_2)$ intersection events
- Boundaries are already sorted

Overall Running Time:

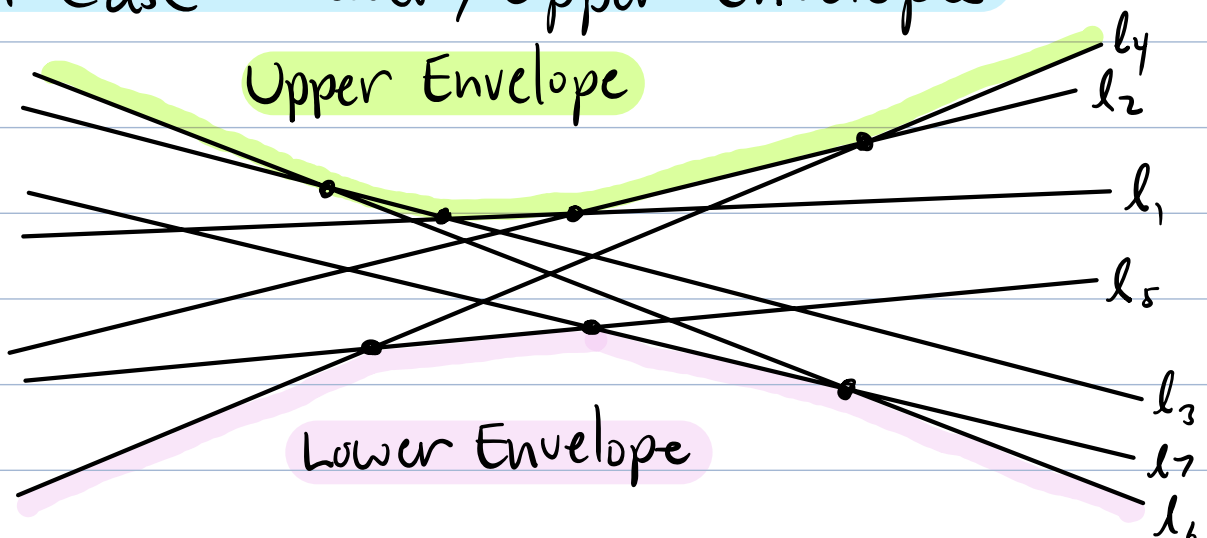
$$T(n) = 2T(n/2) + n$$

2 recursive calls on $n/2$ halfspaces

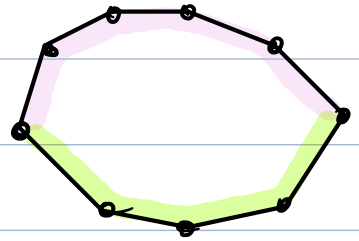
merge in linear time

$$= O(n \log n) \quad [\text{see, e.g., CLRS}]$$

Special Case: Lower/Upper Envelopes



Envelopes of lines $\sim$ Hull of points Related?



Point-Line Duality

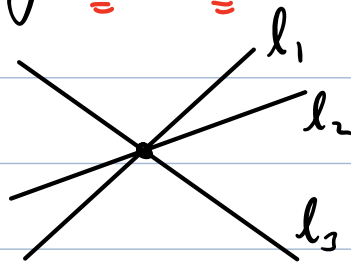
Lines in $\mathbb{R}^2$ are a lot like points:

2 degrees
of freedom

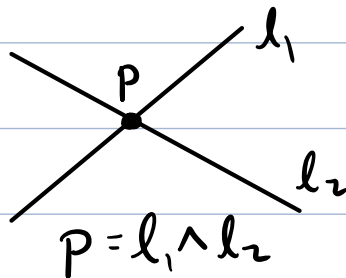
$$y = \underline{a}x + \underline{b}$$

$$p: (\underline{a}, \underline{b})$$

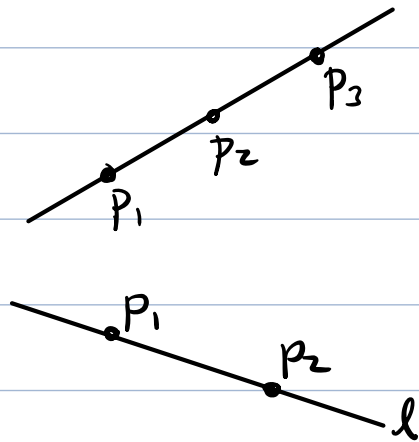
degeneracy:



incidence:



Two lines meet
at a point



Two points join
to form a line

Dual Operator:

Given point $p = (a, b)$

$$a, b \in \mathbb{R}$$

line $l: y = cx - d$

$$c, d \in \mathbb{R}$$

Dual p^* is the line $y = a \cdot x - b$

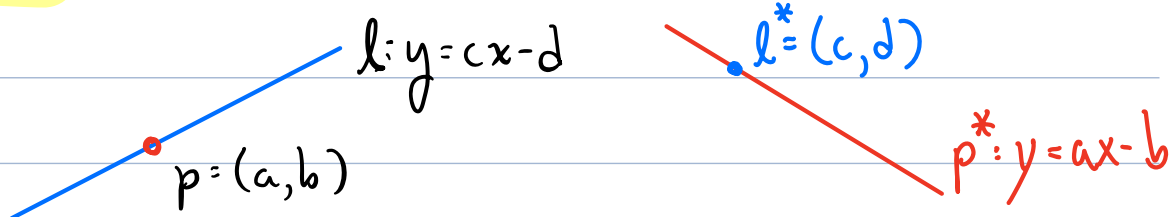
l^* is the point (c, d)

Observations:

Self-inverse: $p^{**} = p$ $l^{**} = l$

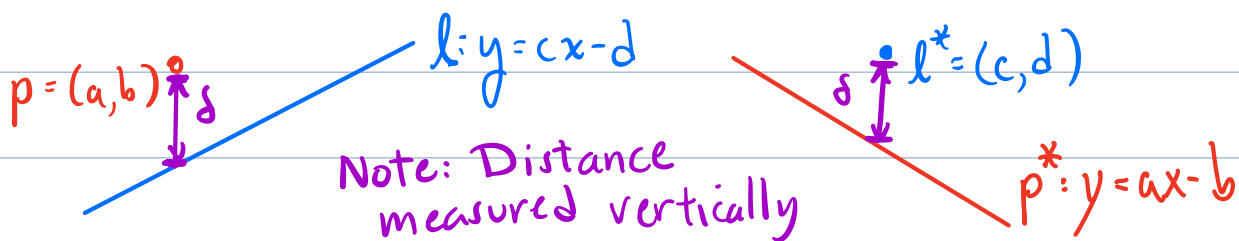
Incidence: p lies on l iff l^* lies on p^*

Proof: $b = c \cdot a - d \Leftrightarrow d = a \cdot c - b$



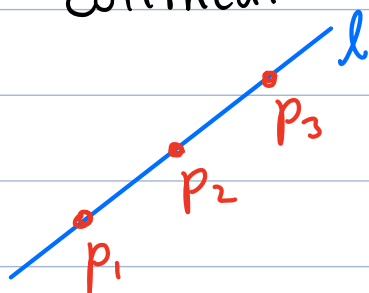
Order reversing: p is at dist δ above l iff l^* is at dist δ above p^*

Proof: $b = c \cdot a - d + \delta \Leftrightarrow d = a \cdot c - b + \delta$

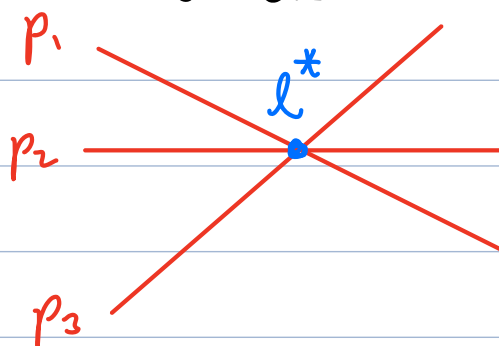


Degeneracy:

p_1, p_2, p_3 are collinear



iff p_1^*, p_2^*, p_3^* are coincident

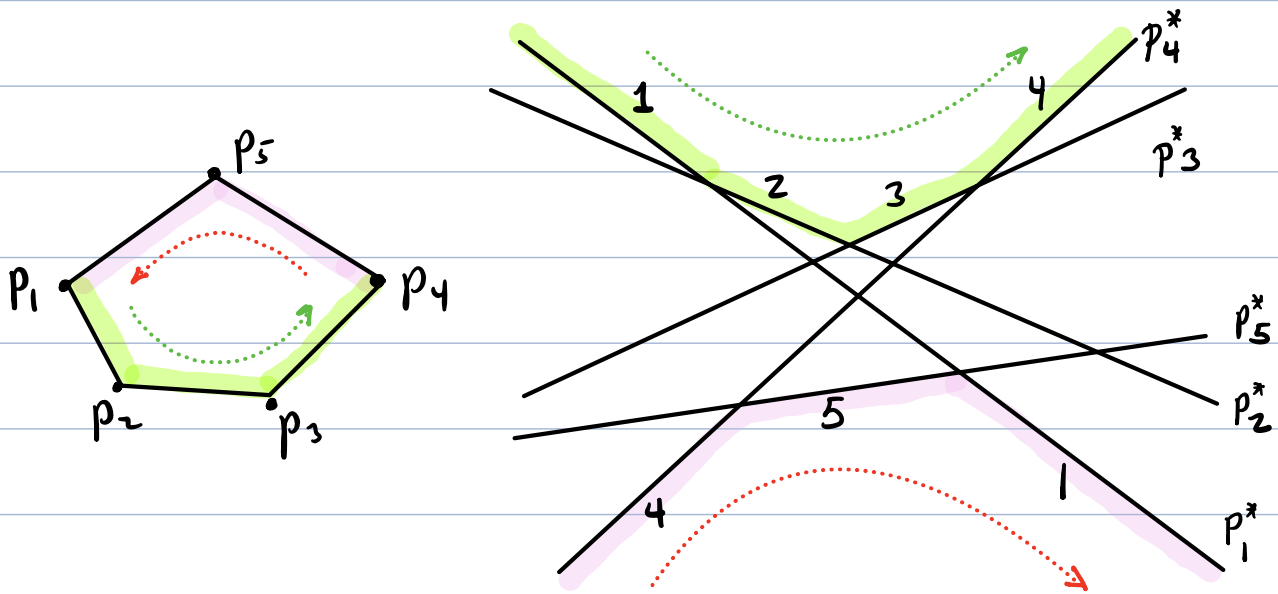


Hulls and Envelopes:

Lemma:

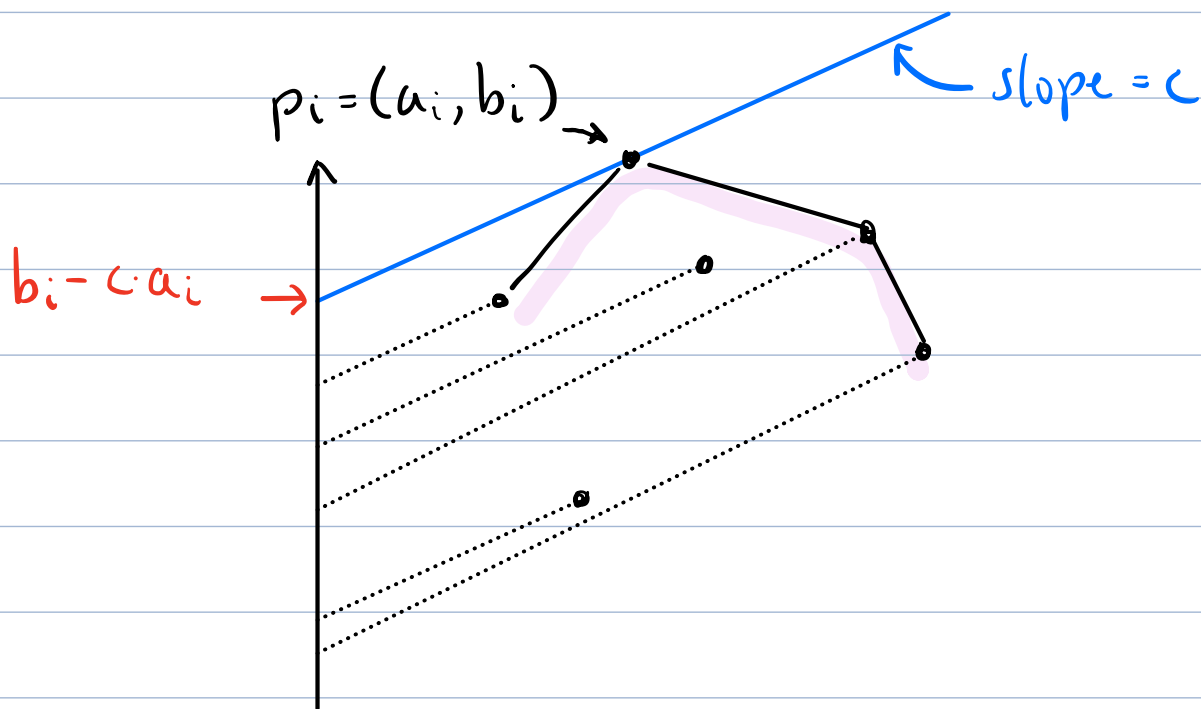
Given a set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^2$, point p_i is on upper (lower) hull of P iff p_i^* contributes an edge to lower (upper) envelope of P^* .

Further - CCW order of vertices on hull $\equiv$ left-to-right order of edges on env.



Proof: We'll show p_i on upper hull iff p_i^* contributes edge to lower envelope (other case is symmetrical)

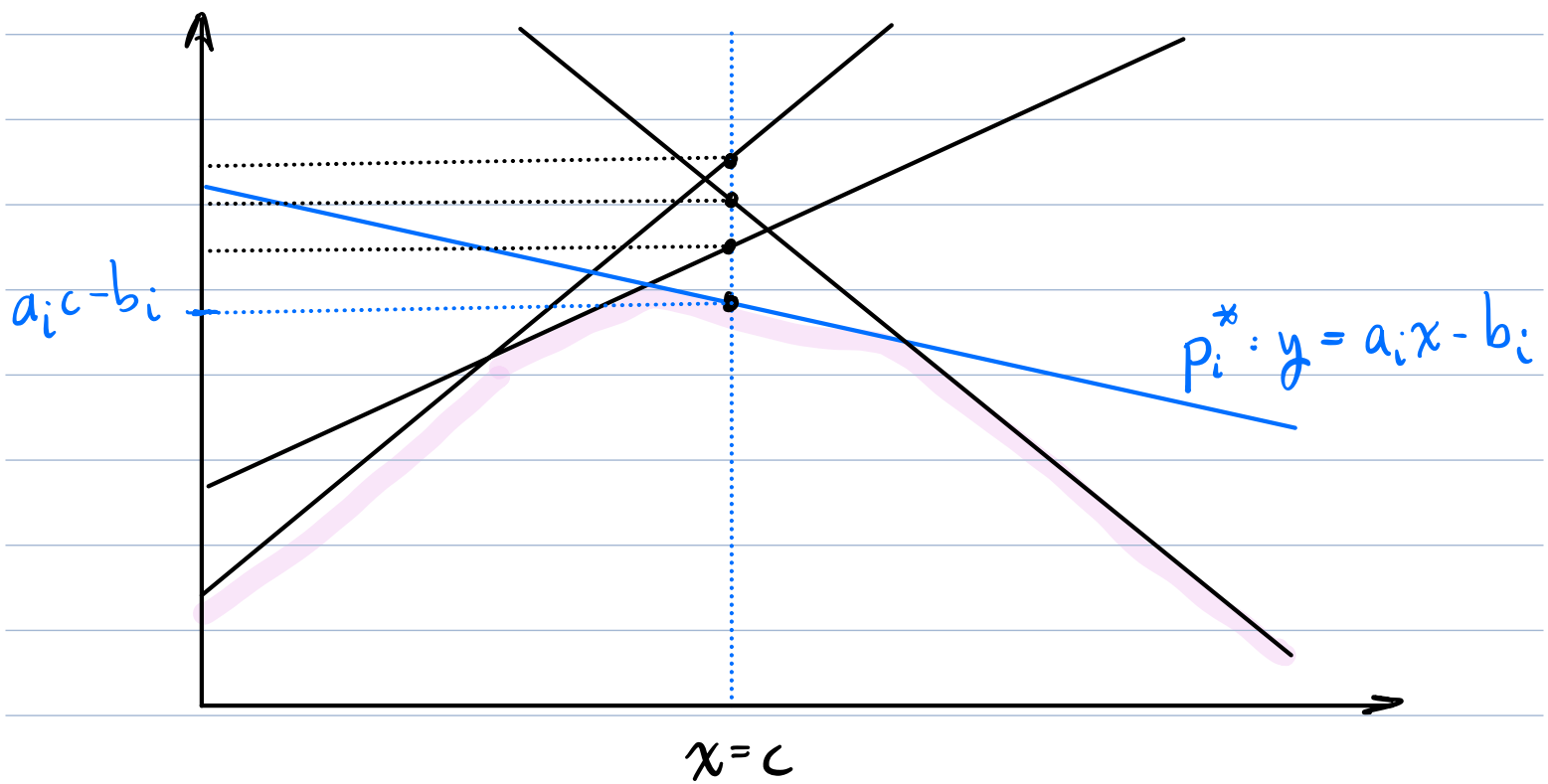
- p_i is vertex of upper hull iff there is a supporting line l of some slope c through p_i



$\Leftrightarrow$ The line with slope c through p_i has max y -intercept

$$\Leftrightarrow i = \operatorname{argmax}_{1 \leq j \leq n} (b_j - c \cdot a_j) \quad (1)$$

- p_i^* contributes an edge to lower envelope iff p_i^* is lowest line to hit the vertical line $x=c$ for some c .

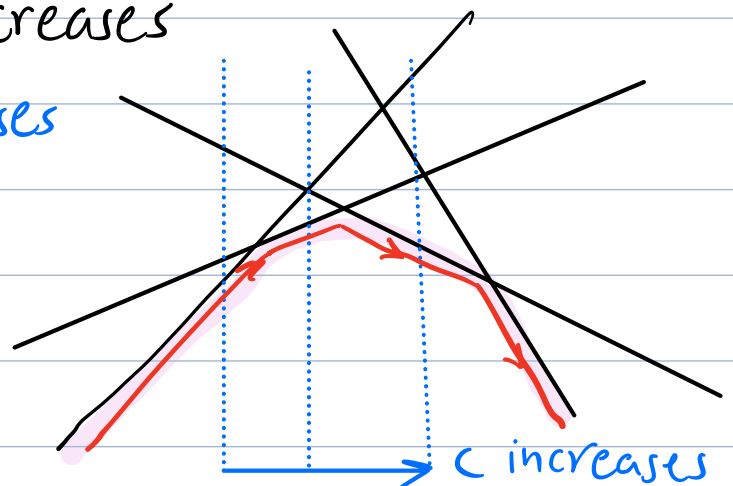
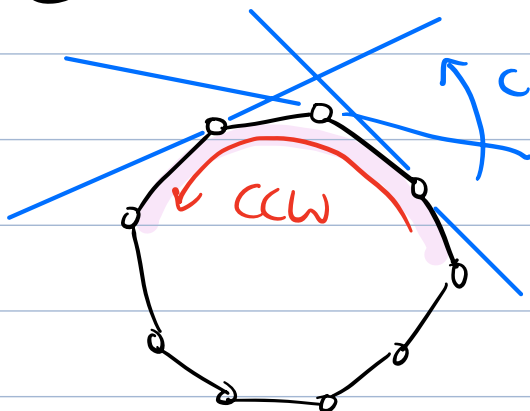


$$\Leftrightarrow i = \operatorname{argmin}_{1 \leq j \leq n} a_j c - b_j$$

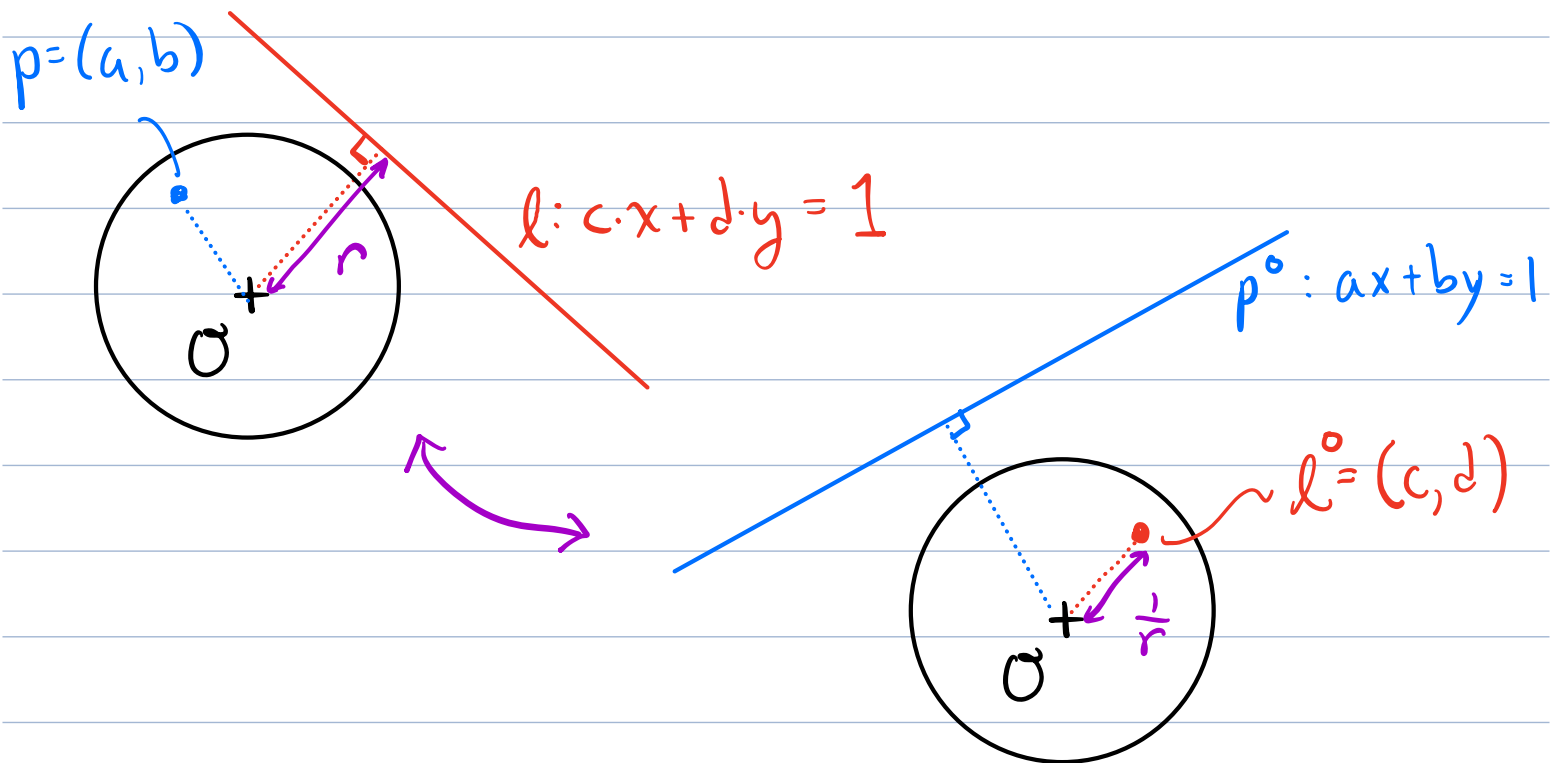
$$i = \operatorname{argmax}_{1 \leq j \leq n} (b_j - c a_j) \quad (2)$$

But (1) $\equiv$ (2)

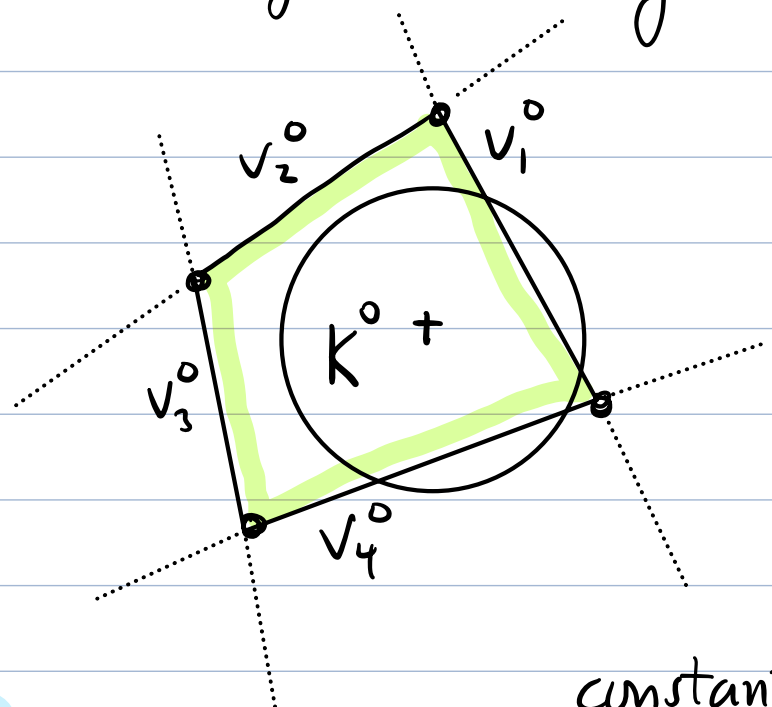
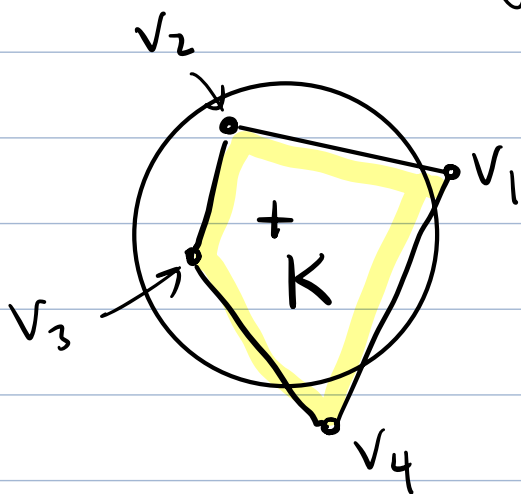
Order? As c increases



Polar Dual: Another dual transform



If K is a convex body containing the origin



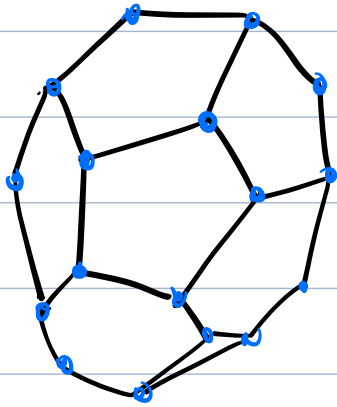
Mahler Volume:

$$\text{vol}(K) \cdot \text{vol}(K^\circ) = \Theta(1)$$

constant
depend.
on
dimension

Higher Dimensions:

Let K be a polytope in $\mathbb{R}^d$



For $0 \leq i \leq d-1$
 $f_i = \text{num. of faces of dim } i$

$i=0$ - vertices

$=1$ - edges

$\vdots$

$=d-1$ - facets

McMullen's Upper-Bound Thm:

- A polytope in $\mathbb{R}^d$ with n vertices
may have $O(n^{\lfloor d/2 \rfloor})$ facets

- n facets...

.... $O(n^{\lfloor d/2 \rfloor})$ vertices

Tight - Cyclic Polytopes

Summary:

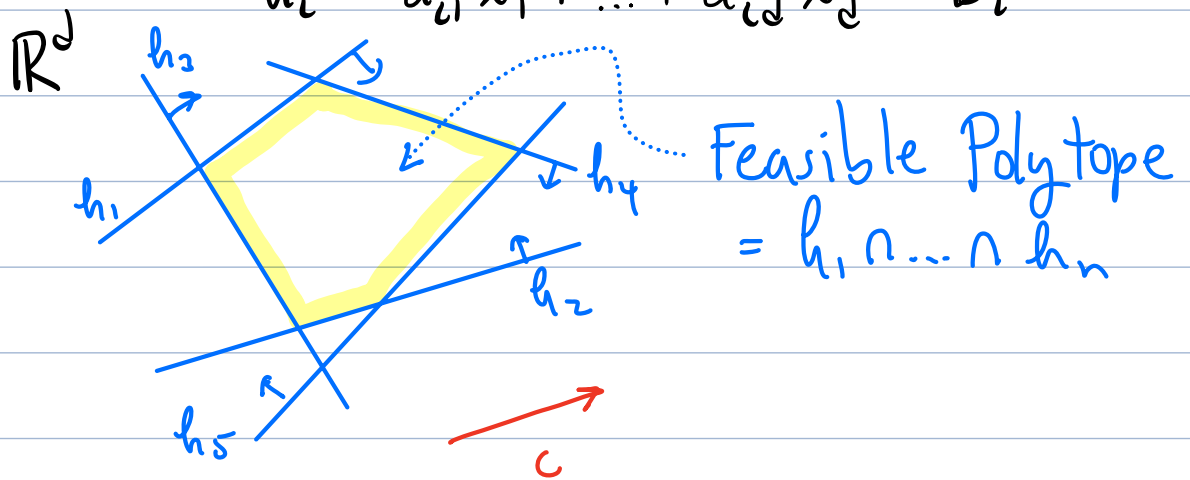
- Halfplane intersection - $O(n \log n)$
- Special case - Upper/Lower envelopes
- Point-line duality
 - Incidence/Order properties
- Upper/Lower hulls $\equiv$ Lower/Upper envelopes
- Polar duality
- Polytopes in $\mathbb{R}^d$

CMSC 754 - Computational Geometry

Lecture 7: Linear Programming

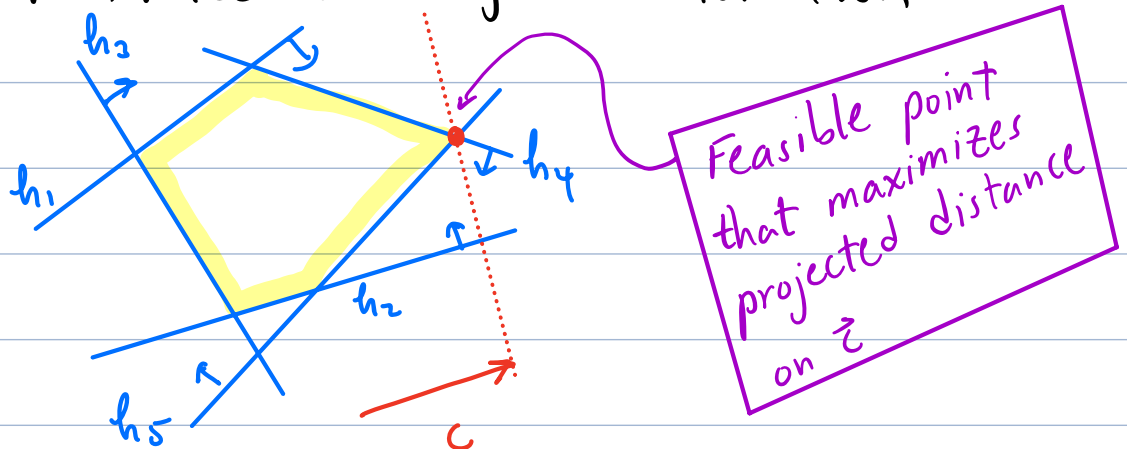
Linear Programming (LP):

- Fundamental optimization problem in $\mathbb{R}^d$
- Given a set of n linear constraints (halfspaces) $H = \{h_1, \dots, h_n\}$
 $h_i: a_{i1}x_1 + \dots + a_{id}x_d \leq b_i$



- Given a linear objective function
 $f(\bar{x}) = c_1x_1 + \dots + c_dx_d = c^T \bar{x}$

LP: Find the vertex of the feasible polytope that maximizes the objective function



Matrix form:

Given $c \in \mathbb{R}^d$ and $n \times d$ matrix A and $b \in \mathbb{R}^n$
find $x \in \mathbb{R}^d$ to:

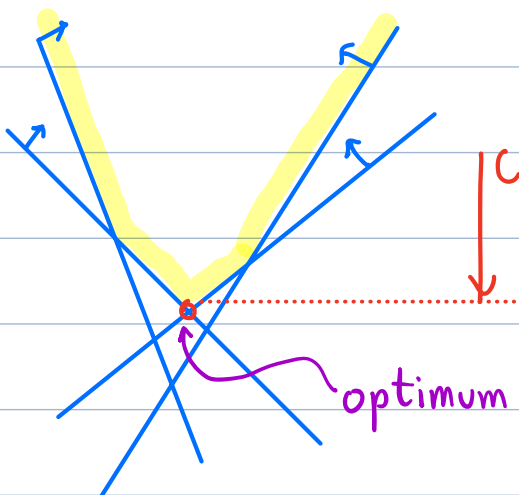
$$\begin{aligned} &\text{maximize: } c^T x \\ &\text{subject to: } Ax \leq b \end{aligned}$$

$\swarrow$ i^{th} row of A corresponds to b_i

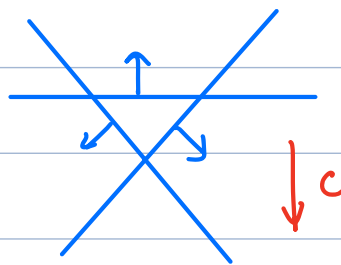
3 Possible Outcomes:

- 😊 **Feasible:** An optimal pt exists (gen'l position: a unique vertex of feasible polytope)
- 😞 **Infeasible:** No solution because feasible polytope is empty
- 😞 **Unbounded:** No (finite) solution because feasible polytope is unbounded in direction of objective fn.

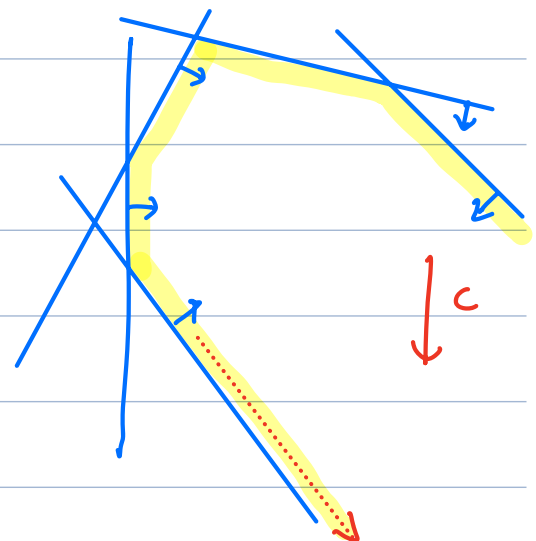
Feasible



Infeasible

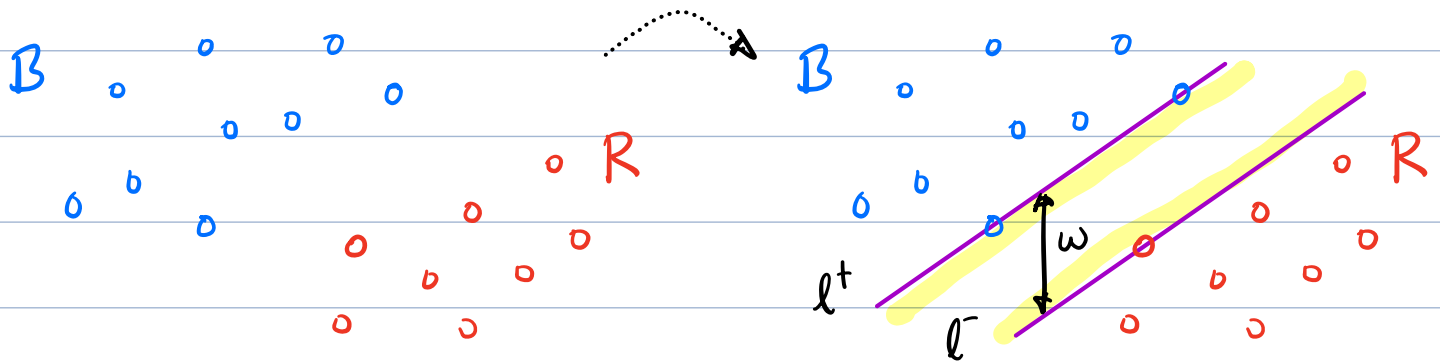


Unbounded



Example:

- Given two point sets B & R in $\mathbb{R}^2$
find lines of max. vertical distance
with B above both & R below both



- Lines: $l^+: y = e \cdot x + f^+$ $l^-: y = e \cdot x + f^-$

- Constraints: $\forall p \in B, p_y \geq e \cdot p_x + f^+$ (above l^+)
 $\forall p \in R, p_y \leq e \cdot p_x + f^-$ (below l^-)

- Objective: maximize $w = f^+ - f^-$

Standard form: Find (e, f^+, f^-)

to maximize $f^+ - f^- \equiv (0, 1, -1) \cdot (e, f^+, f^-)$
subject to:

$$p_{ix} \cdot e + 1 \cdot f^+ + 0 \cdot f^- \leq p_{iy}, \quad \forall p_i \in B$$
$$-p_{jx} \cdot e + 0 \cdot f^+ - 1 \cdot f^- \leq -p_{jy}, \quad \forall p_j \in R$$

LP in constant-dimensional space

- Assume - n is large
 d is a constant
- We'll present a (randomized) algorithm with (expected) running time $O(d!n) = O(n)$

Incremental Approach:

Overview:

- Find d -halfspaces that define an initial vertex v_d (or report that LP is unbounded)

→ $O(dn)$ time (see our text)

- Remove halfspace h_n and recursively compute LP on $n-1$ halfspaces $h_1, \dots, h_{n-1}$

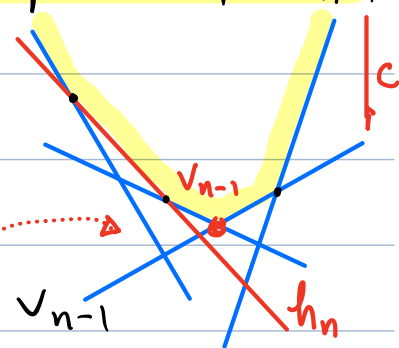
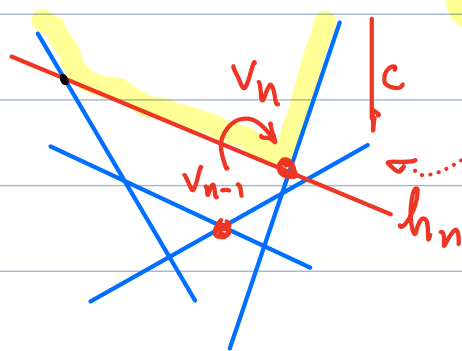
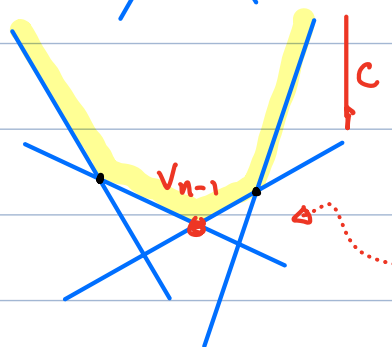
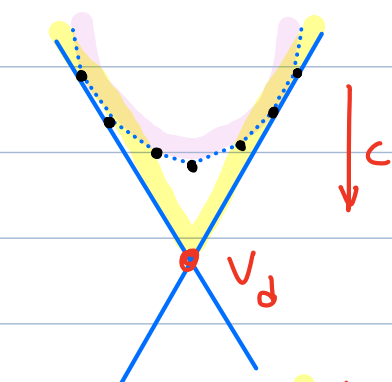
If infeasible → return
else let v_{n-1} be opt

- Add back h_n

- If $(v_{n-1} \in h_n)$ return v_{n-1}

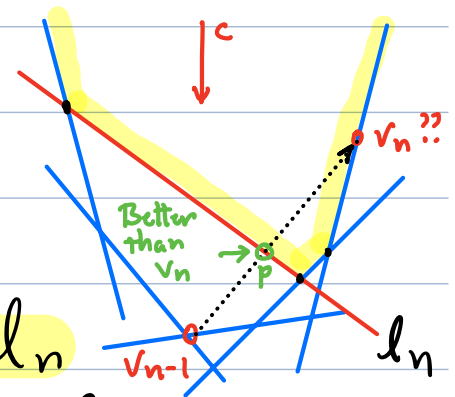
- else ...

How to update opt. vertex?



Lemma: If $v_{n-1} \notin h_n$ then new opt vertex (v_n) lies on the hyperplane bounding h_n .

Proof: Let l_n be hyperplane bounding h_n . Assume c directed downwards.



v_{n-1} - not feasible $\Rightarrow$ below l_n

v_n - if not on $l_n \Rightarrow$ above l_n

Let $p = l_n \cap \overline{v_{n-1}v_n}$

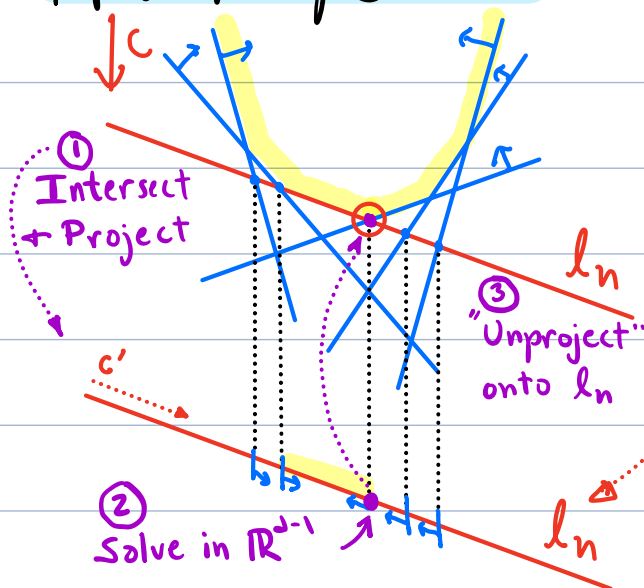
By convexity, $p \in$ feasible polytope

By linearity, obj. function gets progressively worse from $v_{n-1} \rightarrow v_n$

$\Rightarrow$ p is better solution than v_n

$\times$ contradiction!

How to update?



① Intersect $h_1, \dots, h_{n-1}$ with l_n + project $\tilde{c}$ [Yields an LP in $\mathbb{R}^{d-1}$ with $n-1$ constraints]

② Solve this $(d-1)$ -dim LP recursively (If $d=1$, solve by brute force $O(n)$)

③ "Unproject" solution back onto l_n

(See latex notes for details)

Running time? Pretty bad - $\mathcal{O}(n^d)$

- Let $W_d(n)$ be worst-case complexity for n halfspaces in dim d

- Recurrence:

$$W_d(n) = W_d(n-1) + d + [dn + W_{d-1}(n-1)]$$

solve LP on $h_1 \dots h_{n-1}$

Test $v_{n-1} \in h_n$

Project onto h_n

Solve LP on h_n

Claim: $W_d(n) = \mathcal{O}(n^d)$ ← Too slow!

How to fix this?

Easy! Randomize the choice of h_n

Why?

$$W_d(n) = W_d(n-1) + d + dn + W_{d-1}(n-1)$$

This solves to $\mathcal{O}(n)$

Only applies if $v_{n-1} \notin h_n$

This rarely happens!

Randomized Incremental Algorithm

Input: $H = \{h_1, \dots, h_n\}$ constraint halfspaces in $\mathbb{R}^d$
 $c \in \mathbb{R}^d$ objective vector

Output: Optimum vertex v or error $\begin{cases} \text{unbounded} \\ \text{infeasible} \end{cases}$

- (1) If ($d = 1$) solve LP by brute force - $O(n)$
- (2) Find initial subset $\{h_1, \dots, h_d\}$ that provide initial optimum v_d (or return "unbounded") - $O(d \cdot n)$ (see text)
- (3) Randomly select halfspace from $\{h_{d+1}, \dots, h_n\}$ - call it h_n . Recursively solve LP on remaining $n-1$ halfspaces $\rightarrow$ Let v_{n-1} be result
- (4) If ($v_{n-1} \in h_n$) return v_{n-1} $\rightarrow O(d)$
- (5) else, project $\{h_1, \dots, h_{n-1}\} + c$ onto h_n , the bounding hyperplane for h_n . $\rightarrow O(dn)$
Solve recursively, letting v_n be result. Return v_n

Expected Case Running Time:

- Running time depends on (random) choice, h_n
- Let $T_d(n)$ be the expected-case running time, over all choices of h_n .
- Let p_n = probability that $v_{n-1} \in h_n$
- To simplify, assume all halfspaces chosen randomly ($h_1, \dots, h_d$ aren't)

Recurrence:

$$T_d(n) = \begin{cases} 1 & \text{if } n=1 \\ n & \text{if } d=1 \\ T_d(n-1) + d + p_n(dn + T_{d-1}(n-1)) & \text{o.w.} \end{cases}$$

(3) Recursively compute v_{n-1}

(4) test if $v_{n-1} \in h_n$

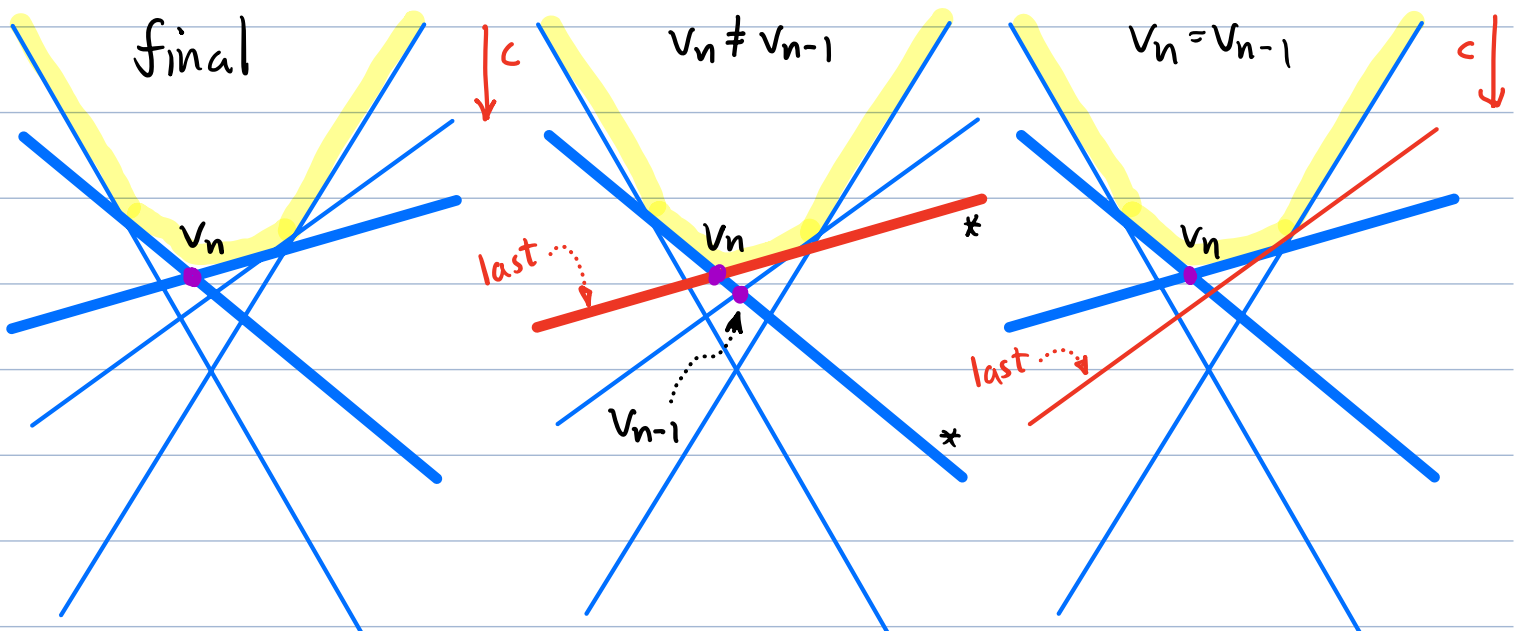
if not

(5) project $h_1, \dots, h_{n-1}$ onto h_n

(5) solve $d-1$ dim LP on projections

What is p_n ? Backwards Analysis

- Let's consider the final configuration and ask - which halfspace came last and how does its choice affect things?



Obs: The optimum is determined by d halfspaces (assuming gen'l position)

- If h_n is any of these, $v_{n-1} \notin h_n + v_n \neq v_{n-1}$ 😞
- Otherwise, $v_{n-1} \in h_n + v_n = v_{n-1}$ 😊

$\Rightarrow p_n = d/n$ If $n \gg d$, p_n very small + bad case unlikely

Why is it called "backwards"?

- We consider final config. and look backwards to our last random choice

Lemma: $T_d(n) \leq \gamma_d d! n$, where γ_d is a constant depending on dimension

Proof: Induction on $n + d$

$$T_d(n) = T_d(n-1) + d + p_n (dn + T_{d-1}(n))$$

by I.H. $\leq \gamma_d d! (n-1) + d + \frac{d}{n} (d \cdot n + \gamma_{d-1} (d-1)! n)$
+ def of p_n

$$= \gamma_d d! (n-1) + d + (d^2 + \gamma_{d-1} d!)$$

$$= \gamma_d d! n + (d + d^2 + \gamma_{d-1} d! - \gamma_d d!)$$

want:

$$\leq \gamma_d d! n$$

Suffices to select γ_d such that

$$d + d^2 + \gamma_{d-1} d! - \gamma_d d! \leq 0$$

$$\Leftrightarrow d! \gamma_d \geq d + d^2 + \gamma_{d-1} d!$$

We can satisfy this by setting:

$$\gamma_1 \leftarrow 1$$

$$\gamma_d \leftarrow \frac{d + d^2}{d!} + \gamma_{d-1}$$

$\Rightarrow \gamma_d$ is a constant depending on $\dim$

□

Summary:

- Randomized algorithm for LP
- Expected run time of LP is $O(d! n) = O(n)$
(since we assume d is constant)
- Variation depends on random choices, not input
- (Seidel) Prob of running slower extremely small

A Bit of History (Optional)

1940's : Used in operations research (Econ, Business)

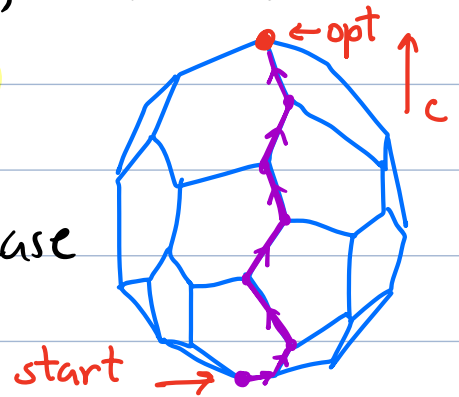
Kantorovich, Dantzig, von Neuman

Dantzig - Simplex algorithm

- (1947)
- fast in practice
 - exponential in worst case

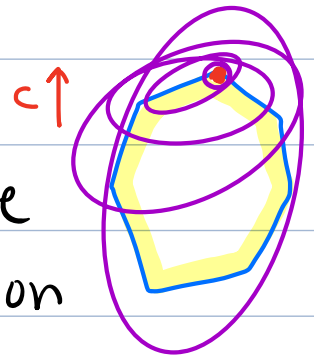
feasible polytope may have $O(n^{L/2})$ vertices

- Karp - Not known to be NP-hard



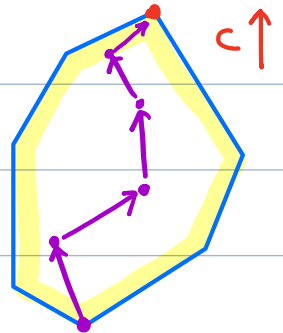
Khachiyan - Ellipsoid Algorithm

- (1979)
- (Weakly) polynomial time
 - ↳ Time depends on precision
 - Compute smaller + smaller ellipsoids containing optimum



Karmarkar - Interior-Point Methods

- (1984)
- Move through polytope's interior
 - (Weakly) polynomial
 - Practical



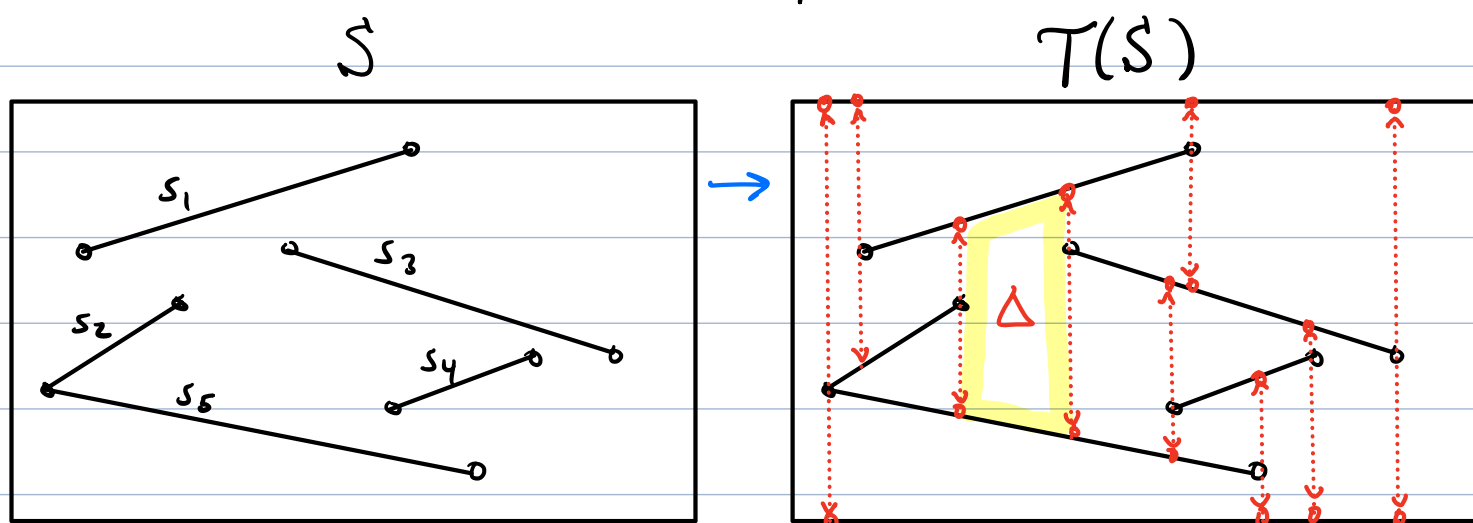
Open - Is there a poly. time (purely combinatorial) algorithm?

CMSC 754 - Computational Geometry

Lecture 8 - Trapezoidal Maps

Trapezoidal Maps:

- Given a set $S = \{s_1, \dots, s_n\}$ of line segments in $\mathbb{R}^2$, which we assume do not intersect (except at their endpoints)
- General position: No duplicate x-coords + no vertical segments
- Enclose in large bounding rectangle
- Shoot a bullet path vertically above + below each endpoint until it hits something

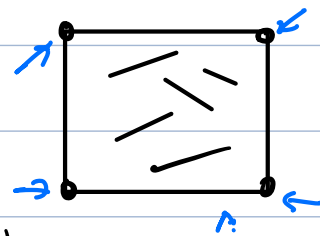
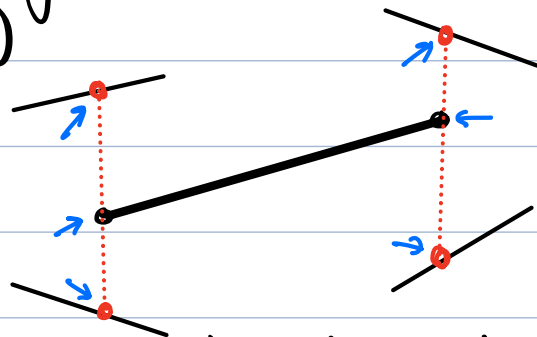


- This subdivides the rectangle into trapezoids (degenerating possibly to triangles)

Lemma: If $|S| = n$, $T(S)$ has $\leq 6n+4$ vertices and $\leq 3n+1$ trapezoids

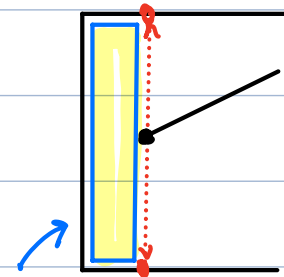
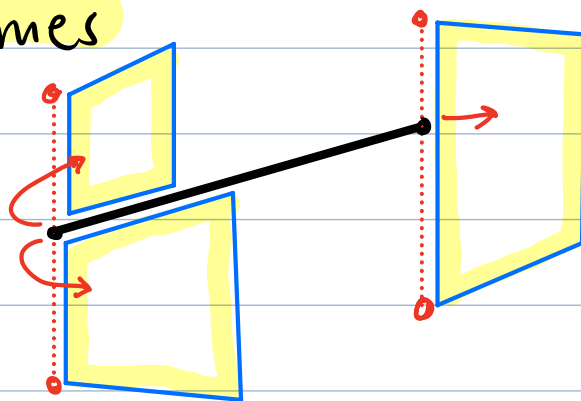
Proof:

- Each segment contributes 6 vertices to $T(S)$



Plus 4 for the bounding rectangle
 $\Rightarrow 6n+4$ vertices

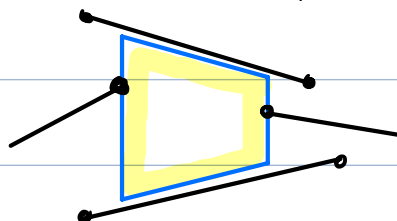
- Charge each trapezoid to vertex on its left side. Each segment is charged 3 times



$\Rightarrow 3n + 1$ for leftmost trap.

□

Obs: Each trapezoid owes its existence to ≤ 4 segments



Construction:

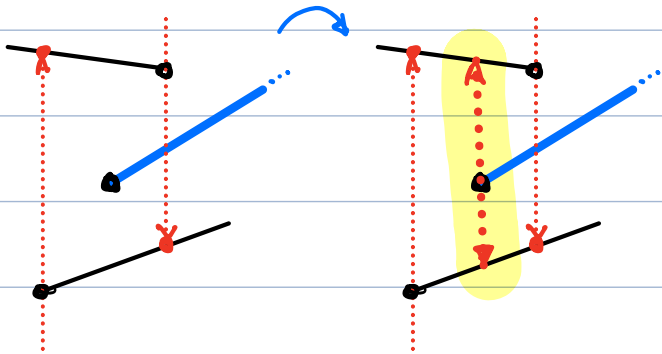
- Plane sweep - $O(n \log n)$ [exercise]
- Randomized incremental - $O(n \log n)$ [this lect]

Incremental Construction:

Random order

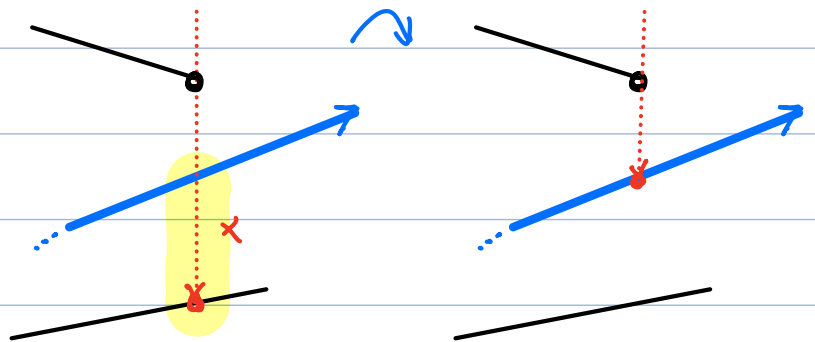
- Add segments one-by-one
- Update the map after each insertion
- Two types of updates:
 - Endpt of new segment
 - shoot bullet paths up + down
 - Crash through a vertical wall
 - trim the wall back

End point:



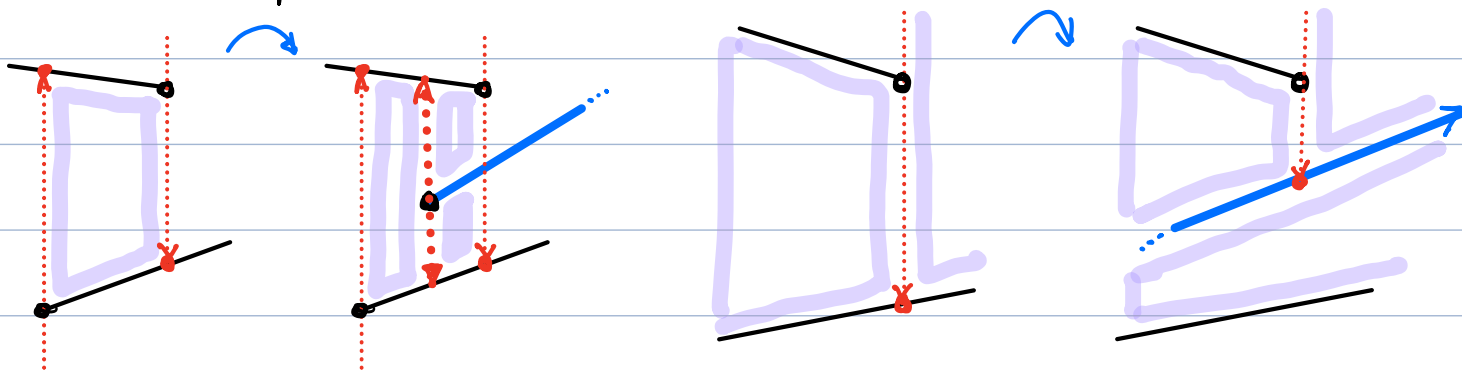
Find the trapezoid containing this end point, and add vertical segments to top + bottom

Crash through wall:



Determine whether the shooting vertex is above or below, and trim away the excess

These updates implicitly generate new trapezoids + destroy old ones



Running time: to insert segment $s_i, i=1, \dots, n$

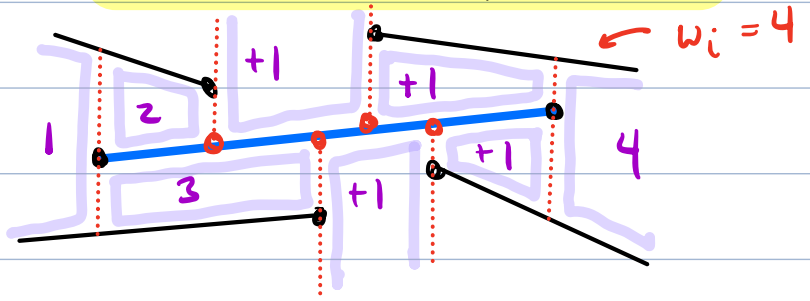
- Find trapezoid containing s_i 's left endpt $\left. \begin{array}{l} \} O(\log n) \\ \text{(next lect.)} \end{array} \right\}$
- Trace segment through trapezoids + update $\left. \begin{array}{l} \} \text{Depends!} \\ O(1) \dots O(n) \end{array} \right\}$

Lemma: For $1 \leq i \leq n$, let k_i denote the number of new trapezoids created by insertion of i^{th} segment. Ignoring the time to locate the left endpt, the insert time is $O(k_i)$

(Note: k_i is a random variable, depending on insertion order $O(1) \dots O(n)$)

Proof: Let w_i denote num. of walls hit.

$$k_i = 4 + w_i$$



Insert time $\sim O(2 + 2 + w_i) = O(k_i)$

Bullets
for left
end pt

Bullets
for right

Trim walls
hit

□

Overall run time: (Ignoring endpt location)

Worst-case: Adding segment i can create $O(i)$ new trapezoids

$$\Rightarrow T(n) = \sum_{i=1}^n i = O(n^2)$$

Expected-case: We will show that if segs are inserted in random order, $E(k_i) = O(1)$

Wow - This does not depend on i !

$$\Rightarrow \text{Exp. time} = \sum_{i=1}^n E(k_i) \leq n \cdot O(1)$$

$$= O(n)$$

(ignoring left
endpt location)

Lemma: Assuming segments are inserted in random order, $E(k_i)$ (the expected number of new trapezoids with i^{th} insert) is $O(1)$.

Proof: (Backwards analysis)

$\mathcal{T}_i$ does not depend on insert order

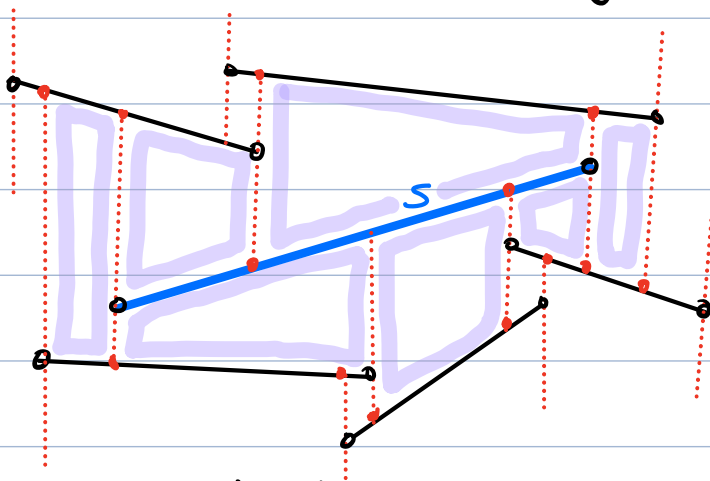
- Let $\mathcal{T}_i = \text{trapezoidal map after } S_i = \{s_1, s_2, \dots, s_i\}$

- Each seg. is equally likely to be last

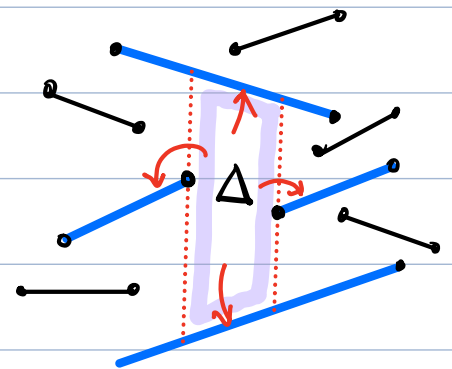
$$\text{Prob}(s_i \text{ is last inserted}) = 1/i$$

- Given any trapezoid $\Delta \in \mathcal{T}_i$ and any segment $s \in \{s_1, \dots, s_n\}$ we say Δ depends on s if Δ would have been created if s was inserted last.

$$\delta(\Delta, s) = \begin{cases} 1 & \text{if } \Delta \text{ depends on } s \\ 0 & \text{ow.} \end{cases}$$



Trapezoids that depend on s



Segments on which Δ depends

Note: $\delta(\Delta, s)$ does not depend on insertion order

$$E(k_i) = \sum_{s \in S_i} \text{Prob}(s \text{ inserted last}) \cdot \left(\begin{array}{c} \text{Num. of traps} \\ \text{that depend on} \\ s \end{array} \right)$$

$$= \sum_{s \in S_i} \left(\frac{1}{i} \right) \sum_{\Delta \in \tau_i} \delta(\Delta, s)$$

(Reverse sum. order)

$$= \frac{1}{i} \sum_{s \in S_i} \sum_{\Delta \in \tau_i} \delta(\Delta, s)$$

$$= \frac{1}{i} \sum_{\Delta \in \tau_i} \sum_{s \in S_i} \delta(\Delta, s)$$

≤ 4

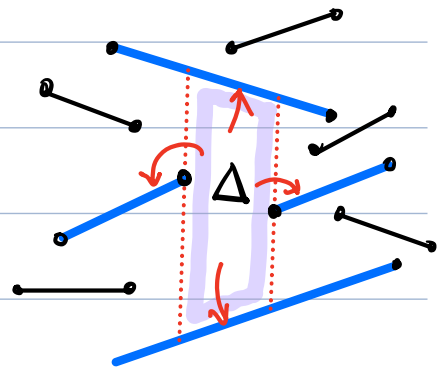
Δ depends on at most 4 segs

$$\leq \frac{1}{i} \sum_{\Delta \in \tau_i} 4$$

$$= \frac{4}{i} |\tau_i| \leq \frac{4}{i} (3i+1)$$

$$= 12 + \frac{4}{i} = O(1)$$

□



Line Segment Intersection - Revisited

- We showed how to compute intersection of line segments in $O((n+m) \log n)$ time.
 n = no. of segs. m = no. of intersects

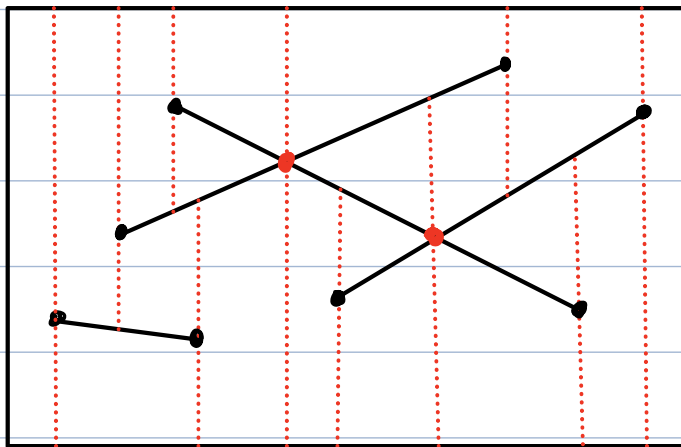
- We gave a lower bound of $\Omega(n \log n + m)$

$\underbrace{\hspace{1.5cm}}$ element uniqueness $\underbrace{\hspace{1.5cm}}$ output size

- Can we achieve this?

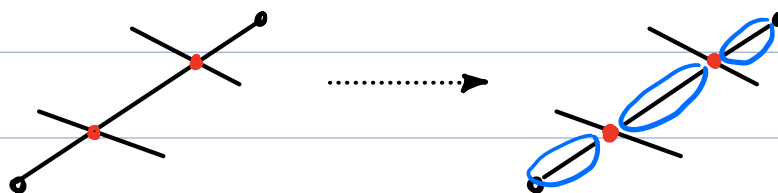
Yes! (but randomized)

- Build trapezoidal map
- Also add bullet paths when segs. intersect



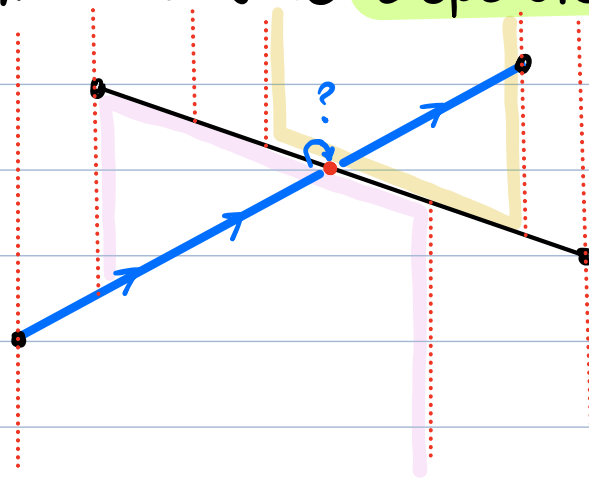
$O(n+m)$
trapezoids

- Think of intersection pts as breaking up into subsegments



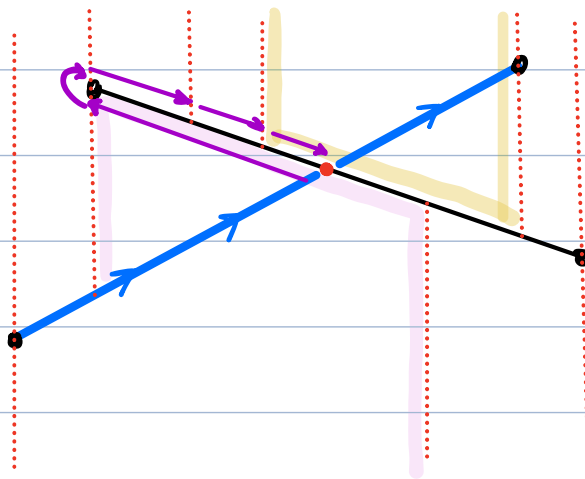
- What changes?

- Whenever intersection detected,
need to find trapezoid to continue with



on other side
of segment

- Do this by walking along the segment
from one side to the other



- Can show that if segs. inserted in random
order - expected time is $O(n+m)$

Summary:

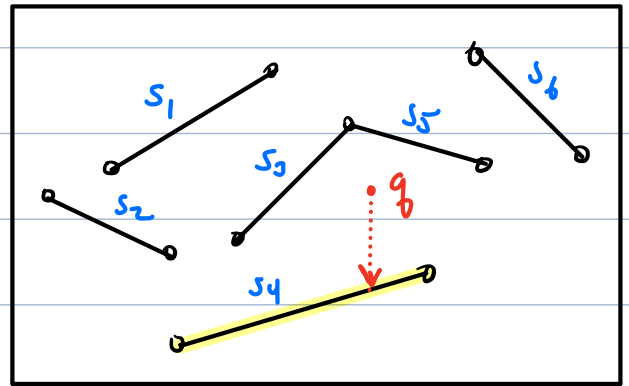
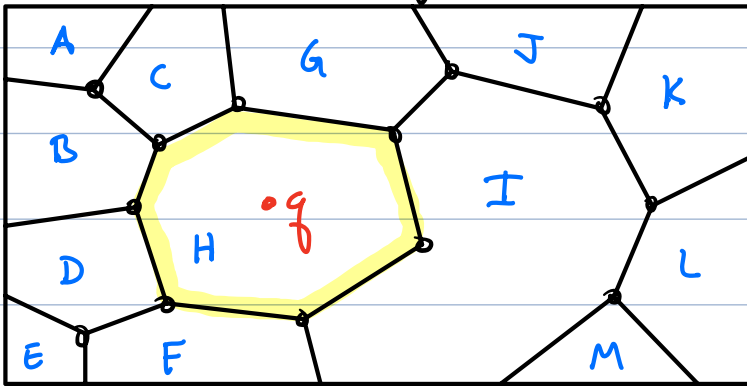
- Can build trap. map in expected linear time
(ignoring time to locate starting trapezoid.)
- Next - Finding start traps. in $O(n \log n)$ time.

CMSC 754 - Computational Geometry

Lecture 9: Planar Point Location (+Trap. Maps)

Planar Point Location:

Given a subdivision of the plane (cell complex), build a data structure so that for any query pt, can find the cell containing it.

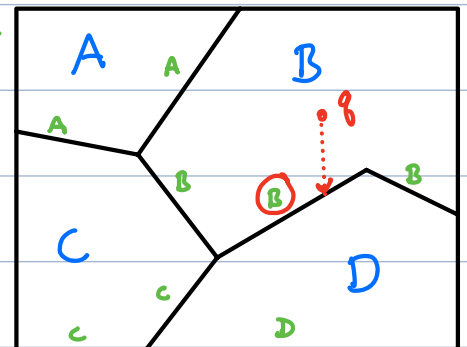


Vertical Ray Shooting:

Given a set of disjoint line segments in the plane, build a data structure s.t. given any query pt q , can report the segment immediately below.

Ray Shooting $\Rightarrow$ Point Location

Label each segment with region just above



Data structure for vertical ray shooting:

Approach: Build trapezoidal map + ray shooting structure simultaneously

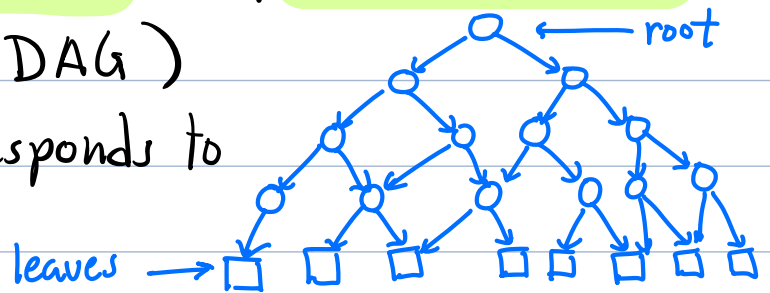
$S = \{s_1, \dots, s_n\}$ Randomly permuted $\rightarrow \mathcal{T}(S)$
 $S_i = \{s_1, \dots, s_i\} \rightarrow$ Partial map $\mathcal{T}(S_i) = \mathcal{T}_i$

Recall: In expectation, each insertion results in $O(1)$ changes to structure.

Overview:

- Rooted binary tree with shared subtrees (a rooted DAG)

- Each leaf corresponds to a trapezoid

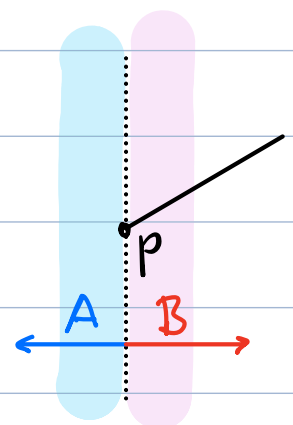
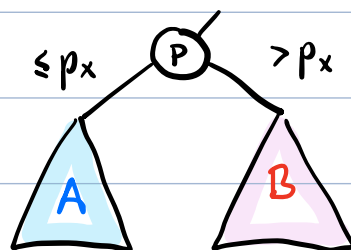


- Each trapezoid occurs exactly once as leaf

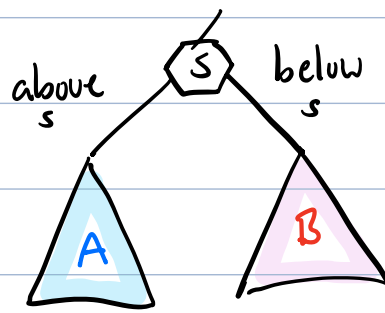
- Internal nodes - two types

x-Node:

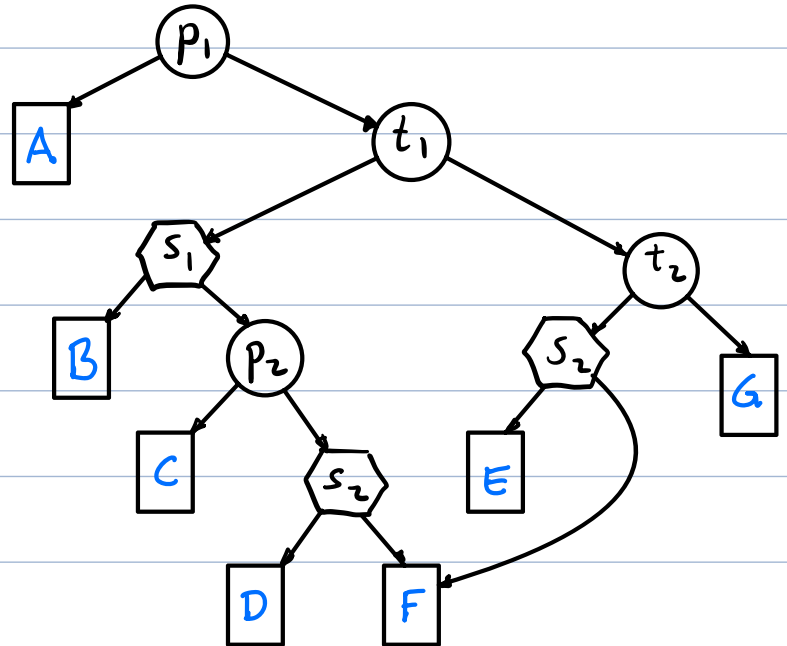
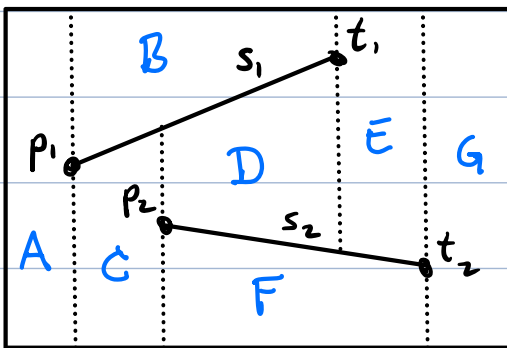
labeled with an endpt p



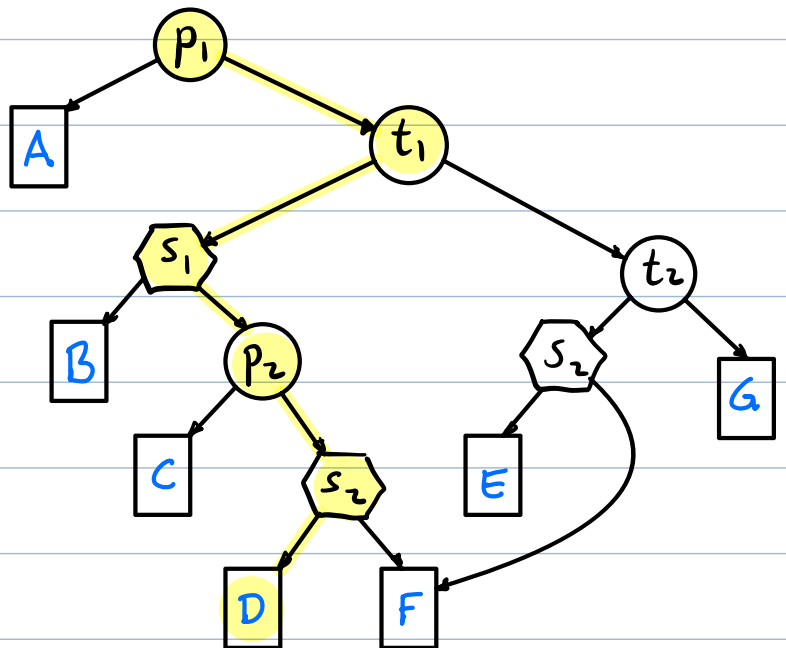
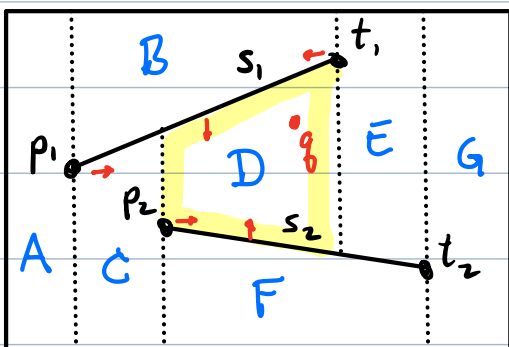
y-Node:
Labeled with
a segment s



Example:



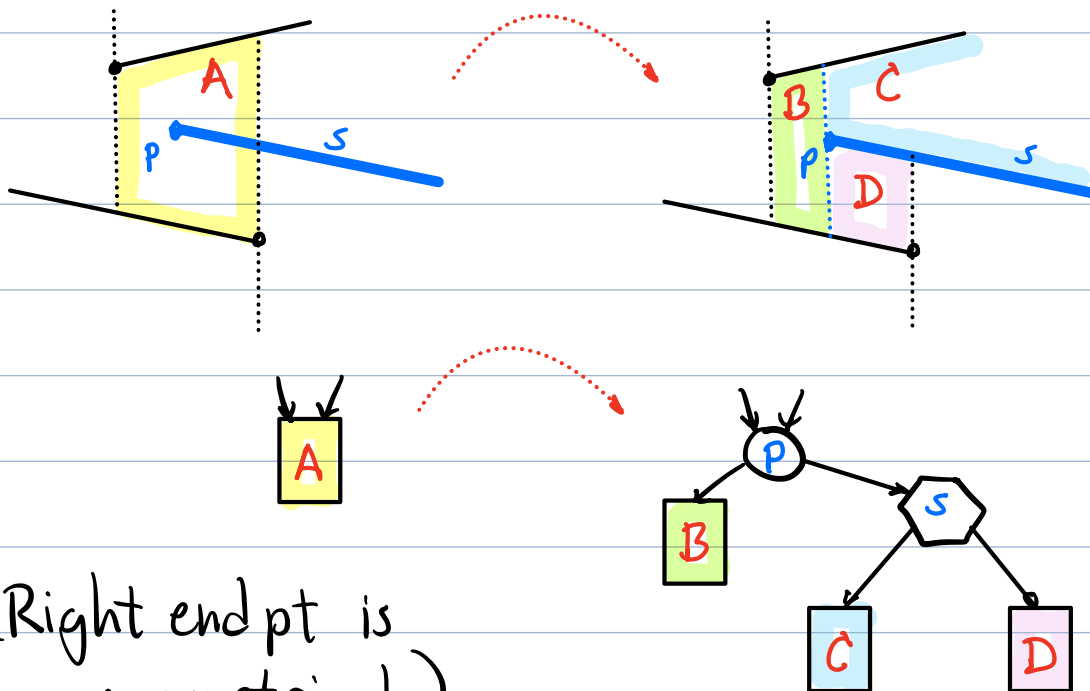
Query processing:



Incremental Construction:

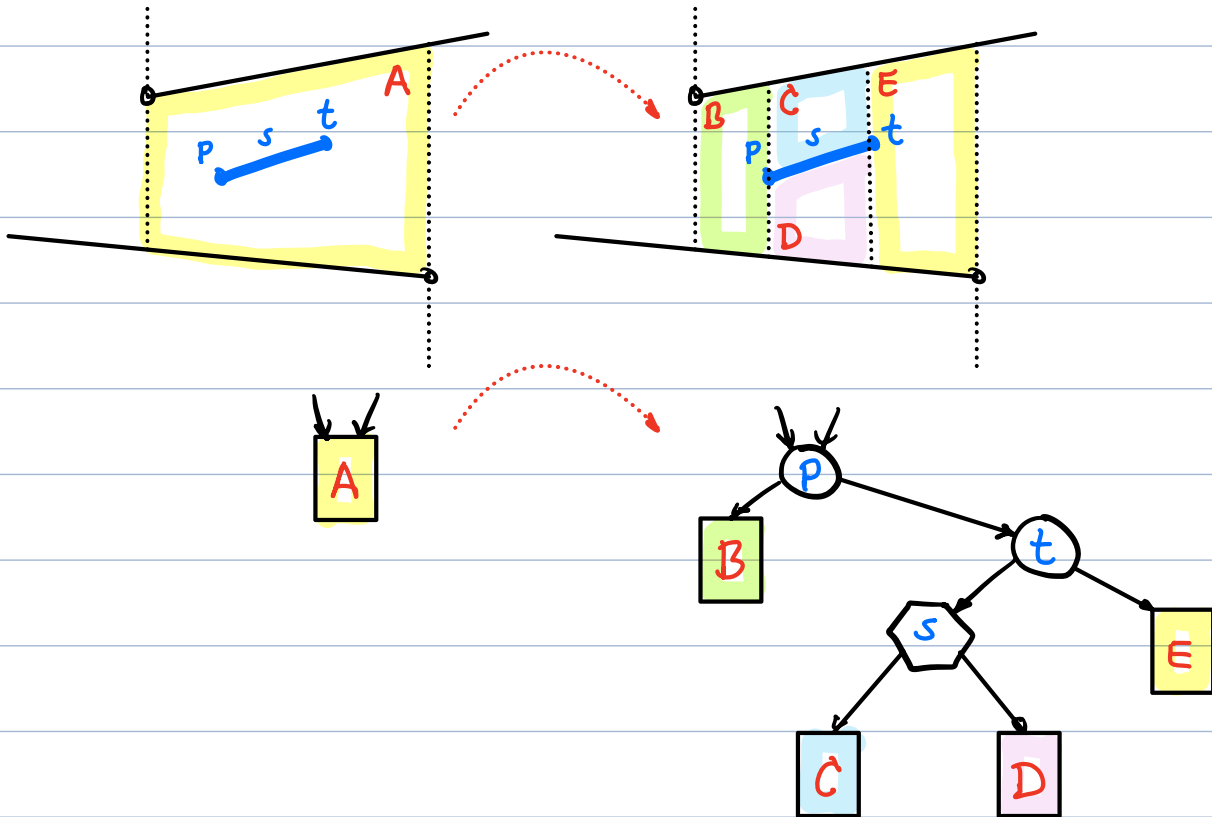
- As segments are added: $s_1, s_2, \dots, s_i$
we build structure for $\mathcal{T}(s_1), \mathcal{T}(s_2) \dots \mathcal{T}(s_i)$
- Update process:
 - Each added segment causes some trapezoids to go away + others created
 - We replace old leaves with new structures
 - By sharing, only one leaf per trapezoid

1: Single endpt in trapezoid (left or right):

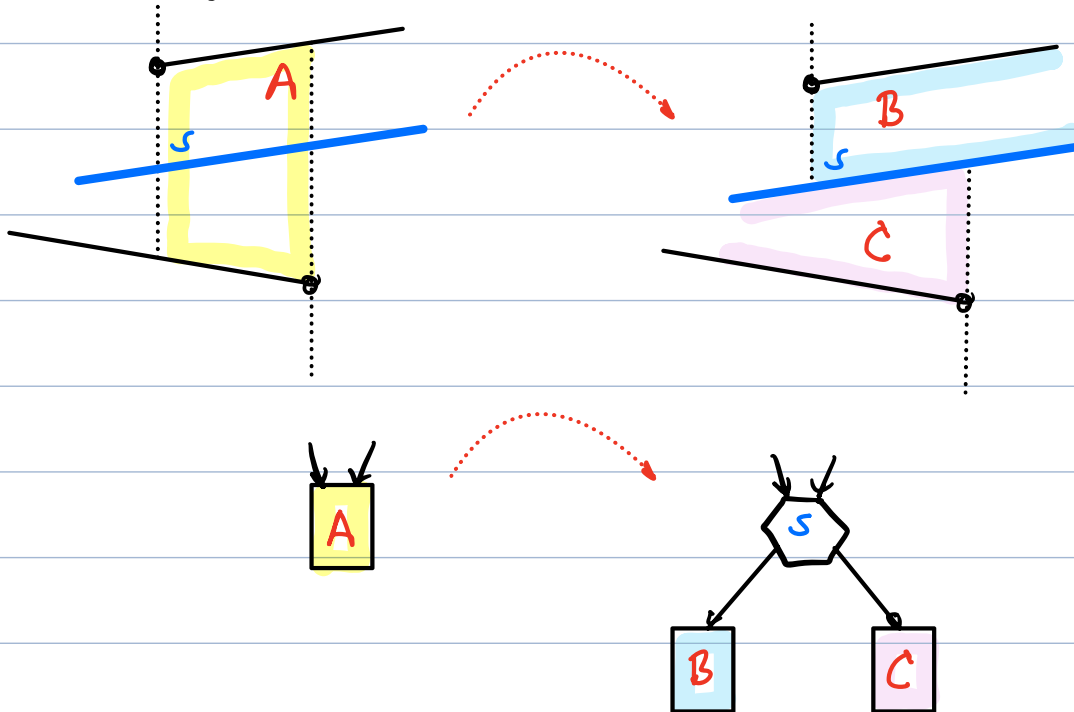


(Right endpt is symmetrical)

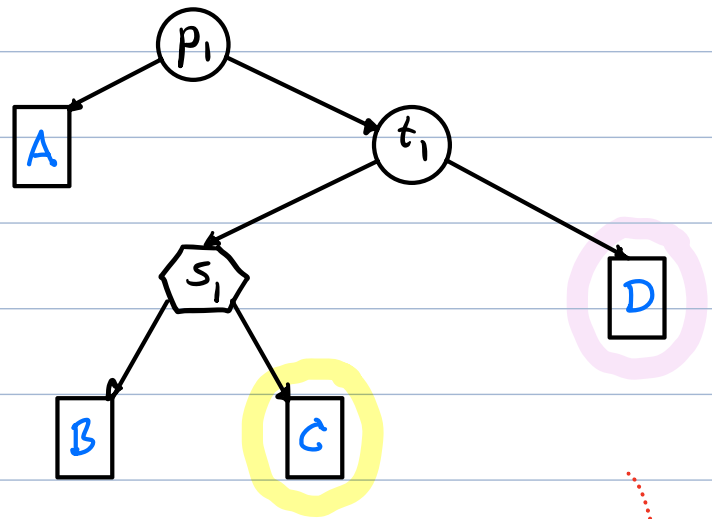
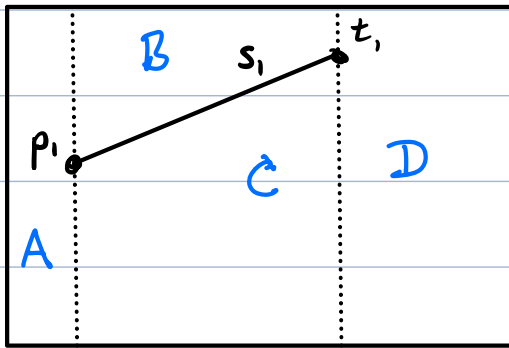
2: Two segment endpts in same trapezoid



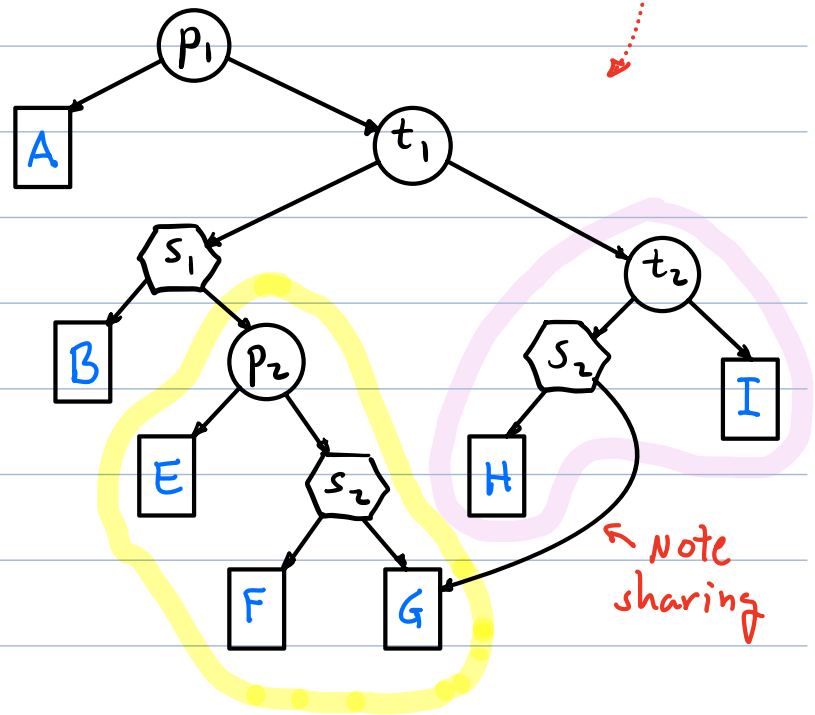
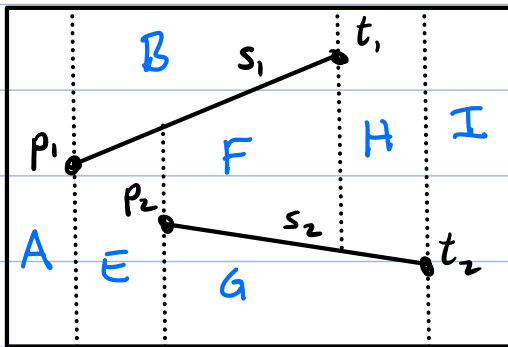
3: No segment endpoint in trapezoid



Example:



insert $s_2 = \overline{p_2 t_2}$



Analysis:

Will show if segs are inserted in random order, expected space is $O(n)$ + expected search time for any fixed query pt is $O(\log n)$

Thm: The expected case space is $O(n)$

Proof: Last lecture we showed that expected no. of changes is $O(1)$ per seg $\Rightarrow$ total changes $O(n)$

Number of new nodes $\sim$ number of changes
 $\Rightarrow$ final expected size is $O(n)$

Thm: Given a fixed query pt $q \in \mathbb{R}^2$, the expected search depth for q is $O(\log n)$

Huh? Does this imply that depth of search tree is $O(\log n)$ in expectation?
No - But see our text for a proof of this.

Proof:

- Let q be any fixed query pt.

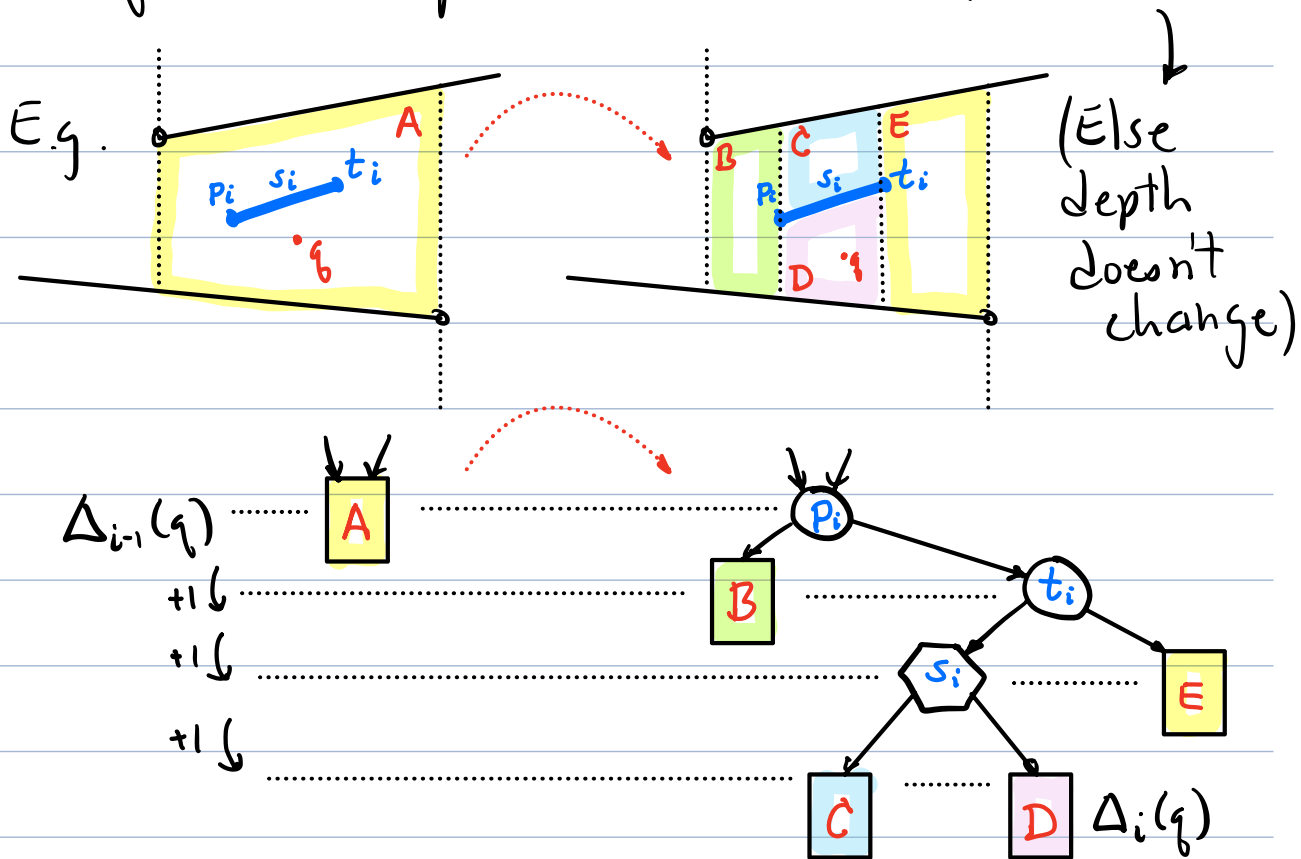
- Let $\Delta_i(q)$ be the trapezoid containing q after the insertion of s_i ($1 \leq i \leq n$)

- Note: Sometimes $\Delta_i(q) = \Delta_{i-1}(q)$
(s_i had no impact)

- What if $\Delta_i(q) \neq \Delta_{i-1}(q)$?

- For $1 \leq i \leq n$, let $\bar{X}_i(q) = \begin{cases} 1 & \text{if } \Delta_i(q) \neq \Delta_{i-1}(q) \\ 0 & \text{o.w.} \end{cases}$

- If $\bar{X}_i(q) = 1$, $\text{depth}(\Delta_i) \leq 3 + \text{depth}(\Delta_{i-1})$



Let $D(q)$ the expected depth of q 's trapezoid in the final structure.

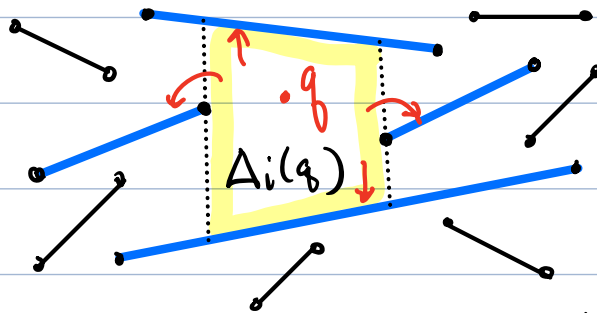
$$\begin{aligned} D(q) &\leq 3 \sum_{i=1}^n E(X_i(q)) \\ &= 3 \sum_{i=1}^n \text{Prob}(\Delta_i(q) \neq \Delta_{i-1}(q)) \end{aligned}$$

- We assert that $\text{Prob}(\Delta_i(q) \neq \Delta_{i-1}(q)) \leq 4/i$

- Backwards analysis:

- Each of the existing i segs is equally likely to be last (prob = $1/i$)

- $\Delta_i(q) \neq \Delta_{i-1}(q)$ iff last segment is one of the 4 segments incident to $\Delta_i(q)$



$$\Rightarrow \text{Prob}(\Delta_i(q) \neq \Delta_{i-1}(q)) \leq 4/i$$

- Substituting: Expected depth of q 's trapezoid

$$\begin{aligned} D(q) &\leq 3 \sum_{i=1}^n E(X_i(q)) = 3 \sum_{i=1}^n \text{Prob}(\Delta_i \neq \Delta_{i-1}) \\ &\leq 3 \cdot \sum_{i=1}^n 4/i = 12 \sum_{i=1}^n 1/i \quad (\text{Harmonic series}) \end{aligned}$$

$$\approx 12 \ln n = O(\log n) \quad \square$$

Summary:

- Last time we showed that randomized incremental alg. took $O(1)$ time in expectation per segment, ignoring time to locate left end pt.
- Today, we presented a data structure with query time $O(\log n)$ for pt location
 - $\Rightarrow$ Total expected construction time is $O(n \log n)$
- Space + Query time are in expectation
 - Can we guarantee them?
 - $\rightarrow$ Yes: Just rebuild if things go wrong (Increases expected construct time slightly, but still $O(n \log n)$.)
- Works even if segments intersect:
 - Exp. time: $O(n \log n + m)$ $\rightarrow$ Beats plane sweep!
 - Space: $O(n + m)$
 - Query time: $O(\log n)$

CMSC 754 - Computational Geometry

Lecture 10: Voronoi Diagrams

Metric Spaces: Distances modeled as **metric space** (X, f) : $f: X \times X \rightarrow \mathbb{R}^{\geq 0}$, s.t. for all $p, q, r \in X$:

Symmetry: $f(p, q) = f(q, p)$

Positivity: $f(p, q) \geq 0$ and $f(p, q) = 0$ iff $p = q$

Triangle Inequality: $f(p, q) \leq f(p, r) + f(r, q)$

Euclidean Distance: for $p, q \in \mathbb{R}^d$:

$$\|p - q\| = \left[\sum_i (p_i - q_i)^2 \right]^{1/2}$$

Voronoi Diagram:

A fundamental structure for metric spaces.

Given a point set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^d$ called **sites**, we want to subdivide space based on each site's **"region of influence"**

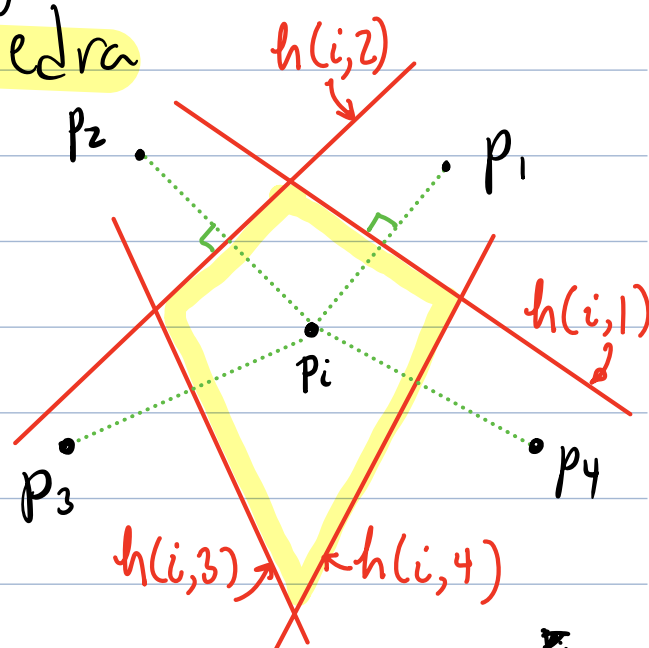
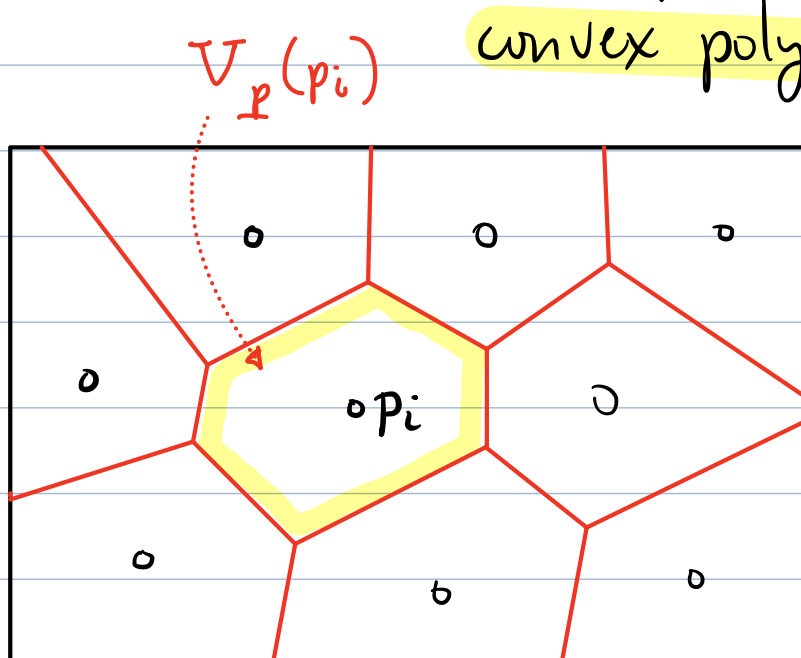
Def: Voronoi cell for site p_i

$$V_P(p_i) = \{q \in \mathbb{R}^d \mid \|p_i - q\| < \|p_j - q\|, \forall j \neq i\}$$

Obs: - Voronoi cells are disjoint

- For Euclidean dist, Voronoi cells are (possibly unbounded)

convex polyhedra



$$\text{Let } h(i,j) = \{q \mid \|p_i - q\| < \|p_j - q\|\}$$

$h(i,j)$ - halfspace bounded by perpendicular bisector between p_i & p_j

$$\text{Vor}(p_i) = \bigcap_{j \neq i} h(i,j) \quad \text{— intersection of halfspaces} \Rightarrow \text{polytope}$$

Def: $\text{Vor}(P)$ is the subdivision (cell complex) induced by P 's voronoi cells.

- $\text{Vor}(P)$ covers $\mathbb{R}^d$
- Has n cells (faces of dim d)
- Polyhedral subdivision (for Euclidean dist)
- Combinatorial complexity:
 - $\mathbb{R}^2$: $O(n)$ edges + vertices
 - $\mathbb{R}^d$: $O(n^{\lceil d/2 \rceil})$ size [Closely related to convex polytopes in $\mathbb{R}^{d+1}$]

Many applications:

Nearest neighbor search:

Preprocess a set of sites $P = \{p_1, \dots, p_n\} \subset \mathbb{R}^d$ s.t. given any query point $q \in \mathbb{R}^d$ can find q 's nearest site

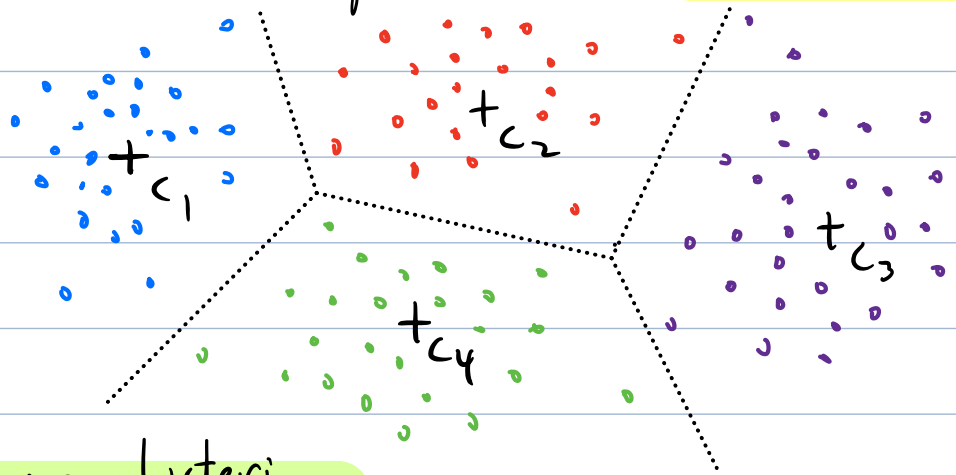
How? - Compute $\text{Vor}(P)$

- Build a point-location data structure for $\text{Vor}(P)$

[Optimal in $\mathbb{R}^2$. Not as good in $\mathbb{R}^d$.]

Point-based Clustering:

- Given set T of training points, group them into k clusters
- Clusters are defined by k cluster centers $\{c_1, \dots, c_k\}$
- Cluster membership based on closest center



- k-means clustering

Variations:

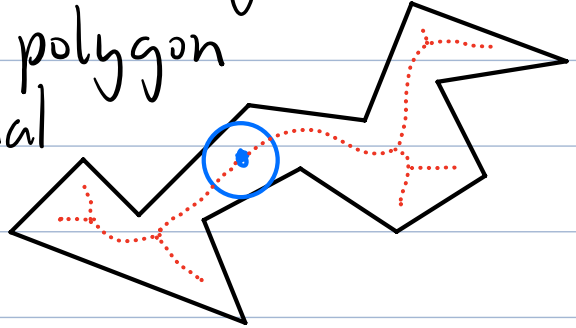
- Other metrics: L_1 -Vor diagram (Manhattan distance)
- Weighted pts:
 - Multiplicative: $\text{dist}(q, p_i) = \alpha_i \|p_i - q\|$
 - Additive: $\text{dist}(q, p_i) = \|p_i - q\| + w_i$
- k^{th} Nearest:
 - $\text{Vor}_k(P) = \text{subdivide based on } k^{\text{th}}$
closest

$Vor_n(P)$ = farthest point Vor. diag

- Other shapes:

- Voronoi diagram of line segments

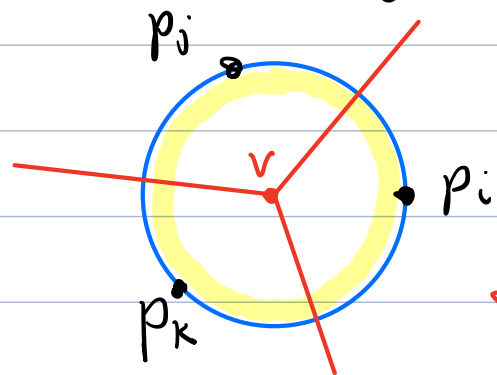
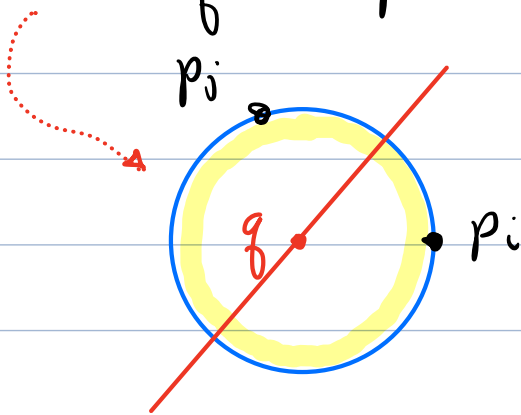
- Medial axis of polygon
centers of maximal
disks



Properties of the Voronoi Diagram:

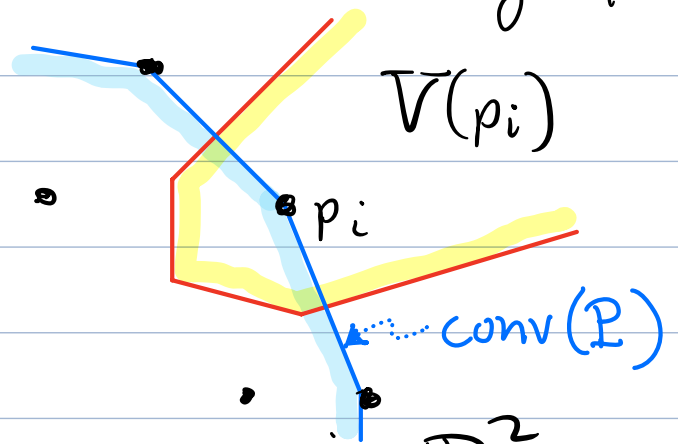
Empty-circle Property:

A pt q is on an edge of the Vor. diag iff there is a circle centered at q that passes through 2 sites & is otherwise empty.



Circumcircle Property: A pt v is a vertex of the diagram iff it is the center of a circle passing through 3 sites & is otherwise empty.

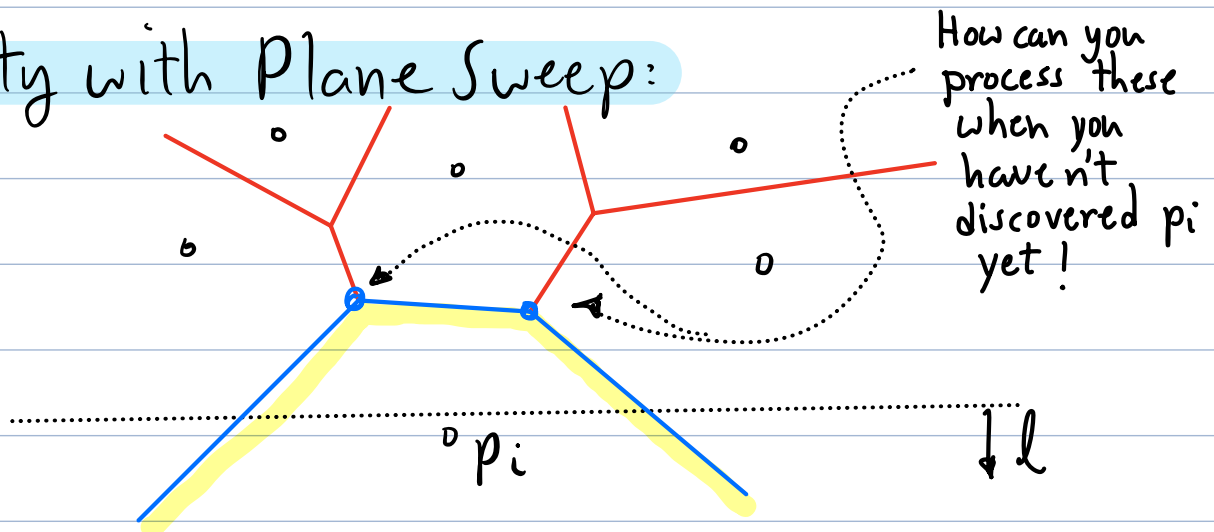
Hull Property: A site p_i has an unbounded Voronoi cell iff p_i is on boundary of convex hull of P .



Constructing Voronoi Diagrams in $\mathbb{R}^2$

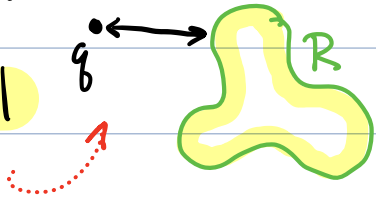
- **Incremental** - add a site; update (best if randomized)
- **Divide + Conquer** - $O(n \log n)$
- **Plane Sweep** (this lecture)
 - Fortune's Algorithm - $O(n \log n)$

Difficulty with Plane Sweep:



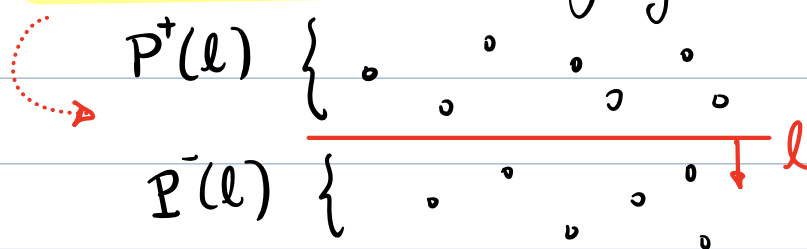
Clever twist: We'll maintain two sweeping structures: **sweep line** + **beach line**

Def: Given a set of pts R and pt q , define

$$\text{dist}(q, R) = \min_{p \in R} \|p - q\|$$


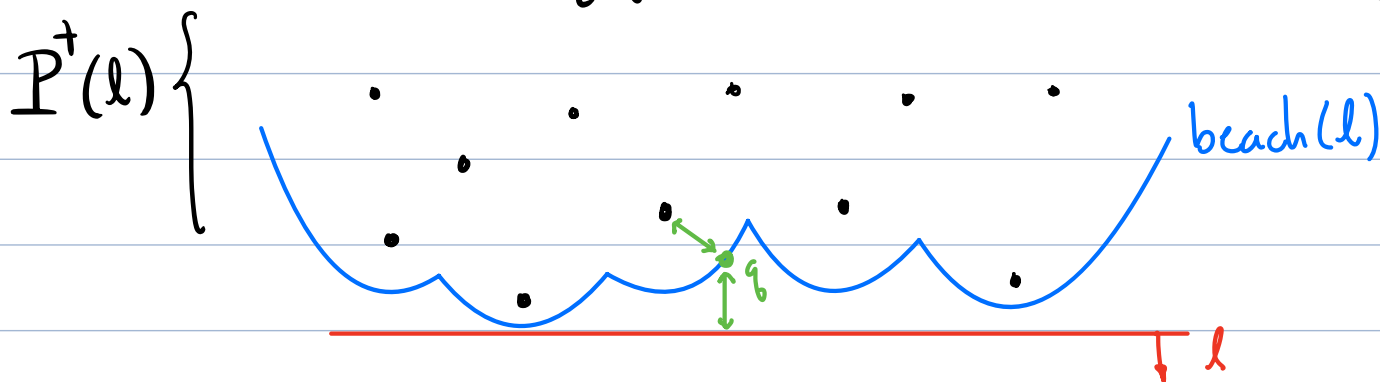
Given a sweep line l (horizontal + moving down) define

$P^+(l)$ to be sites lying above l



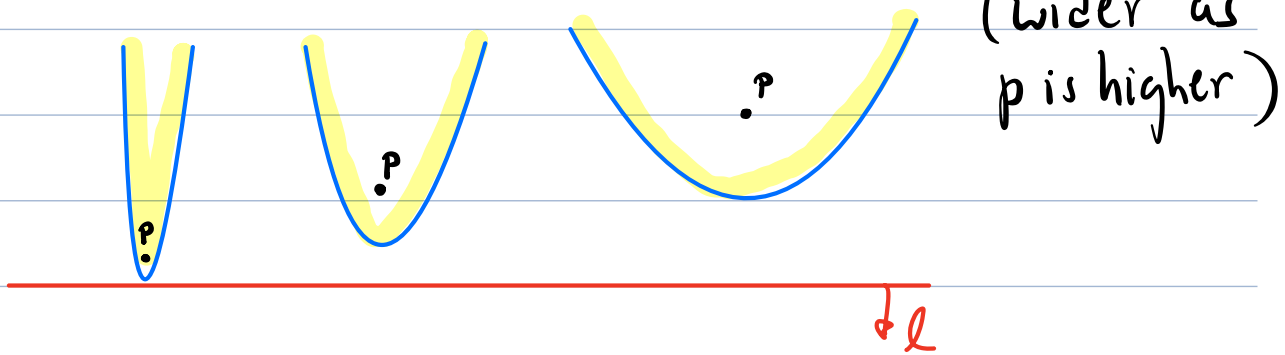
Given sweep line l , define the **beach line** to be set of pts $q \in \mathbb{R}^2$ that are **equidistant** from $P^+(l)$ and l

$$\text{beach}(l) = \{q \in \mathbb{R}^2 \mid \text{dist}(q, P^+(l)) = \text{dist}(q, l)\}$$



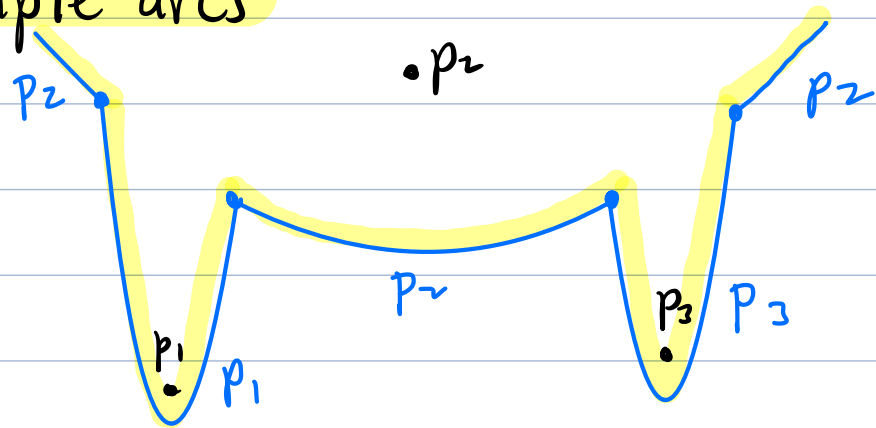
Beach-line Structure:

The points equidistant to a site p + line l form a parabola



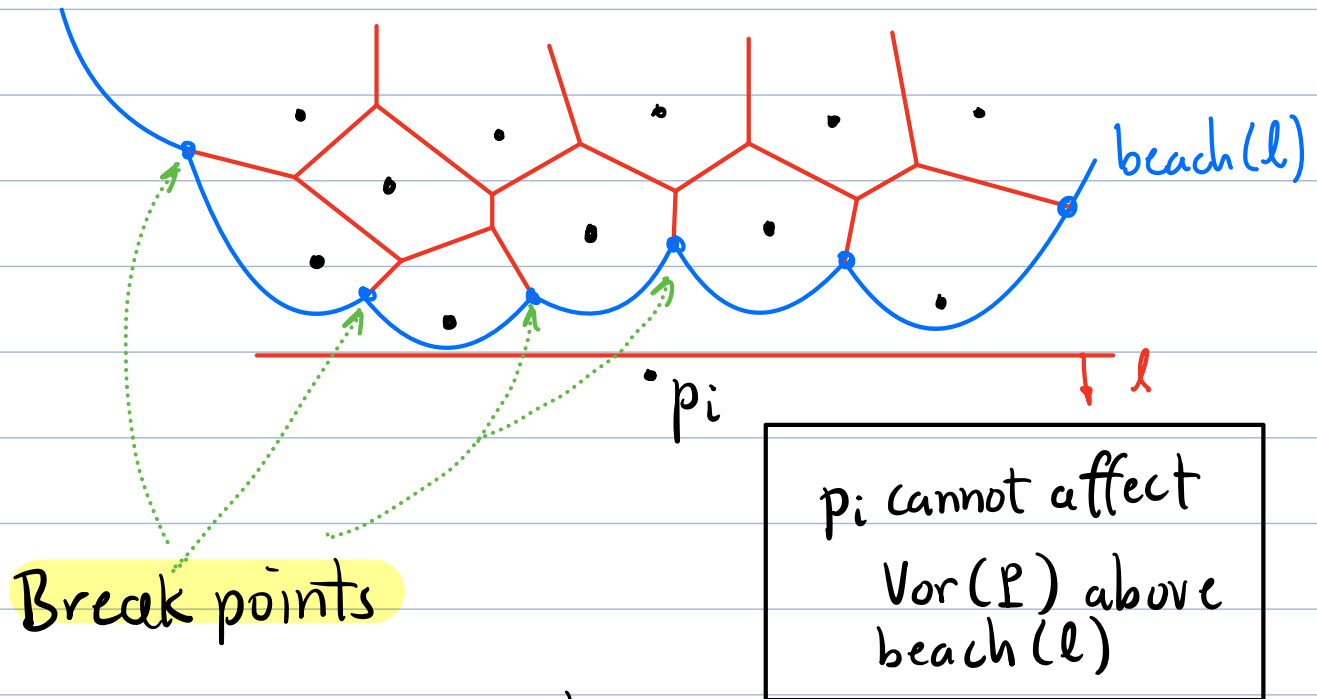
The beach line is the lower envelope of these parabolas for all sites in $P^+(l)$

- Beach line is x -monotone
- A single site may contribute 0, 1, or multiple arcs



- Total complexity is $O(|P^+(l)|) = O(n)$
[Proof: Exercise]

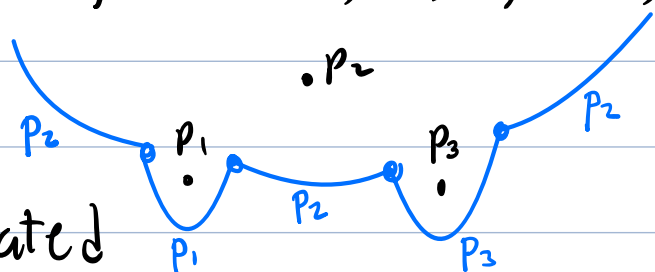
Key: The portion of $\text{Vor}(P)$ above the beach line is "safe" from sites lying below l .



Fortune's Algorithm:

Sweep-line status:

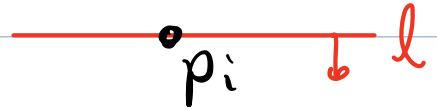
- y -coord of sweep line
- seq. of sites (left to right) that contribute arc to beach line (eg. $\langle 2, 1, 2, 3, 2 \rangle$)
- Parabolic arcs not computed
- Breakpts generated as needed



Voronoi diagram: Portion of Voronoi diagram (rep. as DCEL) above beach line is stored/updated.

Events:

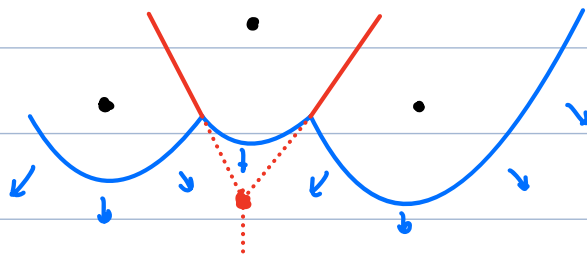
Site event: Sweep line passes over a site



Vertex event (circle event):

- A new Voronoi vertex is discovered

$\equiv$ An arc on beach line vanishes

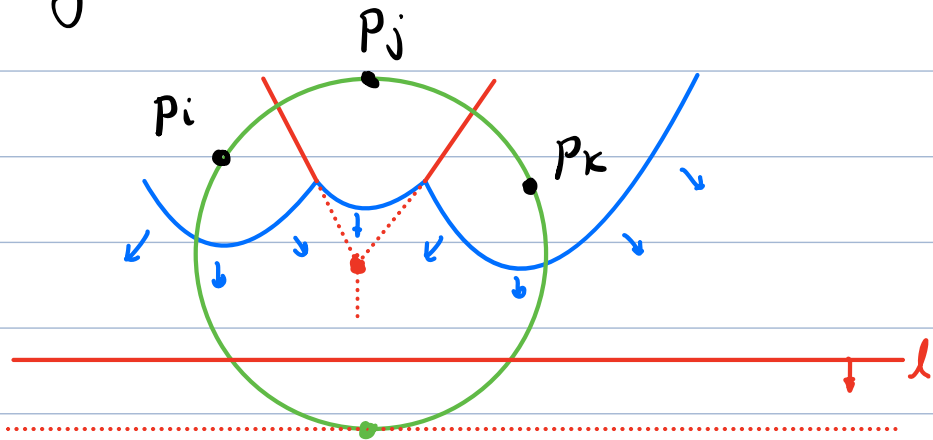


Priority Queue: Stores y-coords for sweep line at events.

Site events: Easy - just y-coord of site (static)

Vertex events: Tricky! (see below)

Scheduling vertex events:



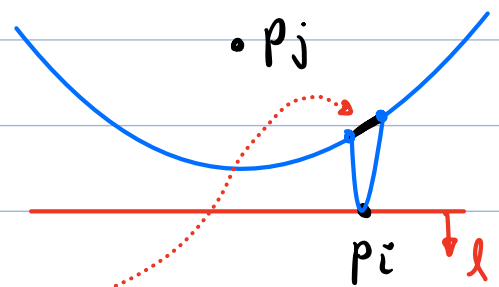
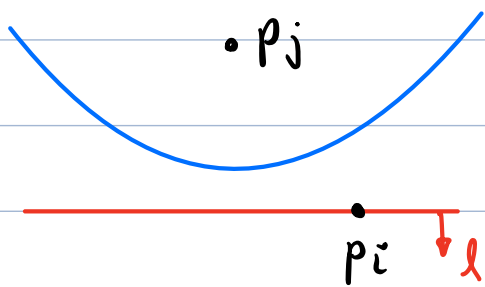
- For each consecutive triple $\langle \dots p_i, p_j, p_k \dots \rangle$ on beach line compute lowest y-coord of circumcircle (p_i, p_j, p_k)
- Schedule vertex event when sweep line reaches this y-coord.

Site Event: for site p_i :

(1) Find arc of beach line above p_i

(2) Split this arc:

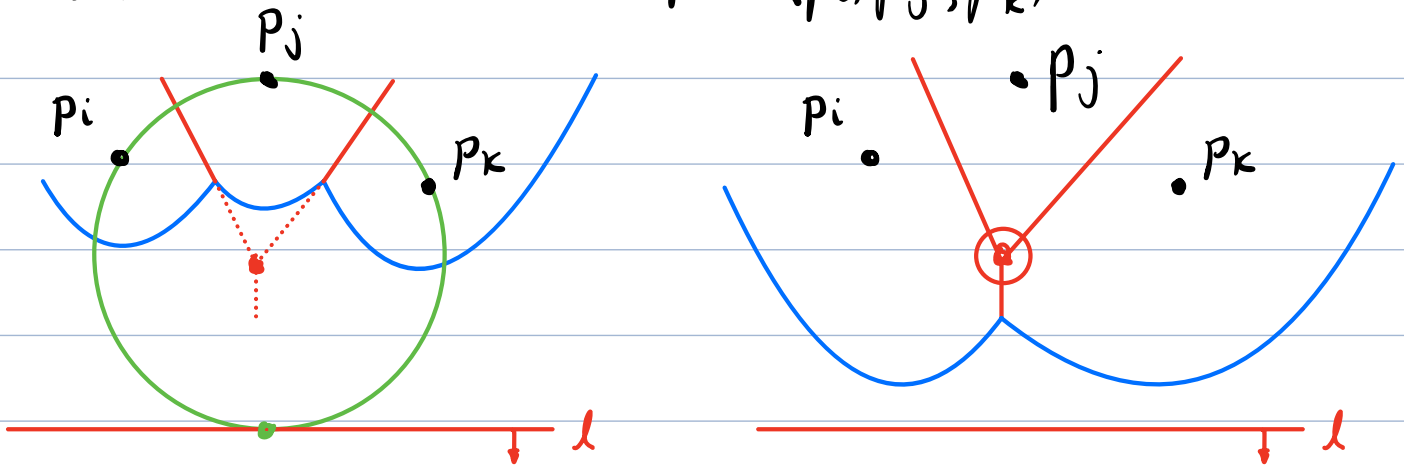
$$\langle \dots, p_j, \dots \rangle \Rightarrow \langle \dots p_j, p_i, p_j, \dots \rangle$$



(3) Create a new "dangling" edge (between p_i and p_j) add to Voronoi diagram.

(4) Update priority queue vertex events (below)

Vertex Event: for triple $\langle p_i, p_j, p_k \rangle$



(1) Delete p_j 's arc from beach line

$$\langle \dots p_i p_j p_k \dots \rangle \Rightarrow \langle \dots p_i p_k \dots \rangle$$

(2) Create new Voronoi vertex joining edges $p_i p_j$ + $p_j p_k$ in diagram

(3) Start new (partial) Voronoi edge for $p_i p_k$

(4) Update priority queue vertex events

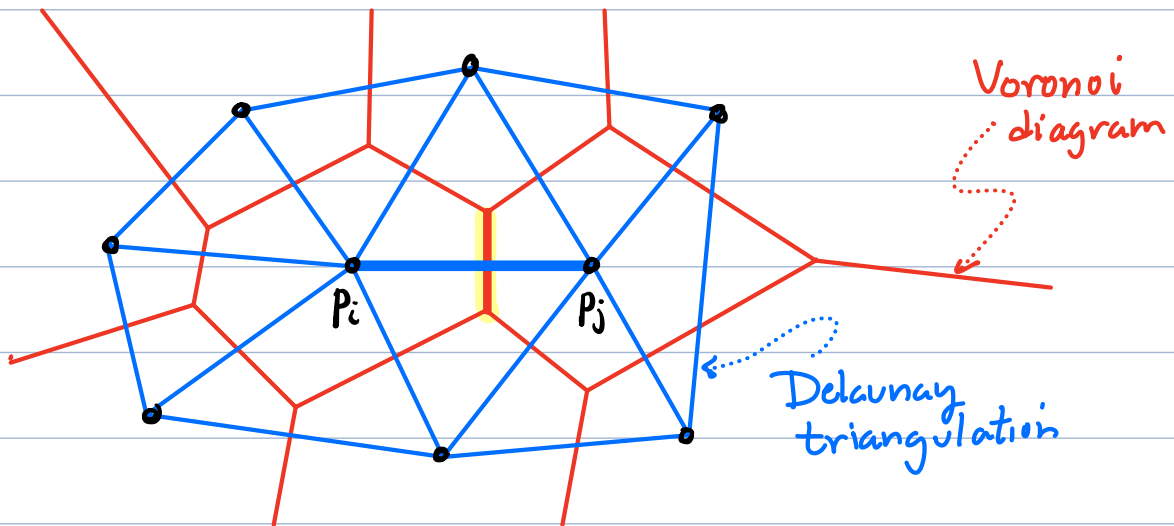
Analysis:

- $O(n)$ events
- $O(\log n)$ per event
- $O(n \log n)$ total time

CMSC 754 - Computational Geometry

Lecture 11: Delaunay Triangulations (Properties)

Last lecture - Voronoi Diagrams
This - The dual structure - Delaunay Triangulations



Delaunay Triangulation:

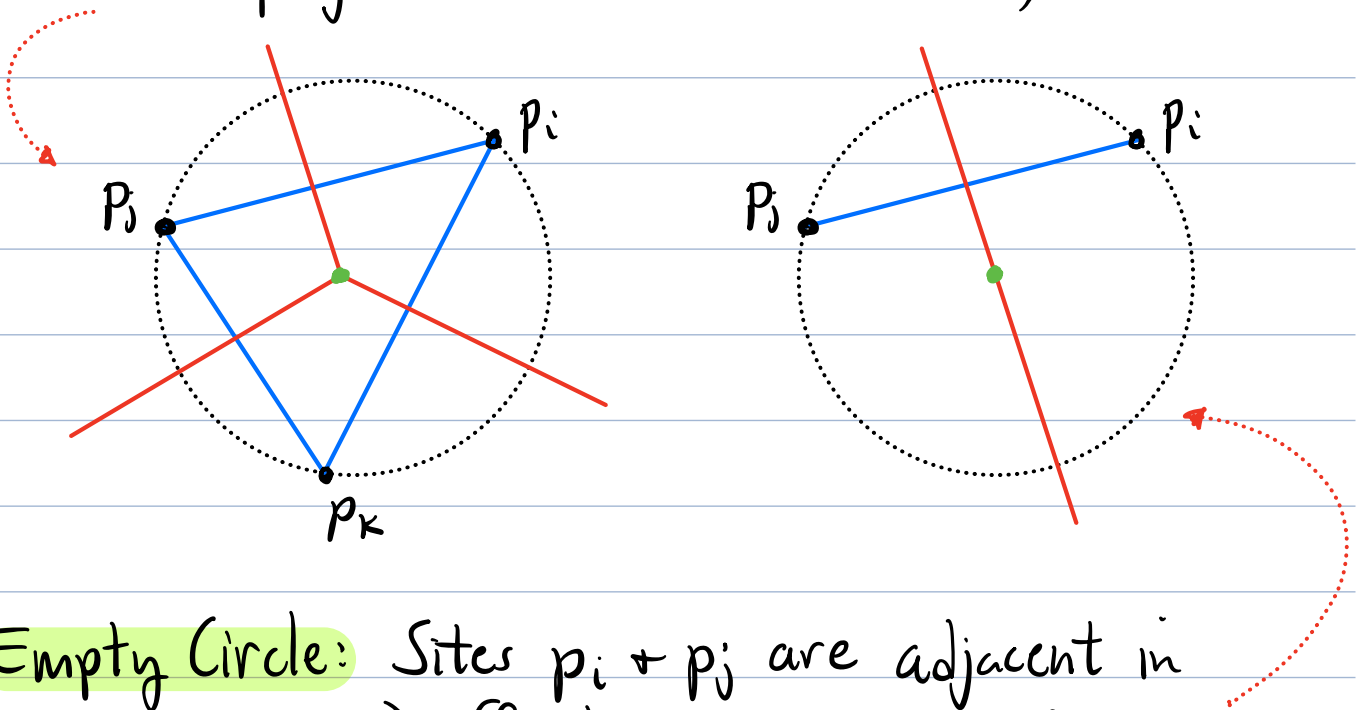
Given a set $P = \{p_1, \dots, p_n\}$ of sites in $\mathbb{R}^2$, the Delaunay Triangulation is the cell complex whose vertices are sites & there is an edge $\overline{p_i p_j}$ iff $V(p_i) \& V(p_j)$ share a common edge. Called $DT(P)$

Properties:

Triangulation: If general position (no four sites cocircular), the internal faces are all triangles

Hull: The boundary of the external face is the boundary of $\text{conv}(P)$

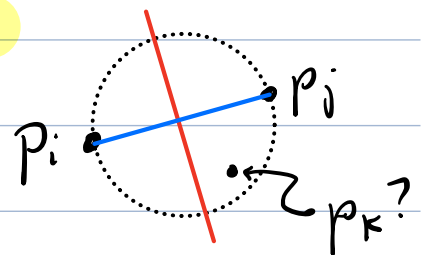
Circumcircle: The circumcircle of any triangle is empty (no sites in its interior)



Empty Circle: Sites p_i & p_j are adjacent in $\text{DT}(P)$ iff there is an empty circle through p_i & p_j .

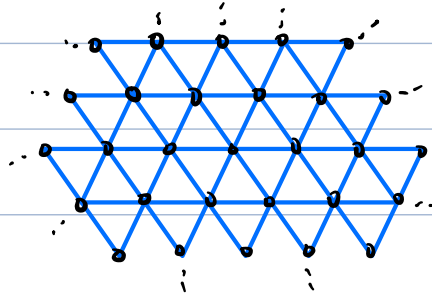
Closest Pair: The closest pair of sites are Delaunay neighbors

- Consider the circle with diameter $\overline{p_i p_j}$.
- No site p_k can lie within
- Apply empty circle prop.



Combinatorial Complexity:

By applying Euler's formula, there are at most $2n$ triangles and at most $3n$ edges



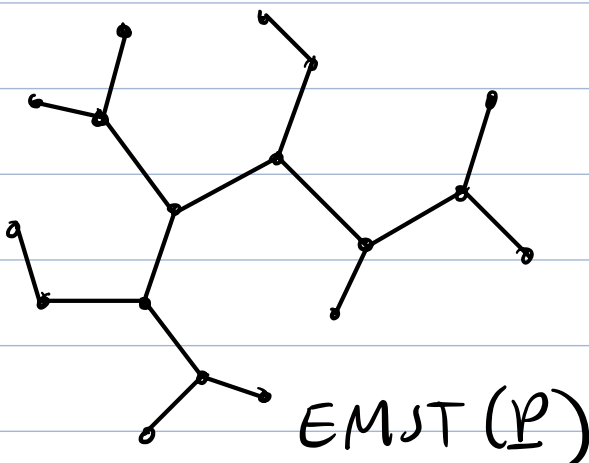
[In $\mathbb{R}^d$, size is $\mathcal{O}(n^{\lceil d/2 \rceil})$]

Euclidean Minimum Spanning Tree: (EMST)

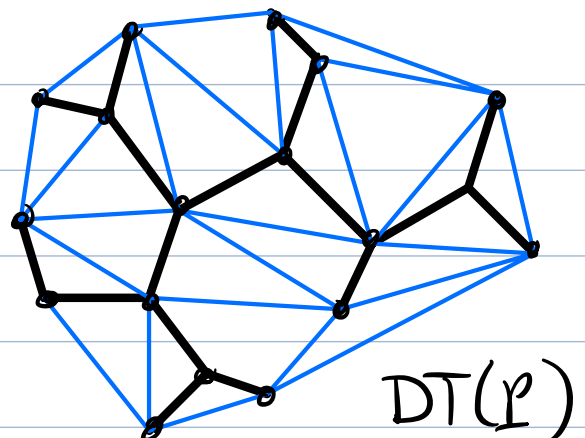
Euclidean graph: Complete graph on vertex set $P = \{p_1, \dots, p_n\}$, where edge weight is Euclidean distance ($w(p_i, p_j) = \|p_i - p_j\|$)

$EMST(P) =$ MST of Euclidean graph
(lowest weight tree spanning P)

Thm: $EMST(P) \subseteq DT(P)$



$\subseteq$



Proof: (Contradiction)

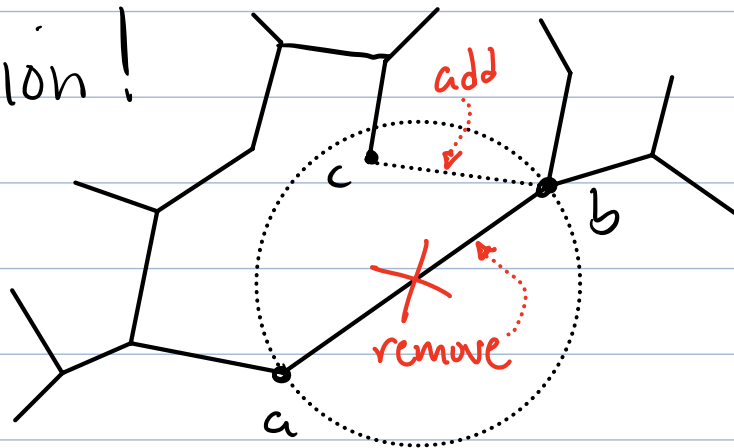
- Suppose some edge $\overline{ab} \in \text{EMST}(P)$
but not in $\text{DT}(P)$

- Empty circle $\Rightarrow$ circle with diameter $\overline{ab}$ contains site c

- $\|ac\| < \|ab\|$
 $\|bc\| < \|ab\|$

- Can remove $\overline{ab}$ from EMST + replace
with either $\overline{ac}$ or $\overline{bc}$ to produce
a spanning tree of lower weight

- Contradiction!



Minimum Weight Triangulation: No!

$\text{MWT}(P)$ = triangulation of P whose
sum of edge lengths is minimum

Generally $\text{MWT}(P) \neq \text{DT}(P)$

Notation: Given graph $G=(V,E)$ and vertices $u,v \in V$, let $\delta_G(u,v)$ = shortest path distance in G from u to v .

Spanner Properties:

Given a graph G and $t \geq 1$, a t -spanner is a subgraph G' of G on same vertex set s.t. $\forall u,v \in V$,

$$\delta_{G'}(u,v) \leq t \cdot \delta_G(u,v)$$

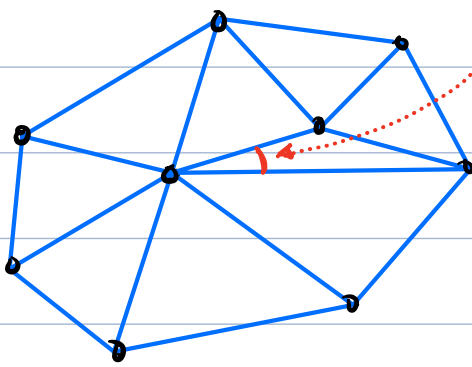
(Path lengths don't stretch too much)

Theorem (Keil + Gutwin, '92) Given a set P of sites in the plane, $DT(P)$ is a $4\pi\sqrt{3}/9 \approx 2.418$ spanner of the Euclidean graph. That is, $\forall p,q \in P$

$$\delta_{DT(P)}(p,q) \leq \frac{4\pi\sqrt{3}}{9} \cdot \|p-q\|$$

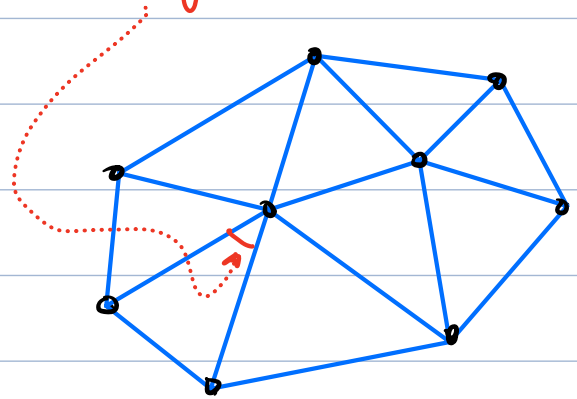
Avoids skinny triangles:

Let P be a set of sites in the plane. Among all possible triangulations of P , $DT(P)$ maximizes the size of the smallest angle.



other triangulation

smallest angle

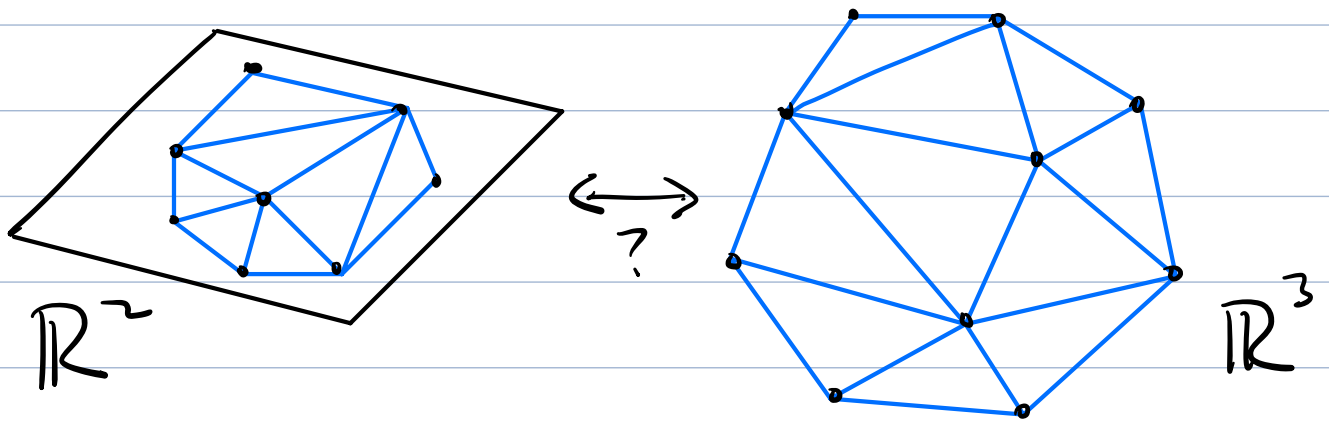


$DT(P)$

Thm: If all angles of all triangulations are ordered small to large, $DT(P)$ is the largest lexicographically compared to all triangulations of P .

(See full lecture notes)

Relationship to polytopes in $\mathbb{R}^{d+1}$



Delaunay triangulation in $\mathbb{R}^d$ is the projection of a lower convex hull in $\mathbb{R}^{d+1}$

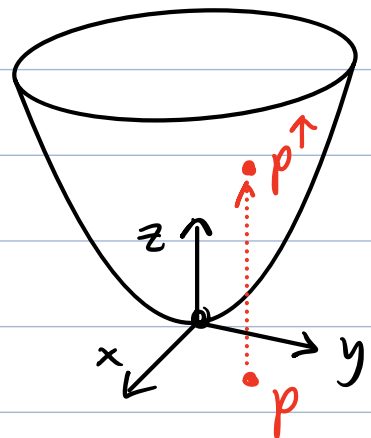
Voronoi diagram in $\mathbb{R}^d$ is the projection of a lower envelope of hyperplanes in $\mathbb{R}^{d+1}$

→ We'll prove the first only: $\mathbb{R}^2 \rightarrow \mathbb{R}^3$

Consider the paraboloid:

$$z = f(x, y) = x^2 + y^2$$

Given $p = (p_x, p_y)$, define $p^\uparrow$ to be $(p_x, p_y, p_x^2 + p_y^2)$

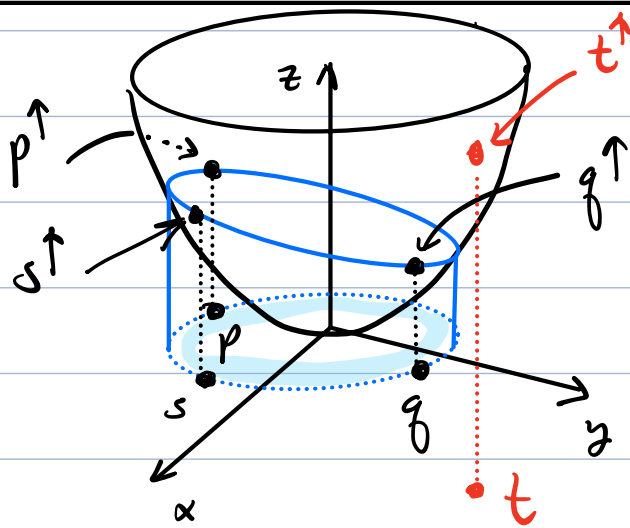
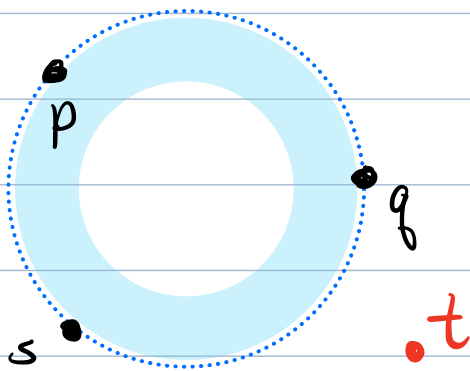


Lemma:

Three pts $p, q, s \in \mathbb{R}^2$
have an empty
circumcircle w.r.t. P

$\Leftrightarrow$

Three pts $p^\uparrow, q^\uparrow, s^\uparrow$
lie on plane h
with all pts of $P^\uparrow$
above h

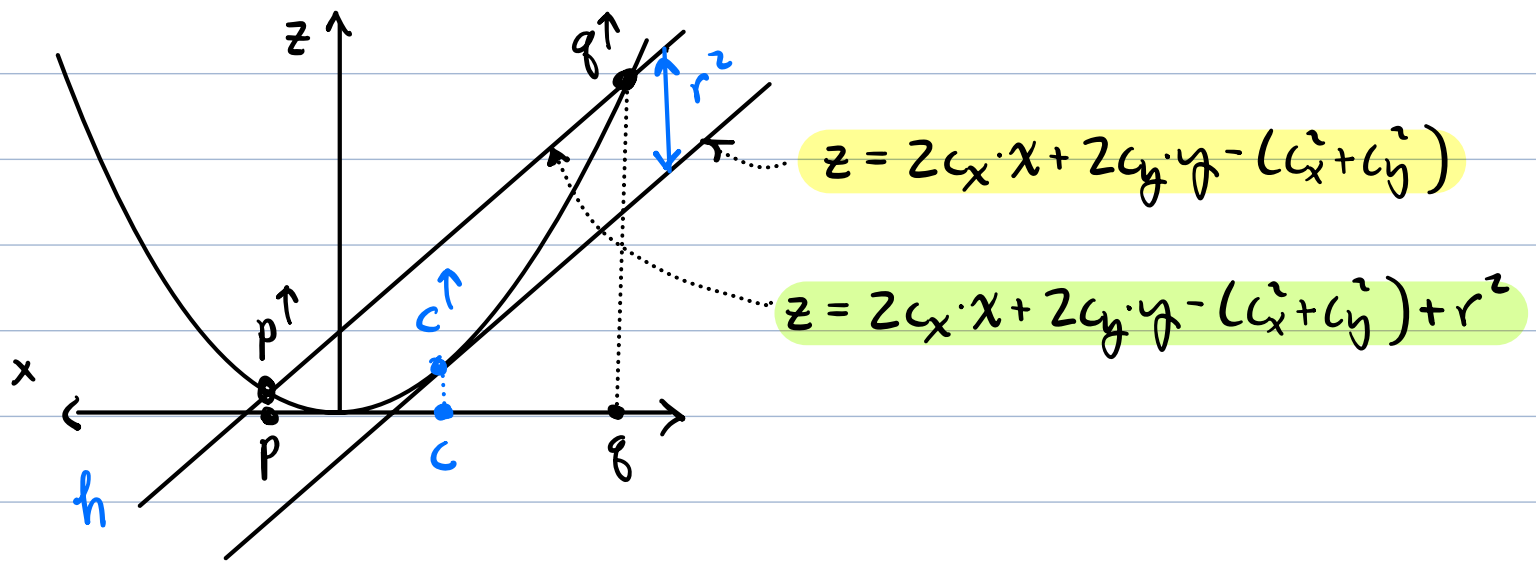


- Let $c = (c_x, c_y)$ be center of circumcircle through p, q, s & let r be its radius
- The plane tangent to paraboloid at $c^\uparrow$ is:

$$z = 2c_x \cdot x + 2c_y \cdot y - (c_x^2 + c_y^2)$$

- Shift this plane up by distance r^2 :

$$h: z = 2c_x \cdot x + 2c_y \cdot y - (c_x^2 + c_y^2) + r^2$$



- All 3 lifted pts lie on this plane:

$$p_x \text{ on circle: } (p_x - c_x)^2 + (p_y - c_y)^2 = r^2$$

$$\Leftrightarrow (p_x^2 - 2p_x c_x + c_x^2) + (p_y^2 - 2p_y c_y + c_y^2) = r^2$$

$$\Leftrightarrow p_x^2 + p_y^2 = 2c_x \cdot p_x + 2c_y \cdot p_y - (c_x^2 + c_y^2) + r^2$$

$$\Leftrightarrow p_z^\uparrow = 2c_x \cdot p_x^\uparrow + 2c_y \cdot p_y^\uparrow - (c_x^2 + c_y^2) + r^2$$

$$\Leftrightarrow p^\uparrow \text{ lies on plane } h$$

- Can also show that t outside/inside circle $\Leftrightarrow t^\uparrow$ above/below h

$\Rightarrow \Delta pqs$ is Delaunay

$\Leftrightarrow$ circumcircle pqs is empty

$\Leftrightarrow \exists$ supporting hyperplane
through $p^\uparrow q^\uparrow s^\uparrow$

$\Leftrightarrow p^\uparrow q^\uparrow s^\uparrow$ define facet of
convex hull of $P^\uparrow$

$\Rightarrow$ Delaunay triangulation of P

$\equiv$

Convex hull of $P^\uparrow$

Summary:

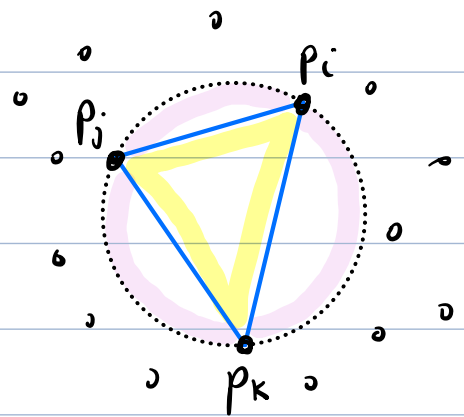
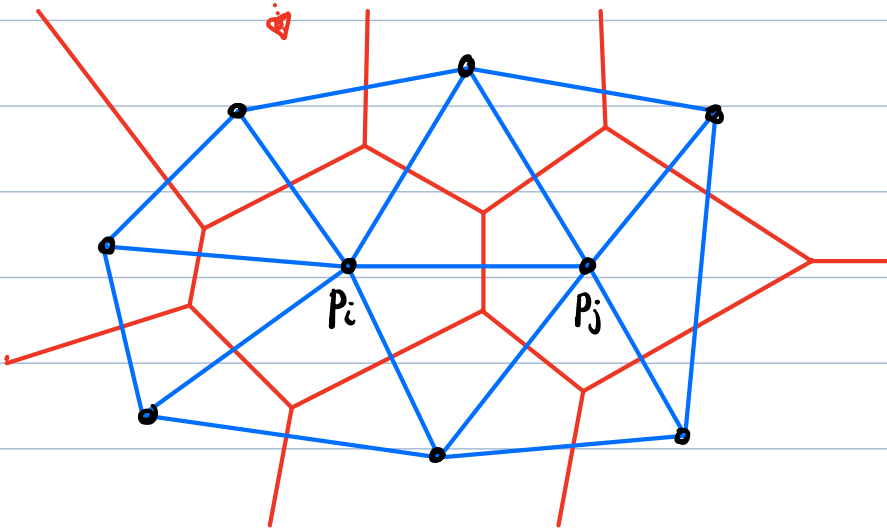
Delaunay triangulations

- Dual of Voronoi diagram
- Empty circumcircle condition
- $EMST \subseteq DT$
- Not min weight triangulation
- Maximizes min angle (in $\mathbb{R}^2$)
- Is a ~ 2.42 spanner
- Convex hull of lifted points

CMSC 754 - Computational Geometry

Lecture 12: Delaunay Triangulations (Construction)

Last lecture: - Delaunay triangulation + properties
- Given a set $P = \{p_1, \dots, p_n\}$ of sites,
 $DT(P)$ is the dual of $Vor(P)$

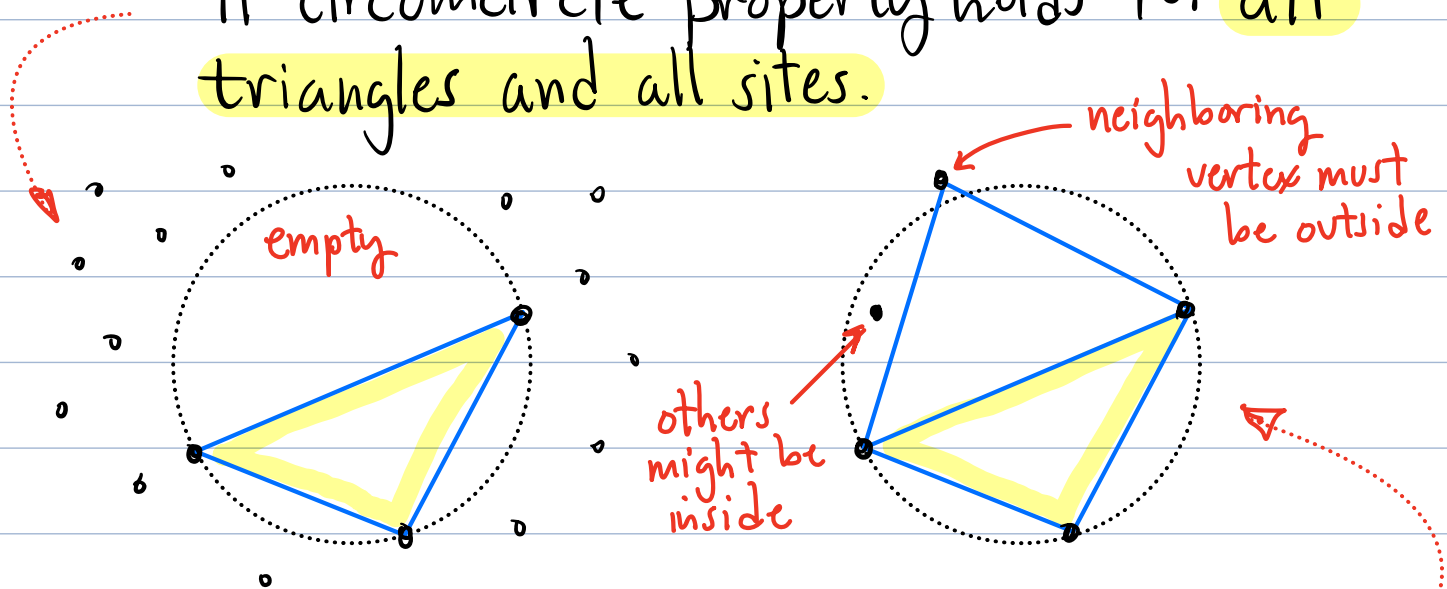


Circumcircle Property:

$\Delta p_i p_j p_k \in DT(P)$ iff circumcircle
of p_i, p_j, p_k contains no sites

Local/Global Delaunay:

- A triangulation is globally Delaunay if circumcircle property holds for all triangles and all sites.



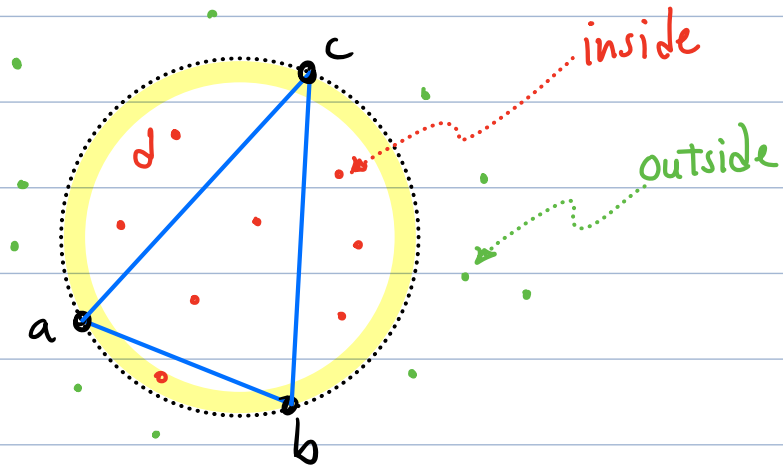
- A triangulation is locally Delaunay if circumcircle property holds the vertices of every pair of adjacent triangles.

Does it matter? No.

Thm (Delaunay): A triangulation is globally Delaunay iff it is locally Delaunay.

(See lecture notes / text for proof)

Incircle Test: Given points $a, b, c + d \in \mathbb{R}^2$,
 does d lie in circumcircle of Δabc ?
 (Assume a, b, c given in CCW order)



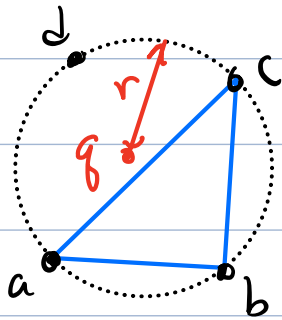
inCircle($a, b, c; d$): d is inside if

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

Obs:

- This is an **orientation test in $\mathbb{R}^3$**
- Generalizes to **any dimension**
- Computable in **$O(1)$ time** in any fixed dimension.

Why? Consider boundary case $\rightarrow$ cocircular
 Center $q = (q_x, q_y)$ radius $= r$



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

$$\Rightarrow \boxed{-2q_x} a_x \boxed{-2q_y} a_y + \boxed{1} (a_x^2 + a_y^2) + \boxed{(q_x^2 + q_y^2 - r^2)} = 0$$

Same applies to $b, c, d \Rightarrow$

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

$\rightarrow$ A linear combination of columns is identically 0

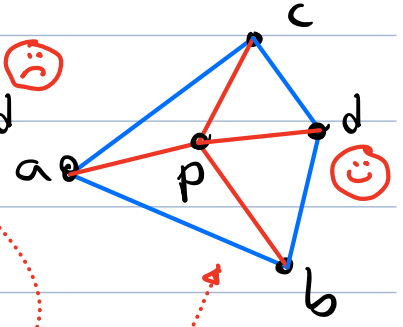
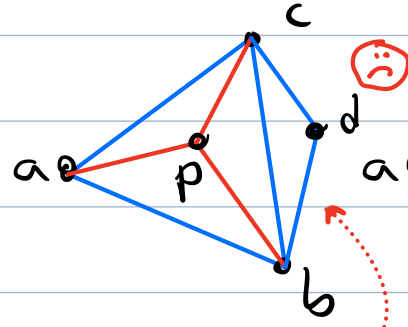
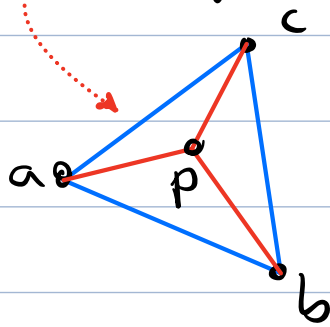
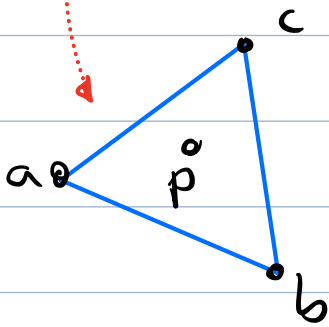
$\Rightarrow$ column vectors are lin. dependent
 $\Rightarrow$ det of matrix is 0

(Randomized) Incremental Construction:

- Add sites one-by-one in random order + update the triangulation after each.

① Find triangle Δabc containing the new site p .

② Add edges connecting p to a, b, c



③ Check neighboring triangles for violations of local Delaunay

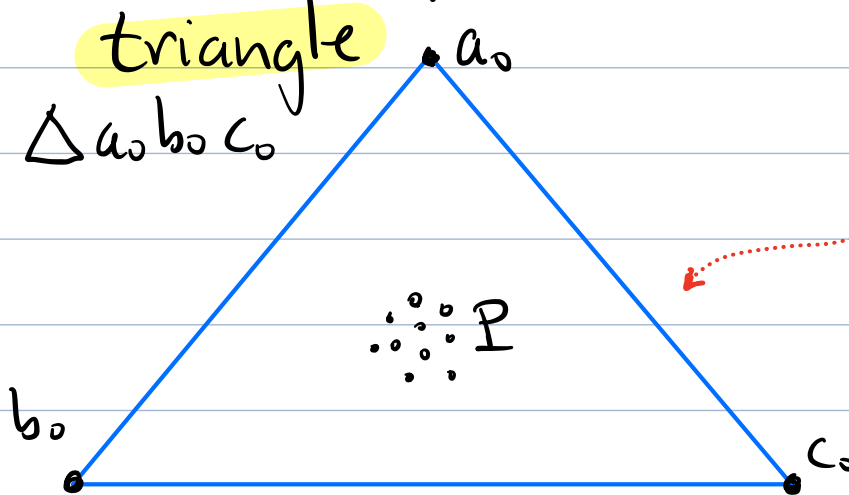
④ Apply edge flips to correct these

⑤ Repeat until local Delaunay for all neighbors

Sentinel sites:

- If new site is not in convex hull - it's not in any triangle!

- Fix: Enclose points in a **HUGE** triangle



How huge? No circumcircle from P should contain a_0, b_0 or c_0

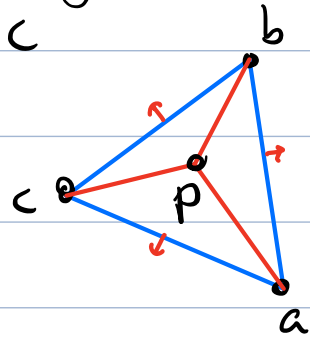
(see text for how)

Build-Delaunay($P = \{p_1, \dots, p_n\}$)

- Create sentinel triangle $\Delta a_0 b_0 c_0$ containing P
- Randomly permute P
- for $i = 1$ to n Insert(p_i)

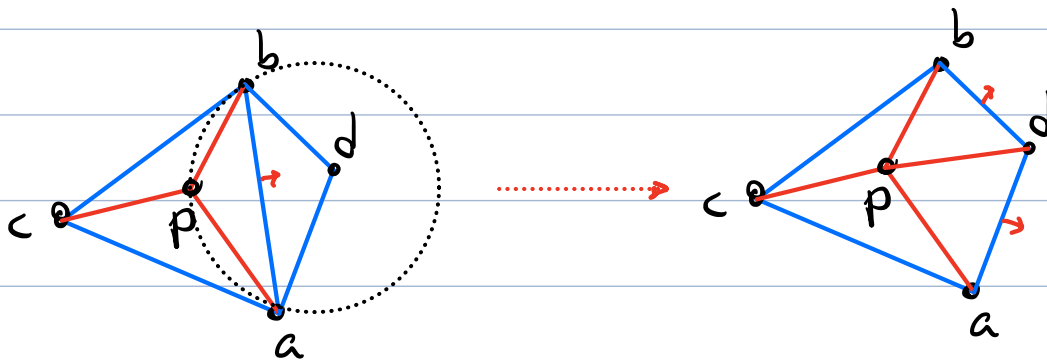
Insert (p):

- Find $\triangle abc$ containing p
- Add edges pa, pb, pc
- SwapTest(ab)
- " (bc)
- " (ca)

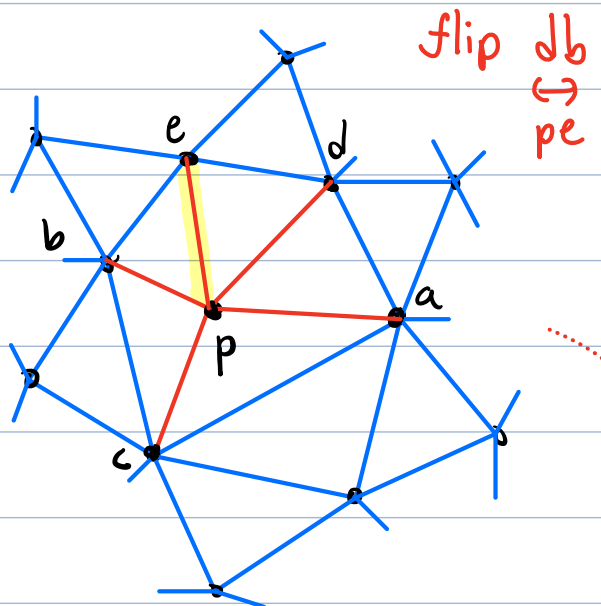
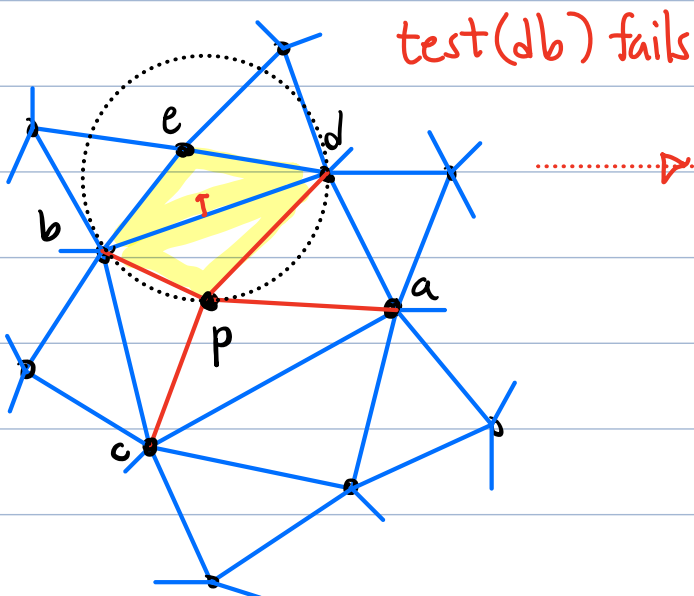
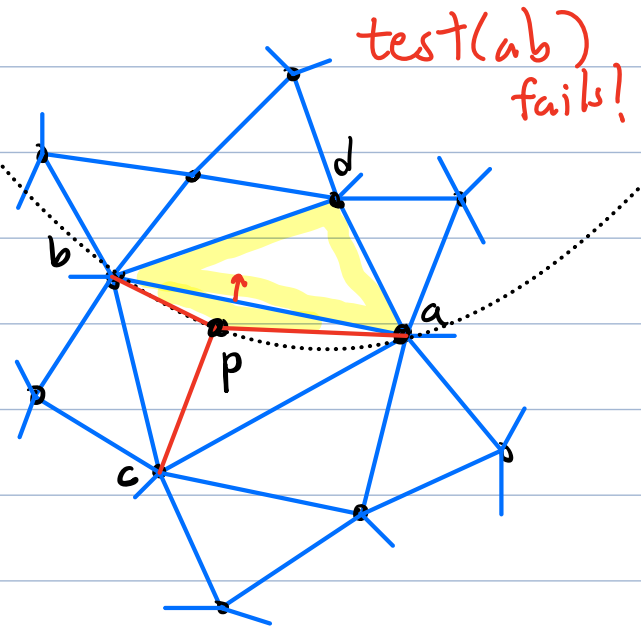
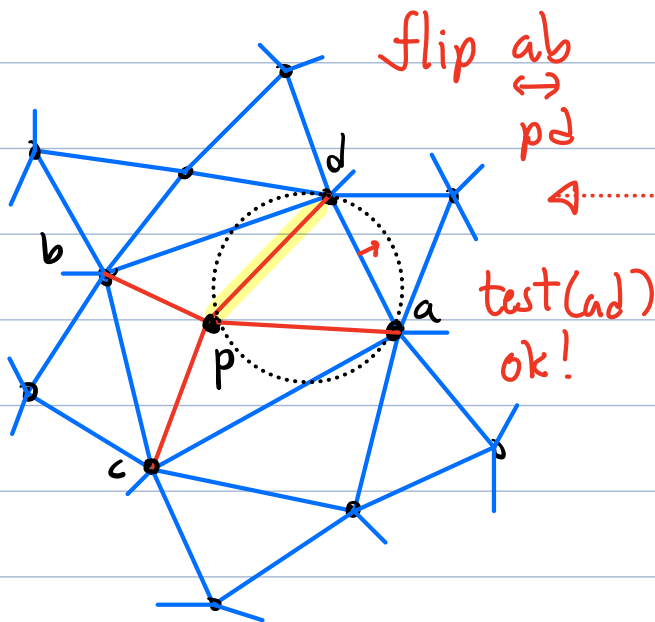
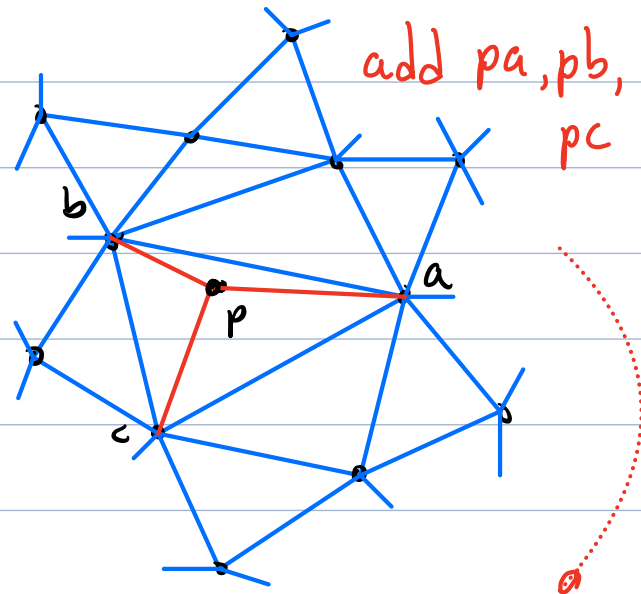
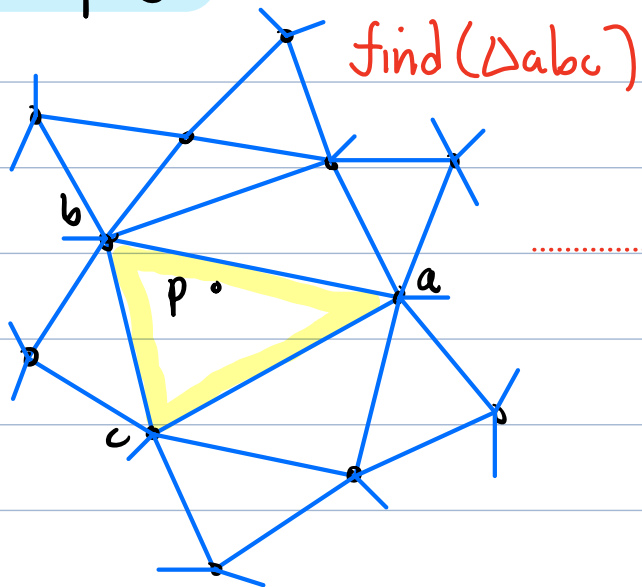


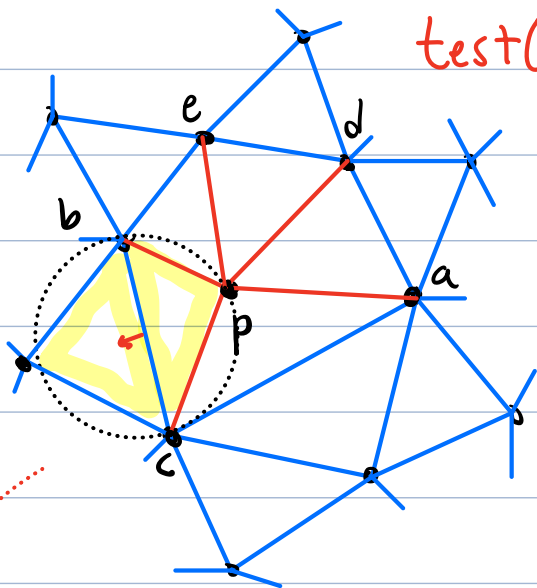
SwapTest(ab):

- if (ab is edge of external face)
 return
- $d \leftarrow$ vertex opposite p on ab
- if (inCircle(p, a, b, d))
- flip edge ab
 (for pd)
- SwapTest(ad)
- SwapTest(db)

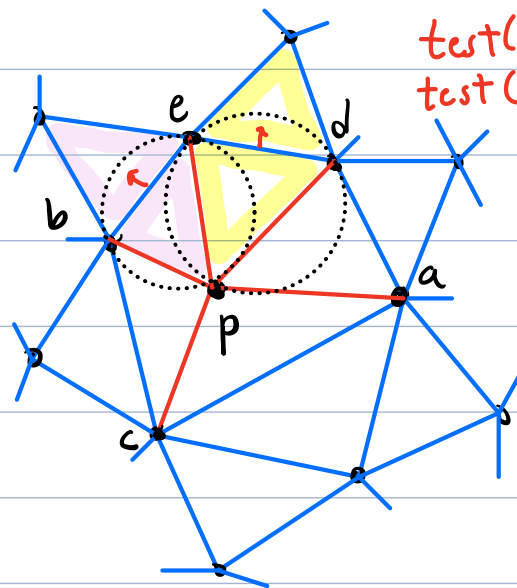


Example:

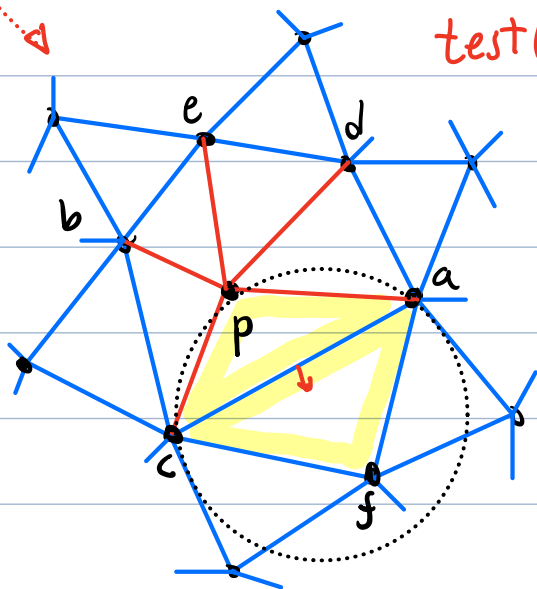




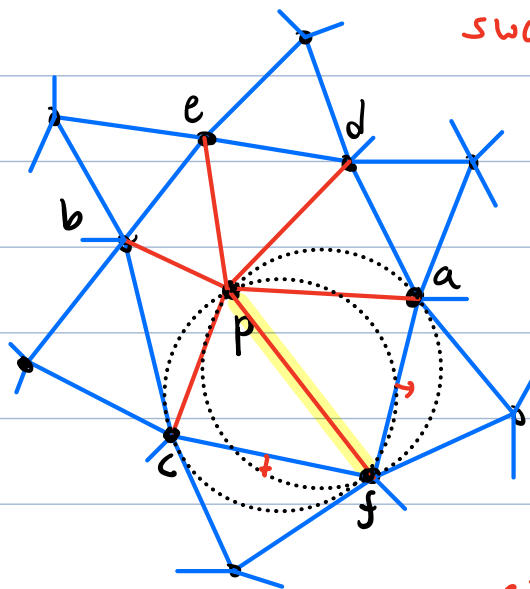
test(bc)
ok!



test(de) ok!
test(cb) ok!



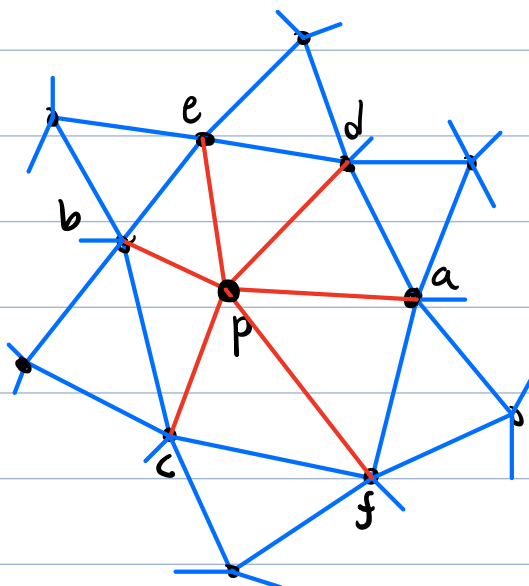
test(ca)
fails



swap ca
↔
pf

Done!

test(cf) ok!
test(fa) ok!



Final

Note: All new
edges are incident
to p

Correctness:

- Only triangles that could violate local Delaunay are incident to p , + we check all
- By Delaunay's Thm, local $\Rightarrow$ global Delaunay

Running time:

- for each insertion $p_1, \dots, p_n$
- find triangle containing $p_i \rightarrow O(\log n)$
- swap tests + edge flips $\rightarrow O(1)$ in expectation
- Total: $O(n \log n)$

Lemma: The expected update time (swap tests + edge flips) is $O(1)$

Proof: (Backwards analysis)

- Update time $\sim$ degree of p in final Δ -tion
- Every pt is equally likely to be last
- $\sum_i \deg(p_i) = 2(\# \text{edges}) \leq 2(3n) = 6n$
- Expected time $\sim$ Average degree $\leq \frac{1}{n} \cdot 6n = 6$

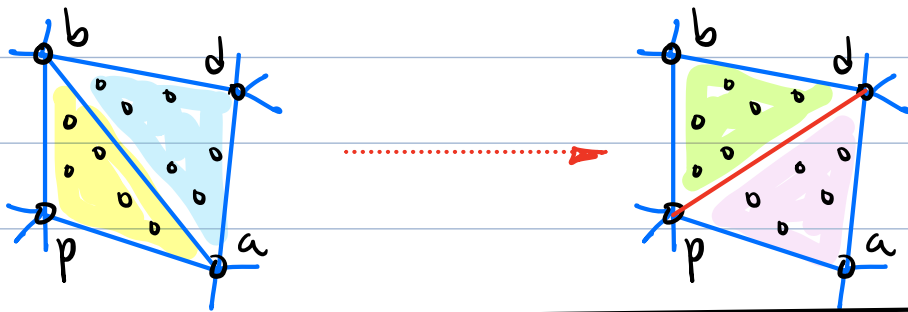
□

Point-Location:

Bucketing:

- Each triangle maintains ↗ future sites in this triangle
- To locate a point, just get its bucket id.
- When an edge flip is performed rebucket the affected sites

called a "bucket"



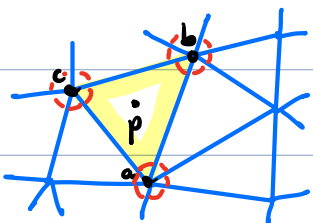
Lemma: Let p be any site. The probability that p is rebucketed as result of i^{th} insertion is $\leq 3/i$

Proof: (Backwards Analysis)

- Sites are rebucketed only if in new triangle
- All new triangles are incident to last site
- Each site is equally likely to be lost
- $\text{Prob}(p \text{ is rebucketed})$ [let $p \in \Delta abc$]

$$\leq \text{Prob}(a, b, \text{ or } c \text{ was last inserted}) \\ \leq 3/i$$

□



Lemma: Total time for rebucketing is $O(n \log n)$ in expectation

Proof: Rebucket time (expected)

$$= \sum_{p \in P} \sum_{i=1}^n 1 \cdot \text{Prob}(p \text{ was rebucketed in } i^{\text{th}} \text{ insertion})$$

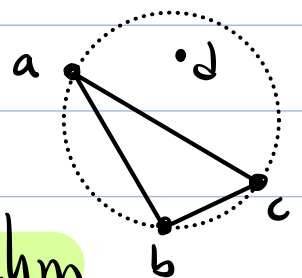
$$\leq \sum_{p \in P} \sum_{i=1}^n 3/i \approx \sum_{p \in P} 3 \cdot \ln n \quad (\text{Harmonic series})$$

$$= 3n \ln n$$

$$= O(n \log n) \quad \square$$

Summary:

- Local vs. global Delaunay
 - Proved local $\Rightarrow$ global
- Incircle test
 - Orientation test in $\mathbb{R}^2$
- Randomized incremental algorithm for DT
- Update time - $O(1)$ per pt. expected
- Point location - $O(\log n)$ per pt. "

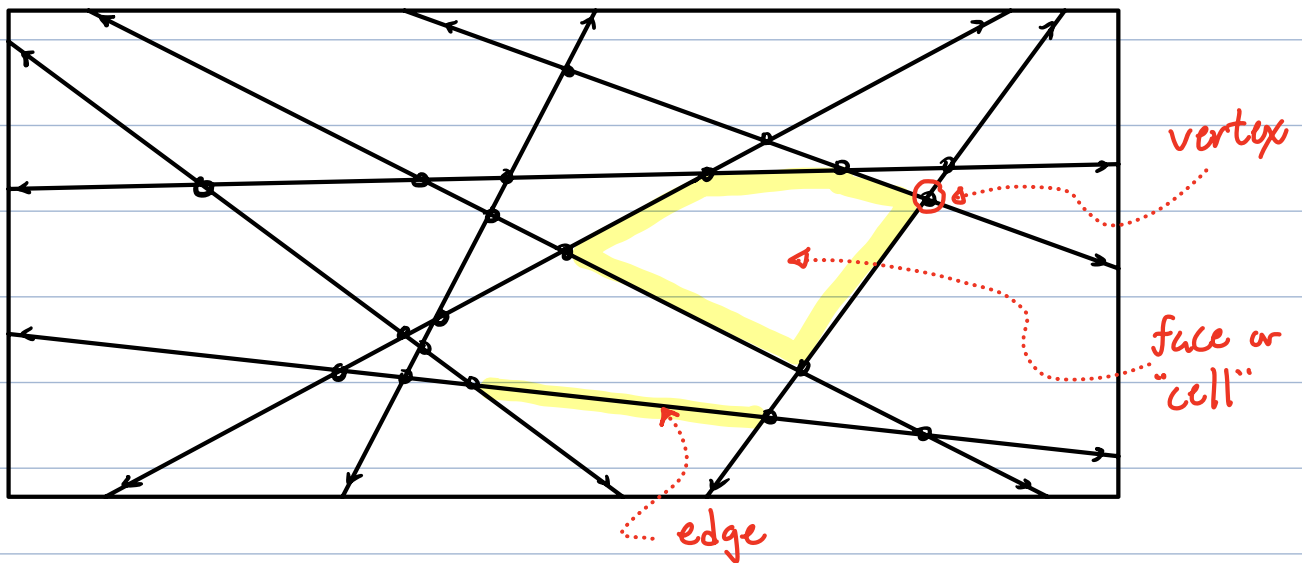


CMSC 754 - Computational Geometry

Lecture 13 - Line Arrangements

Arrangement:

Given a set $L = \{l_1, \dots, l_n\}$ of lines in $\mathbb{R}^2$ (generally $(d-1)$ -dim hyperplanes in $\mathbb{R}^d$), they subdivide the plane into a cell complex called the arrangement of L , or $A(L)$.



Combinatorial Properties:

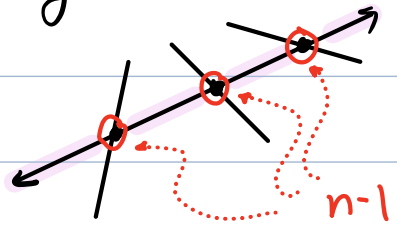
Lemma: Given n lines L in gen'l position in $\mathbb{R}^2$:

- (i) $A(L)$ has $\binom{n}{2} = \frac{1}{2} \cdot n(n-1)$ vertices
- (ii) $A(L)$ has n^2 edges
- (iii) $A(L)$ has $\binom{n}{2} + n + 1 = \frac{1}{2}(n^2 + n + 2)$ cells

Proof:

- (i) Each pair intersects once = $\binom{n}{2}$ ✓

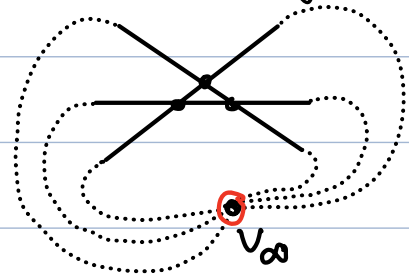
(ii) Each line is split by $n-1$ others into n edges
 $\Rightarrow n^2$ total $\checkmark$



(iii) Add a vertex at ∞ of degree n to tie off all unbounded edges

$$v = \binom{n}{2} + 1$$

$$e = n^2$$



By Euler's formula:

$$v - e + f = 2$$

$$\Rightarrow \left(\binom{n}{2} + 1\right) - n^2 + f = 2$$

$$\Rightarrow f = 2 + n^2 - \left(\binom{n}{2} + 1\right)$$

$$\Rightarrow f = 2 + n^2 - \frac{n(n-1)}{2} - 1$$

$$= \frac{1}{2}(n^2 + n + 2) \checkmark$$

□

[In $\mathbb{R}^d$, complexity is $\Theta(n^d)$]

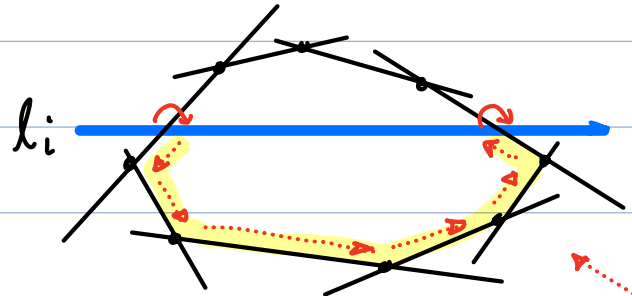
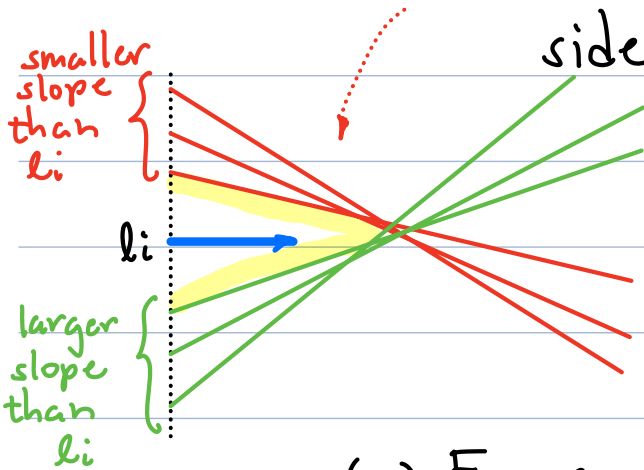
Incremental Construction: (not randomized)

Idea: Add lines one by one (in any order)
 Update the structure after each

Notation: $L_i = \{l_1, \dots, l_i\}$

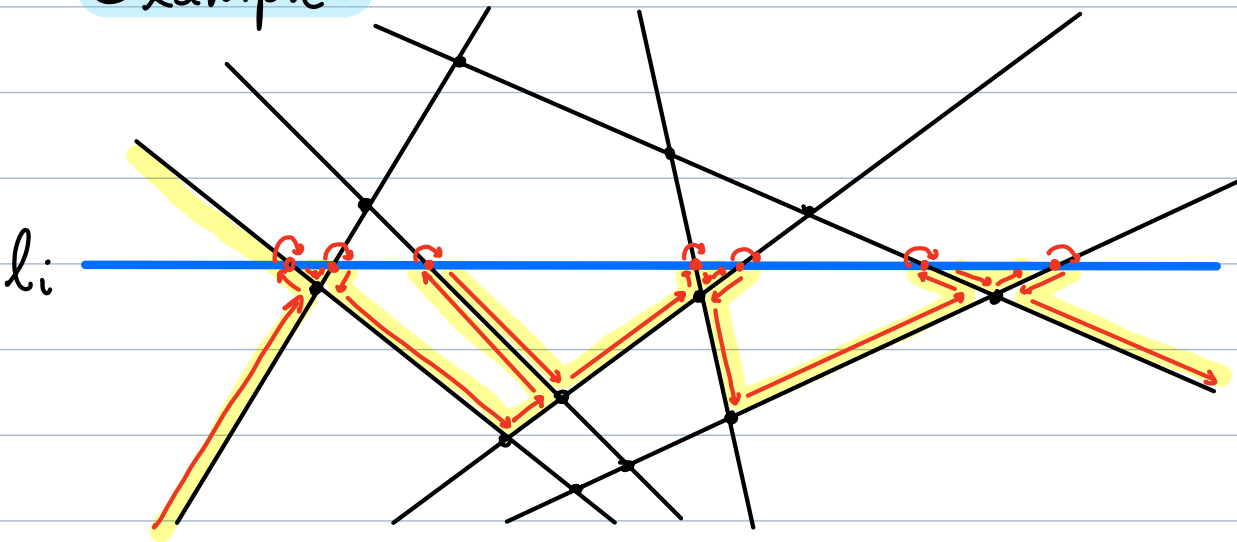
How to add the i^{th} line? l_i

(1) Find the unbounded cell on left side where l_i starts (slope based)



(2) For each face of $A(L_{i-1})$ that intersects l_i , walk along its lower boundary to determine where it exits this cell

Example:



- Once we know entry-exit points on each face - we update arrangement in $O(i)$ time (DCEL)
- How long to crawl around edges?

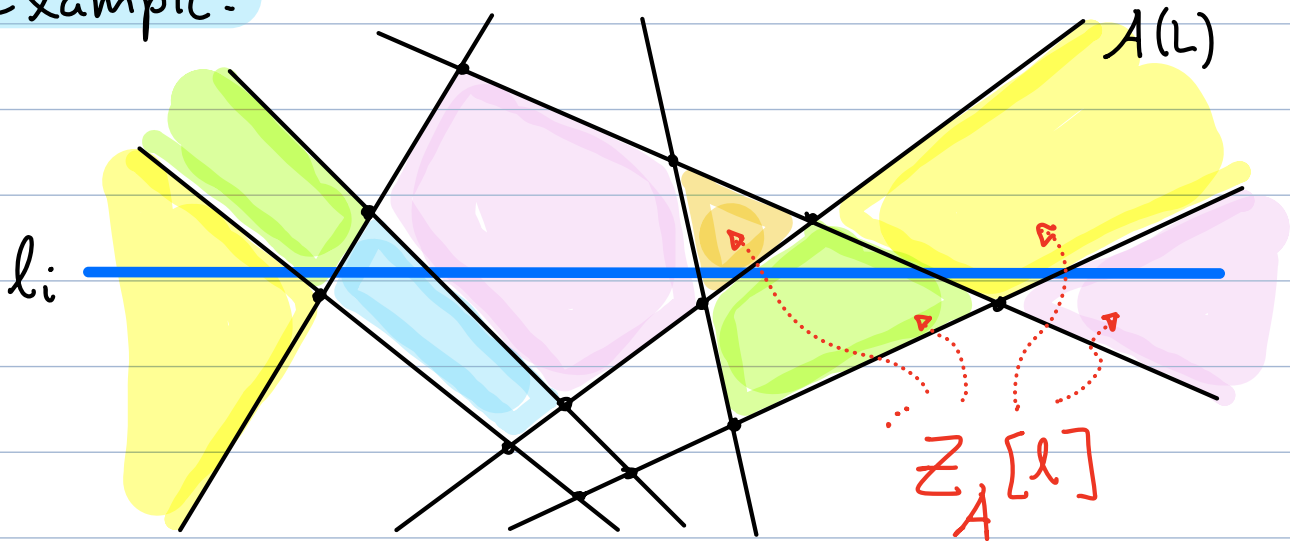
Naive analysis: On adding l_i

- l_i crosses i cells
- each cell may have as many as $i-1$ edges
- crawl takes $O(i(i-1)) = O(i^2)$ time
- total time $\approx \sum_{i=1}^n i^2 = O(n^3)$

Can it really be this bad?

Zone: Given an arrangement $A = A(L)$ and a line $l \notin L$, zone of l in A , $Z_A(l)$ is the set of cells of A that l intersects.

Example:

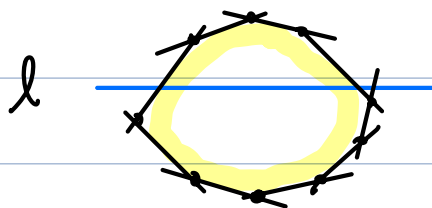


Obs: Crawl time $\leq$ no. of edges on the zone of l_i in $A(L_{i-1})$ [$Z_{A(L_{i-1})}(l_i)$]
 $\rightarrow$ We'll show this is $O(i)$ not $O(i^2)$

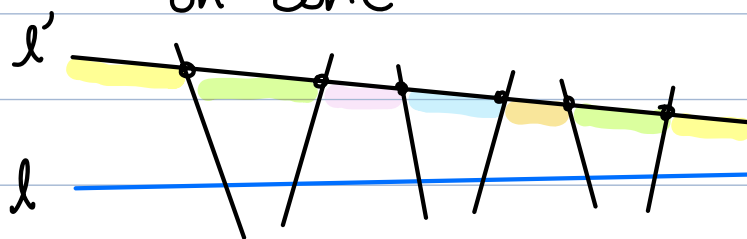
Theorem: (Zone Theorem) Given an arrangement $A(L)$ where $|L| = n$ and any line $l \in L$, the number of edges in $Z_A(l) \leq 6n$

How to prove this?

cell by cell? Some cells have high complexity



line by line? Some lines appear many times on zone



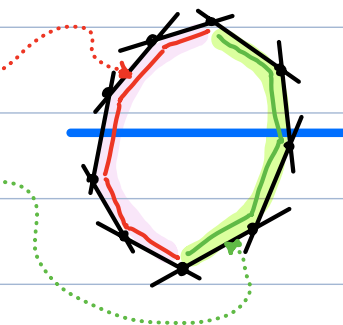
Our approach:

- Partition edges of zone into two classes (left side + right side)
- Show (by induction) at most $3n$ of each

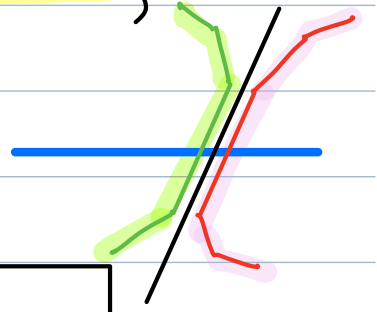
A zone edge is:

left bounding: on left side of cell

right bounding: on right side of cell



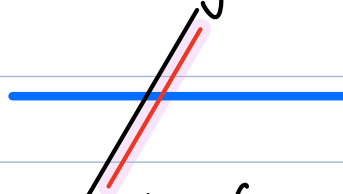
Note: Some edges appear twice in the zone, both as left/right bounding



Claim: At most $3n$ left-bounding edges.

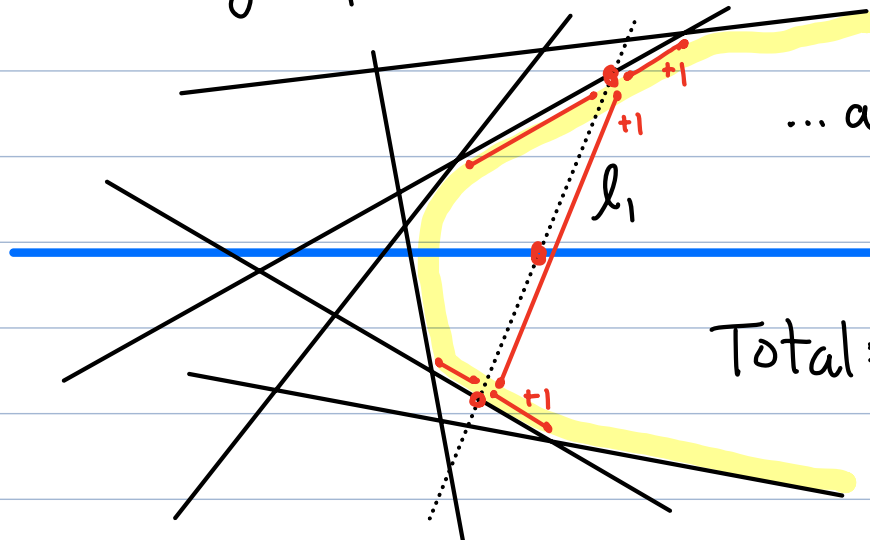
Proof: By induction on n

$n=1$: Just one LB edge $1 \leq 3 \cdot 1 \checkmark$



$n \geq 2$: I.H. arrangement of $n-1$ lines has $\leq 3(n-1)$ LB edges in zone

- Let $l_1 \in L$ be rightmost line to cross l
- Removing $l_1 \Rightarrow$ at most $3(n-1)$ LB edges
- Adding l_1 back creates ...



... at most 3 new LB edges

$$\text{Total: } \leq 3(n-1) + 3 = 3n \quad \square$$

Thm: Given a set L of n lines in $\mathbb{R}^2$,
 $A(L)$ can be built in time $O(n^2)$
[and has size $O(n^2)$... so this is optimal]

Proof: - Apply incremental construction

- Inserting l_i takes time $\sim$ no. of
edges in $Z_{A(L_{i-1})}(l_i) \leq 6(i-1)$

- Total time $\leq \sum_{i=1}^n 6(i-1) = 6 \sum_{i=0}^{n-1} i = O(n^2)$

Applications:

Line arrangements can be used to solve
many problems - mostly $O(n^2)$ time
- often using duality

How to process an arrangement?

- Build it + traverse it like a graph
 $O(n^2)$ time, $O(n^2)$ space

- Plane sweep

$O(n^2 \log n)$ time, $O(n)$ space

- Topological plane sweep

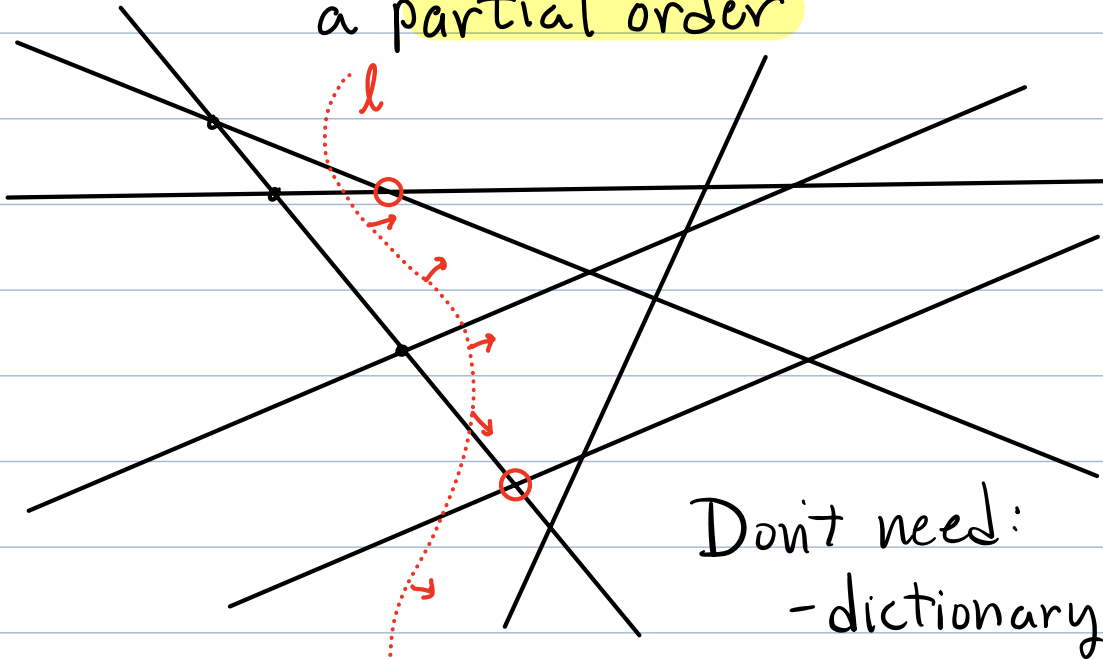
$O(n^2)$ time, $O(n)$ space

Not covered, but applicable pretty much
whenever plane sweep is.

you may
assume
this

Topological plane sweep:

- A relaxed version of plane sweep
- Vertices are not swept in strict left to right order, but based on a partial order

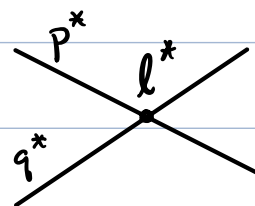
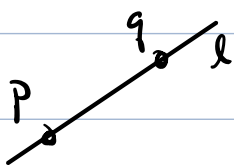


Don't need:

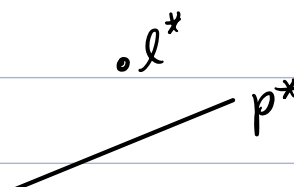
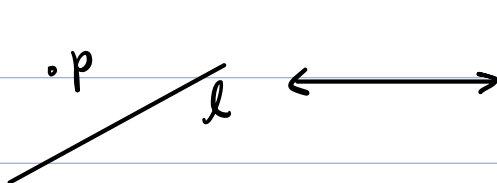
- dictionary
 - priority queue
- ⇒ saves $\log n$ factor

Recall: Dual transformation

$$\begin{aligned} p = (a, b) &\longleftrightarrow p^*: y = ax - b \\ l: y = ax - b &\longleftrightarrow l^*: (a, b) \end{aligned}$$



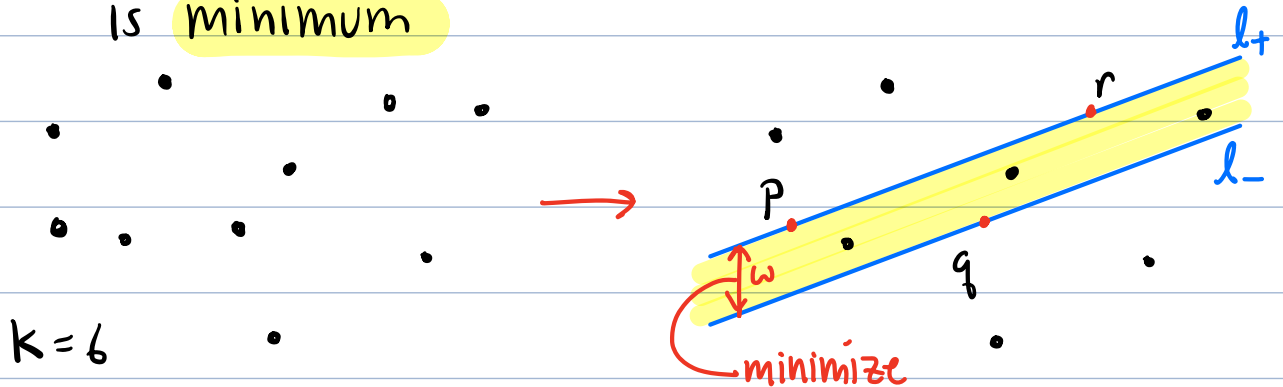
incidence preserving



order reversing

Narrowest k-corridor:

- Given a set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^2$ and integer $3 \leq k \leq n$, find pair of parallel (non vertical) lines that enclose k pts so that vertical distance between lines is minimum



Primal form:

- Let $l_+ + l_-$ be upper + lower lines of "slab"

$$l_+: y = ax - b_+ \quad b_+ \leq b_-$$
$$l_-: y = ax - b_-$$

parallel $\Rightarrow$ same slope

- Vertical width: $w = b_- - b_+$
- k pts of P lie on or between $l_- + l_+$

order reversed due to negation

Local optimality:

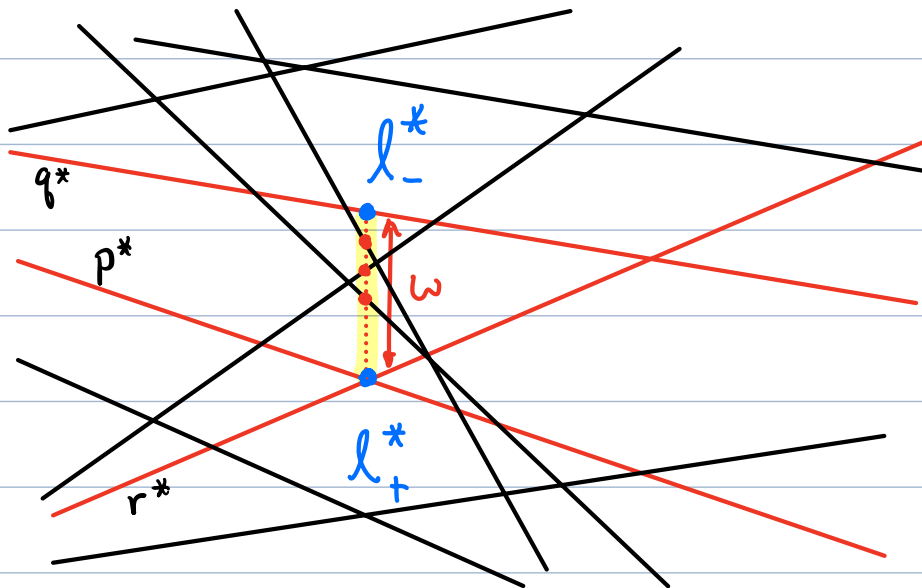
3 pts of P will lie on $l_+ + l_-$, 2 on one edge + 1 on other

- If 0, 1, or 2 can make width smaller
- If 4 or more - not gen'l position

Dual form:

- $l_+^* + l_-^*$ are pts $(a, b_+) + (a, b_-)$
- vertical distance $b_- - b_+$
- k lines of P^* pass through or between these pts

vertical line segment

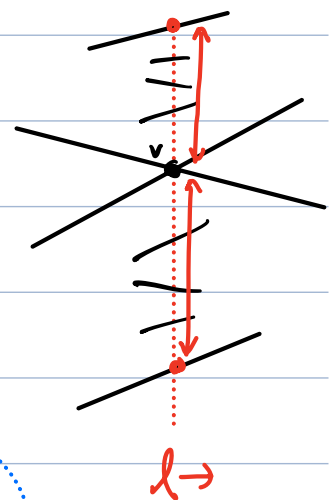


Local optimality:

3 lines will pass through $l_-^* + l_+^*$ with 2 on one side + one on other

Narrowest-Corridor (P, k):

- (1) $P^* \leftarrow$ dual lines of P
- (2) Plane sweep through P^* .
- (3) On arriving at each vertex v , compute vertical distance to lines $k-2$ above + $k-2$ below
- (4) Return smallest such distance



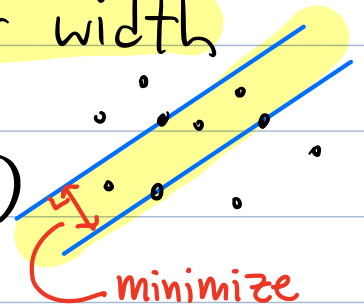
Correctness: (Argued above)

Can access in $O(1)$ time since sweep line can be stored in array

Time: $O(n^2 \log n)$ time + $O(n)$ space

↳ can reduce to $O(n^2)$ by topol. plane sweep.

Aside: It is easy to generalize this to minimize perpendicular width (Just apply a correction factor when computing widths)

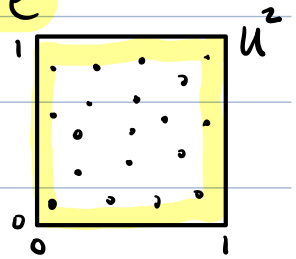


Halfplane Discrepancy:

Let $U^2 = [0,1]^2$ denote the unit square

Given n pts $P = \{p_1, \dots, p_n\} \subset U^2$,

how close is P to being uniformly distributed over U^2 ?



Idea:

For any halfplane h , let

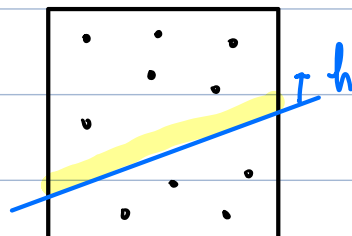
$$\mu(h) = \text{area}(h \cap U^2)$$

$$[0 \leq \mu(h) \leq 1]$$

the fraction of P in h

$$\mu_P(h) = |h \cap P| / |P|$$

$$[0 \leq \mu_P(h) \leq 1]$$



$$\mu(h) = 2/3 = 0.666...$$

$$\mu_P(h) = 6/10 = 0.6$$

If P is uniformly distrib., we expect

$$\mu(h) \approx \mu_P(h) \quad \forall h$$

To measure how uniform is P , define:

$$\Delta(P) = \max_h |\mu(h) - \mu_P(h)|$$

Called the halfplane discrepancy of P $[0 < \Delta(P) \leq 1]$
can't be perfect

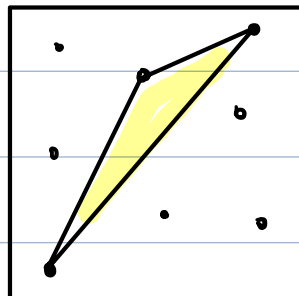
Questions:

- Given $P \subset U^2$, what is $\Delta(P)$?
- How low can $\Delta(P)$ be for any set of size n ?
- How to generate optimally uniform set P_{opt} of a given size n ?
($\Delta(P_{\text{opt}})$ is min. possible)
- Other measures of discrepancy?
 - Triangle discrepancy
 - Heilbronn's Triangle Problem:

Given any set of n pts P in U^2 ,
how large can the min area
triangle be?

conj: $O(1/n^2)$

Open for a century!



Computing $\Delta(P)$ for a set $P \subset U^2$.

- Key: Identify $O(n^2)$ candidates for halfplane that maximizes discrepancy.
- Compute discrepancy for each
 - Return the max

Lemma: Given pt set P , let h be halfplane of max discrepancy. Let l be h 's bounding line. Either:

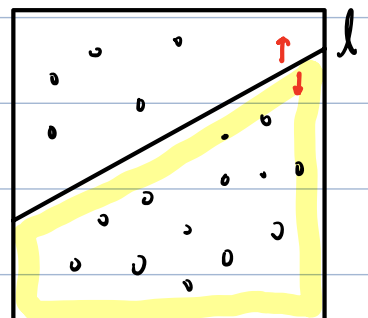


- (i) l passes through pt $p_i \in P$, and p_i is midpoint of $l \cap U^2$
- (ii) l passes through two pts of P .

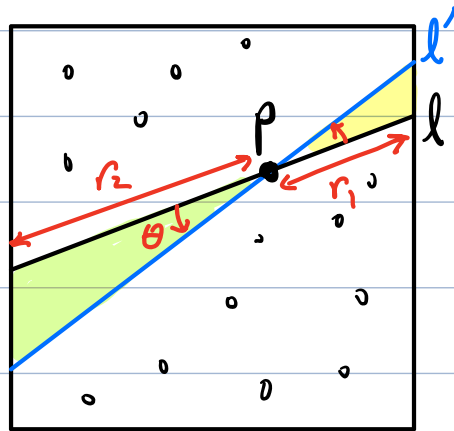
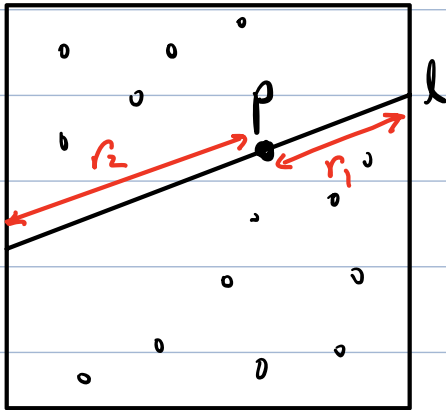
Proof:

Approach: Consider any line l . We'll show unless it satisfies (i) or (ii) we can perturb it to increase discrepancy.

Case 1: l passes through no pt of P - perturbing l up or down increases discrepancy.



Case 2: l passes through a pt $p \in P$, but p is not midpt of $l \cap U^2$



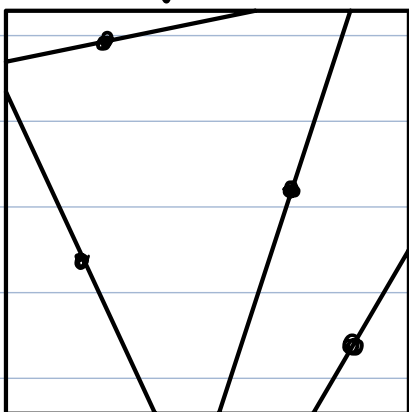
p splits $l \cap U^2$ into two segments of lengths $r_1 + r_2$. Since p is not midpt, may assume w.l.o.g. $r_2 > r_1$

If we rotate l by small angle θ about p we increase/decrease area by $\sim$

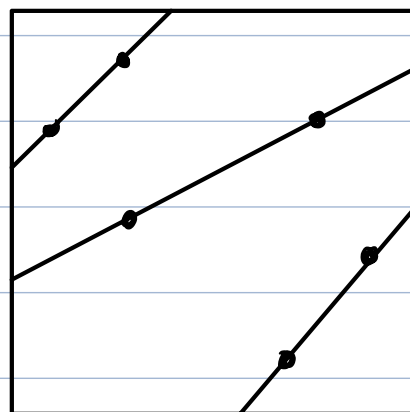
$$r_2^2 \cdot \theta - r_1^2 \cdot \theta = (r_2^2 - r_1^2) \theta > 0$$

Some small rotation will increase discrepancy.

Type (i)



Type (ii)



□

Computing $\Delta(P)$:

Type (i):

- for each $p_i \in P$, compute lines l s.t. p_i on midpt of $l \cap U^2$
- Count no. of pts on either side of l
 $\rightarrow n$ pts ; $O(1)$ lines each ; $O(n)$ time to count $\Rightarrow O(n^2)$ time

Type (ii):

- Dualize P to P^*
- Perform plane sweep of arrangement $A(P^*)$
- For each vertex of arrangement maintain no. of lines above + below on sweep line
- Compute discrepancy in $O(1)$ time for each vertex

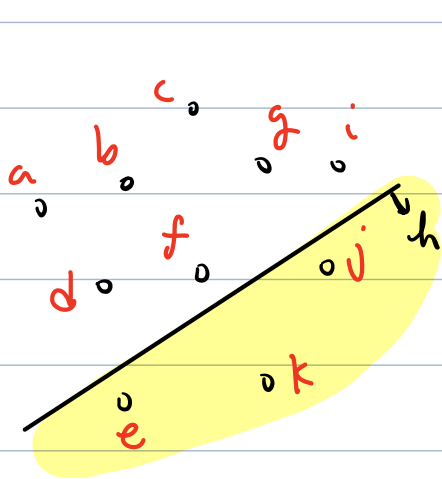
$\rightarrow O(n^2)$ vertices

Can maintain counts in $O(1)$ time
 $\Rightarrow O(n^2 \log n)$ time + $O(n)$ space

$O(n^2)$ by topol plane sweep

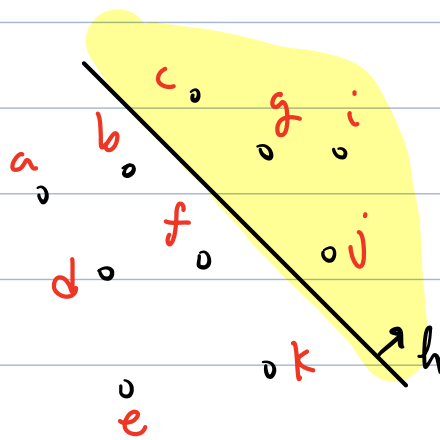
Computing k -sets:

Given a set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^2$ and integer k , $1 \leq k \leq n-1$, a k -set is a k -element subset of P of the form $P \cap h$, for some halfplane h .



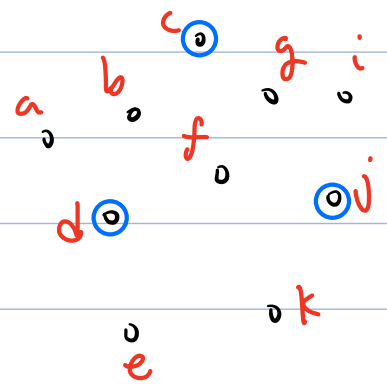
$\{e, k, j\}$

is a 3-set



$\{c, g, i, j\}$

is a 4-set



$\{c, d, j\}$

is **not** a 3-set

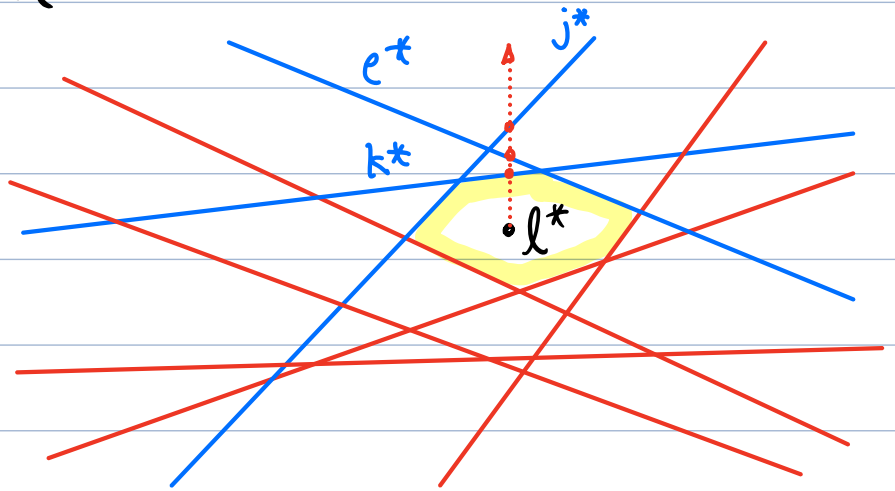
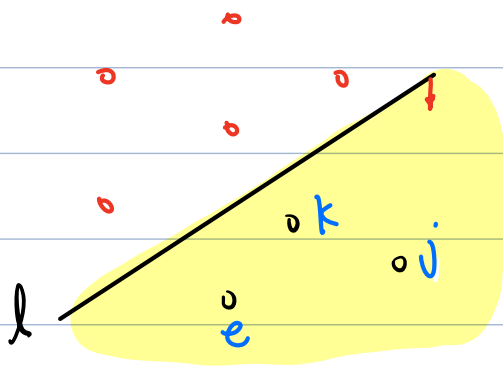
Problem: Given P and k , enumerate all k -sets of P .

How many? Naive $\leq \binom{n}{k} = O(n^k)$

Better $\leq \binom{n}{2}$ (see below)

Best theoretic bounds: $O(n \log k)$, $O(nk^{1/2})$

Dual equivalent?



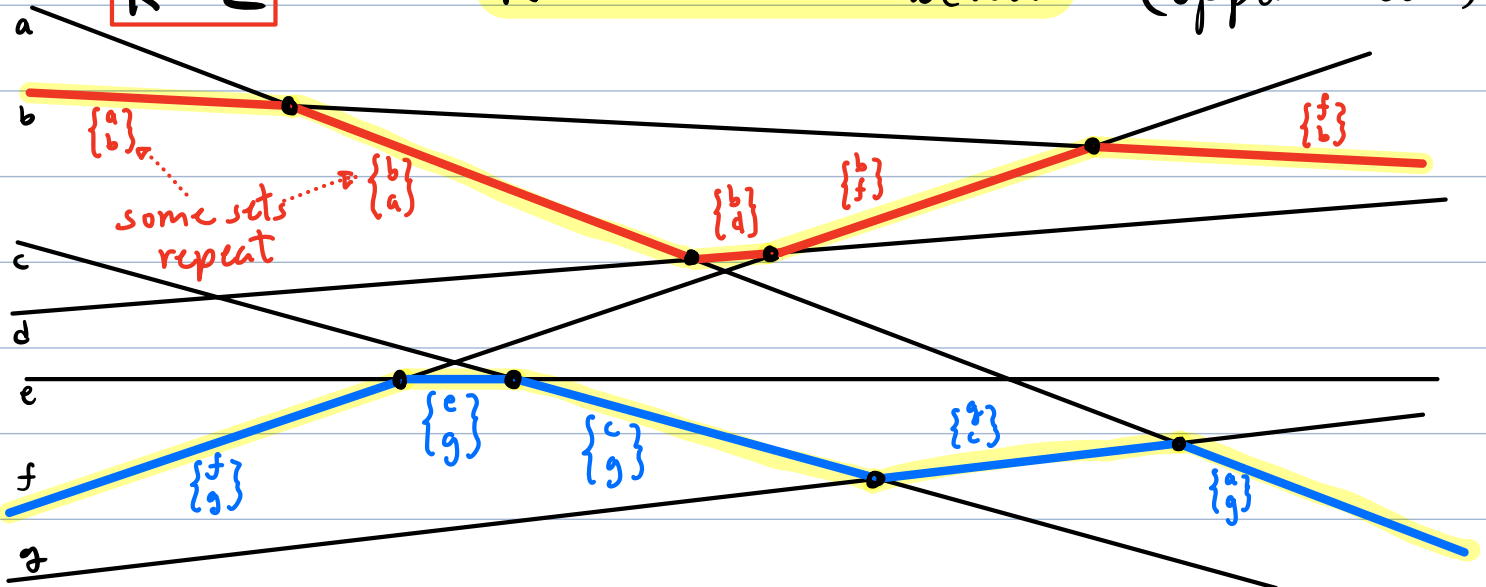
By order reversal:

k -pts of P lie below $l \iff k$ -lines of P^* pass above l^*

Approach:

- Traverse the arrangement $A(P^*)$
- Identify all edges with k lines on or above (lower k -set)
- k " " " below (upper k -set)

$n=7$
 $k=2$



Level: Given an arrangement of n lines $A(L)$, for $1 \leq k \leq n$, define level k , L_k , to be set of pts in $A(L)$ with

- $\leq k-1$ lines (strictly) above
- $\leq n-k$ lines (strictly) below

In above figure, we have shown L_2 and L_6

Obs: By applying plane sweep through $A(L)$, we can construct all levels in time $O(n^2)$

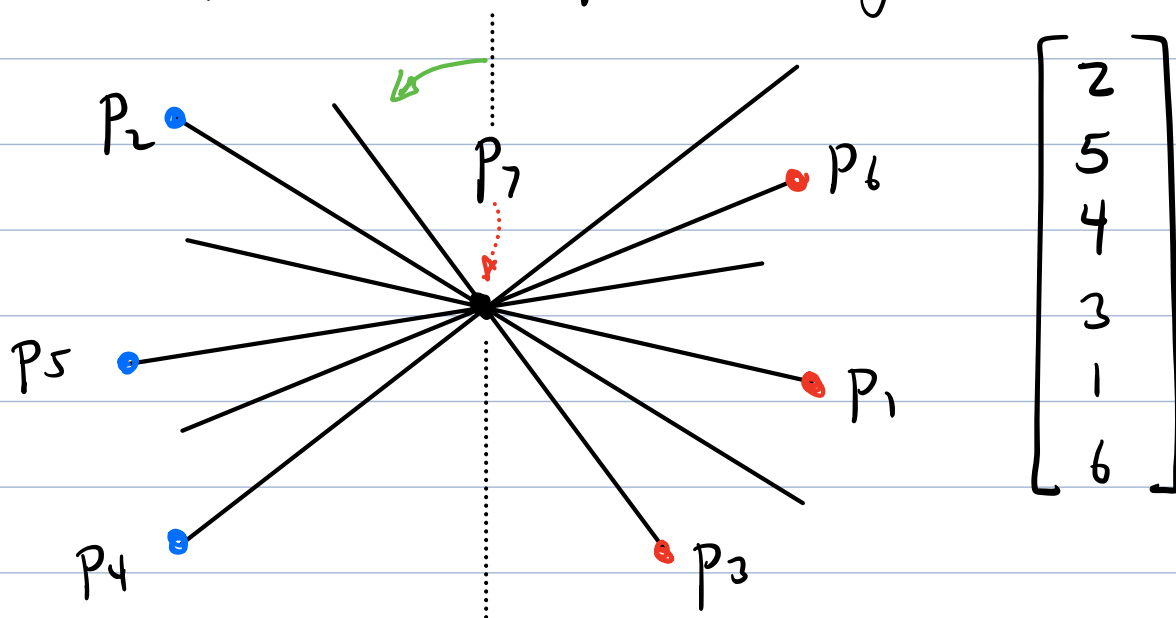
$\Rightarrow$ Can identify all k -sets of P in time $O(n^2)$ by sweeping $A(P^*)$ + extracting levels $L_k + L_{n-k+1}$

Note: To actually list the sets adds additional k factor, total $O(k \cdot n^2)$

Avoid duplicates? Exercise

Sorting angular sequences:

Given a set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^2$,
for each p_i , sort the remaining
 $n-1$ pts around p_i in angular order.



Naive: $O(n(n \log n)) = O(n^2 \log n)$
Sort angles for each point

Better: $O(n^2)$ using arrangements.

[See lect. notes for details]

Summary:

- Arrangements of lines in $\mathbb{R}^2$
- Complexity = $O(n^2)$
- Constructable in $O(n^2)$ time
(incremental, deterministic)
- Zone Theorem: Total zone complexity is $O(n)$. [$O(n^{d-1})$ in $\mathbb{R}^d$]
- Topological plane sweep
 $O(n^2)$ time ; $O(n)$ space
- Applications:
 - Narrowest k-corridor
 - Halfplane discrepancy
 - k-sets
 - Sorting angular sequences

CMSC 754 - Computational Geometry

Lecture 14 - Well-Separated Pair Decompositions

Geometric Approximations:

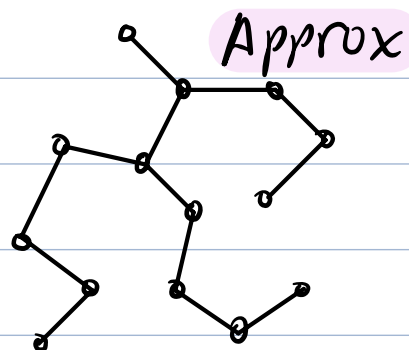
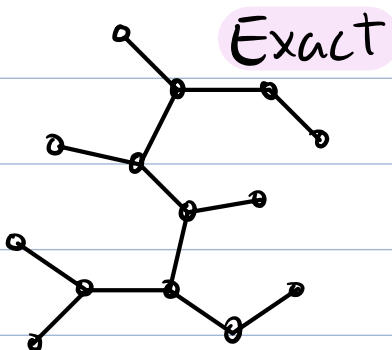
- Useful when exact computation is too costly
- Geometric inputs are "measurements" and often are uncertain.
So approximate solutions are fine.

Examples:

Euclidean MST of pt set $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$

Exact: $O(n \log n)$ in $\mathbb{R}^2$
 $O(n^{2-4/d})$ in $\mathbb{R}^d$ [Nearly quadratic]

Approx: Given $\epsilon > 0$, compute a spanning tree of weight $\leq (1+\epsilon) \cdot \text{EMST}(P)$



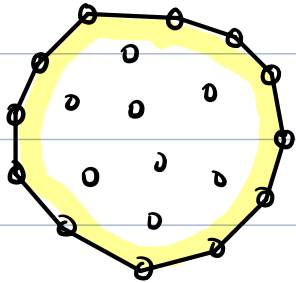
Convex Hull of a set $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$

Exact: $O(n \log n)$ in $\mathbb{R}^2$

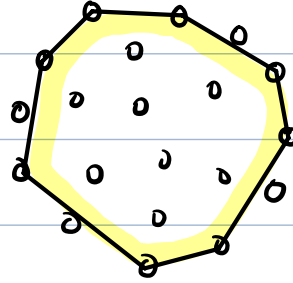
$O(n^{d/2+1})$ in $\mathbb{R}^d$

Approx: Compute a subset $P' \subseteq P$ s.t.
 $\text{conv}(P)$ and $\text{conv}(P')$ are
very similar

Exact

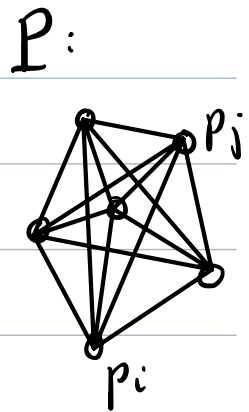


Approx



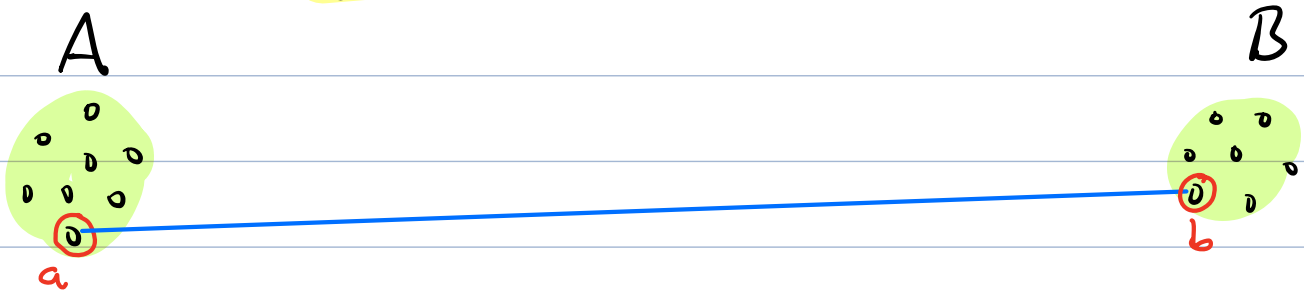
Well-Separated Pair Decomposition:

Given set $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$, the **Euclidean graph** is complete graph on P , where $w(p_i, p_j) = \|p_i - p_j\|$



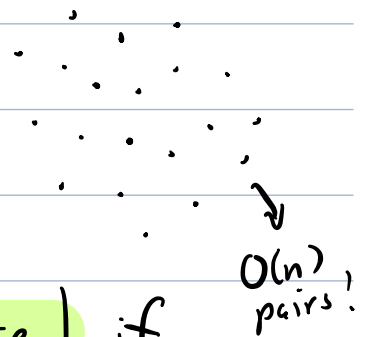
- Has $\binom{n}{2} = O(n^2)$ edges
- Can we **encode** this using a structure of size $O(n)$?

Intuition: If two point clusters $A, B \subseteq P$ well separated, we can represent many edges of $A \times B$ using a single edge connecting a representative $a \in A$ + $b \in B$

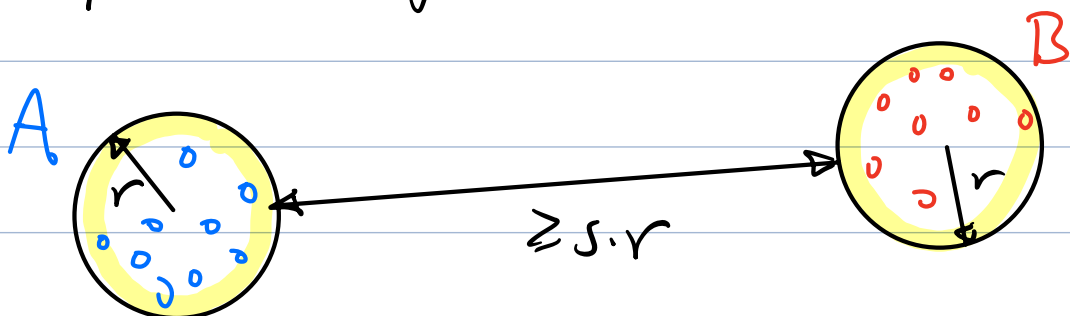


If we do this for all well-separated clusters, how many edges do we need?

Def: Given $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$ and scalar $s > 0$



- Two sets $A, B \subseteq P$ are s -well separated if $A \cup B$ can be enclosed in balls of some radius r , s.t. these balls are separated by distance $\geq s \cdot r$



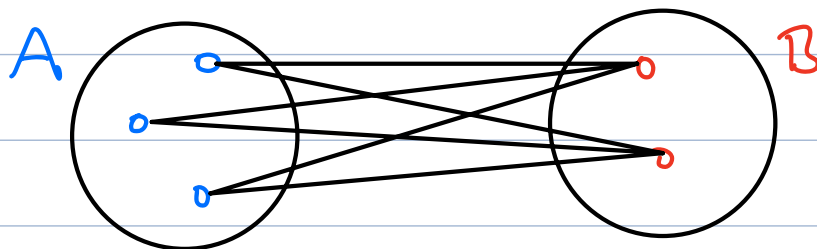
Obs:

- If A and B are s -well separated, they are s' -well separated for any $0 < s' \leq s$
- Two singleton sets $A = \{a\}$, $B = \{b\}$ are s -well separated for any $s > 0$. ($a \neq b$)



Def: Given sets A, B , define

$$A \otimes B = \{\{a, b\} \mid a \in A, b \in B, a \neq b\}$$



Obs: $P \otimes P$ = set of all $\binom{n}{2}$ pairs of P .

Def: Given $P + s > 0$, an s -well separated pair decomposition of P (s -WSPD) is collection of pairs

$$\{\{A_1, B_1\}, \dots, \{A_m, B_m\}\}$$

such that:

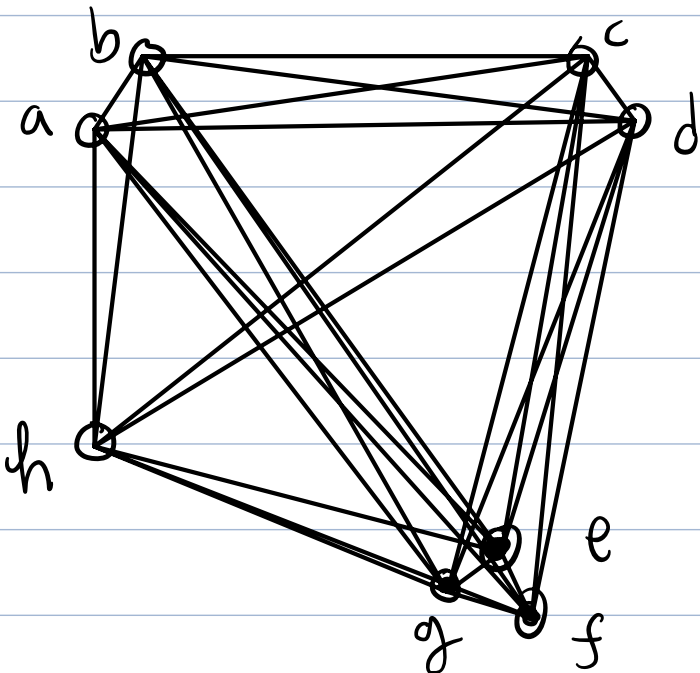
(1) $A_i, B_i \subseteq P$ for $1 \leq i \leq m$

(2) $A_i \cap B_i \neq \emptyset$ " " (disjoint)

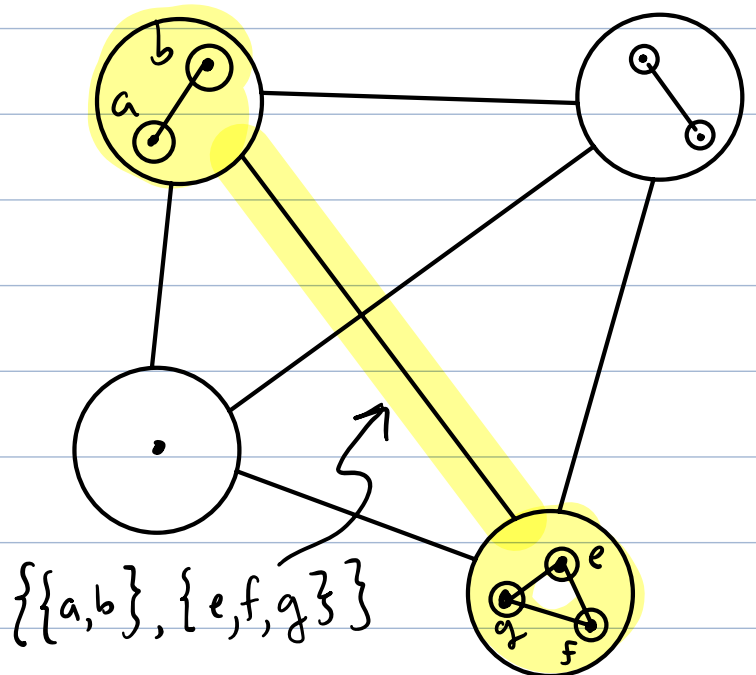
(3) $\bigcup_{i=1}^m A_i \otimes B_i = P \otimes P$ (cover)

(4) $A_i + B_i$ are s -well separated for $1 \leq i \leq m$

28 pairs



11 well-sep pairs



Obs: For any $s > 0$ there is always a trivial s -WSPD consisting of $\binom{n}{2}$ singleton pairs.

Can we do better? d is constant; hidden d^d

Yes! $\rightarrow O(s^d \cdot n)$ pairs

If s, d constants: $O(n)$!

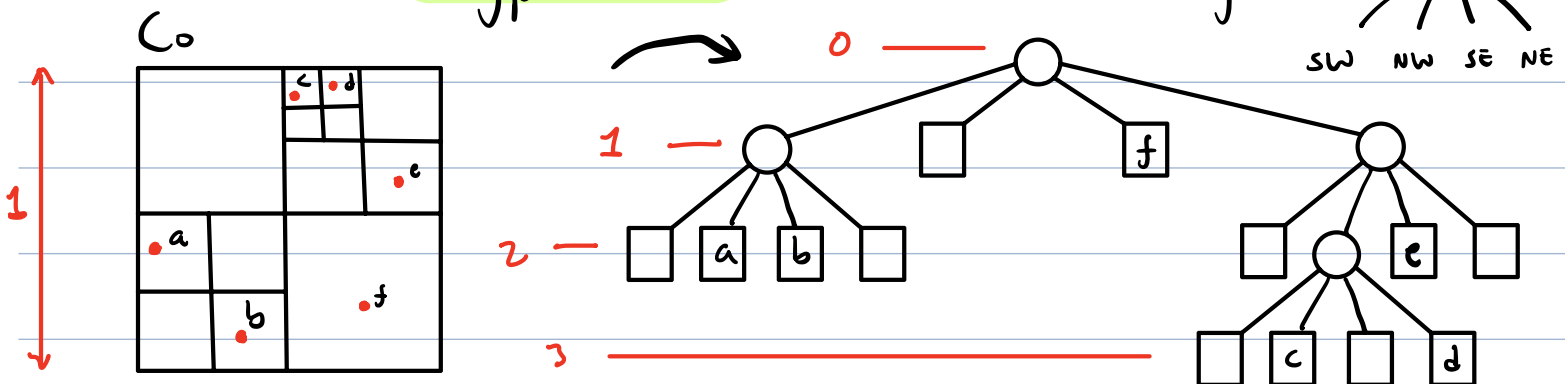
Can compute in time:

$O(n \log n + s^d n)$

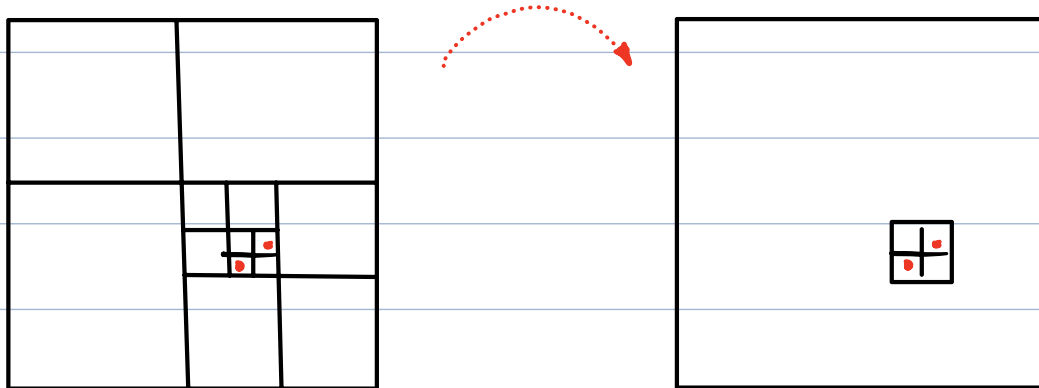
Quadtrees:

A tree storing P based on recursive subdiv. into hypercubes.

- Let C_0 be a bounding hypercube for P
- While a cell of subdivision has 2 or more pts of P , split it into 2^d hypercubes of half side length



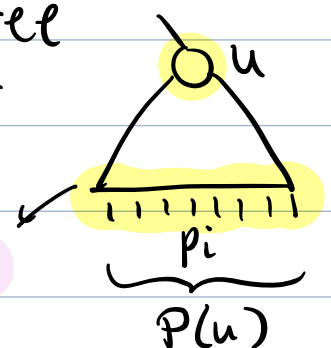
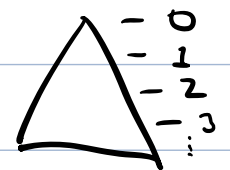
Note: A quadtree may have more than $O(n)$ nodes, but we can reduce storage to $O(n)$ by path compression. (see latex notes)



Thm: Given a set of pts $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$ can construct a (compressed) quadtree of space $O(n)$ in $O(n \log n)$ time.

Additional information (provided by construction)
Given node u in tree:

- $\text{level}(u)$ = level of u in tree
- $P(u)$ = set of pts in u 's subtree
- $\text{rep}(u)$ = an arbitrary element of $P(u)$



We will represent each WSP as pair of nodes $\{u, v\}$. Actual pair is $\{P(u), P(v)\}$

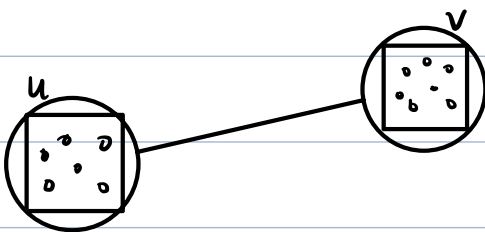
Constructing the WSPD:

Given P + $s > 0$:

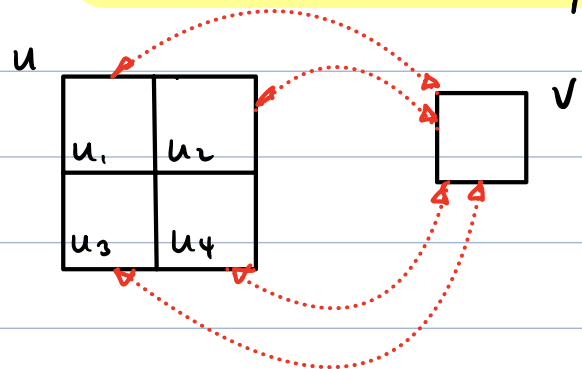
- Build quadtree for $P \rightarrow$ Let $u_0 = \text{root}$
- Invoke: $\text{ws-pairs}(u_0, u_0, s)$

```
ws-pairs (Node  $u$ , Node  $v$ , Scalar  $s$ ) {  
  if ( $u$  +  $v$  are both leaves +  $u = v$ ) return  $\emptyset$   
  ← if ( $\text{rep}(u)$  or  $\text{rep}(v)$  is empty) return  $\emptyset$   
  else if ( $u$  +  $v$  are  $s$ -well sep)  
    [ return  $\{u, v\}$  //  $\text{WSP} = \{P(u), P(v)\}$   
  else // not w.s.  
    [ if ( $\text{level}(u) > \text{level}(v)$ )  
      [ swap  $u \leftrightarrow v$  //  $u$  is not deeper than  $v$   
      let  $u_1, \dots, u_k$  be  $u$ 's children  
      return  $\bigcup_{i=1}^k \text{ws-pairs}(u_i, v, s)$   
}
```

Cases: u + v are well sep



u + v not well-sep



Analysis: We'll show $O(s^d \cdot n)$ pairs generated

- Assume: Quadtree is not compressed (simpler)
 $s \geq 1$ (else just use $s' = \max(1, s)$)

① Terminal / Non-Terminal:

- To count no. of WSP's, we'll count no. of calls to ws-pairs
- A call is:
 - terminal: makes no recursive calls
 - non-terminal: otherwise
- It suffices to count just no. of non-terminal calls (each generates at most $2^d = O(1)$ term. calls)

② Charging: We'll count no. of non-term calls by charging each to node of tree.

Preview: - Each node receives $O(s^d)$ charges

- $O(n)$ nodes in tree
 $\Rightarrow O(s^d \cdot n)$ total charges

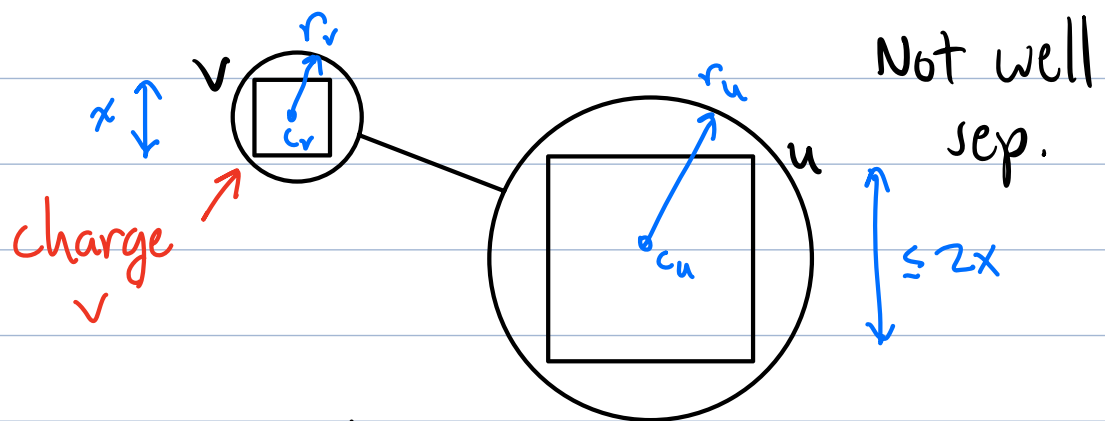
③ Who gets charged?

Let $ws\text{-pairs}(u, v, s)$ be non-term call

$\Rightarrow u, v$ not well sep.

$\rightarrow$ Assume (w.l.o.g.) $lev(u) \leq lev(v)$

$\rightarrow$ We will charge v
(smaller node is charged)



- Let x be side length of v 's cell
- We always split larger cell first
 $\Rightarrow u$'s side length $\leq 2x$

- Let r_v = radius of ball enclosing v 's cell
 r_u = " " " " " u 's cell

$$\Rightarrow r_u \leq 2r_v$$

and

$$r_v = x\sqrt{2}/2$$

- Let c_u, c_v be centers of u & v 's cells

This call is non-term

$\Rightarrow u, v$ not well separated

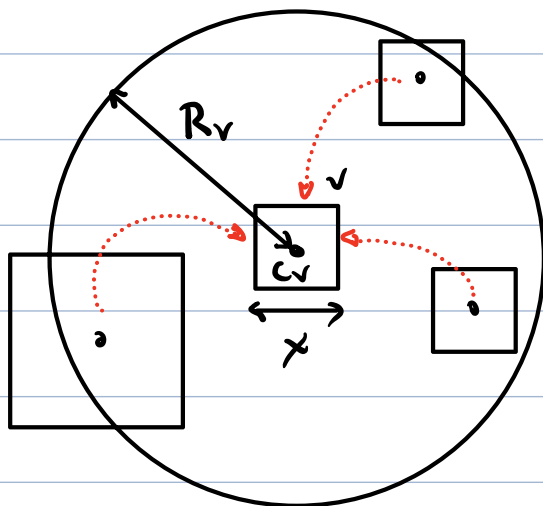
$\Rightarrow$ Distance between balls is
 $< s \cdot \max(r_u, r_v) \leq s \cdot r_u \leq s(2 \cdot r_v)$
 $= s \cdot x \cdot \sqrt{d}$

$\Rightarrow$ Distance between centers

$$\begin{aligned} \|c_u - c_v\| &\leq r_v + r_u + s x \sqrt{d} \\ &\leq x \sqrt{d} / 2 + x \sqrt{d} + s x \sqrt{d} \\ &= \left(\frac{1}{2} + 1 + s\right) x \sqrt{d} \\ &< 3s x \sqrt{d} \quad (\text{since } s \geq 1) \end{aligned}$$

Def: $R_v = 3s x \sqrt{d}$

Summary: A node v of side length x is charged by nodes u of side length x or $2x$ whose cell centers lie within a ball of radius $R_v = 3s x \sqrt{d}$ of c_v .



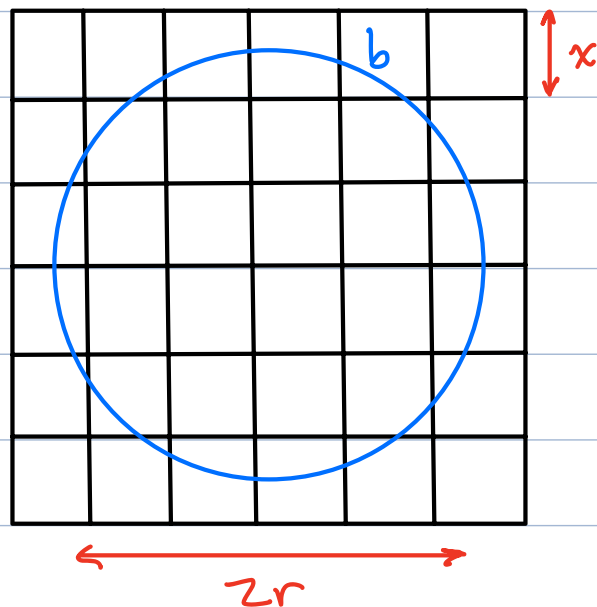
How many such nodes can there be?

Packing lemma: Given a ball b of radius r in $\mathbb{R}^d$ + any collection X of disjoint quadtree cells of side length $\geq x$ that overlap b , then

$$|X| \leq \left(1 + \left\lceil \frac{2r}{x} \right\rceil\right)^d \leq O\left(\max\left(2, \frac{r}{x}\right)^d\right)$$

Proof: To maximize no. of cells, assume they are as small as possible $\Rightarrow x$

These cells form a grid of side length x that overlaps b



No. of grid squares of side length x overlapping an interval of length $2r$ is

$$\leq 1 + \left\lceil \frac{2r}{x} \right\rceil$$

$$\Rightarrow \text{Total: } \left(1 + \left\lceil \frac{2r}{x} \right\rceil\right)^d$$

□

Returning to WSPD analysis:

- No. of charges to $v \leq$

No. of nodes of side length $\geq x$
overlapping a ball of radius
 $R_v = 3sx\sqrt{d}$

- By Packing Lemma, no. of nodes

$$\begin{aligned} &\leq \left(1 + \left\lceil \frac{2R_v}{x} \right\rceil\right)^d \\ &\leq \left(1 + \left\lceil \frac{6sx\sqrt{d}}{x} \right\rceil\right)^d \\ &\leq (2 + 6s\sqrt{d})^d \end{aligned}$$

$$\leq O(s^d) \quad \text{since } s \geq 1 \text{ and } d \text{ is constant}$$

So, each node charged $O(s^d)$ times

→ $O(n)$ nodes in quadtree

→ $O(n \cdot s^d)$ non-term calls to ws-pairs

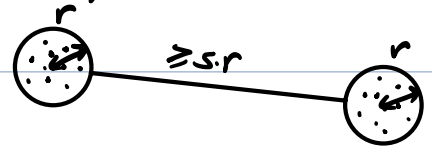
→ $O(n \cdot s^d)$ pairs generated

when !!

Theorem: Given a point set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^d$ (d is constant) and $s \geq 1$, in $O(n \log n + s^d n)$ time, can build an s -WSPD for P of size $O(s^d \cdot n)$

Summary:

- Geometric approximations
- Well-separated pairs
 - A concise way to encode pairs of point sets.
- WSPD - Encodes all $\binom{n}{2}$ pairs using $O(n)$ well-separated pairs
- Can construct an s -WSPD for n pts in $\mathbb{R}^d$ of size $O(s^d \cdot n)$ in time $O(n \log n + s^d \cdot n)$

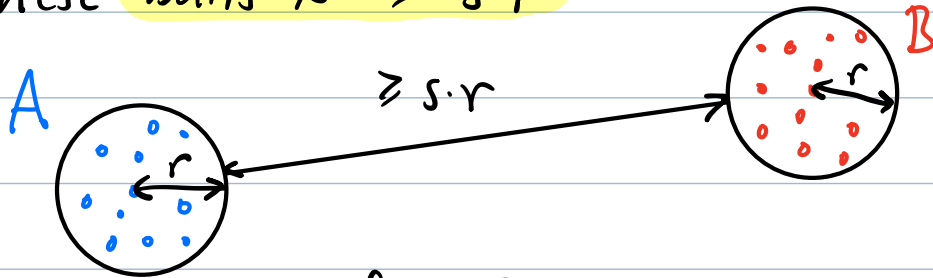


CMSC 754 - Computational Geometry

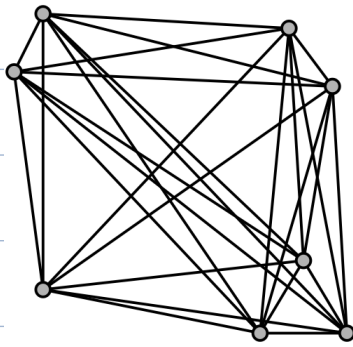
Lecture 15: Applications of WSPDs

Review of WSPDs:

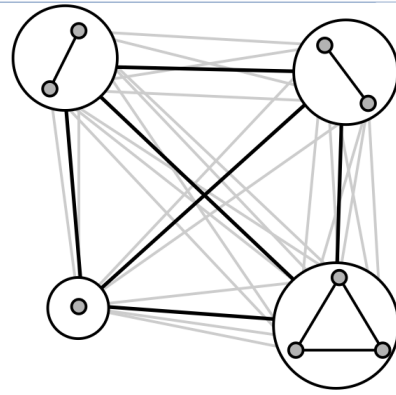
- Given a point set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^d$ (d a fixed constant) and separation factor $s > 0$, two sets $A + B$ are s -well separated if they can be contained in two balls of some radius r s.t. the distance between these balls is $\geq s \cdot r$



- An s -WSPD for P is a collection: $\{\{A_1, B_1\}, \dots, \{A_m, B_m\}\}$ such that
 - $A_i, B_i \subseteq P$
 - $A_i \cap B_i = \emptyset$ (disjoint)
 - $\bigcup_i A_i \otimes B_i = P \otimes P$ (cover all pairs)
 - $A_i + B_i$ are s -well separated
- Given $P + s \geq 1$, in time $O(n \log n + s^d \cdot n)$ we can construct an s -WSPD for P of size $O(s^d \cdot n)$.



28 pairs



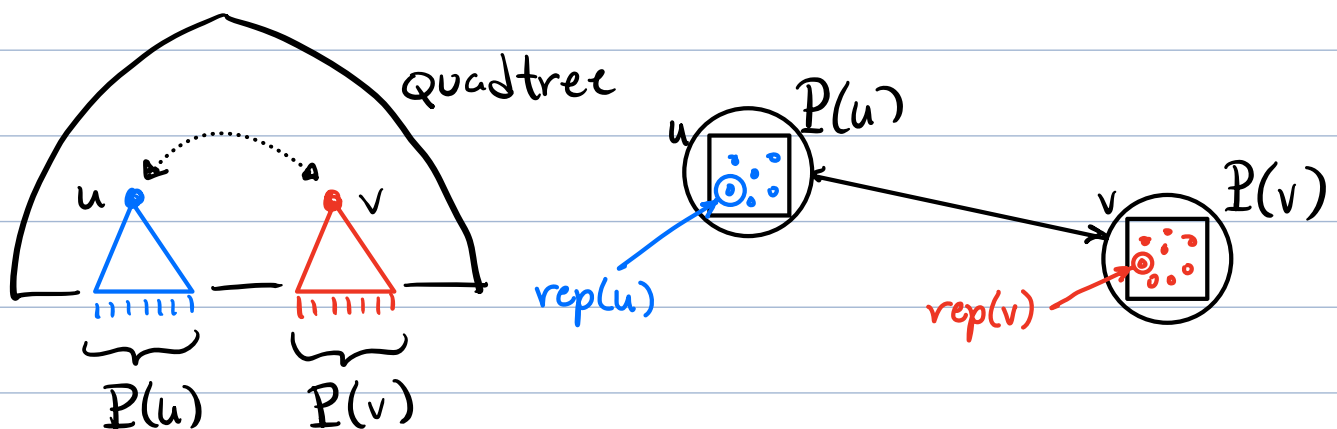
11 well-separated pairs

- Construction is based on d -dim quad tree

- Given nodes u, v in tree let

$P(u)$ - points in u 's subtree

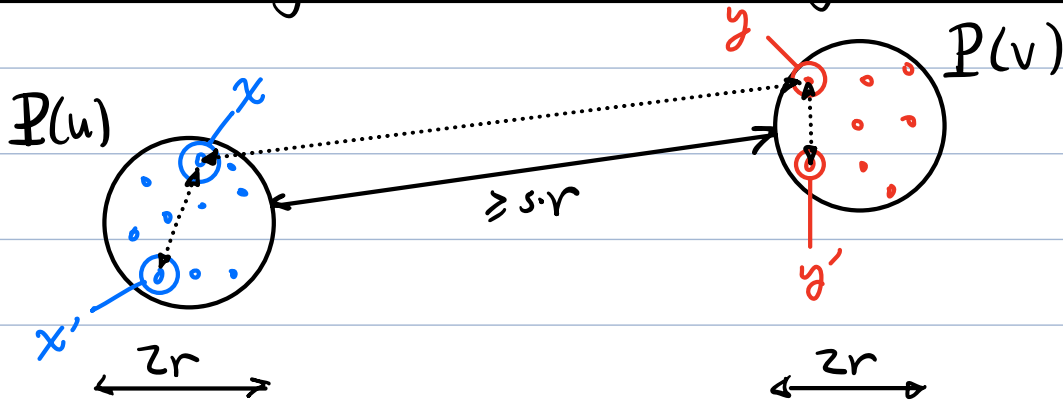
$rep(u)$ - an arbitrary pt of $P(u)$
(u 's representative)



The WSP $\{P(u), P(v)\}$
is represented by the
pair $\{u, v\}$

Utility Lemma: Given an s -WSP $\{P(u), P(v)\}$
and $x, x' \in P(u)$ + $y, y' \in P(v)$:

- (i) $\|x - x'\| \leq \frac{2}{s} \cdot \|x - y\|$
(ii) $\|x' - y'\| \leq (1 + \frac{4}{s}) \cdot \|x - y\|$

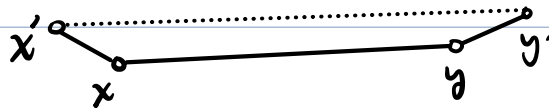


Intuition: (i) Same side closer than cross side
(ii) Cross side dists similar

Proof: (i) $\|x - x'\| \leq 2 \cdot r$
 $= 2 \cdot r \frac{s \cdot r}{s \cdot r} \leq \frac{2 \cdot r}{s \cdot r} \|x - y\|$
 $= (\frac{2}{s}) \|x - y\| \quad \checkmark$

(ii) Observe: $\|x - y\| \geq s \cdot r \Rightarrow 4 \cdot r \leq \frac{4}{s} \|x - y\|$

By the triangle inequality:



$$\begin{aligned} \|x' - y'\| &\leq \|x' - x\| + \|x - y\| + \|y - y'\| \\ &\leq 2r + \|x - y\| + 2r \\ &\leq \|x - y\| + 4r \\ &\leq \|x - y\| + \frac{4}{s} \|x - y\| \\ &= (1 + \frac{4}{s}) \|x - y\| \quad \checkmark \end{aligned}$$

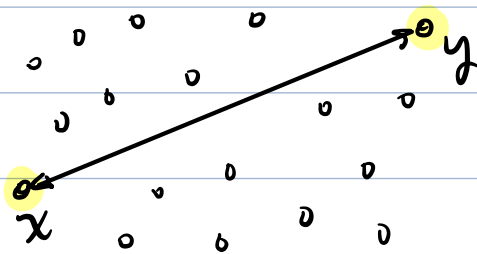
Applications:

- $(1+\epsilon)$ approx to diameter (farthest pair)
- exact closest pair
- Computing a t -spanner (for any $t > 1$)
- $(1+\epsilon)$ approx to Euclidean MST

$(1+\epsilon)$ Approx Diameter: in time $O(n \log n + n/\epsilon^d)$

Given $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^d$

$$\text{diam}(P) = \max_{x, y \in P} \|x - y\|$$



Exact:

In $\mathbb{R}^2$: Can compute in $O(n \log n)$

[Convex hull + rotating calipers]

$\mathbb{R}^d$: (Nearly) quadratic in n

$(1+\epsilon)$ -Approx:

- Set $s = 4/\epsilon$

- Compute an s -WSPD for P

- for each WSP $\{u, v\}$:

$$\text{dist}_{u,v} = \|\text{rep}(u) - \text{rep}(v)\|$$

- return $\max \text{dist}_{u,v}$ as approx diam

$\rightarrow O(n \log n + n/\epsilon^d)$

$\rightarrow O(n/\epsilon^d)$

Correctness:

Plan:

① Since $\text{reprs} \leq \mathbb{P}$, $\text{approx diam} \leq \text{diam}(\mathbb{P})$

② We will show

*: $\exists \text{ WSP } u, v \text{ s.t.}$

$$\text{dist}_{u,v} \geq \text{diam}(\mathbb{P}) / (1 + \epsilon)$$

$$\Rightarrow \max \text{dist}_{u,v} \geq \text{diam}(\mathbb{P}) / (1 + \epsilon)$$

$$\textcircled{1} + \textcircled{2} \Rightarrow \frac{\text{diam}(\mathbb{P})}{1 + \epsilon} \leq \text{approx diam} \leq \text{diam}(\mathbb{P}) \quad \checkmark$$

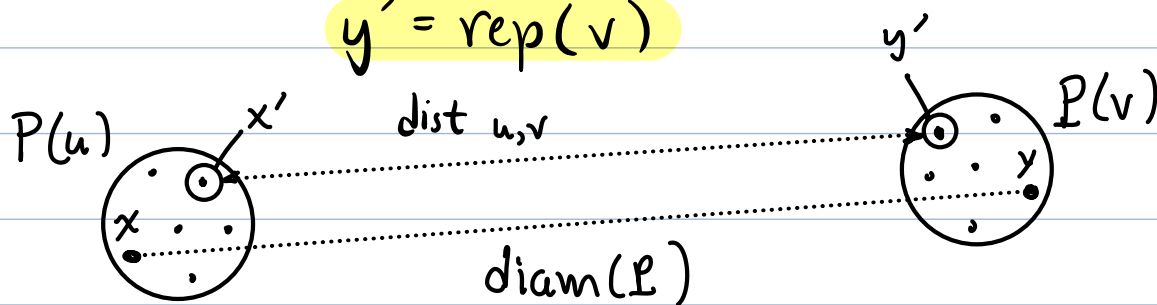
Need to show *

- Let x, y be diameter pair

- $\exists \text{ WSP } \{u, v\} \text{ s.t. } x \in P(u) \ y \in P(v)$

- Let $x' = \text{rep}(u)$

$y' = \text{rep}(v)$



By WSPD utility lemma:

$$\text{diam}(\mathbb{P}) = \|x - y\| \leq \left(1 + \frac{4}{s}\right) \|x' - y'\|$$

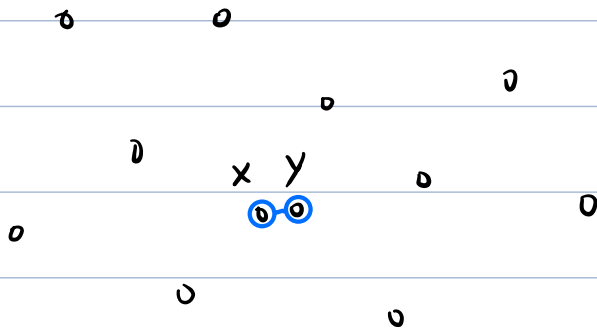
$$= (1 + \epsilon) \|x' - y'\| \quad (s = 4/\epsilon)$$

$$= (1 + \epsilon) \text{dist}_{u,v} \Rightarrow \text{dist}_{u,v} \geq \text{diam}(\mathbb{P}) / (1 + \epsilon)$$

(Exact) Closest Pair: in time $O(n \log n)$

Given $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^d$ find $x, y \in P$

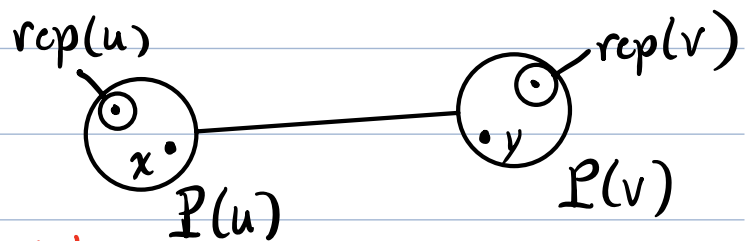
$$\min_{x, y \in P} \|x - y\|$$



Intuition: Some WSP $\{u, v\}$ must cover the pair $\{x, y\}$

Huh? It looks like

$x + y$ not closest!



It must be that $\text{rep}(u) = x$
+ $\text{rep}(v) = y$

Exact Closest Pair:

- Let $s > 2$ (eg. $s = 2.0001$)
- Build s -WSPD for P
- for each WSP $\{u, v\}$

$$\text{dist}_{u,v} = \|\text{rep}(u) - \text{rep}(v)\|$$

- return $\min_{u,v} \text{dist}_{u,v}$ as closest dist

$$\left. \begin{array}{l} O(n \log n + 2^d \cdot n) \\ = O(n \log n) \end{array} \right\}$$

Correctness:

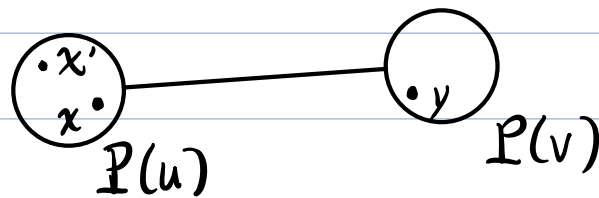
Follows directly from the following lemma:

Lemma: If $s > 0$ + x, y are closest pair in P , then any s -WSPD of P contains the pair $\{\{x\}, \{y\}\}$

That is, x, y are singletons in WSPD

Proof:

- Suppose not.
- Let $\{u, v\}$ be WSP with $x \in P(u), y \in P(v)$
- May assume w.l.o.g. that $P(u)$ has another pt x'



- By WSPD Utility Lemma:

$$\begin{aligned} \|x - x'\| &\leq \frac{2}{s} \cdot \|x - y\| \\ &< \|x - y\| \quad (\text{since } s > 2) \end{aligned}$$

$\Rightarrow x, y$ not closest pair

$\rightarrow$ contradiction

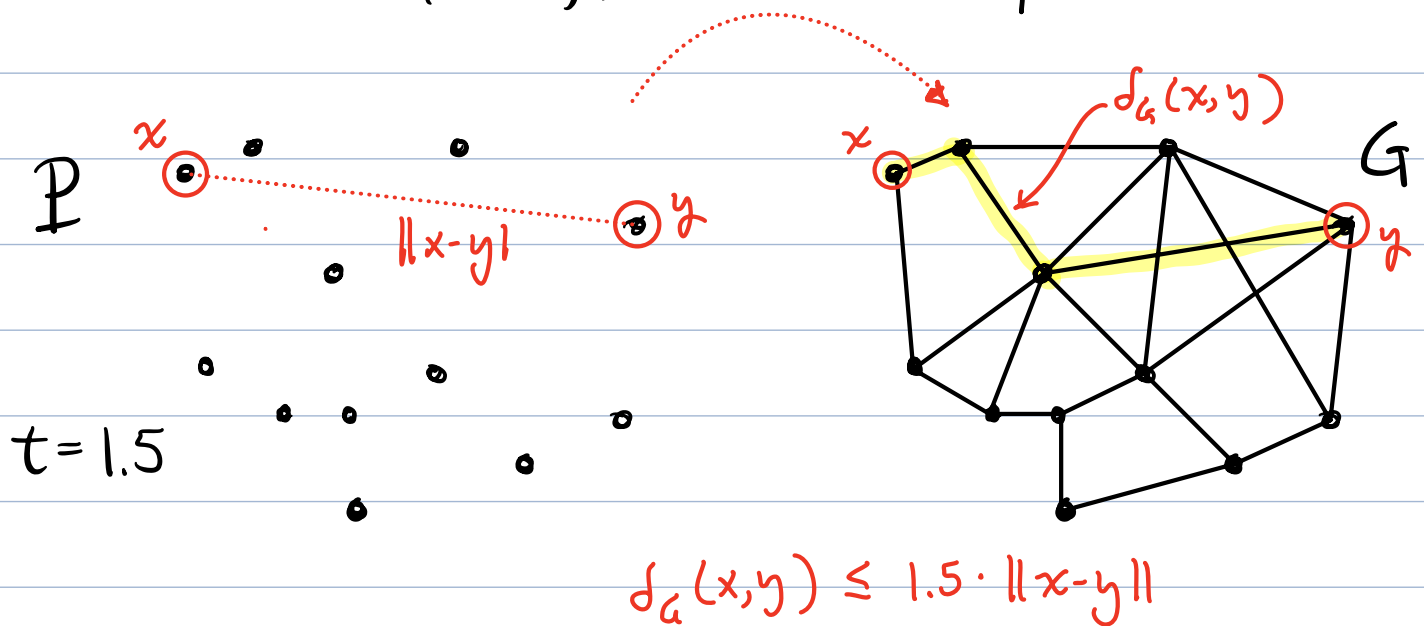
Spanners:

Recall def. of t -spanner (from lect. on Delaunay Tri.)

Given point set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^d$ and $t \geq 1$, a t -spanner is a graph on P s.t. $\forall x, y \in P$:

$$\|x - y\| \leq d_G(x, y) \leq t \cdot \|x - y\|$$

where $d_G(x, y)$ is shortest path dist in G



We will show that given $P \subseteq \mathbb{R}^d$ + $t > 1$ can build a $(1+\epsilon)$ -spanner for P in time $O(n \log n + n/\epsilon^d)$ consisting of $O(n/\epsilon^d)$ edges

Spanner construction (Given P & $t > 1$)

- Let $s = \frac{4(t+1)}{t-1}$
- $G \leftarrow$ graph with vertex set P & no edges
- Build an s -WSPD for P
- for each WSP $\{u, v\}$:
 - add edge $(\text{rep}(u), \text{rep}(v))$ to G
- return G

Time: If $t = 1 + \varepsilon$, $s = O(1/\varepsilon)$ [$0 < \varepsilon < 1$]
 $\Rightarrow O(n \log n + n/\varepsilon^2)$

Size: $O(n/\varepsilon^2)$ WSPs $\Rightarrow O(n/\varepsilon^2)$ edges

Correctness:

Will show that for all $x, y \in P$

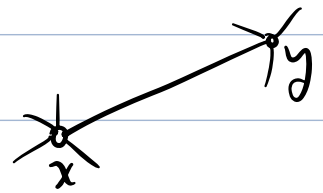
$$\|x - y\| \stackrel{\textcircled{1}}{\leq} d_G(x, y) \stackrel{\textcircled{2}}{\leq} t \cdot \|x - y\|$$

① Trivially true since G is a subgraph of complete Euclidean graph

② Rest of the proof...

Induction on num. of edges in path from x to y in G

Basis: Edge (x, y) is in G



$$\Rightarrow \delta_G(x, y) = \|x - y\| \leq t \cdot \|x - y\| \quad \checkmark$$

(since $t > 1$)

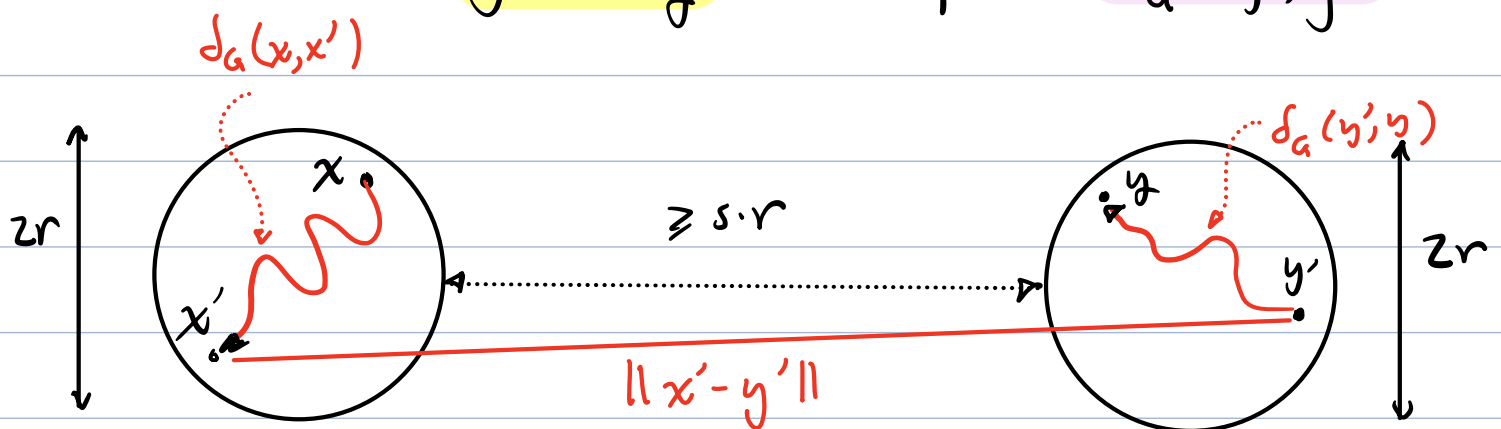
Induction step:

- $\exists$ pair $\{u, v\}$ in WSPD that covers the pair (x, y)

- Let $x' = \text{rep}(u)$ $y' = \text{rep}(v)$
(possibly $x' = x$ or $y' = y$)

- To get from x to y in G we can:

- x to x' $\rightarrow$ path $\delta_G(x, x')$
- x' to y' $\rightarrow$ direct edge: $\|x' - y'\|$
- y' to y $\rightarrow$ path $\delta_G(y', y)$



By the induction hyp: $\delta_G(x, x') \leq t \cdot \|x - x'\|$

$\delta_G(y', y) \leq t \cdot \|y' - y\|$

$$\Rightarrow \delta_a(x, y) \leq t \cdot \|x - x'\| + \|x' - y'\| + t \cdot \|y' - y\|$$

$$= t(\|x - x'\| + \|y' - y\|) + \|x' - y'\|$$

By WSPD Utility Lemma:

- $\|x - x'\| \leq \frac{2}{s} \|x - y\|$
- $\|y' - y\| \leq \frac{2}{s} \|x - y\|$
- $\|x' - y'\| \leq (1 + \frac{4}{s}) \|x - y\|$

$$\Rightarrow \delta_a(x, y) \leq t \left(\frac{2}{s} \|x - y\| + \frac{2}{s} \|x - y\| \right) + \left(1 + \frac{4}{s} \right) \|x - y\|$$

$$= \left(t \frac{4}{s} + 1 + \frac{4}{s} \right) \|x - y\|$$

$$= \left(1 + \frac{4(t+1)}{s} \right) \|x - y\|$$

$$= t \|x - y\| \quad \left(\text{since: } s = \frac{4(t+1)}{t-1} \right)$$

□

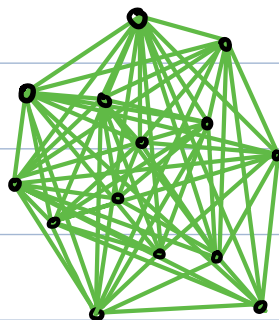
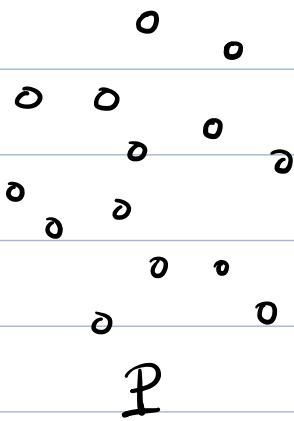
To obtain a $(1+\epsilon)$ -spanner, set $t = 1+\epsilon$ + apply this construction

Approx. to Euclidean MST

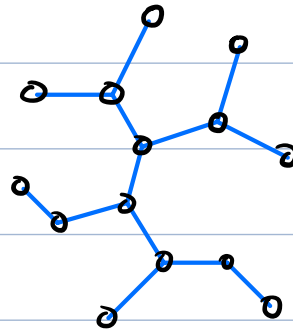
Given a point set $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$
define:

$EMST(P)$ = Min. spanning tree of
complete Euclidean graph
on P (where $w(u, v) = \|u - v\|$)

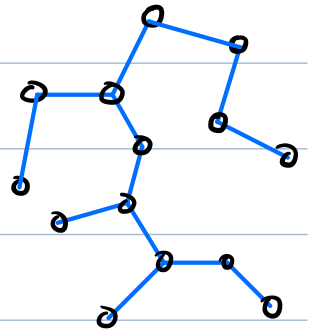
Let: $emst(P) = \sum_{(x, y) \in EMST(P)} \|x - y\|$
= total weight of $EMST(P)$



Euclidean Graph



$EMST(P)$



approx EMST

A graph H is an $(1 + \epsilon)$ -approx EMST if:

(1) H is a spanning tree for P

(2) $w(H) \leq (1 + \epsilon) \cdot emst(P)$

where $w(H)$ = total weight of
 H 's edges

We'll show how to compute an $(1 + \epsilon)$ -approx EMST
in time $O(n \log n + n/\epsilon^2)$

approx-EMST(P, ϵ)

- $G \leftarrow (1+\epsilon)$ -spanner for P
- return $\text{MST}(G)$

Time: Compute G : $O(n \log n + n/\epsilon^d)$

Compute $\text{MST}(G)$:

- Can compute MST of a graph with v vertices + e edges in time

$$O(v \log v + e)$$

- G has n vertices + n/ϵ^d edges

- $\text{MST}(G)$ takes $O(n \log n + n/\epsilon^d)$

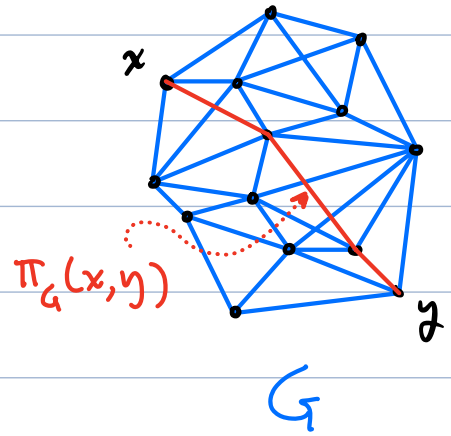
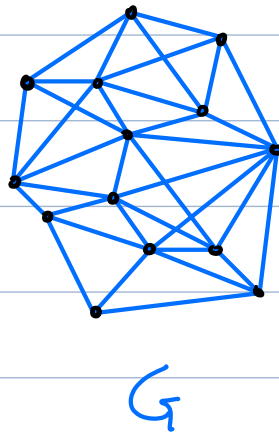
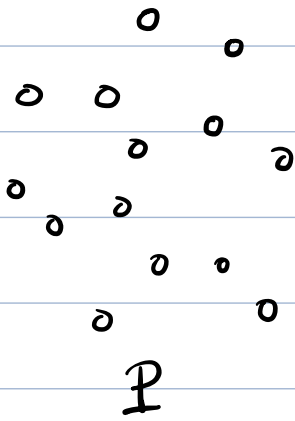
Correctness:

- We'll show that G has a connected subgraph that contains all pts of P (spans P) and has weight $\leq (1+\epsilon) \cdot \text{emst}(P)$
- If G has a spanning subgraph H of weight W , then the weight of its MST is no larger

For each $x, y \in P$, let $\pi_G(x, y)$ be shortest path from x to y in G .

Let $\ell_G(x, y)$ be length of this path

Know that $\delta_G(x, y) \leq (1 + \epsilon) \|x - y\|$



H :

for each $(x, y) \in \text{EMST}(P)$
add the edges of $\pi_G(x, y)$ to H

Obs:

① H is connected and spans all pts of P

② Total weight:

$$w(H) \leq \sum_{(x, y) \in \text{EMST}(P)} \delta_G(x, y)$$

$$\leq \sum_{(x, y) \in \text{EMST}(P)} (1 + \epsilon) \cdot \|x - y\|$$

$$= (1 + \epsilon) \sum_{(x, y) \in \text{EMST}(P)} \|x - y\|$$

$$= (1 + \epsilon) \cdot \text{emst}(P)$$

$$\textcircled{1} + \textcircled{2} \Rightarrow \omega(\text{MST}(G))$$

$$\leq \omega(H) \leq (1+\varepsilon) \cdot \text{cost}(P)$$



Summary:

- WSPD Utility Lemma
 - A useful tool in WSPD applications
- $(1+\varepsilon)$ approx to diameter (farthest pair)
 - $O(n \log n + n/\varepsilon^d)$ time
- exact closest pair
 - $O(n \log n)$ time
- Computing a $(1+\varepsilon)$ -spanner (for any $\varepsilon > 0$)
 - $O(n \log n + n/\varepsilon^d)$ time $O(n/\varepsilon^d)$ space
- $(1+\varepsilon)$ approx to Euclidean MST
 - $O(n \log n + n/\varepsilon^d)$ time

CMSC 754 - Computational Geometry

Lecture 16: Coresets and Kernels

Approximation by Sampling:

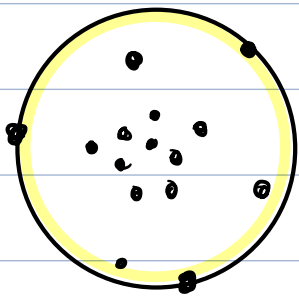
- Running time too slow?
- Maybe your data size is too large!
- Idea:
 - Extract a small subset, $P' \subseteq P$
 - Run solve problem exactly on P'
 - Prove that the answer on P' is "close to optimal" on P .

How to compute P' ?

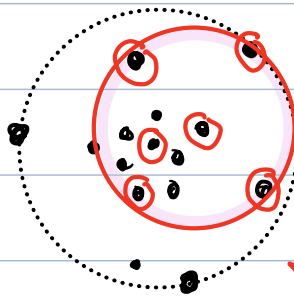
- Depends on your problem
- Random sampling is most common, but not necessarily best

Example: Minimum Enclosing Ball (MEB)

- Given a set $P = \{p_1, \dots, p_n\}$ in $\mathbb{R}^d$ compute the Euclidean ball of min. radius enclosing P .

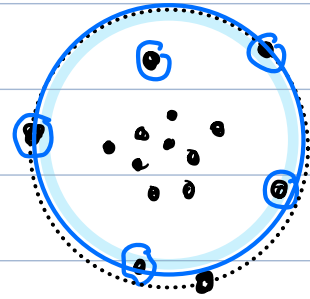


$MEB(P)$



$MEB(P')$

$P' = \text{random}$



$MEB(P'')$

$P'' = \text{coreset}$

Problem with random sampling:

- $MEB(P)$ depends on points near periphery
- Random sample extracts many irrelevant points.
- Smarter: Use a sampling method that gives priority to peripheral points

Coreset: Let P be input set.

$f^*(P) \rightarrow \mathbb{R}$ is our objective function
(eg. $f^*(P) = \text{radius of MEB}$)

Given $\epsilon > 0$, an ϵ -coreset is a subset $Q \subseteq P$ s.t.

$$1 - \epsilon \leq \frac{f^*(Q)}{f^*(P)} \leq 1 + \epsilon$$

The opt. soln. for Q is close to opt. for P

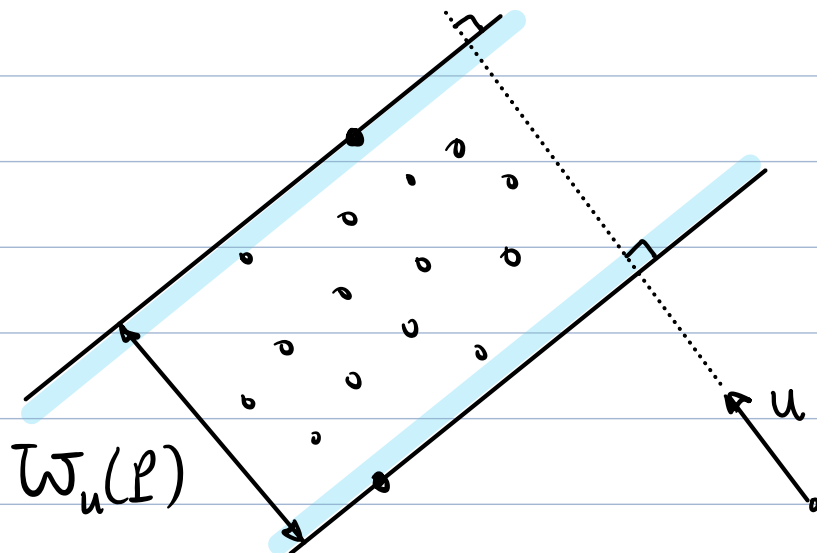
Questions:

- For what optimization problems do (small) coresets exist?
- (As a function of $n + \epsilon$) how small is the coreset?
- How fast can we compute a coreset?

Coreset for Directional Width: (also called ϵ -kernel)

- Given a pt set $P \subseteq \mathbb{R}^d$
- Given a unit vector $\vec{u}$
- Directional width of P in direction $\vec{u}$ is:

$$W_u(P) = \max_{p \in P} (\vec{p} \cdot \vec{u}) - \min_{p \in P} (\vec{p} \cdot \vec{u})$$



Given $\varepsilon > 0$, an ε -coreset for direc. width (also called ε -kernel) is a subset $R \subseteq P$ s.t.

$\forall$ unit vect. $\vec{u}$:

Trivially true
since $R \subseteq P$

$$(1-\varepsilon) \bar{W}_u(P) \leq \bar{W}_u(R) \leq \bar{W}_u(P)$$

Getting this is
the objective

Aside: When computing approx. lower bounds we sometimes write:

$$(1-\varepsilon) \cdot \text{exact} \leq \text{approx}$$

and other times:

$$\frac{\text{exact}}{1+\varepsilon} \leq \text{approx}$$

Does the form matter?

Not really. If $0 < \varepsilon < 1$, then

$$1-\varepsilon \leq \frac{1}{1+\varepsilon} \leq 1-\frac{\varepsilon}{2}$$

- Only constant factors are affected

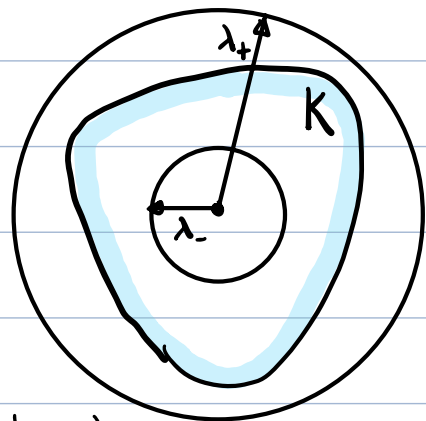
Useful Facts:

Chain Property: If X is an ε -kernel for Y
and Y is an ε' -kernel for Z
then X is an $(\varepsilon + \varepsilon')$ -kernel for Z

Union Property: If X is an ε -kernel for P
 X' is an ε -kernel for P'
then $X \cup X'$ is an ε -kernel for $P \cup P'$

Canonical Position: We like fat things...

Fat: Given $0 \leq \alpha \leq 1$, a convex body K is α -fat if K can be sandwiched between two concentric balls of radii $\lambda_- \leq \lambda_+$ where $\alpha = \lambda_- / \lambda_+$



Canonical Position: Convex body K is in α -canonical form if it is sandwiched between balls of radius $\lambda_- = \frac{1}{2}\alpha + \lambda_+ = \frac{1}{2}$ centered at the origin.

Why $1/2$? $\Rightarrow K$'s diameter ≤ 1

A point set P is $\left\{ \begin{array}{l} \alpha\text{-fat} \\ \alpha\text{-canonical form} \end{array} \right\}$ if $\text{conv}(P)$ is.

We can convert any pt set into canonical form.

Affine Transformation: Is a
linear transformation
(scaling + rotation + shearing)
+ translation

Lemma: Given any n -element pt. set $P \subseteq \mathbb{R}^d$, there exists an affine transformation T that maps P into $(1/d)$ -canonical form

- $R \subseteq P$ is an ε -kernel for P
iff $T(R)$ is an ε -kernel for $T(P)$
- T can be computed in $O(n)$ time

Proof makes use of important fact: (1948)

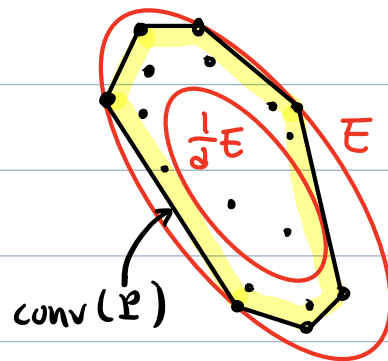
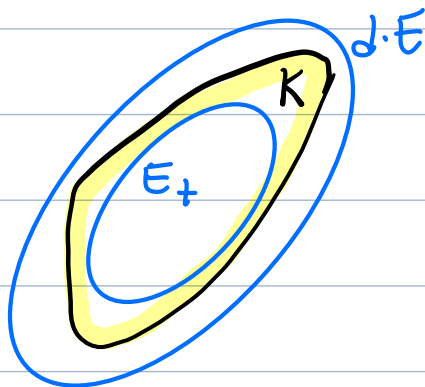
John's Theorem: Given any convex body $K \subseteq \mathbb{R}^d$, let E be max volume ellipsoid contained in K , then

$$E \subseteq K \subseteq d \cdot E$$

where $d \cdot E$ is a factor- d scaling E about its center.

Equiv: Given pt. set P , let E be min vol. ellipsoid containing P , then

$$\frac{1}{d} E \subseteq \text{conv}(P) \subseteq E$$

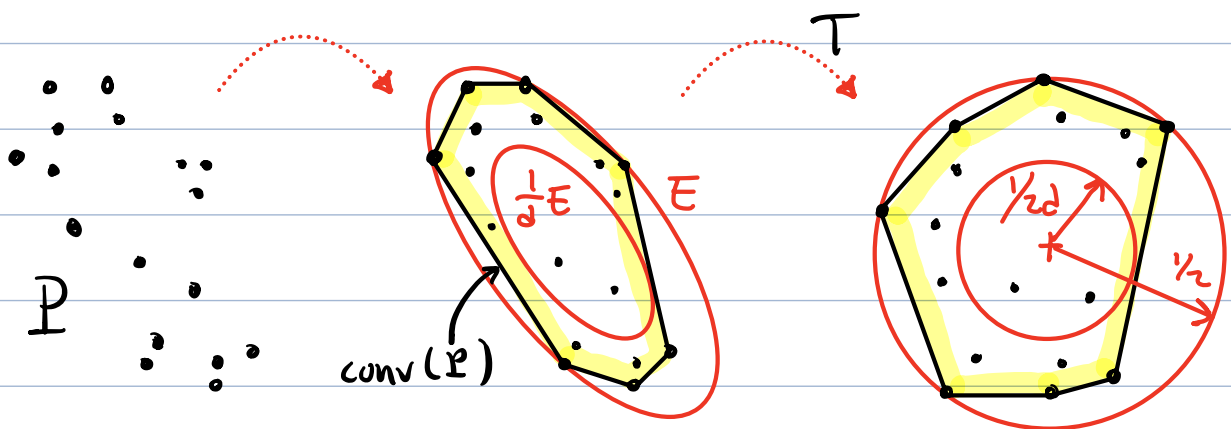


- The ellipsoid is called the **John Ellipsoid** or **Löwner-John Ellipsoid**
- Can compute it in $O(n)$ time (randomized incremental)

Fact: Given any ellipsoid E , there exists an affine transformation that maps E to a unit ball, centered at origin.

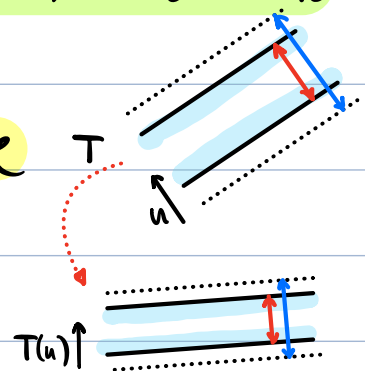
Proof (of canonical form lemma):

- ① Compute P 's outer John ellipsoid E
- ② Find affine transformation mapping E to unit ball centered at origin
- ③ Scale by $1/2 \rightarrow$ output resulting transformation T



Why are directional width approximations preserved?

- Affine transformations preserve ratios of parallel lengths (Details omitted)



Quick + Dirty Kernel: Simple but not optimal size
 $- \mathcal{O}(1/\epsilon^d)$

Given $P \subseteq \mathbb{R}^d$ + $\epsilon > 0$:

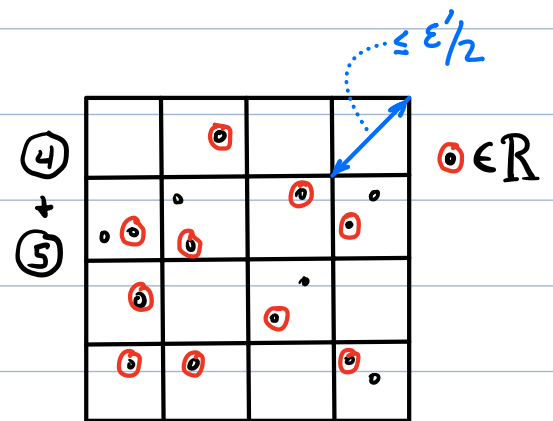
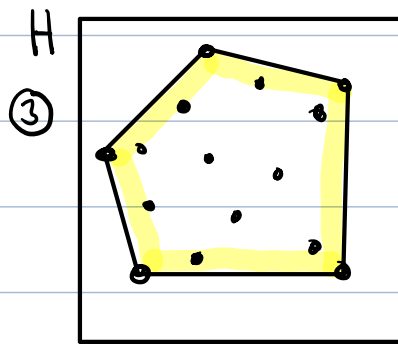
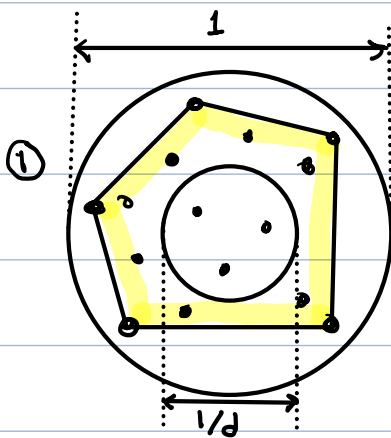
① Map P to $1/2$ -canonical position

Note: $\forall u, 1/2 \leq W_u(P) \leq 1$

$\Rightarrow$ absolute error of $\epsilon/d \Rightarrow$ rel. error $\leq \epsilon$

② Let $\epsilon' = \epsilon/d$

③ Let $H = [-1/2, +1/2]^d$ be unit hypercube containing P



④ Subdivide H into square grid of diameter $\leq \epsilon'/2$ (equiv., side length $= \epsilon'/2\sqrt{2}$)

Note: No. of grid cells is $\left(\frac{1}{\epsilon'/2\sqrt{2}}\right)^d = \mathcal{O}(1/\epsilon^d)$

⑤ $R \leftarrow$ take one pt of P from each occupied cell

Note: $|R| = \mathcal{O}(1/\epsilon^d)$. Computable in $\mathcal{O}(n)$ time

Running time: $O(n + 1/\epsilon^d)$

- Canonical position - $O(n)$

- Place pts in grid cells - $O(n)$

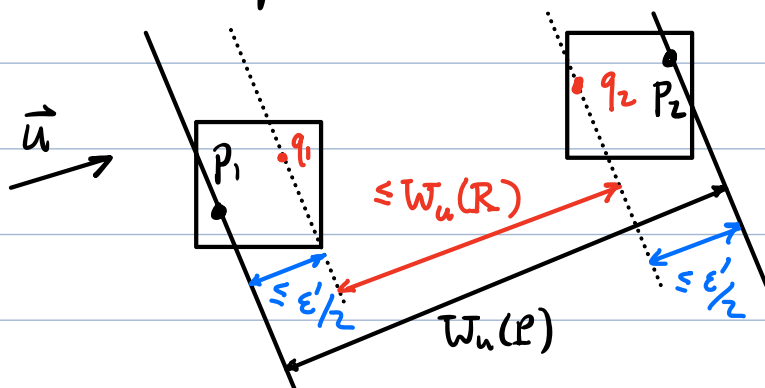
[integer division + hashing]

- Output R - $O(1/\epsilon^d)$

Correctness:

- Given any direction $\vec{u}$, let $p_1, p_2 \in P$ be pts that define $W_u(P)$

- Let $q_1, q_2 \in R$ be corresponding representatives from p_1 + p_2 's cells



- Since cell diameter $\leq \epsilon'/2$, it follows that

$$W_u(P) \leq \epsilon'/2 + W_u(R) + \epsilon'/2$$

$$= \epsilon' + W_u(R) = \epsilon/d + W_u(R)$$

- By canonical form, $W_u(P) \geq 1/d$

$$W_u(P) \leq \epsilon \cdot W_u(P) + W_u(R)$$

$$\Rightarrow (1 - \epsilon) W_u(P) \leq W_u(R) \leq W_u(P)$$

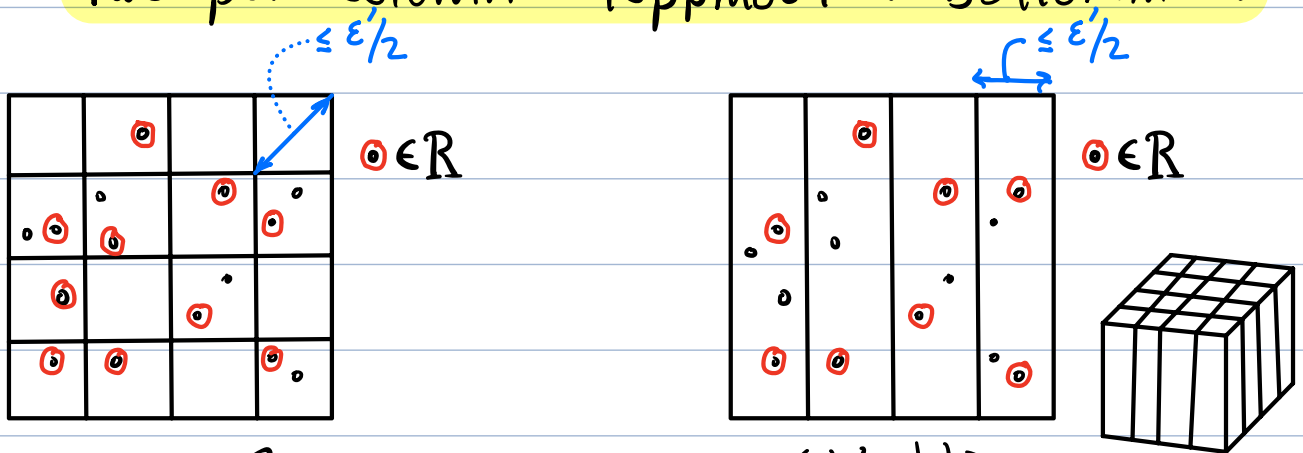
$\Rightarrow R$ is an ϵ -kernel

since $R \subseteq P$



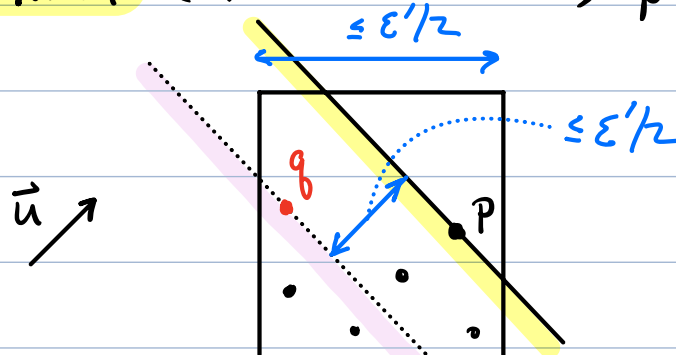
Small Improvement: ~~$O(1/\epsilon^d)$~~ $\rightarrow O(1/\epsilon^{d-1})$

- Quick + dirty's grid includes many internal points $\rightarrow$ wasteful
- Rather than take:
 - one representative per cell, instead
 - two per column - toppmost + bottommost



- How many? Top grid has $O(1/\epsilon^{d-1})$ cells
 $|R| = 2 \cdot O(1/\epsilon^{d-1}) = O(1/\epsilon^{d-1})$

- Correctness? Let p be extreme pt in direction $\vec{u}$ + let $q \in R$ be topmost (or bottommost) pt in column



Directional distance betw. $q + p$ is $\leq \epsilon/2$
 ... remaining details omitted

Big Improvement - ϵ -kernel of size $O(1/\epsilon^{\frac{d-1}{2}})$ [Optimal in the worst case]

Construction based on idea discovered
(independently) by Dudley + Bronshteyn +
Ivanov (~1974)

① Map P to $1/d$ -canonical position

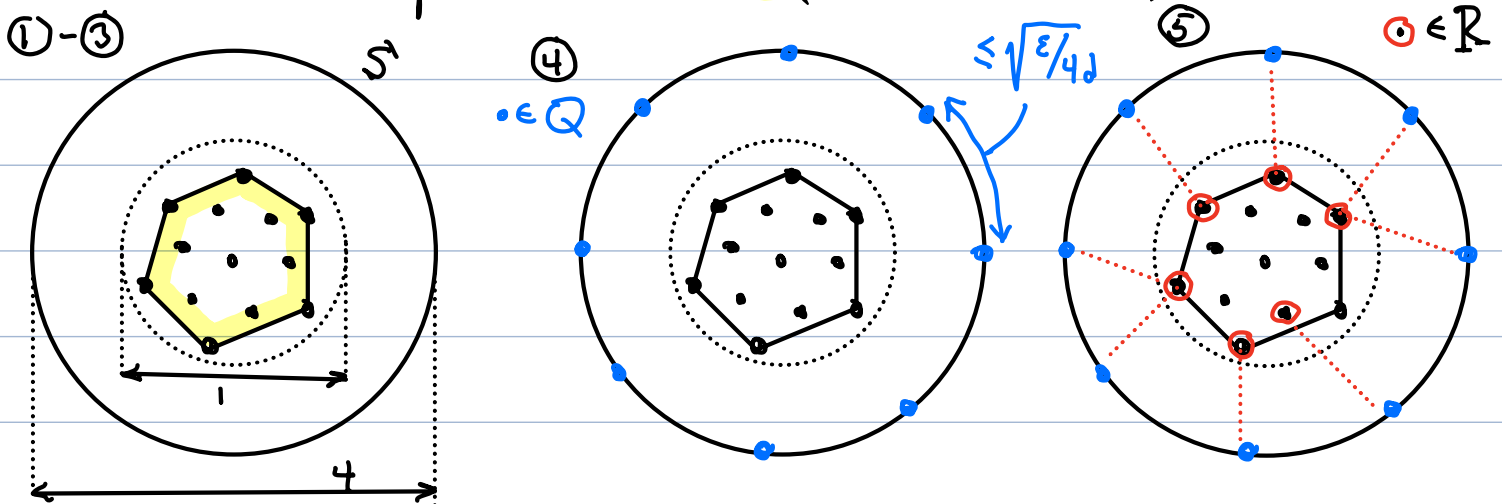
Note: $\forall u, 1/d \leq W_u(P) \leq 1$

$\Rightarrow$ absolute error of $\epsilon/d \Rightarrow$ rel. error $\leq \epsilon$

② Let $\epsilon' = \epsilon/d$

③ Let $S =$ sphere of radius 1 centered
at origin, let $\delta = \sqrt{\epsilon'/4d}$

④ Let Q be a set of points on S s.t.
any point of S is within distance δ of
some pt of Q . (Q is " δ -dense")



⑤ For each $q \in Q$, let $nn(q) \in P$ be its closest pt.
Return: $R = \bigcup_{q \in Q} nn(q)$

Size: $|R| \leq |Q|$

- Claim that $|Q| = O((1/\sqrt{\epsilon})^{d-1}) = O(1/\epsilon^{\frac{d-1}{2}})$

- Intuition: Each $q \in Q$ covers a spherical cap of radius $\delta \approx \sqrt{\epsilon}$

- Such a cap has surface area
 $\approx \delta^{d-1} \approx \sqrt{\epsilon}^{d-1} \approx \epsilon^{(d-1)/2}$

- S has constant radius $\Rightarrow$ constant area

- No. caps needed to cover S
 $\approx \text{const} / \epsilon^{(d-1)/2} = O(1/\epsilon^{(d-1)/2})$

$\Rightarrow |R| = O(1/\epsilon^{(d-1)/2})$

Running Time:

- Canonical position: $O(n)$

- Computing δ -dense Q
 $O(|Q|) = O(1/\epsilon^{(d-1)/2})$

How? Enclose S in a hypercube

Cover hypercube with grid $\sim \delta$

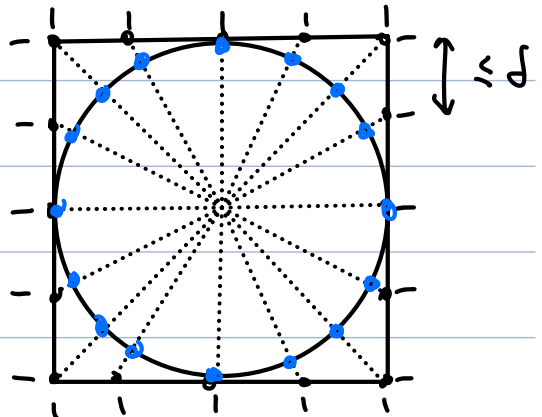
Project onto S

- Compute $nn(q)$

- Suffices to use

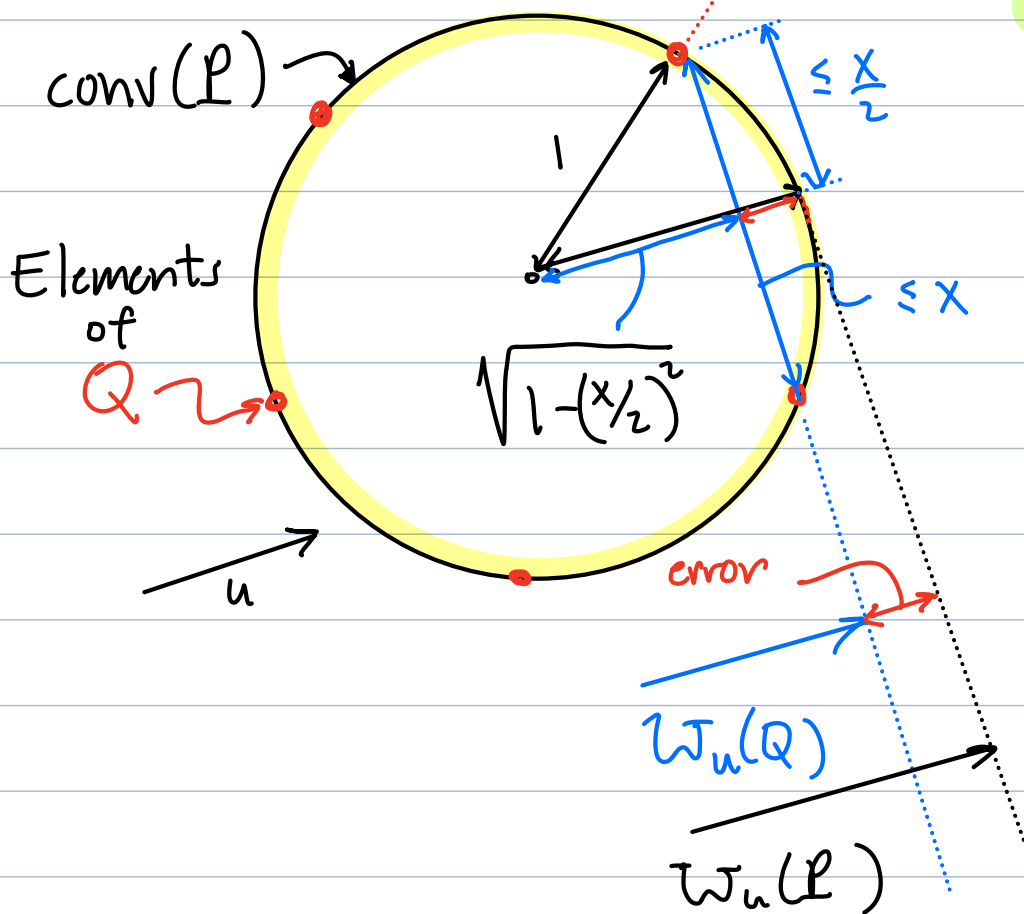
approx nn

- $O(\text{poly}(1/\epsilon) \cdot \log n)$



Correctness: (Complex - See latex notes)
 We'll consider a simpler question:

Why $\sqrt{\epsilon}$ spacing? Simple case: Suppose $\text{conv}(P)$ is a unit ball



To generate a large error select u to hit $\text{conv}(P)$ midway between pts of Q .

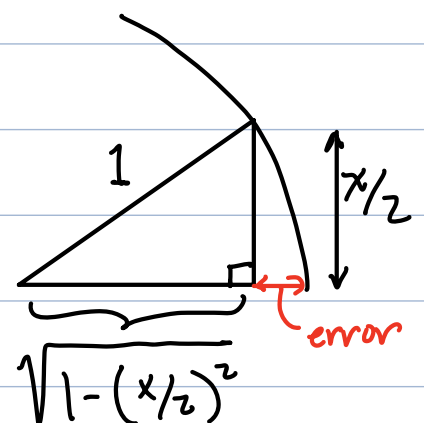
By Pythagorean Thm:

$$\text{error} \leq 1 - \sqrt{1 - (x/2)^2}$$

We want

$$\text{error} \leq \epsilon$$

That is, want x s.t.



$$1 - \sqrt{1 - (x/2)^2} \leq \varepsilon$$

Solving for x , we have:

$$\Leftrightarrow 1 - (x/2)^2 \geq (1 - \varepsilon)^2 = 1 - 2\varepsilon + \varepsilon^2$$

$$\text{if } \varepsilon \leq 1, \text{ then } \varepsilon^2 \leq \varepsilon \Rightarrow 1 - 2\varepsilon + \varepsilon^2 \leq 1 - \varepsilon$$

$$\Leftarrow 1 - (x/2)^2 \geq 1 - \varepsilon$$

$$\Leftrightarrow x/2 \leq \sqrt{\varepsilon}$$

$$x \leq 2\sqrt{\varepsilon}$$

- This explains why spacing $\sim \sqrt{\varepsilon}$ is the right thing to do

- Notice this is tight up to constant factors.

Summary -

- Coresets

- ε -coreset for directional width

- Quick + dirty - $O(1/\varepsilon^d)$

- Improved - $O(1/\varepsilon^{d-1})$

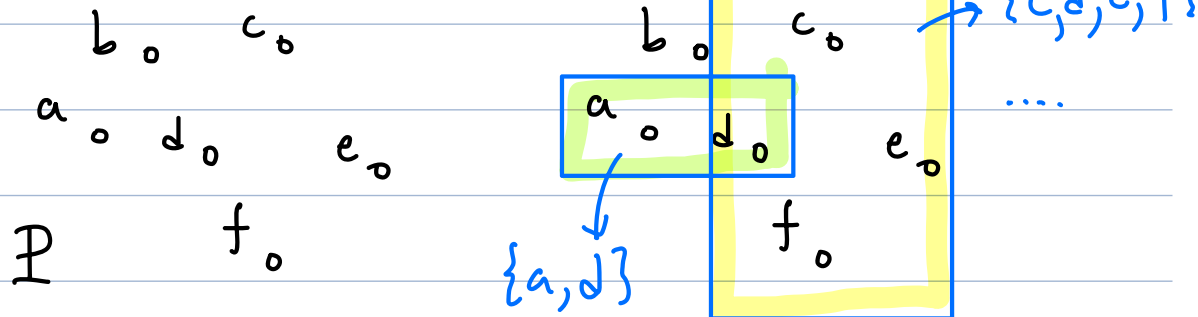
- Dudley - $O(1/\varepsilon^{(d-1)/2})$

CMSC 754 - Computational Geometry

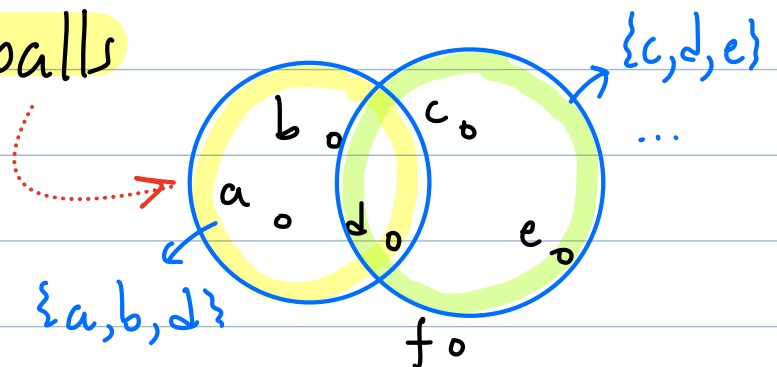
Lecture 17 - Sampling & VC-Dimension

Geometric Set Systems:

- Many problems involve sets of points that are defined by geometric objects
- Example: Given a set $P \subseteq \mathbb{R}^2$, consider all subsets of P contained in:
 - axis-aligned rectangles



- Euclidean balls



Range Space:

Given a set P , let 2^P denote the power set of P , consisting of all subsets of P ($|2^P| = 2^{|P|}$)

Range space is a pair $(X, \mathcal{R})$ where:

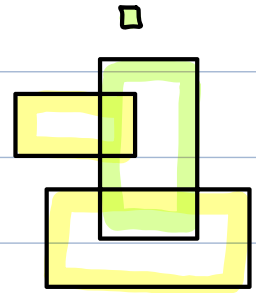
X - domain (a set)

$\mathcal{R}$ - ranges - a subset of 2^X

Eg. $X = \mathbb{R}^2$

$\mathcal{R}$ = set of all axis-aligned rectangles

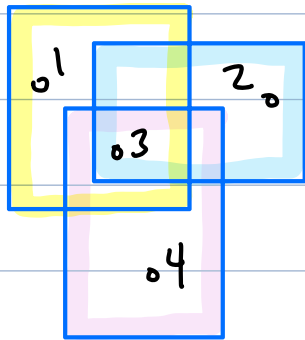
(each is an infinite set)



Restriction: Given $P \subseteq X$, define

$$\mathcal{R}|_P = \{P \cap Q \mid Q \in \mathcal{R}\}$$

the restriction of $\mathcal{R}$ to P



$$\mathcal{R}|_P = \emptyset, \{1\}, \dots, \{4\}, \dots, \{1, 2, 3, 4\}$$

But not: $\{1, 4\}$ or $\{1, 2, 4\}$

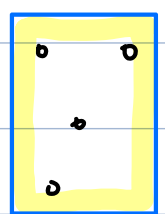
Range space $(X, \mathcal{R})$ is discrete if $|X|$ finite

Given a discrete range space $(P, \mathcal{R})$

and any $Q \in \mathcal{R}$ define Q 's measure

$$\mu(Q) = \frac{|Q \cap P|}{|P|}$$

Q


$$\mu(Q) = \frac{4}{8} = \frac{1}{2}$$

Sampling: Rather than deal with entire point set (may be huge) we would like a "good" sample.

Given $S \subseteq \mathcal{P}$ (presumably $|S| \ll |\mathcal{P}|$)
define

$$\hat{\mu}_S(Q) = \frac{|Q \cap S|}{|S|}$$

(When S is clear, we write $\hat{\mu}(Q)$)

How good is S as a sample?

Given a discrete range space $(\mathcal{P}, \mathcal{R}) + \varepsilon > 0$

ε -sample: $S \subseteq \mathcal{P}$ is an ε -sample if

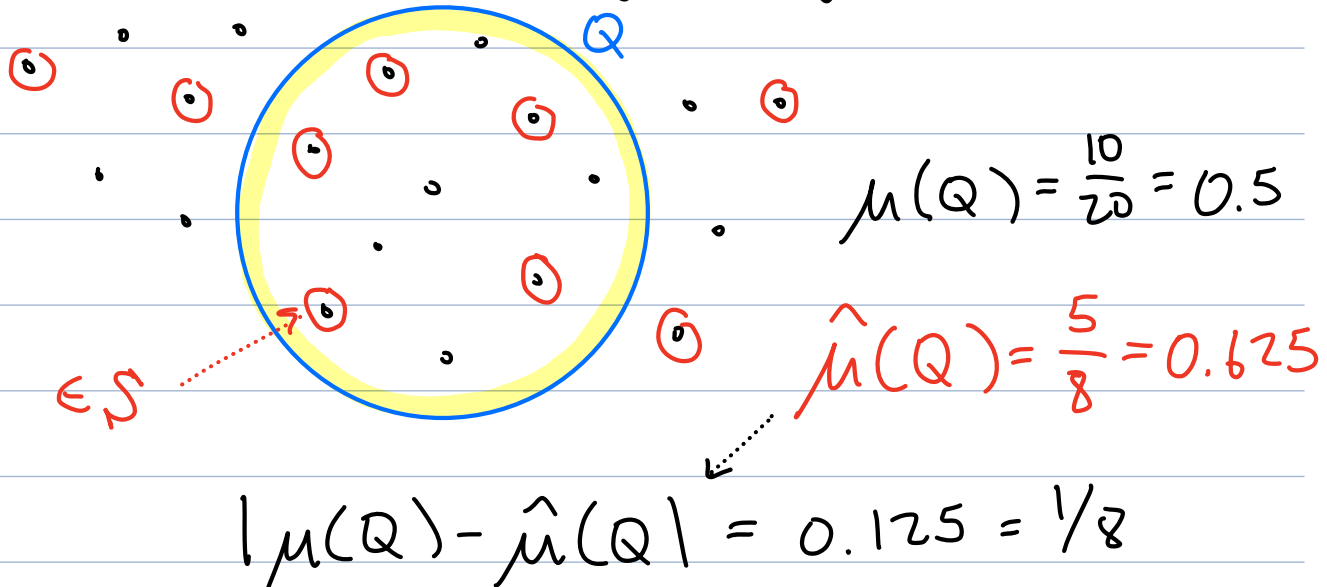
$$|\mu(Q) - \hat{\mu}(Q)| \leq \varepsilon \quad \forall Q \in \mathcal{R}$$

ε -net: $S \subseteq \mathcal{P}$ is an ε -net if

$$\mu(Q) \geq \varepsilon \Rightarrow S \cap Q \neq \emptyset \quad \forall Q \in \mathcal{R}$$

Intuition:

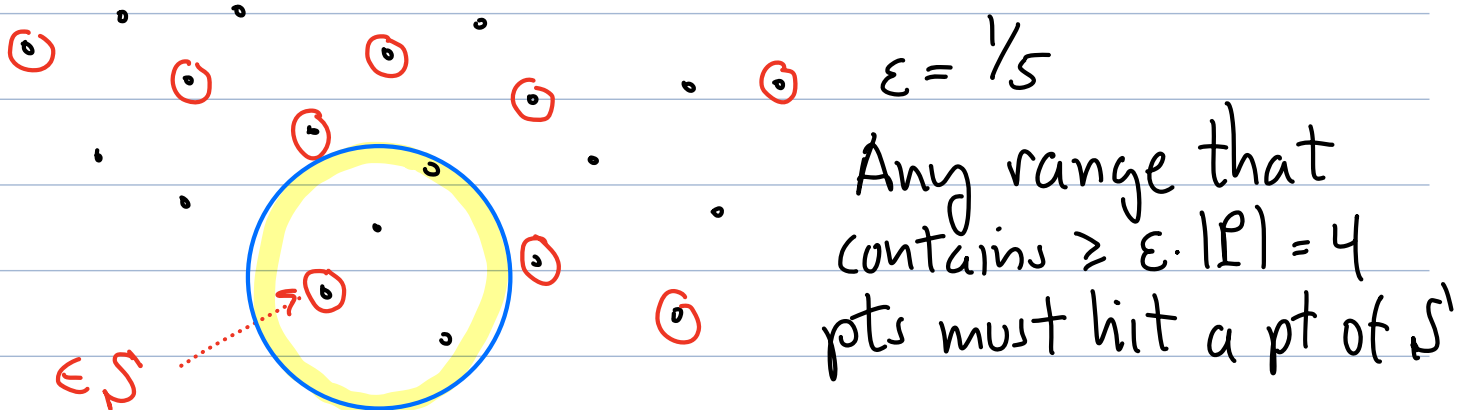
- S is an ϵ -sample if it captures roughly the same proportion of elements for any range



If this holds for all ranges in $\mathcal{R}$
 S is a $1/8$ -sample.

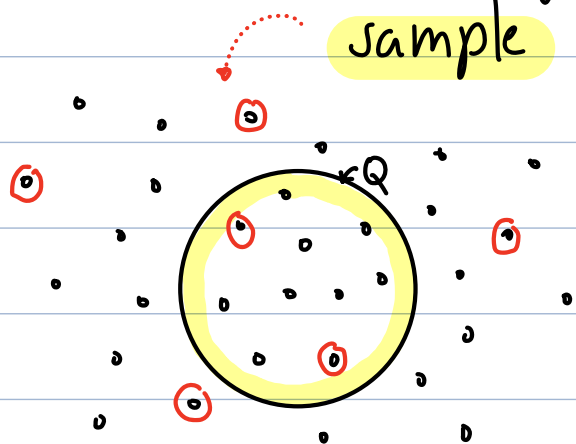
- A range Q is ϵ -heavy if $\mu(Q) \geq \epsilon$

An ϵ -net hits all ϵ -heavy ranges

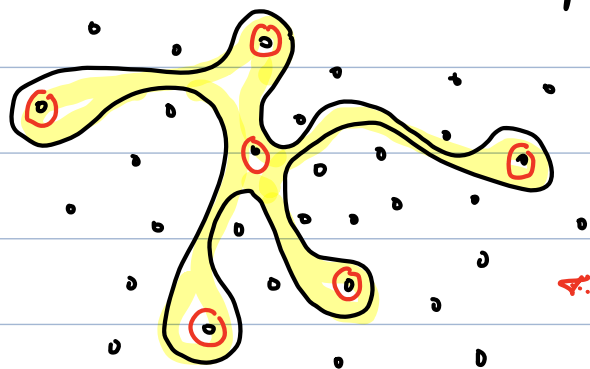


How to construct ϵ -nets + ϵ -samplers?

Intuition: Any sufficiently large random sample should work (with some prob.)



$$\frac{|P \cap Q|}{|P|} = \frac{10}{31} \approx \frac{2}{6} = \frac{|S \cap Q|}{|S|}$$



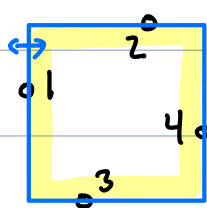
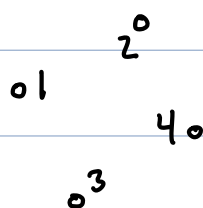
$$\frac{|P \cap Q|}{|P|} = \frac{6}{31} \neq \frac{6}{6} = \frac{|S \cap Q|}{|S|}$$

But this fails if we allow very wild range shapes.
How to formally forbid such ranges?

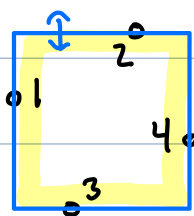
VC-Dimension:

Shattering: A range space $(X, \mathcal{R})$ shatters a pt set P if $\mathcal{R}|_P = 2^P$ (contains all subsets of P)

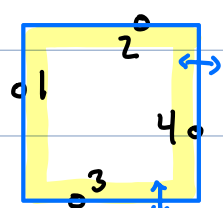
E.g. Axis-aligned rectangles shatter the pt set below:



Can include or exclude 1

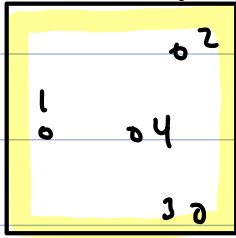


Can include or exclude 2



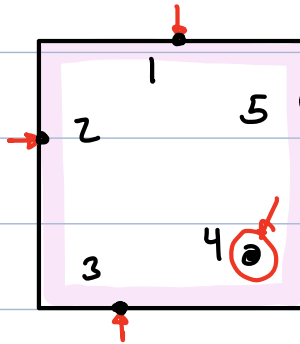
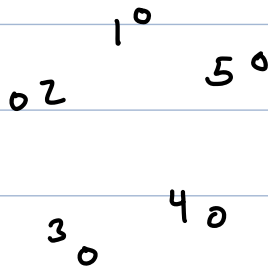
Same for 3 + 4

But they can't shatter everything:



Any rect. containing 1, 2, 3 must contain 4

... and they can never shatter a set of ≥ 5



Any rect. that contains the 1, 2, 3, 5 must contain 4

Def: The VC-dimension of a range space $(X, \mathcal{R})$ is the size of the largest pt set shattered by $\mathcal{R}$.

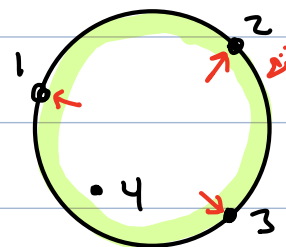
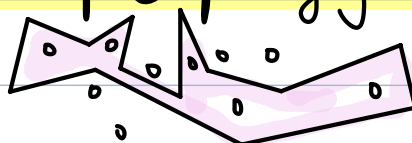
("VC" - Vapnik-Chervonenkis - 1971)

Examples:

→ VC-dim of axis-aligned rects in $\mathbb{R}^2 = 4$

→ VC-dim of Euclidean disks in $\mathbb{R}^2 = 3$

→ VC-dim of simple polygons in $\mathbb{R}^2 = \infty$



Intuitively: Range spaces of constant VC-dim have a constant num. of degrees of freedom

Sauer's Lemma: If $(X, \mathcal{R})$ is a range space of VC-dim d in $|X| = n$, then

$$|\mathcal{R}| = \mathcal{O}(n^d)$$

More precisely:

$$|\mathcal{R}| \leq \Phi_d(n)$$

where:

$$\Phi_d(n) = \binom{n}{0} + \binom{n}{1} + \dots + \binom{n}{d}$$

Observe: Φ satisfies the recurrence:

$$\Phi_d(n) = \Phi_d(n-1) + \Phi_{d-1}(n-1)$$

↳ (Exercise)

Proof: (of Sauer's Lemma) Induction on $d + n$.

Basis: $n = 0$ or $d = 0$ - trivial $\mathcal{R} = \{\emptyset\}$

Step: Fix any $x \in X$

Consider two new range spaces:
over $X \setminus \{x\}$

$$\mathcal{R}_x = \{Q \setminus \{x\} : Q \cup \{x\} \in \mathcal{R} + Q \setminus \{x\} \in \mathcal{R}\}$$

↳ Pairs that differ only on x

$$\mathcal{R} \setminus \{x\} = \{Q \setminus \{x\} : Q \in \mathcal{R}\}$$

↳ Just remove x

Example: $X = \{1, 2, 3, 4\}$ let $x = 4$

Suppose $\mathcal{R}$ has:

$$\{2, 3\} + \{2, 3, 4\} \longrightarrow$$

$$\{1\} + \{1, 4\} \longrightarrow$$

$$\{\} + \{4\} \longrightarrow$$

$\mathcal{R}_x$ has:

$$\{2, 3\}$$

$$\{1\}$$

$$\{\}$$

and $\mathcal{R}$ has: $\{1, 3\}$ but not $\{1, 3, 4\}$
 $\{2, 4\}$ but not $\{2\}$

$$\text{Then: } \mathcal{R}_x = \{\{\}, \{1\}, \{2, 3\}\}$$

$$\mathcal{R} \setminus \{x\} = \{\{\}, \{1\}, \{2, 3\}, \{1, 3\}, \{2\}\}$$

Observe:

- $|\mathcal{R}| = |\mathcal{R}_x| + |\mathcal{R} \setminus \{x\}|$

- $\mathcal{R}_x$ has VC-dim $d-1$

- Both over domain of size $n-1$

$$\Rightarrow |\mathcal{R}| \leq \Phi_{d-1}(n-1) + \Phi_d(n-1) = \Phi_d(n) \quad \square$$

Recall:

Given a discrete range space $(P, \mathcal{R})$ + $\varepsilon > 0$

ε -sample: $S \subseteq P$ is an ε -sample if

$$|\mu(Q) - \hat{\mu}(Q)| \leq \varepsilon \quad \forall Q \in \mathcal{R}$$

ε -net: $S \subseteq P$ is an ε -net if

$$\mu(Q) \geq \varepsilon \Rightarrow S \cap Q \neq \emptyset \quad \forall Q \in \mathcal{R}$$

Range spaces of low VC-dimension have
 ε -samples + ε -nets of small size:

ε -Sample Theorem: Given range space $(X, \mathcal{R})$ of VC-dim d , let P be finite subset of X . There exists constant c s.t. with probability $\geq 1 - \varphi$, a random sample of P of size $\geq$

$$\frac{c}{\varepsilon^2} \left(d \cdot \log \frac{d}{\varepsilon} + \log \frac{1}{\varphi} \right)$$

is an ε -sample for $(P, \mathcal{R})$.

ε -Net Theorem: Given range space $(X, \mathcal{R})$ of VC-dim d , let P be finite subset of X . There exists constant c s.t. with probability $\geq 1 - \varphi$, a random sample of P of size $\geq$

$$\frac{c}{\varepsilon} \left(d \log \frac{1}{\varepsilon} + \log \frac{1}{\varphi} \right)$$

is an ε -net for $(P, \mathcal{R})$.

Too many parameters! $\ddot{\smile}$

tl;dr : - Constant VC-dim
- Constant prob. of success

Size of ε -sample is $O\left(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon}\right)$

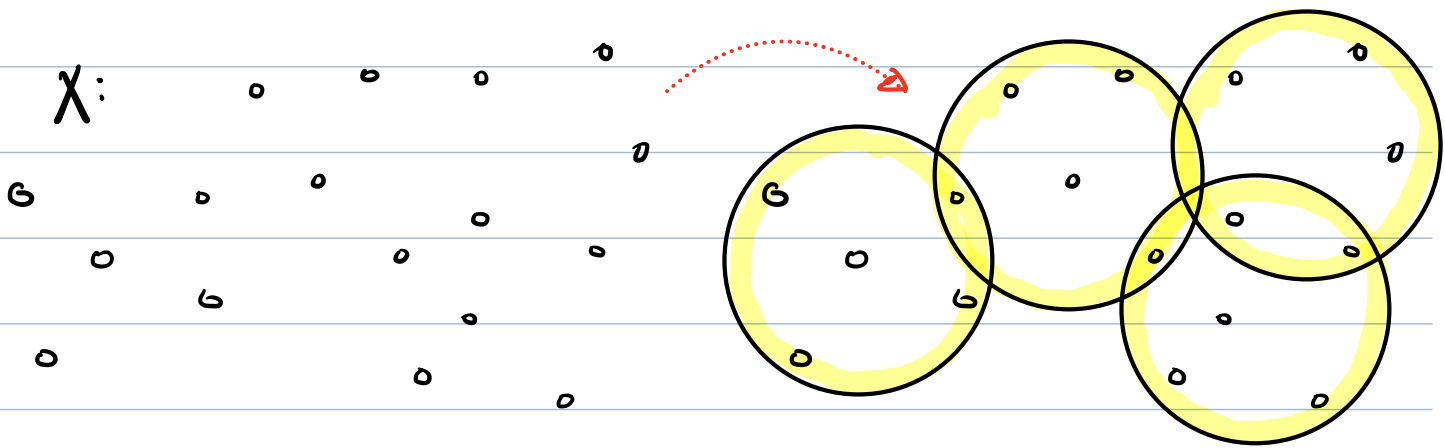
ε -net is $O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$

Proofs? See Har-Peled's book

Application: Geometric Set Cover

Given a pt set X + a collection of sets R over X , a **cover** is a collection of sets from R that contain every pt of X

E.g. X is a set of n pts in $\mathbb{R}^d$
 R = set of all unit Euclidean balls in $\mathbb{R}^d$



Set cover Problem: Given X and R , find the **smallest** cover of X

- Set cover is **NP-hard** ;)
- **No known constant factor approximation** ;)
- Simple greedy algorithm computes a cover of size
 $\leq (\ln |X|) \cdot \text{opt}$

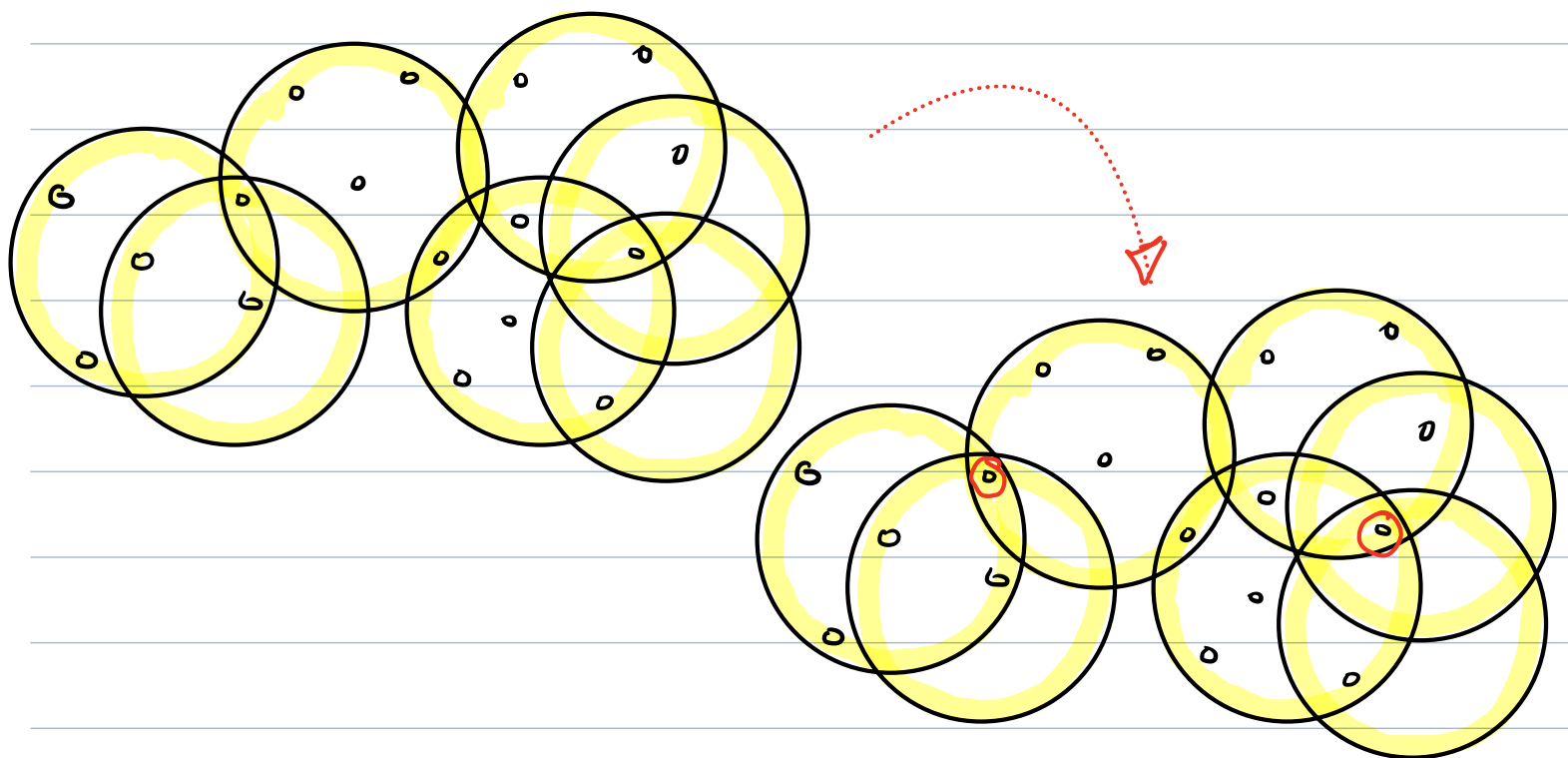
Select set that covers the most uncovered pts

We'll show that if $(X, \mathcal{R})$ is a set system of constant VC-dimension, it is possible to compute an approx. solution of size $\leq (\log k)\text{-opt}$

where k is number of sets in opt. cover
(Note $k < |X|$, so this is always better)

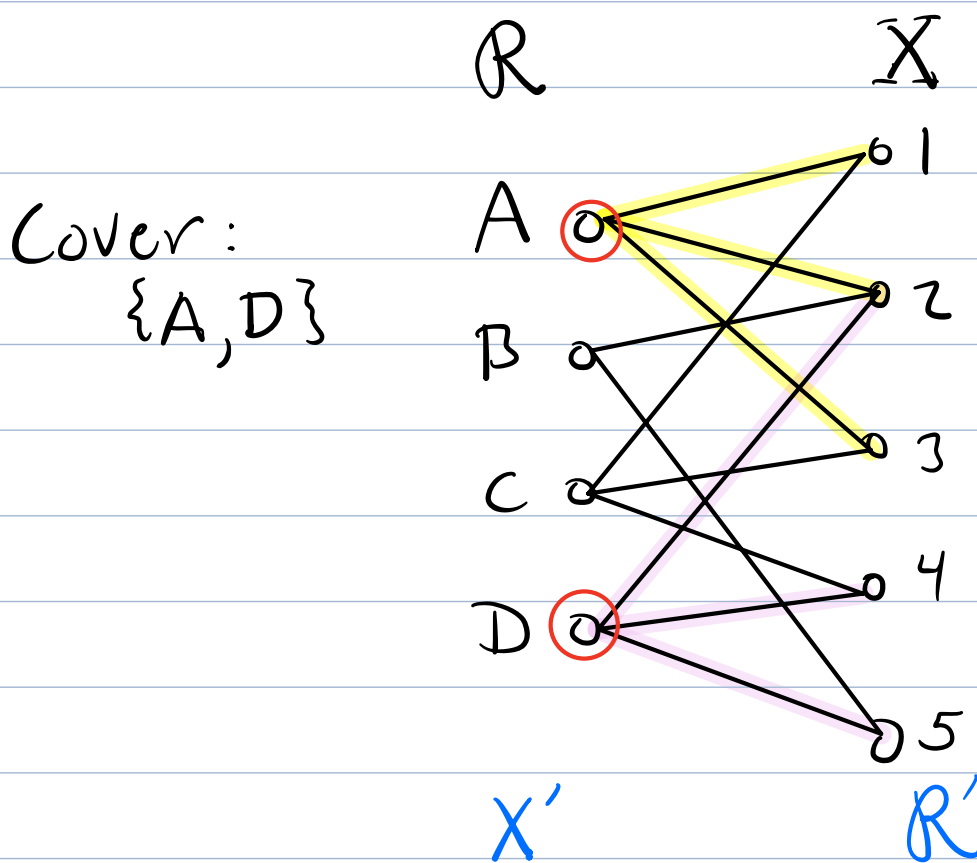
Set cover $\leftrightarrow$ Hitting Set Duality

Hitting Set: Given a collection of sets $\mathcal{R}$ over some domain X , a **hitting set** is a subset of X such that every set of $\mathcal{R}$ contains at least one of them.



Set cover + hitting set are the same problem in disguise

E.g. $A = \{1, 2, 3\}$ $B = \{2, 5\}$
 $C = \{1, 3, 4\}$ $D = \{2, 4, 5\}$



Let's reinterpret: sets $\rightarrow X'$; pts $\rightarrow R'$

1: $\{A, C\}$ 2: $\{A, B, D\}$ 3: $\{A, C\}$ 4: $\{C, D\}$ 5: $\{B, D\}$

Hitting set: $\{A, D\}$

Obs: (X, R) has set cover of size k iff
 (X', R') has hitting set of size k

Theorem: Given a set system $(X, \mathcal{R})$ of constant VC-dimension, in polynomial time it is possible to compute a hitting set of size $O(k^* \log k^*)$ where k^* = size of optimal hitting set.

Note: A set has constant VC-dim iff its dual has constant VC-dim.

Iterative Reweighting:

Weighted ϵ -Nets: Given a set system $(X, \mathcal{R})$ where each $x \in X$ has a positive weight $w(x)$. Let $w(X)$ be total weight:

$$w(X) = \sum_{x \in X} w(x)$$

A set $S \subseteq X$ is an ϵ -net if

$$\forall Q \subseteq \mathcal{R} \text{ if } \frac{w(Q \cap P)}{w(P)} \geq \epsilon \text{ then } Q \cap S \neq \emptyset$$

Standard ϵ -net $\equiv$ all pts have $w(x) = 1$

Weighted sampling:

ϵ -Net Theorem still holds, but rather than random sample of size $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ sample each point with probability proportionate to its weight to get a set of this size.

Iterative Reweighting:

- Guess the size k of opt hitting set (binary search to get best k)
- Set all weights to 1
- Repeat:
 - $S \leftarrow$ weighted ϵ -net of X
 - Is this a hitting set? yes $\rightarrow$ success
 - No? Find any set $Q \subseteq R$ not hit + double weights of all $x \in Q$
- Too many iterations? Fail $\rightarrow$ try larger k

Intuition: If we fail to hit we double weights of unhit object - more likely to hit next time.

Why it works: Critical items (in opt. solution) increase in weight rapidly - eventually they are all selected.

Algorithm: Given $(X, \mathcal{R})$

for $k = 1, 2, 4, \dots, 2^i, \dots$ until success

// Guess that $\exists$ hitting set of size k

- $\forall x \in X$ set $w(x) \leftarrow 1$

- Set $\epsilon \leftarrow 1/4k$

(for suitable const. c)

- Repeat until success or $2k \cdot \lg n/k$

iterations

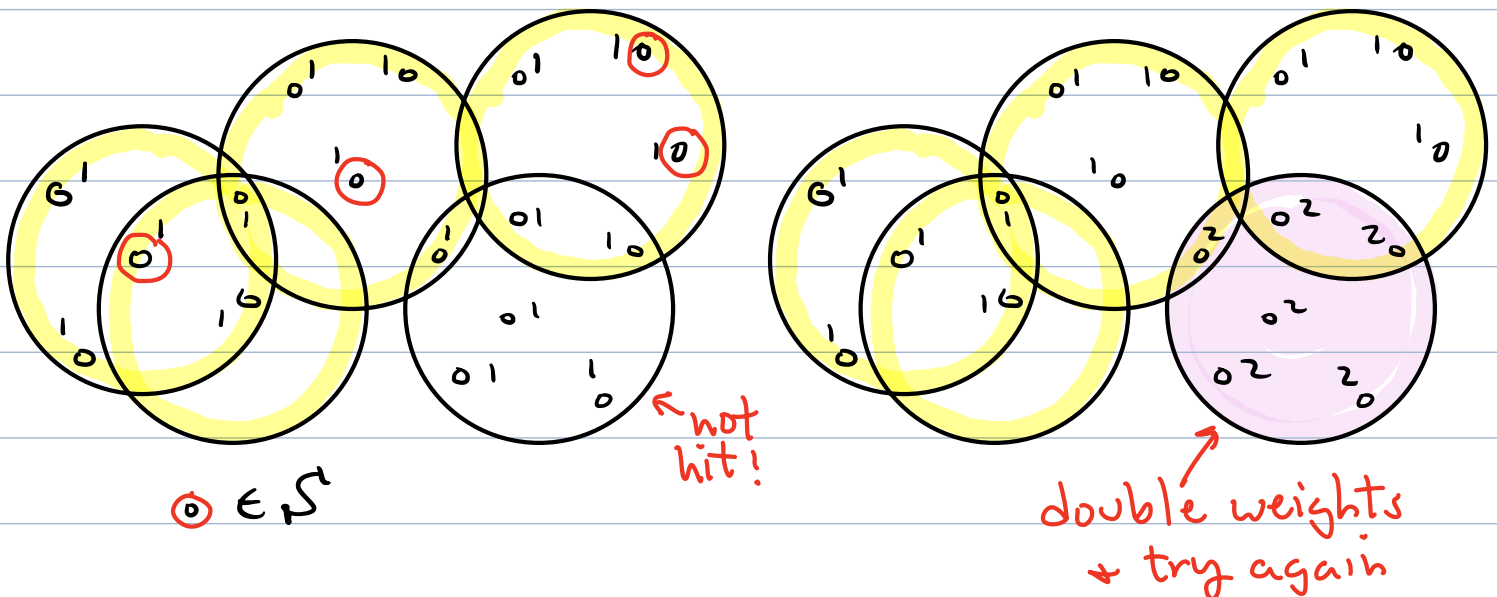
- $S \leftarrow$ wgt ϵ -net of size $c \cdot k \cdot \log k$

- are all sets of $\mathcal{R}$ hit by S ?

- yes $\rightarrow$ return with success!

- no $\rightarrow$ find any set $Q \in \mathcal{R}$
not hit

$\forall x \in Q, w(x) \leftarrow 2 \cdot w(x)$



Why this works? Assume k is correct

- Since opt hitting set hits all sets, at least one point of opt doubles in weight
- Weight of opt hitting set grows exponentially fast
- Total weight of pt set grows much more slowly
- Soon, opt hitting set's weight is so high we must sample it.

Lemma: If $(X, \mathcal{R})$ has hitting set of size k , then the repeat-loop has success within $2k \cdot \lg^{n/k}$ iterations. ($\lg \equiv \log_2$)

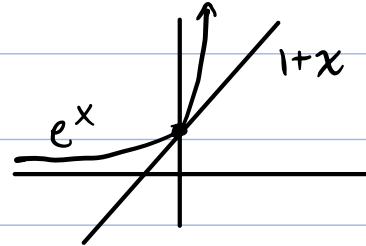
Proof: Let $n = |X|$ $m = |\mathcal{R}|$

- Let H be hitting set of size k
 $W_i(X)$ = total weight after i^{th} iteration
 $W_i(H)$ = weight of H " " "
- Note: $W_0(X) = |X| = n$
- Since S is an ϵ -net, if we fail to hit a set Q , then $w_i(Q) < \epsilon W_i(X)$

$$\begin{aligned}\Rightarrow W_i(X) &= W_{i-1}(X) + w_{i-1}(Q) \\ &\leq W_{i-1}(X) + \varepsilon \cdot W_{i-1}(X) \\ &= (1 + \varepsilon) W_{i-1}(X)\end{aligned}$$

$$\begin{aligned}\Rightarrow W_i(X) &\leq (1 + \varepsilon)^2 W_{i-2}(X) \\ &\leq (1 + \varepsilon)^3 W_{i-3}(X) \\ &\vdots \\ &\leq (1 + \varepsilon)^i W_0(X) = (1 + \varepsilon)^i \cdot n\end{aligned}$$

Fact: $1 + x \leq e^x$



$$\Rightarrow W_i(X) \leq n \cdot e^{i \cdot \varepsilon}$$

Since H hits all sets, it hits Q

$\Rightarrow$ in each (unsuccessful) iteration, at least one element of H doubles

$\Rightarrow$ growth rate of $W_i(H)$ is slowest if all its members double at same rate (Jensen's Ineq.)

$\Rightarrow$ After i^{th} iteration, each of the k elements of H doubled i/k times

$$\Rightarrow W_i(H) \geq k \cdot 2^{i/k}$$

Since $H \subseteq X$, we know $W_i(H) \leq W_i(X)$

$$\Rightarrow k \cdot 2^{i/k} \leq n \cdot e^{i \cdot \varepsilon}$$

Recall, we set $\varepsilon \leftarrow 1/4k$

$$\Rightarrow k \cdot 2^{i/k} \leq n \cdot e^{i/4k}$$

$$\Rightarrow \lg k + \frac{i}{k} \leq \lg n + \frac{i}{4k} \lg e$$

$\lg e < 2$

$$\leq \lg n + \frac{i}{2k}$$

$$\Rightarrow \frac{i}{k} - \frac{i}{2k} = \frac{i}{2k} \leq \lg n - \lg k = \lg n/k$$

$$\Rightarrow \text{No. of iterations } i \leq 2k \cdot \lg n/k$$

(If we exceed this number, we know $|H| > k$, and we fail)

□

Total time:

$$(2k \cdot \log n/k) \cdot [(k \cdot \log k) + m \cdot k]$$

since $k \leq n$

$$= O(n^2 \cdot m \cdot \log n)$$

Summary -

- Range spaces
- ϵ -samples (approx. counts)
- ϵ -net (hits all ϵ -heavy sets)
- VC-dimension (largest shatterable set size)
- Sauer's Lemma: No. of ranges is $O(n^{\text{VC-dim}})$
- ϵ -Sample Thm - size $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$
- ϵ -Net Thm - size $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$
- Approximate set-cover for sets of finite VC-dim
 - $O(\log k^*)$ -approximation, where k^* = size of opt. cover
 - Based on iterative reweighting

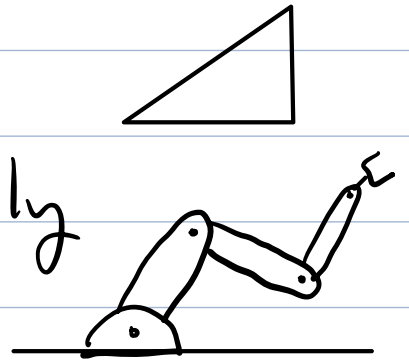
CMSC 754 - Computational Geometry

Lecture 18 - Motion Planning

Motion Planning:

Given a robot (with constraints on how it can move), a set of obstacles, and a start + target configurations for the robot, is there a collision-free motion plan?

Robot: May be rigid object or linked/hinged assembly



Motion constraints:

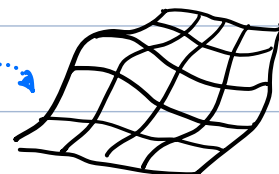
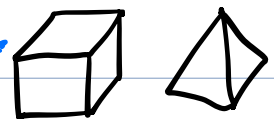
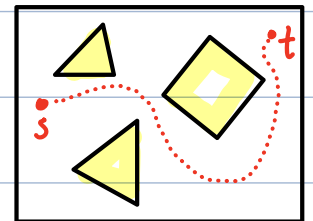
- Translation
- Rotation
- Speed/Acceleration limits
- ⋮

Obstacles: Polygons in 2-D

Polyhedra in 3-D

Curved objects

Terrains



We'll mostly consider the simplest scenario:

Space - $\mathbb{R}^2$

Robot - (convex) polygon

Motion - translation only

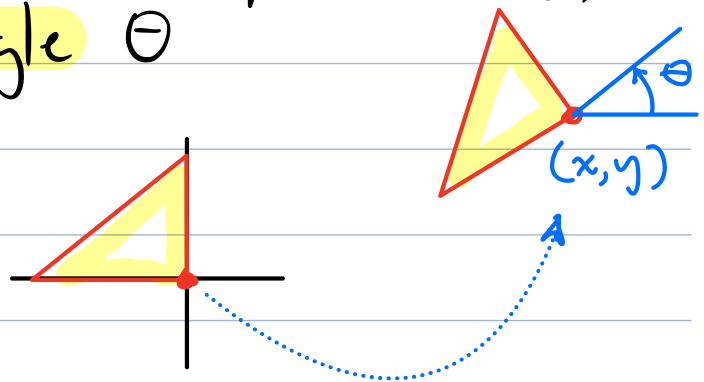
Obstacles - collection of nonoverlapping convex polygons

Configuration: A set of parameters that uniquely specifies the robot's position

E.g. Rigid in 2-D

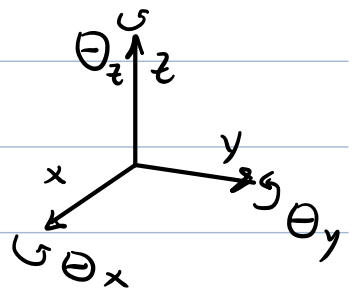
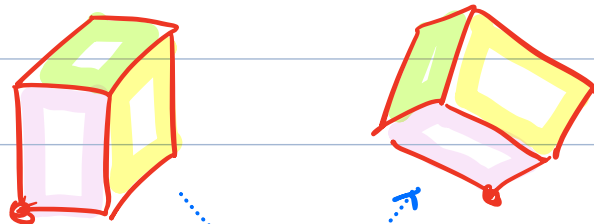
- location of reference point (x, y)
- rotation angle θ

Reference position:



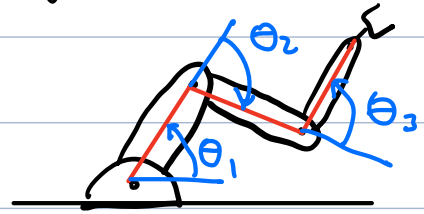
Rigid in 3-D

- location (x, y, z)
- rotation
 - Euler angles $(\theta_x, \theta_y, \theta_z)$
 - Quaternion



Linked/Hinged: Joint angles

$(\theta_1, \theta_2, \theta_3)$



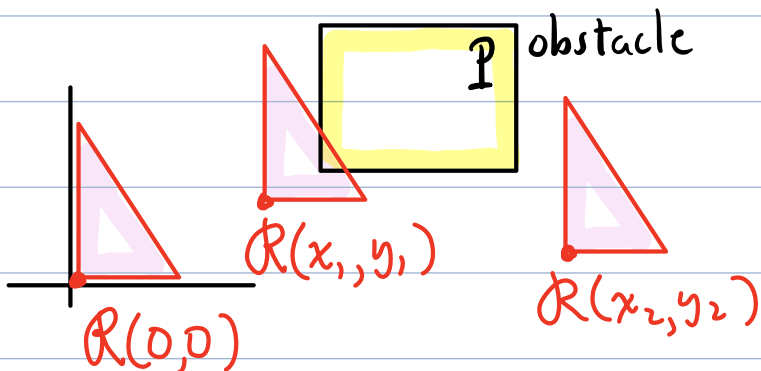
Motion Planning in Config. Space:

- Rather than moving a robot amidst obstacles
 - instead -
- Move a point in the robot's configuration space

Need to distinguish between:

free configuration - robot does not collide
forbidden configuration - robot collides

E.g. Translation only (configuration = location of ref. point)



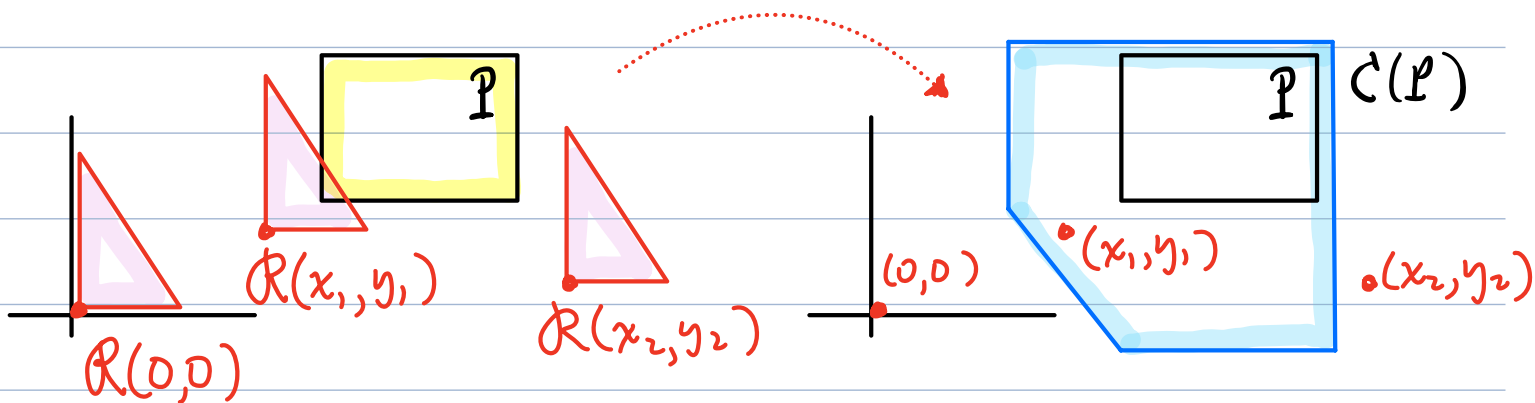
(x_1, y_1) - forbidden
 $R(x_1, y_1) \cap P \neq \emptyset$

(x_2, y_2) - free
 $R(x_2, y_2) \cap P = \emptyset$

Configuration Obstacle (or C-Obstacle)

Given robot R , config vector v , obstacle P
the C-obstacle for P is:

$$C_R(P) = \{v \mid R(v) \cap P \neq \emptyset\}$$



C-Obstacles for Translation - Minkowski Sum

The easiest C-obstacles are for translational motion.

Def: Given $P, Q, S \subseteq \mathbb{R}^d$ + $\alpha \in \mathbb{R}$

$$P \oplus Q = \{p + q : p \in P, q \in Q\}$$

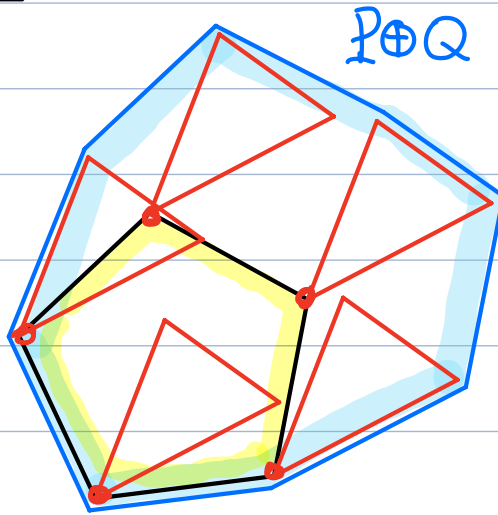
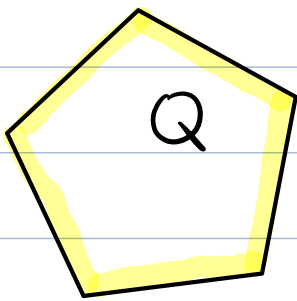
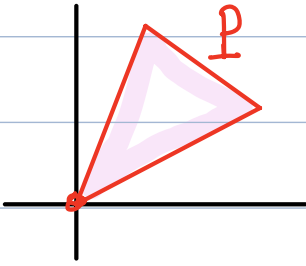
Minkowski Sum

$$\alpha P = \{\alpha \cdot p : p \in P\}$$

$$-P = \{-p : p \in P\}$$

Intuition: $P \oplus Q$ - Place P so its ref. pt. is at origin

- Sweep P 's ref pt around Q
+ see what's swept out



Support Function:

$$h_P(\vec{u}) = \max_{\vec{p} \in P} (\vec{u} \cdot \vec{p})$$

Lemma:

$$h_{P \oplus Q} = h_P + h_Q$$

Lemma: Given a translating robot R + obstacle P :

$$C(P) = P \oplus (-R)$$

Proof: For any translation vector t

$$t \in C(P) \Leftrightarrow R(t) \text{ collides with } P$$

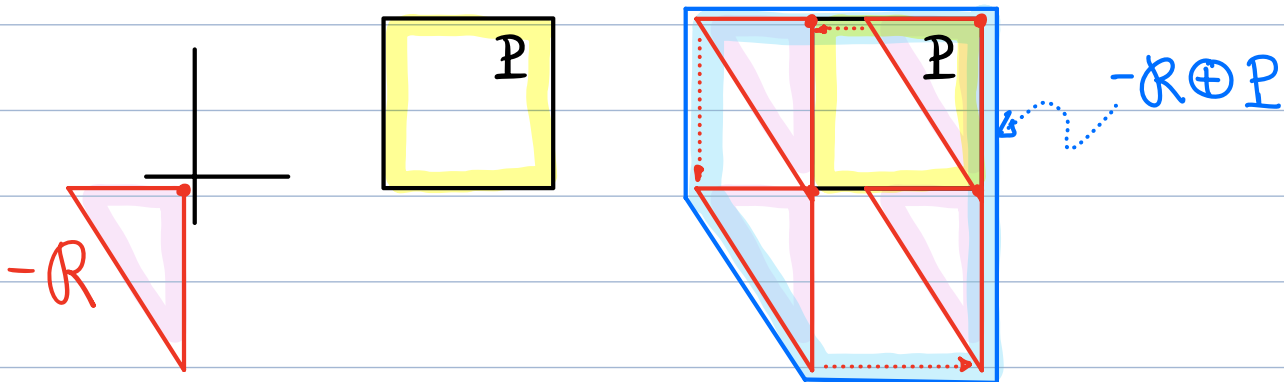
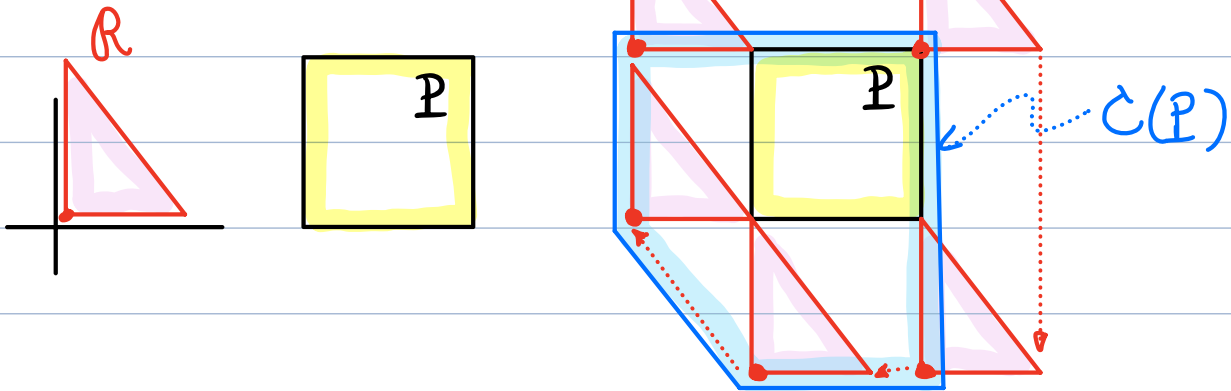
$$\Leftrightarrow R + t \cap P \neq \emptyset$$

$$\Leftrightarrow \exists r \in R, p \in P \quad r + t = p$$

$$\Leftrightarrow \quad \quad \quad t = p - r$$

$$\Leftrightarrow t \in P \oplus (-R)$$

Proof by picture:



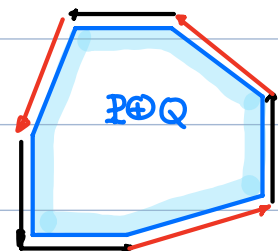
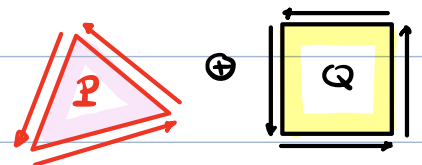
Computing the Minkowski Sum:

If P is a convex m -gon

Q is a convex n -gon

can compute $P \oplus Q$ in time $O(m+n)$

- Direct edges (CW) (vectors)
- Sort them by angle
- Join them tail to head

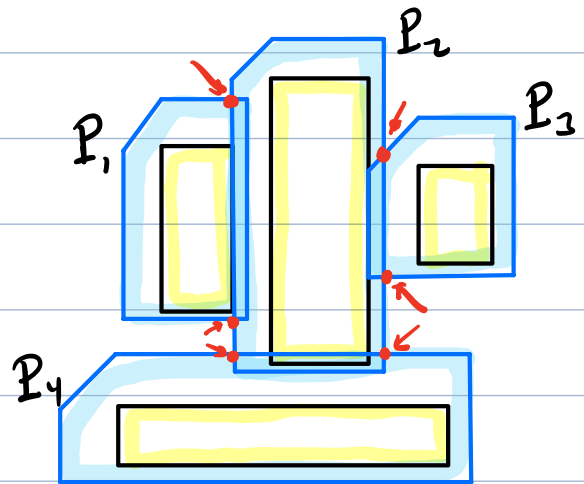
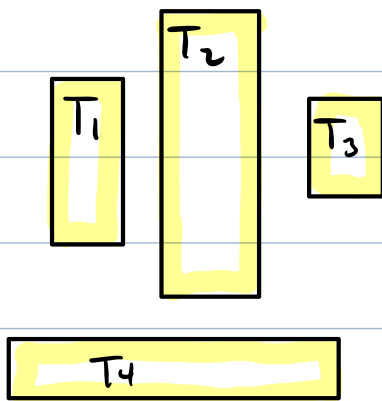


Complexity of C-Obstacles:

- Suppose we have an m -sided convex robot R and a collection of disjoint convex obstacles $T_1, \dots, T_k$. Let $n_i = \text{num. of sides in } T_i$. Let $n = \sum n_i$.
- What is total size of config. obstacles?

$$\text{Let } P_i = T_i \oplus (-R) \quad C_R(T) = \bigcup_{i=1}^k P_i$$

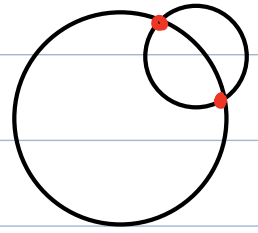
- Although T_i 's are disjoint, P_i 's may overlap



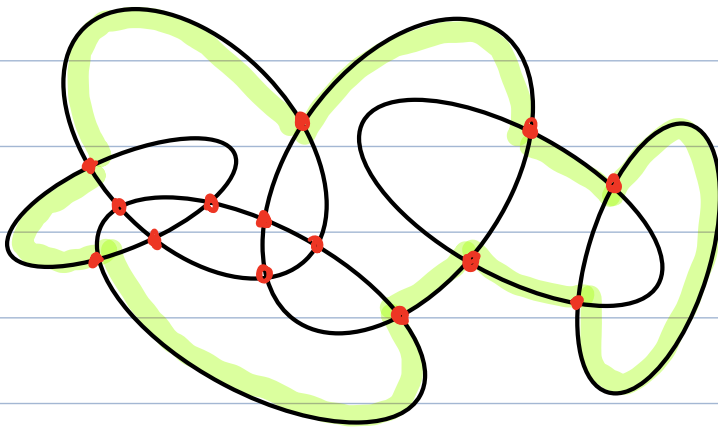
- Points of boundary overlaps create additional vertices - How many? $O(n)$ $O(n^2)$?

Pseudodisks:

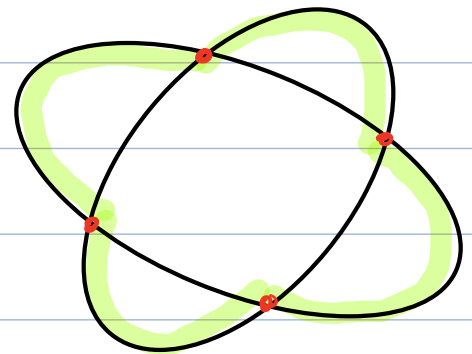
- The boundaries of two circular disks intersect at most twice.



- A collection of convex objects $\{P_1, \dots, P_k\}$ is a collection of pseudodisks if the boundaries of any pair intersect at most twice.



Collection of pseudodisks



Not pseudodisks

Lemma: Given a set $T_1, \dots, T_k$ of disjoint convex bodies in $\mathbb{R}^2$ and convex Q

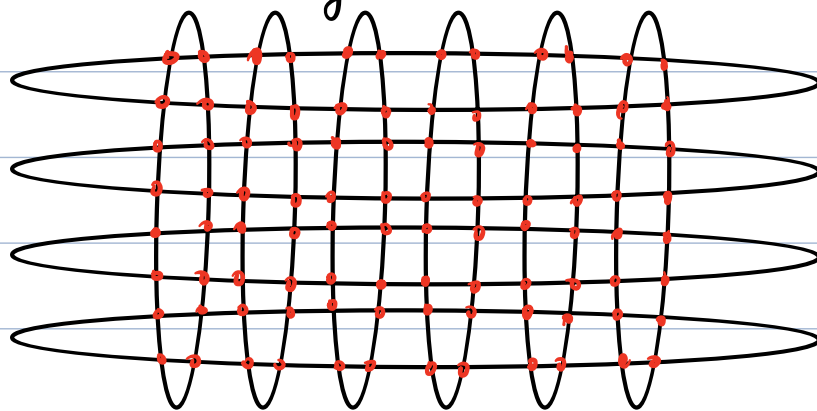
$$\{T_i \oplus Q : 1 \leq i \leq n\}$$

is a collection of pseudodisks.

Proof: See latex notes

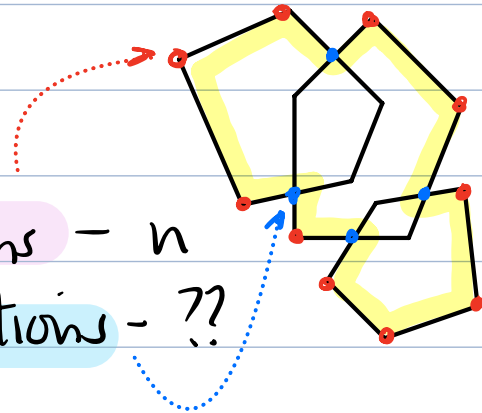
Theorem: Given a collection of pseudodisks with a total of n vertices, their union has a total of at most $2n$ vertices.

In general, union may have $O(n^2)$ vertices

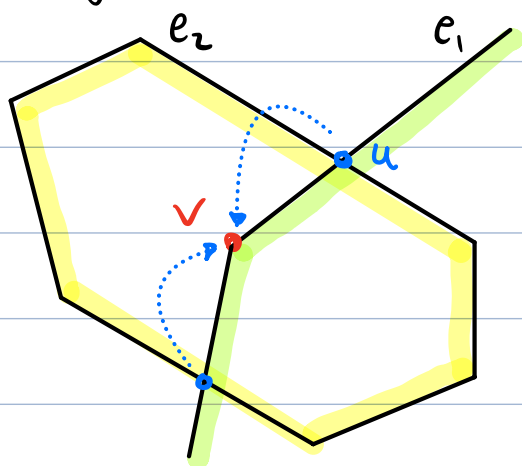


Vertex types:

- Vertices of original polygons - n
- Vertices caused by intersections - ??

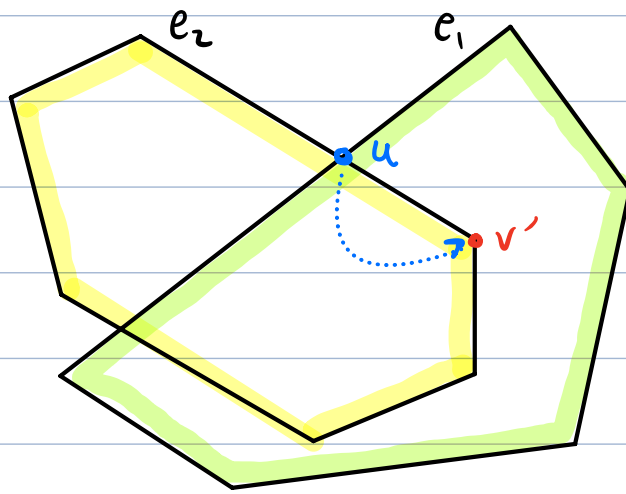


- We'll "charge" intersection vertices to vertices hidden in the interior
- Suppose edges e_1 + e_2 intersect at u



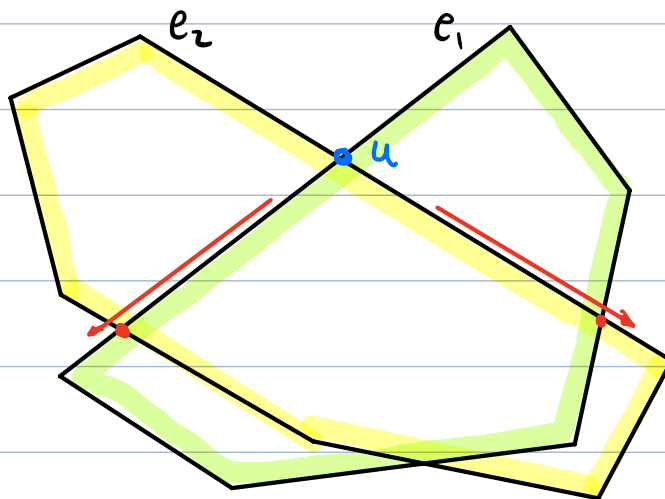
- if e_1 leads to internal vertex v , charge u to v
- v gets ≤ 2 charges

- Otherwise, if e_1 cuts through, but e_2 leads to internal vertex v' , charge u to v'



(Again v' can be charged at most twice)

- Otherwise both $e_1 + e_2$ cut through the other polygon



But this cannot happen since these are pseudodisks!

Since every vertex is charged at most twice
union has at most $2n$ vertices. $\square$

Theorem: Given a convex m -sided robot and a collection of n disjoint obstacles, each with $O(1)$ sides, the total boundary complexity of the union of C -obstacles is $O(m \cdot n)$.

Proof: We have a collection of n pseudodisks each with $O(1) + m = O(m)$ sides.

$\Rightarrow$ Total vertices is $O(m \cdot n)$

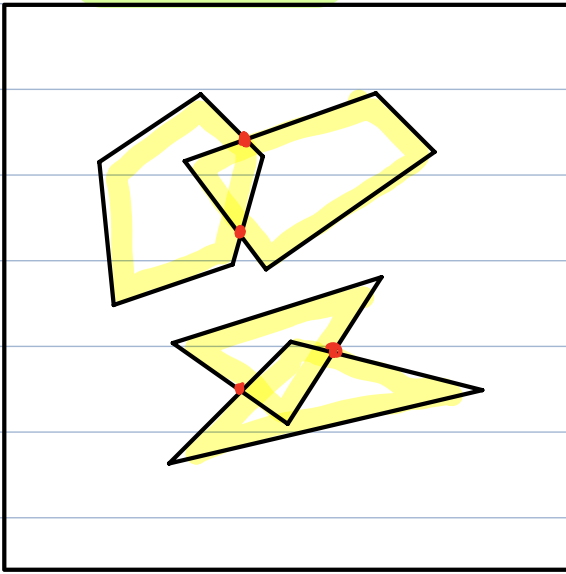
$\Rightarrow$ Union complexity is $O(m \cdot n)$.

Path Planning in Config Space:

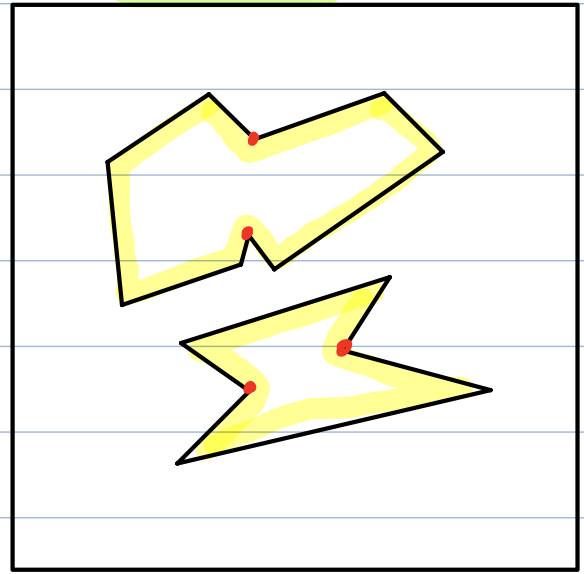
Once we have computed the C -obstacles, how to find a path between start + target?

- Compute union of C -obstacles
- Compute a decomposition of the complement space (outside the C -obstacles)
Eg. Triangulate or trapezoid map
- Compute dual graph, joining pairs that can reach each other

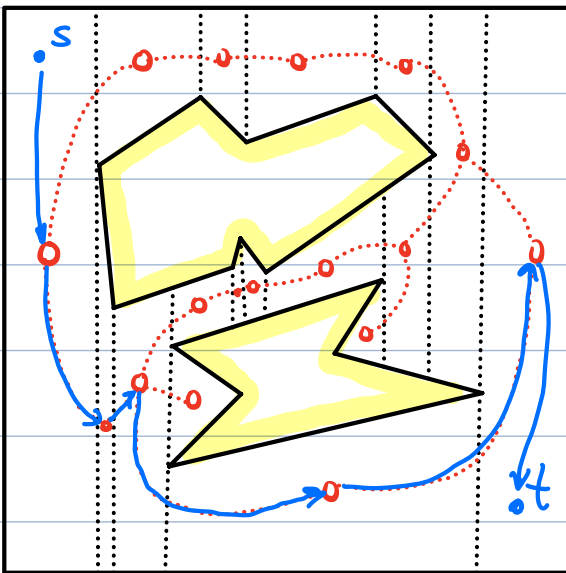
C-obst.



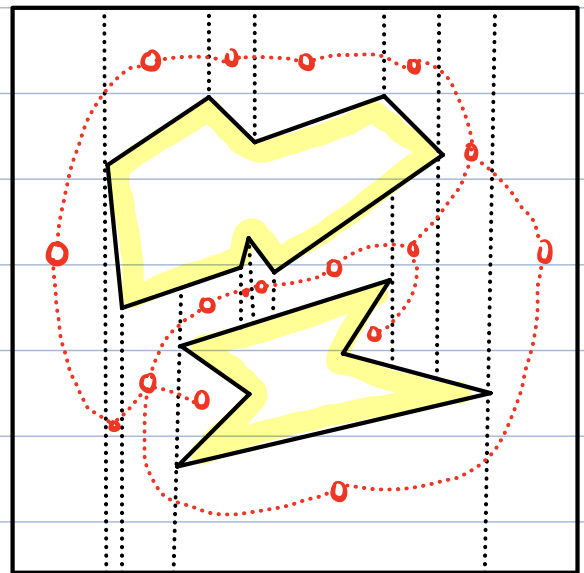
Union



Path $s \rightsquigarrow t$



Trap. Map



Finally: Given start s + target t ,

- find trapezoids containing them
- if reachable in dual graph
 - create path joining them
- else - out put "unreachable"

Note: Not the shortest path

Summary:

- Basic definitions (robot, obstacles,...)
- Configuration space / C-obstacles
- C-Obstacle for
 - convex robot R
 - convex obstacle Pis $P \oplus (-R)$ - Minkowski sum
- Complexity of union of C-obstacles
 - Collection of pseudo disks
 - Total complexity linear (for convex objects in $\mathbb{R}^2$)
- Path planning in config space
 - via trapezoidal maps