CS 274

Joon Kim

Convex Hulls

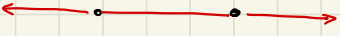
Let P be a set of points in a plane.

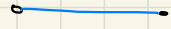
Def) Convexity: $\forall p, q \in P$, the line segment $\overline{pq} \subseteq P$. ex) 

Def) Convex Hull: $H = \text{conv}\, P$, multiple definitions: ex) 

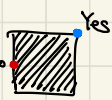
- (informal) stretch an elastic band around P. H is band + all inside.
- (formal ①) the "smallest" convex set that includes P. ↳ no convex strict subset
- (formal ②) intersection of all convex sets that include P.
- (formal ③) the set of all points that are convex combinations of points in P.

Def) Affine Combination: any point $\sum_{i=1}^{j} w_i p_i$ where $p_i \in P$, $\sum_{i=1}^{j} w_i = 1$.

↳ ex) 

↳ ex) 

Def) Convex Combination: affine comb. where every $w_i \geq 0$.

v is a vertex of convex hull H iff $v \notin \text{conv}(H \setminus \{v\})$. ex) 

2D CH Algorithms

Input: finite point set P. Output: line segments on boundaries of conv P. (in ccw order)

We need one geometric predicate.

$$\text{Orient2D}(p,q,r) = \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix} = \begin{vmatrix} p_x - r_x & p_y - r_y \\ q_x - r_x & q_y - r_y \end{vmatrix} > 0 \Rightarrow \text{ccw}, = 0 \Rightarrow \text{colinear}, < 0 \Rightarrow \text{cw}.$$

signed area of parallelogram

ccw(p,q,r) := return Orient2D(p,q,r) > 0.

Graham's Scan (Andrew's Modification): Constructs the lower boundary.

Q ← points in P sorted in lexicographic order. ex) 

S ← empty stack

while Q is not empty:

- remove point from front of Q; push it to S.
- while |S| ≥ 3 and top 3 stack items p,q,r (r is top)
  - do not satisfy ccw(p,q,r): remove q from S, leaving r on top.

output segment $\overline{pq}$ for each adjacent pair p,q on S.

runs in O(n) time (except sorting) since removal is at most O(n) times.

* Upper boundary: invert Q (reverse lex. order), then run the algorithm.

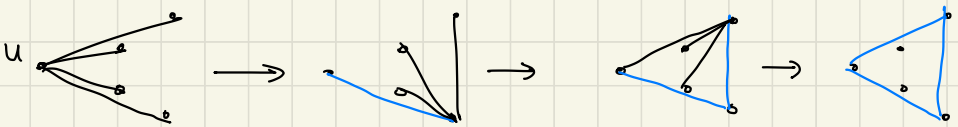
Obs) First & last points in Q must be vertices of H.

Obs2) 3 consecutive points survive only if they are ccw.

Runtime: Dominated by sorting. O(n) with radix sort. O(nlogn) with comparison.

Jarvis' March (gift-wrapping): assume no 3 points are colinear (for now).

Idea: start at leftmost point. Walk around hull, identify each segment.



$u \leftarrow$ lexicographically minimum point of P.

$p \leftarrow u$.

repeat:

- $q \leftarrow$ any point in $P \setminus \{p\}$.
- for each $r \in P \setminus \{p, q\}$, if $\text{ccw}(p, r, q)$: $q \leftarrow r$.
- output $\overline{pq}$.
- $p \leftarrow q$.

until $p = u$.

"pivoting" step

find the most cw point from p's view — $O(n)$ times

$O(h)$ times

Runtime: $O(nh)$ where $h := |H|$. ← output sensitive!

↳ good when h is small, but worst case is $O(n^2)$.

Shattering Algorithm [Tim Chan]: $O(n \log h)$ time, comparison-based

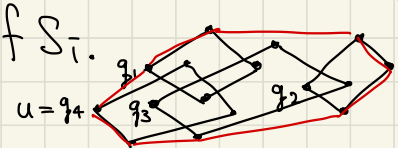
Suppose we know the output size h of conv P.

Partition P into $\lceil n/h \rceil$ subsets $S_1, \ldots, S_{\lceil n/h \rceil}$ of h points each.

$\forall i \in [\lceil n/h \rceil]$, compute $H_i \leftarrow$ conv S_i with any $O(n \log n)$ algorithm.

↳ now in $O(h \log h)$!

$\forall i$, $q_i \leftarrow$ minimum point of S_i.



$u \leftarrow$ minimum of all q_i.

$p \leftarrow u$.

repeat:

- for $i \leftarrow 1 \dots n/h$: (O(n) calls since each q_i walks around H_i once.)
  - $q_i \leftarrow$ cw-most point of H_i from p's view ex) p 
  - $q \leftarrow$ cw-most point of all q_i.
- output $\overline{pq}$.
- $p \leftarrow q$.

until $p = u$. (# of S_i · subroutine on h points)

Runtime: $\frac{n}{h} \cdot O(h \log h) = O(n \log h)$.

... but we don't know h.

↳ Guess h, say $h^* = 3$. Find first h^* edges of H by shattering.

If $h^* \geq h$, we are done. If $h^* < h$, start over with $h^* \leftarrow (h^*)^2$.

Final iteration: $h^* < h^2$. $\Rightarrow$ time $\in O(n \log h^*) \subseteq O(n \log h^2) = O(n \log h)$.

Sum of all iterations $= O(n \log 3 + n \log 3^2 + \cdots + n \log h^*)$

$= O(n \log 3 + 2n \log 3 + 4n \log 3 + n \log h^*) = O(2n \log h^*) = O(n \log h)$.

Line Segment Intersection

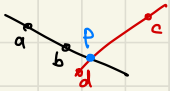
Input: line segment on the plane $\xrightarrow{\text{LSI}}$ Output: planar straight line graph (PSLG)

Def) PSLG: set of vertices & segments where segments intersect only at shared endpoints.

→ input to triangulation algorithm (covered next week!)

A geometric constructor:

Intersection of 2 lines $\overleftrightarrow{ab}, \overleftrightarrow{cd}$:



→ affine!

$$p = a + \alpha(b-a) = c + \beta(d-c).$$

$$\Rightarrow \begin{bmatrix} b_x-a_x & d_x-c_x \\ b_y-a_y & d_y-c_y \end{bmatrix} \begin{bmatrix} \alpha \\ -\beta \end{bmatrix} = \begin{bmatrix} c_x-a_x \\ c_y-a_y \end{bmatrix}.$$

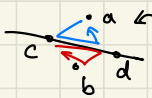
Cramer's Rule: $\alpha = \dfrac{\begin{vmatrix} c_x-a_x & d_x-c_x \\ c_y-a_y & d_y-c_y \end{vmatrix}}{\begin{vmatrix} b_x-a_x & d_x-c_x \\ b_y-a_y & d_y-c_y \end{vmatrix}} = \dfrac{\begin{vmatrix} c_x-a_x & d_x-a_x \\ c_y-a_y & d_y-a_y \end{vmatrix}}{|\ \ /\!/\ \ |} = \dfrac{\text{Orient2D}(c,d,a)}{|\ \ /\!/\ \ |}$.

→ 0 iff $\overleftrightarrow{ab} /\!/ \overleftrightarrow{cd}$.
if numerator is also 0, two lines are identical.
else, they are different lines.

Take α, substitute back to $p = a + \alpha(b-a)$ to get p.

What about line segments? Do $\overline{ab}, \overline{cd}$ intersect? (predicate)

- Iff $\alpha, \beta \in [0,1]$.



opposite sides → signs differ!

- Iff $\text{Orient2D}(c,d,a) \cdot \text{Orient2D}(c,d,b) \le 0$ & $\text{Or2D}(a,b,c) \cdot \text{Or2D}(a,b,d) \le 0$.

* Second test is slightly easier to make robust.

* Second test doesn't handle the parallel edge case, exercise to make it work.

Now consider n segments.

Worst Case: $\Theta(n^2)$ intersection points.

→ Worst Case optimal algorithm: test every pair. Not interesting.

→ What about output-sensitive?

Plane Sweep Algorithm

- Vertical sweep line moves left to right.
- Keep track of segments that (currently) intersect the sweep line.
- Output intersection point when the sweep line crosses it.

 (not rigorous!)

→ Points swept in lexicographical order for y-axis (think of slightly tilted line)

Data Structures

Status: list of all segments that currently intersect the sweep line, ordered.

Event Queue:

① left event → add segment to status.

② right event → remove segment from status.

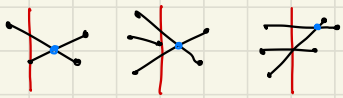
③ intersect event → swap order of segments, also report it.

* EQ does not record types of events! It's just a priority queue of points. This is because a single point can be multiple event types.

Ideas / Intuition

big time save!

- Only adjacent segments on status are checked for intersection.
- Every change to status requires new intersection tests.

ex) 

Details of EQ Q

- Dequeue(Q) gives lexico. minimum key.
- Each key/item is just a point.
- Implemented as a balanced tree / heap for O(log n) operations & lookup.

Details of Status

- Implemented as a balanced tree.
- Keys are segments, not numbers.
- Segments are ordered according to intersections with the sweep line.
- Can also use point as search key to find segments immediately above & below it.  (uses Or2D for comparison, O(log n) time!)
- Each segment in the tree remembers its endpoints (for Or2D)

Each input point maintains a (linked) list of all segments for which it is the left endpoint. 

Algorithm Input: list S of n segments of two endpoints each.

Sort endpoints in S, remove duplicates & build segments list.

Insert all endpoints into Q.

Create empty Status Tree T.

while Q is not empty:

    p ← Dequeue(Q)

    if p is new:

        Handle(p)

Handle(p):

Find set Z of segments in T that contain p. (they should be adj. in T)

R ← segments in Z whose right endpoint is p.

C ← other segments in Z, p is in the center.

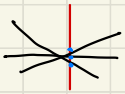
L ← segments in p's list (left endpoint is p)

if |LUCUR| > 1:

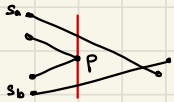
    report p as intersection of RUCUL.

else: report p as a lone endpoint of LUR.

Delete segments in CUR from T.

Insert segments LUC into T, ordered as if the sweep line were to the right of p. 

$s_b, s_a \leftarrow$ segments below & above p in T.

if LUC = $\emptyset$: 

NewEvent(s_b, s_a, p)

else: 

$s' \leftarrow$ lowest segment of LUC in T

$s'' \leftarrow$ highest //

NewEvent(s_b, s', p)

NewEvent(s_a, s'', p)

NewEvent(s_1, s_2, p):

$i = s_1 \cap s_2$ (via constructor)

if $i \neq \emptyset$ and $i > p$:

Enqueue(Q, i)

Runtime: $O(h \log n)$, where $h :=$ output size = # of segment/event point pairs.

Why? Each status operation is $O(\log n)$ time.

Each event involving j segments does $O(j)$ status operations. $\sum_{\text{events}} j = h$.

Each Q operations take $O(\log n)$ time, $|Q| \in O(n^2)$ (every pair).

At most h events, ≤ 3 Q operations per event.

Better bound: $O((n+I)\log n)$, where $I :=$ # of intersections.

Why? In final PSLG, $|E| \geq \frac{h}{2}$, $|V| \leq 2n+I$. For planar graphs, $|E| \leq 3|V|$.

$\Rightarrow h \leq 12n + 6I$. //

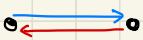
Overlay of Subdivisions

Def) Planar subdivision: A set of segments (and/or curves), vertices, & the decomposition of the plane into faces induced by them (i.e. PSLG+faces)

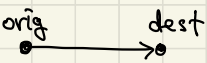
Data structure for subdivisions generally store adjacency info for V/E/F.

Def) Doubly-Connected Edge List (DCEL):

- Each edge represented by two half-edges, called "twins"



- Each half-edge has an <u>origin</u> vertex & <u>destination</u> vertex



- Each face has boundary of half-edges in ccw around face.

- Holes in faces have half-edges in cw order.

DCEL structures (objects):

- A half-edge $\vec{e}$ has fields:

  Twin($\vec{e}$), Origin($\vec{e}$) [Dest($\vec{e}$) = Origin(Twin($\vec{e}$)), not usually stored]

  Next($\vec{e}$), Prev($\vec{e}$) [half-edge around face]

  IncidentFace($\vec{e}$) [to the left of $\vec{e}$]

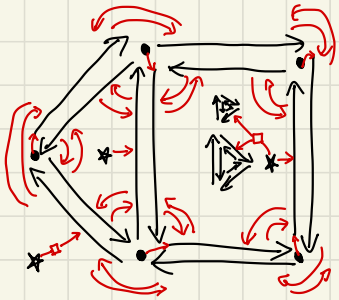
- A vertex v has fields:

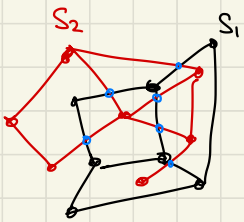
  Coordinates(v), IncidentEdge(v) [<u>one</u> H-E whose origin is v]
  [all edges can be accessed through repeated calls to Twin(Prev($\vec{e}$))]

- A face f has fields:

OuterComponent(f) [one H-E of outer boundary of f]

InnerComponents(f) [(linked) list of one H-E for each hole in f]





How to compute the overlay of two subdivisions S_1, S_2?

Like a PSLG, all intersections must be segment endpoints.

Also, label each face of D with the names of its faces in S_1 & S_2.

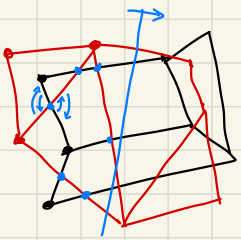
* Assume no overlapping parallel segments (for now)

Algorithm

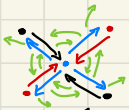
1) Copy all vertices & he structures from S_1 & S_2 into DCEL D.
2) Run sweep-line segment intersection algorithm (with modification!)
   - Each event may trigger modifications to D.
3) Compute face topologies.
4) Compute face identities.

(vertices & he only)

Step 2) Maintain invariant that all of D to left of the sweep line is correct.



- Each "segment" data structure in the status tree has pointers to both half-edges in the DCEL D.

Case 1:  →  [split two edges into four]

Case 2:  →  [next/prev via status before/after]

Case 3:  →  [again, look at the status tree, delete dup v]

Runtime to process an event increases only by a constant factor.

Step 3) Face topologies? Problem is that faces can have holes.

How to match up inner & outer boundaries?

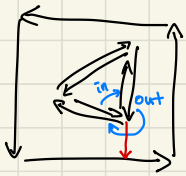
Solution: i) During sweep, note the half-edge below each vertex ($\frac{O(\log n)}{\text{event}}$).

ii) Identify half-edge cycles that are inner boundaries.

After sweep, find the bottom vertex of each he cycle.

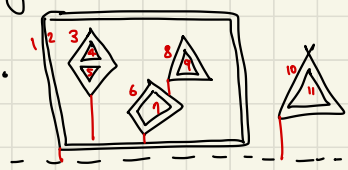
(break ties to the left for equal y-coordinate)

If the cycle makes a right turn at this vertex, it's an inner boundary. (use O-2D) (hole!)

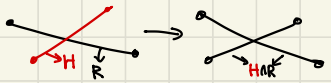


Match inner boundary cycle to the cycle containing the half-edge below the vertex.

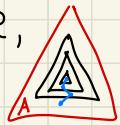
iii) Construct a cycle graph.



* Part ii & iii take $O(h)$ time.

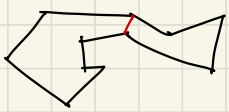
Step 4) Face labeling.  [labels might not be unique!]

Problem: nested holes! use DFS. If a face has a red edge, we know its real label. Propagate red labels by DFS of black edges of output of DCEL. Start with known red label, backtrack o/w.



Total Runtime: $O(h \log h)$ where $h := \#$ of vertices of overlay.

Polygon Triangulations

 $\Rightarrow$ Triangles?

Idea: find a diagonal - line segment in P only intersecting P at its endpoints, which are vertices of P.

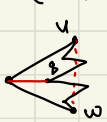
* for today's polygon, multiple colinear vertices are allowed.

Thm) Every polygon with $n > 3$ vertices has a diagonal.

Proof: $v \leftarrow$ lex. minimum vertex. 

Let $u, w \leftarrow$ neighbors of v. If $\overline{uw}$ is a diagonal, we are done.

Check this by testing whether no vertex of P intersects Δuvw (ear).

If not, let q be the vertex in Δuvw farthest from $\overline{uw}$. 

Then, $\overline{vq}$ is a valid diagonal. //

Thm) Every polygon has a triangulation.

Proof: Induct on # of vertices. A diagonal splits P into two polygons with fewer vertices. Base Case ($n=3$). Runtime is $\Theta(n^2)$.

Thm) Every triangulation T of P has $(n-2)$ triangles.

Proof) Induct. Clear when $n=3$. Assume every polygon of $m<n$ vertices has $(m-2)$ triangles. Let e be an interior edge of T. e divides P into polygons with m_1 & m_2 triangles, which have m_1-2 & m_2-2 triangles. $\Rightarrow$ T has $m_1+m_2-4 = n+2-4 = n-2$ trigs. //

$O(n \log n)$ Triangulation

1. Partition P into monotone polygons. [$O(n \log n)$ time]
2. Triangulate them. [$O(n)$ time]

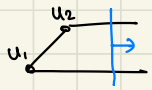
Def) X-monotone Polygon: Made of 2 chains, each with vertices in lexicographic order. ex) 

non-ex) 

Step 2: sweep-line. 

i) merge upper & lower vertices into one sorted list. (like mergesort, $O(n)$).

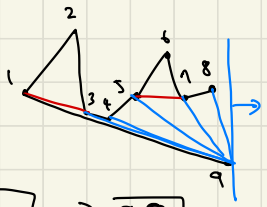
ii) Push u_1, u_2 into stack S.



iii) for $j \leftarrow 3 \dots n$: process(u_j).

process(u_j):

(i) if u_j & top(S) are on different chains:

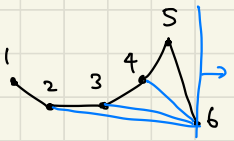


pop all vertices. S [8 7 5 4 3 1] → [9 8]

create diagonal $u_j u_k$ for all popped u_k, except for bottommost one.

push u_{j-1}, u_j.

(ii) if u_j & top(S) are on the same chain:

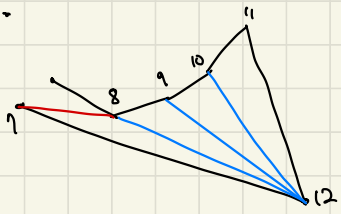


while second(S) is visible from u_j: (Or 2D test!) S [5 4 3 2 1] → [6 2 1]

create diagonal u_j, second(S).

pop top(S).

push u_j.



S [11 10 9 8 7] → ∅

(iii) if $j = n$:

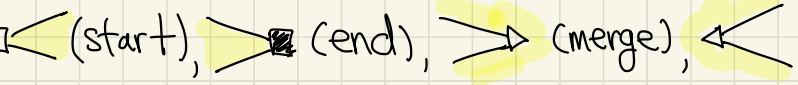
create diagonals to every vertex on stack except top & bottom.

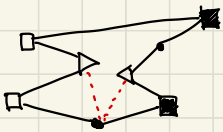
Obs) No vertex is pushed more than twice.

⇒ No vertex is popped more than twice! ⇒ $O(n)$ runtime.

Step 1: Vertex is regular if it has left & right edges.

Otherwise, it is a turn vertex.  categorize by Or2D!

Types:  (start), (end), (merge), (split)

ex) 

Obs) Polygon with no split or merge vertices is x-monotone.

Goal: regularize split & merge vertices by adding diagonals.

Assume vertices are given in ccw order around polygon.

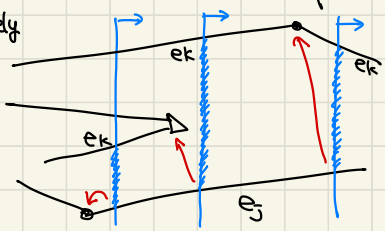
- Determine all vertex types in $O(n)$ time.

Sweep-line algorithm:

- Each vertex is an event. [no heap, just sort.]

- Status tree T only stores edges with polygon's interior above.
- These edges have "helpers".

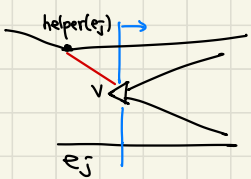
Helper: $\forall e_j$, helper(e_j) := rightmost vertex to left of sweepline that is between or on e_j & e_k ← edge immediately above e_j on sweepline



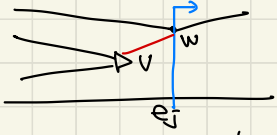
(changes as sweepline moves)

- helper(e_j) is visible from all points on the sweepline between e_j & e_k. ← Invariant!

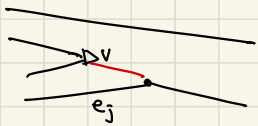
Regularizing split vertices: look up e_j below v in T.
Add diagonal v, helper(e_j).



Regularizing merge vertices: when some helper(e_j) is about to change from v to w, check whether v is a merge vertex. If so, add diagonal $\overline{vw}$.



When e_j is removed from T, check whether v=helper(e_j) is a merge vertex. If so, connect v to endpoint of e_j.



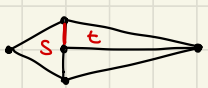
Event handling for v_i: [In Dutch Book, omitted here.]

Runtime: $O(n \log n)$, n vertices, sorting, each event is $O(\log n)$.

Delaunay Triangulation

Def) Triangular Complex: A set T of triangles, edges, & vertices s.t.

- T contains every edge & vertex of every triangle & edge in T.
- $\forall s,t \in T$, $s \cap t$ is empty or an edge/vertex of both s & t.

↳ disallows 

$\rightarrow s \cap t \not\subset s$

Vertex set $V := \{v_1, \ldots, v_n\}$.

A triangular complex T is a <u>triangulation</u> of V if:

- V is the set of vertices in T, and
- T covers conv(V), i.e. $\bigcup_{t \in T} t = \text{conv}(V)$.

↳ every edge of conv(V) is a union of some edges in T. 

Thm) If V is nondegenerate (not all vertices are colinear),
T has $2n-2-v_h$ triangles & $3n-3-v_h$ edges,
where v_h := # of vertices of V on the boundary of conv(V).

Proof: e := # edges in T, f := # faces in T (outside face & f-1 triangles).
Euler's planar graph formula is $n-e+f=2$.
Each triangle has 3 edges; outside face has v_h edges;
each edge is shared by 2 faces.

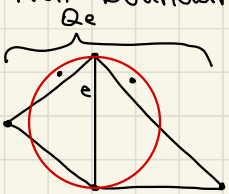
$\Rightarrow e = \frac{3f-3+v_h}{2}$ ← (# face-edge pairs) / (each edge counted twice) $\Rightarrow f = 2n-v_h-1$

$\Rightarrow$ # triangles $= 2n-v_h-2$. $\Rightarrow e = 3n-v_h-3$. //

Def) Circumcircle: circumscribing circle of a triangle. 
↳ <u>empty</u> if it encloses (strictly inside) no point in point set V.
- A triangle is <u>Delaunay</u> if its circumcircle is empty.
- An edge is <u>Delaunay</u> if it has an empty circumcircle. 
- A triangulation of V is <u>Delaunay</u> if all of its triangles are Delaunay.

↳ T is a <u>Delaunay Triangulation</u> of V if it is Delaunay & triangulation of V.

Non-boundary edge e has a containing quadrilateral Q_e.



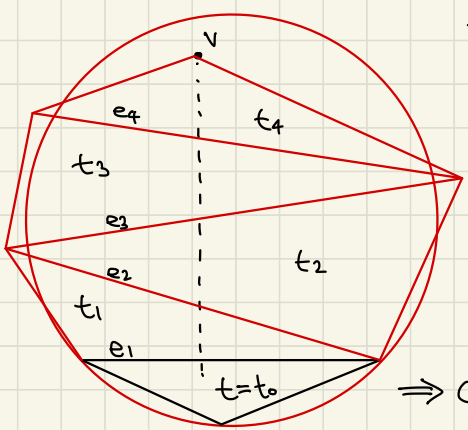
e is <u>locally Delaunay</u> if it has a circumcircle that encloses no vertex of Q_e.

Thm) [A] Every $t \in T$ is Delaunay $\Leftrightarrow$ [B] Every edge of T is Delaunay $\Leftrightarrow$ [C] Every edge of T is locally Delaunay.

Proof: [A] $\Rightarrow$ [B]:  consider the empty circumcircle. trivial. //

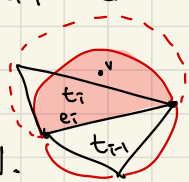
[B] $\rightarrow$ [C]:  consider the empty circumcircle. trivial. //

[C] $\rightarrow$ [A] ("Delaunay Lemma"): Sps for contradiction that some $t \in T$ not Delaunay $\rightarrow$ has vertex $v \in V$ is in t's circumcircle.



Induction on sequence of edges $e_1, \dots, e_j$.

Inductive step: Circumcircle of t_{i-1} encloses v & e_i is locally Delaunay.

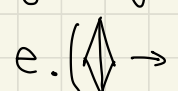


$\rightarrow$ circumcircle of t_i encloses v.

$\Rightarrow$ circumcircle of t_j encloses v <u>and</u> intersect v.

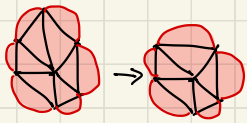
Contradiction. //

The flip algorithm (v1): construct any triangulation of V.

while some edge e is not LD (locally Delaunay), flip e. ()

↳(keep a list of non LD edges)

Note: an unflippable edge is always LD. 

Lemma: A flipped edge never reappears. 

Proof: Union of circumdisks of triangles adjoining a vertex is monotonically shrinking. //

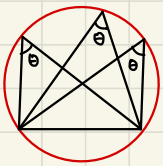
Thm) Flip algorithm terminates after $O(n^2)$ flips.

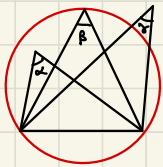
Proof: Only $O(n^2)$ edges defined on n vertices. //

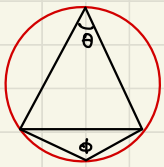
Thm) Every point set V has a DT.

Proof: Every V has a triangulation T. T' ← flip algorithm(T). It terminates.
All edges are LD ⇒ T' is a DT of V. //

Optimality of DT?

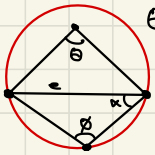
Angles: 

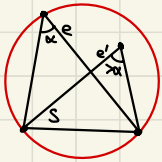




(Thales' Theorem)

Lemma: Flipping an edge e that is not LD increases smallest angle in two affected triangles.

Proof: i) Before flip, smallest angle is not opposite of e. 

ii) For any edges s of Q_e, the angle opposite of s increases after the flip ⇒ even the smallest angle! // 

Def) Angle-Vector: $A(T) :=$ sorted list of the $3 \cdot (\#\text{ triangles})$ angles in T, sorted smallest to largest.

Corollary: Flipping an edge not LD lexicographically increases $A(T)$.

Corollary: If a point set has only one DT, that DT maximizes $A(T)$, compared to all other triangulation of V.

Proof: Every triangulation of V except DT of V can be improved. //

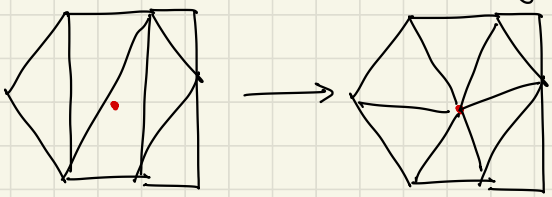
* if points are not cocircular, DT is unique. [explanation in Dutch Book]

Incremental Insertion Algorithm for DT

Let $V := \{v_1, \dots, v_n\}$. $V_i := \{v_1, \dots, v_i\} \subseteq V$ for $i \le n$. $T_i := DT(V_i)$.

Incremental insertion transforms T_{i-1} to T_i by inserting v_i.

ex)

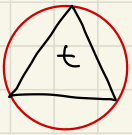


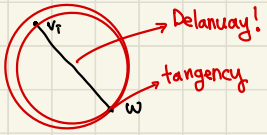
↳ triangles whose circumcircles enclose v_i are no longer Delaunay → must change!

↳ but all others can stay.

Lemma) If t stops being Delaunay when v_i is inserted, for every vertex w of t, $\overline{v_i w}$ is Delaunay.

Proof: t was Delaunay before.



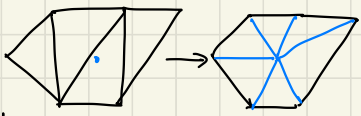


Bowyer-Watson Insertion:

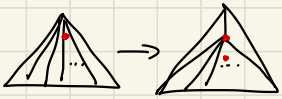
i) Find one triangle whose circumcircle encloses v_i.

ii) Find the m others by DFS in $O(m)$ time.

iii) Delete all of their edges, except the boundary.



iv) Draw new edges from v_i to all vertices that iii deleted their edges.

Runtime: One insertion can take $\Theta(n)$ time. Tight example 

↳ Worst case could be $\Theta(n^2)$ to compute $DT(V)$.

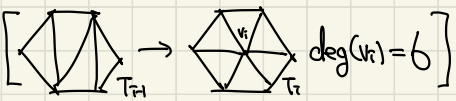
Randomized Incremental Insertion:

- Shuffle vertices in random permutation.
- Insert vertices in that order.

* Each permutation is equally likely. $\Rightarrow O(n \log n)$ expected time.

Lemma) $E[\#$ of structural changes (not counting step i)$] \in O(n)$.

Proof (assume nondegenerate case for now): All $T_1, \ldots, T_n$ are unique.

Structural changes (triangle deletions/creations) to transform $T_{i-1} \to T_i$ are $O(\text{degree}(v_i))$, the degree of v_i in T_i. [

]

Big Idea: Backward Analysis. Analyze a randomized algorithm as if it were running backward in time!

Given T_i, delete a random v_i to delete, yielding T_{i-1}.

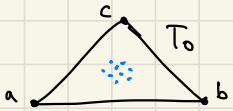
What is the # of structural changes? ⇒ What is the average degree in T_i?

↳ T_i has $\leq 3i-6$ edges, each with 2 endpoints.

→ $E[\text{degree}(v_i)] \leq \frac{6i-12}{i} = 6 - \frac{12}{i}$. → On average, we delete < 6 edges.

Forward in time: On average, inserting v_i creates < 6 edges → < 6 triangles, and deletes < 4 triangles. //

Step (i), point location, is the hard part. To simplify, put V in a huge bounding box.

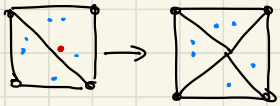


$T_i := DT(\{a, b, c, v_1, \ldots, v_i\})$.

Data Structures:

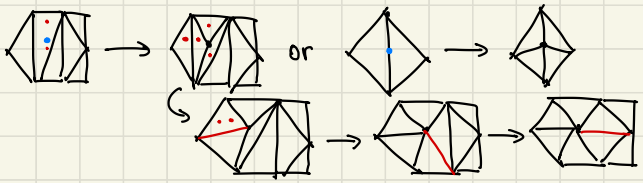
- Each uninserted vertex has a pointer to a triangle that currently contains it.
- Each triangle has a list of uninserted vertices that point to it.

* If a vertex lies on an edge, just choose one to point to.

Each insertion redistributes some lists. 

↳ How to do this quickly?

One way: Lawson's Insertion Alg. (flip version of B-W alg.)

a) Lazy insertion of vertex.  or

b) Flip algorithm.

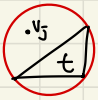
$E[\#\text{ of flips}] \leq 4$ due to the analysis above.
(already accounted for)

Cost per flip is $O(1 + \#\text{ of vertices redistributed})$

$O(1)$ time.

"Vertex Move": transfer of one uninserted vertex from one list to another.

Thm) $E[\#\text{ of Vertex Moves to build } T_n] \in O(n \log n)$



Proof (nondegenerate case): A <u>conflict</u> $\langle v_j, t \rangle$ is a vertex-triangle pair with v_j in t's circumcircle with $t \in T_i$ for some i (clearly $j > i$).

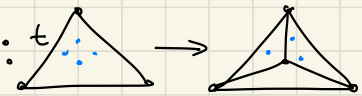
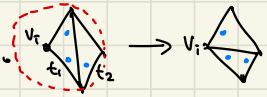
* a triangle created & deleted during one insertion (transient) doesn't count.

C = total # of conflicts created during the algorithm.

= // eliminated //.

= // $\{\begin{matrix}\text{created} \\ \text{eliminated}\end{matrix}\}$ when running the algorithm <u>backwards</u>.

Lemma: Total vertex moves $\leq C$.

Proof:  for each vertex move, a conflict with t is eliminated.  // , a conflict with t_2 is eliminated. //

Backward Analysis: Given T_i, delete a random v_i, yielding T_{i-1}.

RV: X_t (for each $t \in T_i$) := $\mathbb{1}\{t \in T_i \text{ is deleted}\}$. Y_j (for $i \leq j \leq n$) := # of deleted triangles of T_i that v_j conflicts with.

$E[X_t] \leq \frac{3}{i}$ (accounting for bounding box) because t has 3 vertices; t deleted if v_i is one of them.

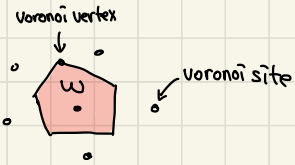
Y_j is sum of expected < 4 of the X_t variables $\rightarrow E[Y_j] < \frac{12}{i}$.

Consider all vertex insertions ... backward, $T_n \rightarrow \ldots \rightarrow T_0$.

# of conflicts eliminated $= C = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[Y_j] \leq \sum_{i=1}^{n} \frac{12n}{i} = 12nH_n \in O(n \log n)$. //

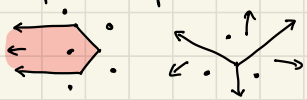
* Lower bound: $\Omega(n \log n)$  reduction from sorting!

Voronoi Diagrams



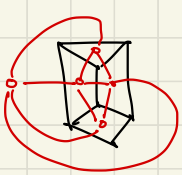
V: set of point sites.

$Vor(w)$: Voronoi cell of site $w \in V := \{p \in E^2 \mid |pw| \leq |pu| \ \forall v \in V\}$.

↳ some of the cells will be unbounded. 

$Vor(V)$: Voronoi Diagram of V, planar subdivision formed by Voronoi cells $\{Vor(w) \mid w \in V\}$, plus their edges & vertices.

Planar Duals



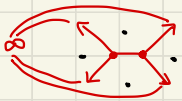
Let X be a planar subdivision.

Let $G(W,E)$ where W contains one node for each face of X.

$(v,w) \in E$ if the corresponding faces share an edge.

If no 4 points of a point set V are cocircular,

DT(V) is the planar dual of Vor(V), and



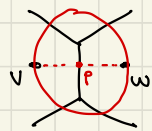
Vor(V) // DT(V) if we include in Vor(V) a "vertex at infinity" where all the infinite rays terminate.

Why??

- If Vor(v) & Vor(w) share edge e, let p be a point on e.

  Closest sites to p are v&w. → $\overline{vw}$ is Delaunay.

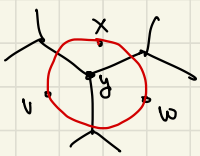
  ↳ Converse argument works too. → Voronoi edge $\overset{\text{dual}}{\Leftrightarrow}$ Delaunay edge.



- If Vor(v), Vor(w), Vor(x) share vertex y, closest sites to y are v, w, x. → $\triangle vwx$ is Delaunay.

  ↳ Converse argument works too. (y is the center of circumcircle.)

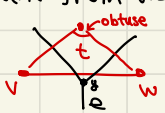
  → Voronoi vertex $\overset{\text{dual}}{\Leftrightarrow}$ Delaunay triangle.



- Obvious property of Vor(V): Voronoi face $\overset{\text{dual}}{\Leftrightarrow}$ vertex of V

All points on Voronoi edge $e = Vor(v) \cap Vor(w)$ are equidistant from v&w.

↳ Voronoi edge e $\perp$ dual Delaunay edge $\overline{vw}$.



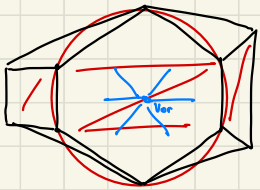
↳ But e might not intersect $\overline{vw}$, because the circumcenter y might be outside.

What if 4+ vertices are cocircular? (The "degenerate case")

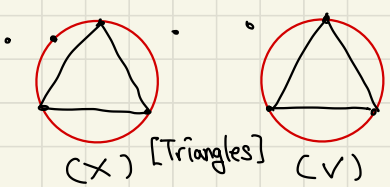
↳ The natural dual of the VD is the Delaunay Subdivision.

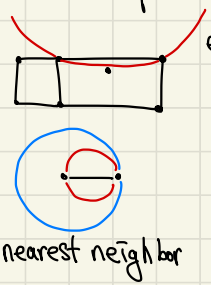
The Delaunay subdivision of V is unique.

DT might not be, but we can triangulate each non-triangles.

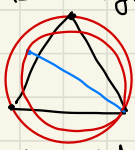


A triangle/edge is <u>Strongly Delaunay</u> if it has a circumcircle with no vertex of inside or on the circle except its own vertices.





every edge on the boundary of conv V is Strongly Delaunay.



→ Every new edge created by inserting v_i is strongly Delaunay in V_i.

Thm) Every DT contains every strongly Delaunay simplex. [Proof omitted]

⇒ Delaunay subdivision contains all strongly Delaunay edges, no others.

Algorithm to compute Vor(V):

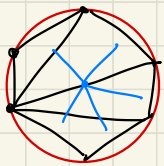
$T \leftarrow DT(V)$ [$O(n \log n)$ exp. time]

$W \leftarrow \{\text{circumcenter}(t) \mid t \in T\} \cup \{w_\infty\}$ ← point at "infinity"

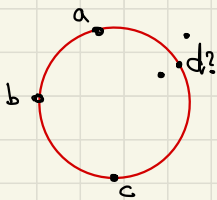
Merge identical circumcenters.

Compute DCEL dual to T, using W as vertices.

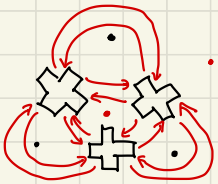
} $O(n)$ time



→ Use $\text{InCircle}(a,b,c,d) = \begin{vmatrix} a_x & a_y & (a_x^2+a_y^2) & 1 \\ b_x & b_y & (b_x^2+b_y^2) & 1 \\ c_x & c_y & (c_x^2+c_y^2) & 1 \\ d_x & d_y & (d_x^2+d_y^2) & 1 \end{vmatrix} = \begin{cases} >0 & \text{if } d \text{ in } (abc) \\ <0 & \text{if } d \text{ out of } (abc) \\ =0 & \text{if } d \text{ on } (abc) \end{cases}$



Guibas & Stolfi: "Quad-edge" data structure represents graph and its planar dual simultaneously. Variation of DCEL.



- Del vertex, Vor face
- Vor vertex, Del face

Allows us to skip dualization step!

Planar Point Location

Input: Planar subdivision S (DCEL), query point(s).

Output: Face(s) containing the query point(s).

Algorithm 1: Check every face ("Point in Polygon"), $\Theta(n)$ time.

Goal: Preprocess (some search structure) S (expensive) so online queries take $o(n)$ time.

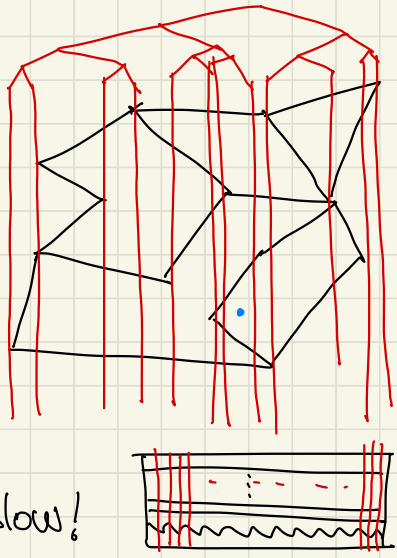
Slab method: Partition S into slabs.

Each slab has ordered list of subsegments.

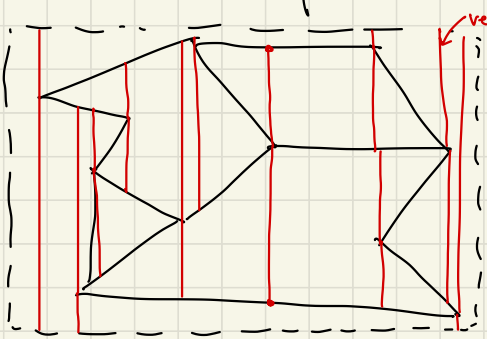
Query: Binary search on x-coordinates, binary search on the slab.

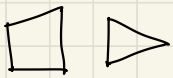
Time: $O(\log n)$ (2 binary searches)

Storage: Up to $\Theta(n^2)$! Preprocessing is also slow!

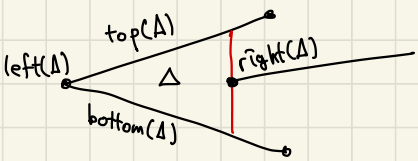


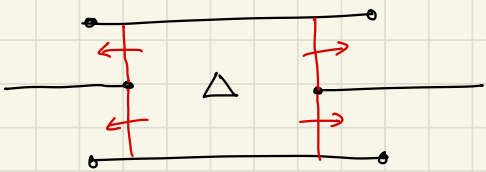
Better idea: Trapezoidal Map T(S) (or "vertical decomposition")



Each trapezoid has one or two vertical sides, and two non-vertical side. 

They are defined by top/bottom segments and the left/right points, and has up to 4 neighbors.





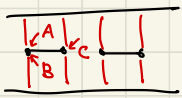
Data structure for a trapezoid stores these 8 things. T(S) is not a DCEL!

Note:  (x)  (v) Lexicographical removal of degeneracy.

Each segment endpoint generates 3 vertices.

T(S) is a planar graph, so linear size.

n segments → $\leq (3n+1)$ trapezoids. (at most 3 charges; leftmost)

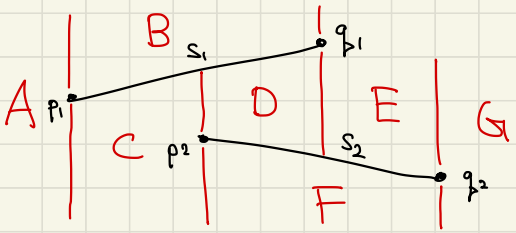


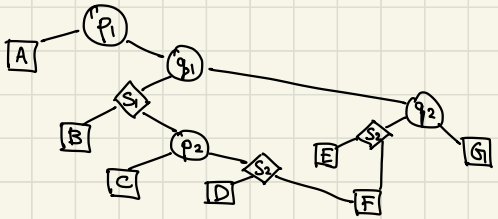
Preprocessing: Random incremental insertion of segments.

- Maintain history DAG.

↳ x-nodes: lexico. comparison with segment endpoint

↳ y-nodes: above/below segment ↳ leaves: trapezoids



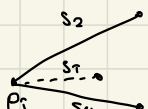


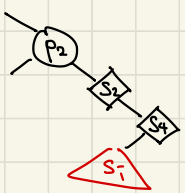
Point location in planar subdivision:

- Find trapezoid Δ containing q via history DAG.
- Δ's top edge has a pointer to the face in the input subdivision.

Inserting segment s_i:

① Locate left endpoint p_i of s_i.

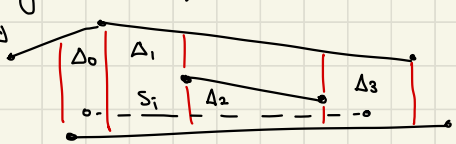




If p_i is already in DAG:

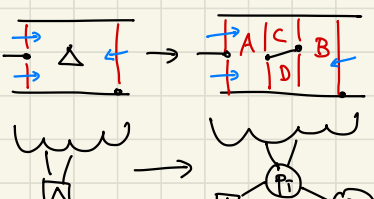
- Go right at x-node with same coordinates.
- At y-node with same left endpoints, "compare slopes"

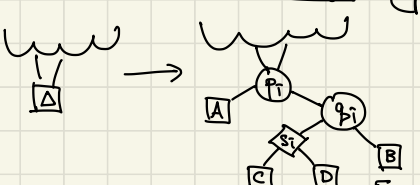
② Trace trapezoids $\Delta_0, \cdots, \Delta_k$ pierced by s_i. $O(k)$ time.



③ Insert s_i.

ex1)



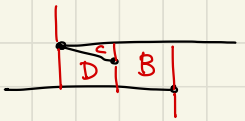


Changes to map:

- Initialize 4 new trapezoids.
- update traps neighboring A & B.

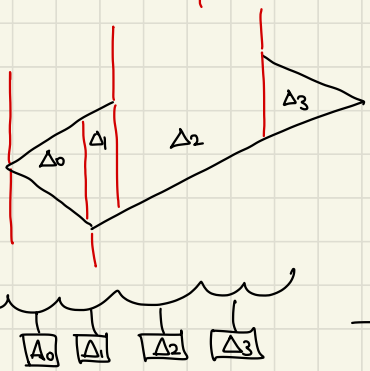
Changes to DAG: update Δ's tree node to this.

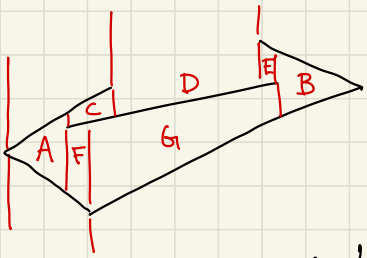
ex2)

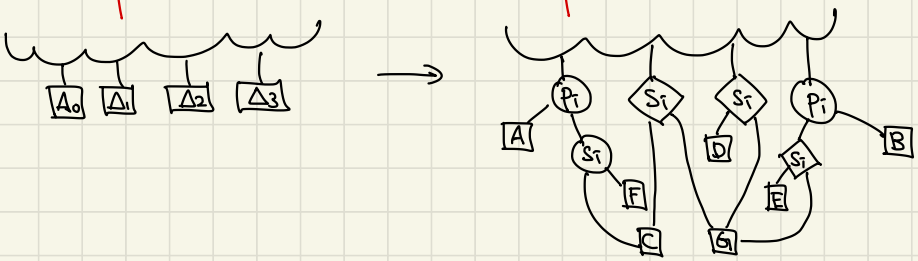


no new (p_i) or $[A]$!

ex3)







- If p_i is new, create (p_i) & $[A]$.
- If q_i is new, create (q_i) & $[B]$. $\Rightarrow O(k)$ time.
- For each $\Delta_0, \ldots, \Delta_k$, create $\langle S_i \rangle$.
- Create new leaves, merging trapezoids where vertical extensions are blocked by S_i.

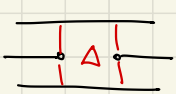
Backward analysis:

① Query time for point q?

Depth of trap containing q is at most 3·(# of changes to q's trapezoid)

Let Δ be a trap containing q is $T(S_i)$. Rip out a segment S_i u.a.r.

Δ disappears only if top(Δ), bottom(Δ), left(Δ), or right(Δ) is removed.



$\rightarrow Pr[\Delta \text{ disappears}] \leq \frac{4}{i}$.

$\rightarrow E[\text{depth of } q \text{ in DAG}_i] \leq \sum_{i=1}^{n} \frac{12}{i} = 12H_n \in O(\log n)$. //

② Size of DAG_i?

$E[\#\text{ of traps ripped out from } S_i] \leq (3i+1)\frac{4}{i} \leq 16$

$\Rightarrow E[\text{size of } DAG_i] \leq \sum_{i=1}^{n} O(16) \in O(n)$. [Worst case is still $\Theta(n^2)$]

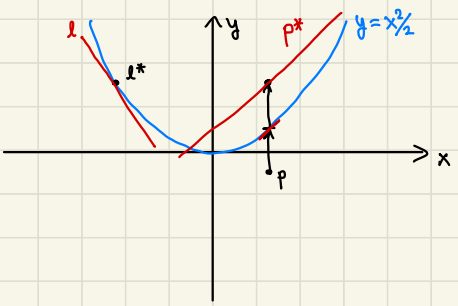
③ Preprocessing time?

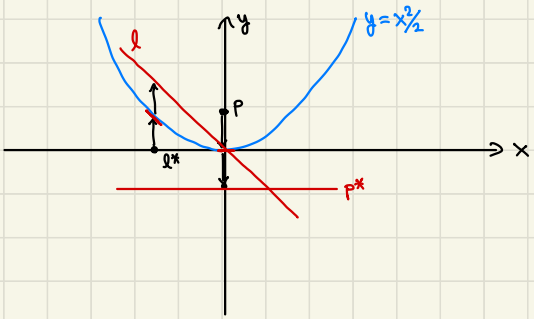
Inserting s_i takes expected $O(\log i)$ (find p_i) $+ O(16)$ (update DAG_i & map) time.

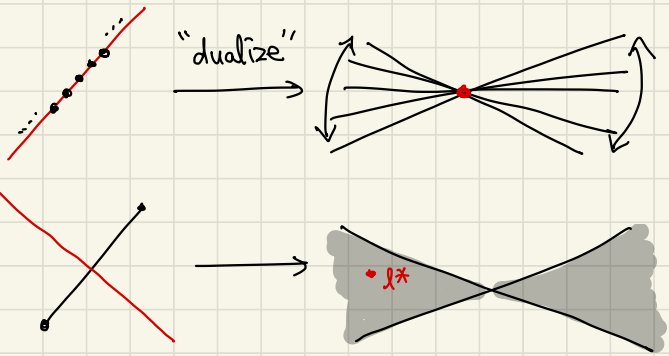
$\Rightarrow E[\text{time to build } T(S_n) + DAG_i] = \sum_{i=1}^{n} O(\log i) = O(n \log n)$

Duality

Primal	Dual
point $p := (p_x, p_y)$	line $p^* := (y = p_x x - p_y)$
line $l := (y = mx + b)$ * no vertical lines!	point $l^* := (m, -b)$
Incidence: l passes p * $p_y = mp_x + b$	Incidence: p^* passes l^* * $-b = p_x m - p_y$
Orientation: p is above l * $p_y > mp_x + b$	Orientation: l^* is above p^* * $-b > p_x m - p_y$

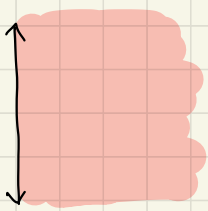


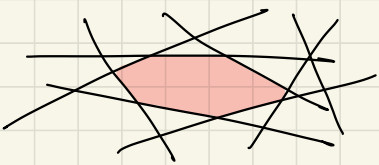




Application: Intersection of half-planes

Def) Half-Plane: a line ∪ set of points on one side.

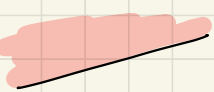


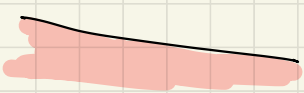
Intersection: 

$H = \{h_1, .., h_n\}$ of hyperplanes.

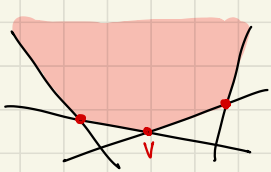
↳ find the intersection (feasible region)

Partition H into $H = L \oplus U \oplus V$.

↳ L := lower bounding HPs. 

↳ U := upper // 

↳ V := vertical HPs. (special case treatment) 

Compute underside first. 

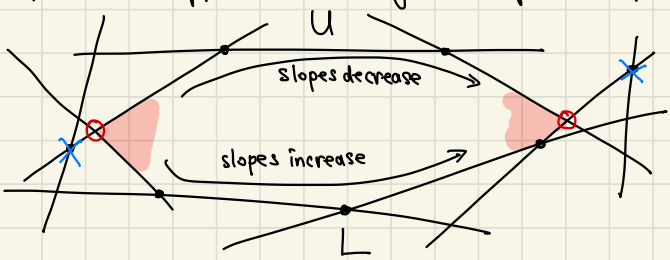
We want each point v where: 2 lines in L intersect v, and no line in L is above v.

↳ Dualize: we want each line v^* where 2 vertices in L^* intersect v^*, and v^* is above no vertex in L^*.

⇒ Lower portion of conv L^*! (Graham scan in $O(n)$)

↳ Dualize the lower hull to get $\bigcap_{h \in L} h^+$ connectivity information.

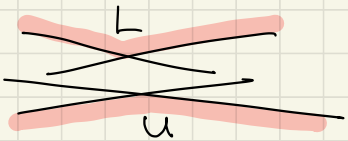
Compute the upper-bounding half-spaces seperately, then merge.



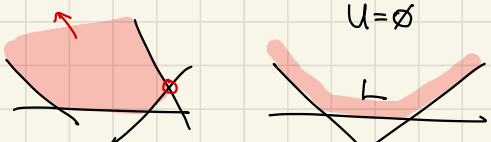
Merge sweeps left to right.

Linear time implementation.

Can be infeasible, $\bigcap_{h \in H} h = \emptyset$.

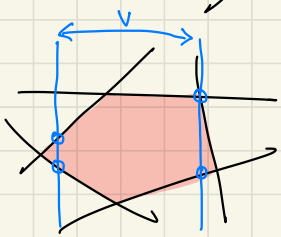


Can be unbounded.



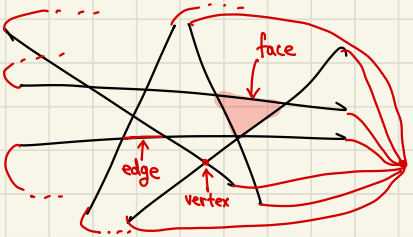
Finally, treat V separately.

↳ Only need 2 lines from V.



Arrangements of Lines

L: set of n lines in plane. → $A(L)$: planar subdivision defined by L. (arrangement)



Has many unbounded faces.

Include a point at infinity.

A(L) is simple if: no parallel lines & no 3 points intersect at a point.

Thm) Complexity of A(L):

- A(L) has $\leq \frac{n(n-1)}{2}$ vertices. (+1 at ∞)
- A(L) has $\leq n^2$ edges.
- A(L) has $\leq \frac{n^2}{2} + \frac{n}{2} + 1$ faces.

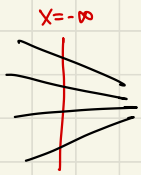
} equality iff A(L) is simple.

Proof:

- There are $\binom{n}{2}$ pairs of lines, each generating at most one vertex.
  Parallel lines or triple intersection $\Leftrightarrow$ reduced # of vertices.
- A line l with v vertices is divided into $(v+1)$ edges.
  At most $(n-1)$ vertices on line $\rightarrow$ at most n edges on one line
  Parallel or triples $\Leftrightarrow$ some line has $< (n-1)$ vertices.
- Faces = edges − vertices + 2 if we count vertex at ∞.
  $\therefore$ simple case $\rightarrow f = n^2/2 + n/2 + 1$ faces.
  Parallel $\rightarrow$ one vertex fewer, two edges fewer $\rightarrow$ one face fewer
  k-way intersection $\rightarrow \frac{k(k-1)}{2}$ vertices fewer, $k^2 - 2k$ edges fewer

$\rightarrow \frac{k^2}{2} - \frac{3k}{2} + 1$ faces fewer, > 0 $\forall k \geq 3$. //

Computing $A(L)$



- Use DCEL.
- Idea 1: Plane sweep line intersection.
  - Lines initially ordered on status tree by decreasing slope.
    Break ties (parallel lines) by increasing y-intercept.
  - Run as usual, then tie loose ends to vertex at ∞.
  - $\rightarrow O(n^2 \log n)$ time.
- Idea 2: Incremental insertion.
  - Insert $l_1, \ldots, l_n$ in order.
  - To insert l_i:

    $v \leftarrow$ vertex at ∞.

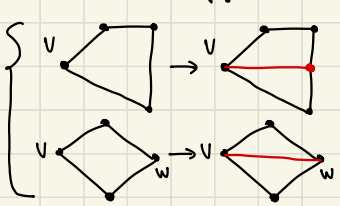
    do:

    determine face to slice; call it f. ← later

    determine edge e or vertex w where v_i leaves f.

    chop face f in two (modify DCEL)

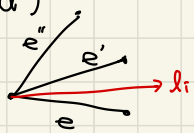
    $v \leftarrow$ vertex where l_i leaves f



while $v \neq$ vertex at ∞.

Determining face to slice:

- Walk l_i from left to right (bottom to top if l_i is vertical)
- From any normal point (not at ∞):



  Look for two consecutive edges e & e' s.t. l_i has slope $> e$ & $< e'$.

- From point at ∞, use slope to determine face.

  Counter clockwise → decreasing slope! (counter intuitive, think 3D?)

  Break ties (parallel lines) by y-intercept (ccw → increasing y-intercept)

  // (vertical lines) by x-coordinates (ccw → decreasing x-coordinates)

Time to insert $l_i \leq$ total complexity of all faces that l_i intersects.

- l_i generally intersects $\Theta(i)$ faces.
- One face can have upto $\Theta(i)$ edges.

↳ Quadratic time to insert one line? The following thm shows it's better.

Thm) Zone Theorem: Let the zone of a line l in arrangement $A(L)$ be the set of faces of $A(L)$ that intersect l. Complexity of the zone is the total # of vertices, edges, and faces in the zone, which is $O(n)$.

Def) Davenport-Schinzel Sequence: $DS(n,s)$ is a string of an alphabet

of n characters such that:

1) no consecutive characters,

2) no alternating sequence of the form $\overbrace{\ldots a \ldots b \ldots}^{2} \overbrace{a \ldots b \ldots}^{s}$ of length $(s+2)$.

eg) $\underline{1}\overline{2}\underline{3}2\underline{4}\overline{2}\underline{1} \in DS(4,2)$, but $\underline{1}2\underline{3}\underline{1}4\underline{3} \notin DS(4,2)$.

Lemma: Every $DS(n,2)$ sequence has $\leq (2n-1)$ characters.

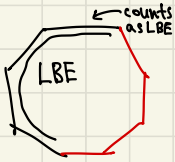
Proof: Let s be a c-char $DS(n,2)$ sequence. Each consecutive pair (a,b) is either the last occurence of a, or the first occurence of b. $(b \ldots \underset{(\neq a)}{ab} \ldots a)$

If former, charge \$1 to a. If latter, charge \$1 to b. Also, charge \$1 each to the first & last chars in s. No char is charged more than \$2 since it gets charged only for its first & last occurunce in s.

→ Total charge is at least $(c-1)$ pairs + \$2 = $\$(c+1) \leq \$2n$. → $c \leq 2n-1$. //

Proof of Zone Thm: Change the coordinates s.t. ℓ is the x-axis.

Left bounding edge (LBE) of a face := edge with a face to the right.

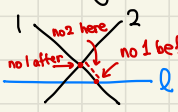


Walk along ℓ, listing all LBEs in ℓ's zone above ℓ.

ℓ 1323145416 1 → $DS(n,2)$ sequence!

⇒ length $< 2n$ by the lemma. Why?

intersecting below/on ℓ (or parallel)

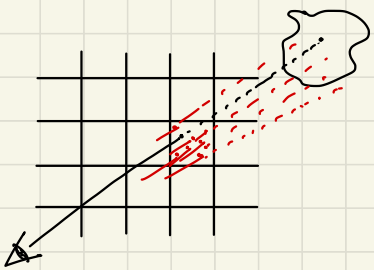


2…2…1…1…1…2…2 is possible, but not 2121. ⇒ This satisfies $DS(n,2)$.

→ (LBE + RBE) × (above l + below l) $< 8n$ for $|E|$. //

Corollary: Incremental Insertion runs in $O(n^2)$ time.

Numeric Integration, Ray Tracing, & Discrepancy



What points should we select to supersample?

↳ Naïvely, selecting grids sounds good.

→ causes discontinuity!

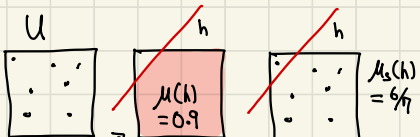
We want "low-discrepancy sequence", e.g. Sobol sequence.

Discrepancy

Let $U := [0,1] \times [0,1]$, unit square, represent a pixel.

Let $S :=$ set of sample points in U.

Let $H :=$ set of <u>all</u> half-planes.



For any $h \in H$, <u>continuous measure</u> $\mu(h)$ and <u>discrete measure</u> $\mu_S(h)$ is ideally equal, but we have the discrepancy $\Delta_h(S) := |\mu(h) - \mu_S(h)|$.

→ Discrepancy of H: $\Delta_H(S) := \sup_{h \in H} \Delta_h(S)$.

* Why sup, not max?  $\Delta_h(S)$ approaches 0.8, but drops to (0.8−0.5).

↳ Consider the greater of closed and open half-plane.

Computing the discrepancy of S: Let h be a half-plane (open or closed) that maximizes $\Delta_H(S)$. Let ℓ be the boundary of h.

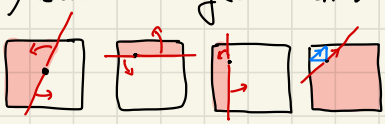
Lemma: ℓ passes through a point in S.  (at least one direction increases $\Delta_h(S)$.)

Lemma: For some maximizer, either:

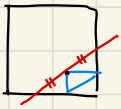
- ℓ passes through 2 points, or
- $\ell \cap [0,1]^2$ is split into two equal segments by a point in S.

Proof: Let h be a maximizer through some $s \in S$. Rotate h about pivot s.

How do $\mu(h)$, $\mu_s(h)$ vary? $\mu_s(h)$: changes when ℓ hits a second point.

$\mu(h)$: varies continuously. 

Algorithm:

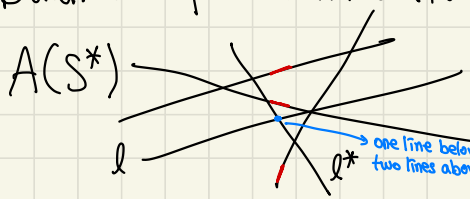
$O(n^3)$
- for each $s \in S$: 
  - $O(n)$
    - $\ell \leftarrow$ "optimal" line through s (construct line // to corner triangle)
    - for each h with boundary ℓ, open/closed, above/below, compute $\Delta_S(h)$.
  - $O(n^2)$
    - for each $s' \in S \setminus \{s\}$:
      - $\ell \leftarrow$ line through s & s'
      - for each h with boundary ℓ, open/closed, above/below, compute $\Delta_S(h)$.

Runtime: $O(n^3)$ time since each $\Delta_S(h)$ computation takes $O(n)$.

Can we do the last part better? → Use duality!

Goal: $\forall s, s' \in S$, count # points in S above/below $\overleftrightarrow{ss'}$.

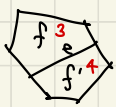
Dualize: $\forall$ pair of lines $\ell, \ell' \in S^*$, count lines in S^* below/above $\ell \cap \ell^*$.



Level of a vertex v in $A(L) :=$ # lines above v.

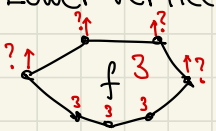
" face f " " " f.

Let f, f' be faces sharing an edge e, with f above f'.



↳ level$(f') =$ level$(f) + 1$ by definition. → Compute all levels by DFS.

Lower vertices of f have the same level as f (except left/rightmost)



But every vertex has a face on top of it. → computable!

New Algorithm (Second half):

Construct $A(S^*)$.

Compute level(f) for some face f (brute force)

Compute levels of all faces by DFS across edges

Label vertices with their levels

Repeat with "upside-down" levels (# lines below v)

all $O(n^2)$

for each $s, s' \in S$:

for each h, open/closed, above/below $\overleftrightarrow{ss'}$, compute $\Delta_h(S)$

$O(1)!!!$ → $O(n^2)$

⇒ Overall runtime improved to $O(n^2)$!

Higher Dimensions

Coordinate axes : $x_1, x_2, \ldots, x_d$.

In E^d, a k-flat is like a k-dimensional subspace, but need not contain the origin.

(a.k.a. affine subspace)

0-flat: point (vertex)

1-flat: line

2-flat: plane

(d-1)-flat: hyperplane, set of x s.t. $n \cdot x = \alpha$ (n: normal vector, α: constant, x: variable point)

d-flat: space (E^d)

Def) Affine Hull (aff S): two equivalent definitions,

① the lowest dimensional flat that includes S,

② the set of all affine combinations of points in set S.

↳ The dimension of S is the dimension of aff S.

Def) Halfspace: one side of a hyperplane (open or closed)

(Convex) Polytopes:

Def) k-simplex: convex hull of (k+1) affinely independent points P. (affinely independent: none is an affine combination of the others)

↳ -1-simplex: $\emptyset$, 0-simplex: vertex, 1-simplex: edge, 2-simplex: triangle ,

3-simplex: tetrahedron  , (d-2)-simplex: ridge, (d-1)-simplex: facet.

Face of a k-simplex is conv P' for any $P' \subseteq P$ (vertices of simplex), including the k-simplex itself.

A d-simplex has $\binom{d+1}{k+1}$ k-faces.

↳ Note that # of k-faces = # of (d-k-1)-faces.

e.g. |vertices| = |facets| (think of tetrahedron), $\lfloor d/2 \rfloor$-faces dominate, $\gg 2^{\lceil d/2 \rceil}$, $\ll (2+\lfloor d/2 \rfloor)^{\lceil d/2 \rceil}$

General Polytopes:

A supporting hyperplane intersects the polytope but not its (relative) interior.

Def) Proper Face: intersection of a polytope with a supporting hyperplane.

↳ vertex, edge, 2-face, ..., ridge ((d-2)-face), facet ((d-1) face)

Def) Face: P, $\emptyset$, or a proper face.

ex) Hypercube: Intersection of halfspaces ($2d$ halfspaces) $x_i \le 1, x_i \ge -1 \ \forall i \in [d]$

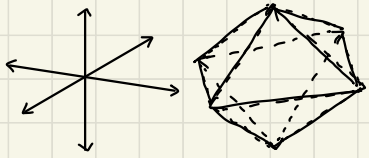
$2d$ facets

↳ point, edge, square, cube, 4-cube, ...

Vertices are $(\pm 1, \pm 1, \ldots, \pm 1)$, 2^d vertices in total!

ex) Cross-polytope: conv $\{e_i, -e_i : i \in [d]\}$ ($2d$ vertices) where e_i is a unit vector along axis x_i.

Faces are conv $\{\pm e_1, \pm e_2, \ldots, \pm e_d\}$, total 2^d facets!



Def) V-polytope: The convex hull of a finite point set (simplex, cross-polytope,...)

Def) H-polytope: The bounded intersection of a finite halfspace set (hypercube)

Thm) Main Theorem of Polytope Theory: Every $\begin{matrix}\text{V-}\\\text{H-}\end{matrix}$ polytope is a $\begin{matrix}\text{H-}\\\text{V-}\end{matrix}$ polytope.

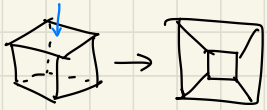
Conversion Algorithms: V→H "convex hull", H→V "halfspace intersection"

Def) H-polyhedron: Like H-polytope, but can be unbounded (from Matousek)

Def) Simple polytope: every vertex has degree d (simplex, hypercube)

Def) Simplicial polytope: every facet is a simplex (simplex, cross-polytope)

Convex polyhedra in 3D:



Edges form a planar graph. Pick some "outer face" and project.

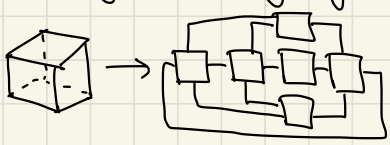
→ O(vertices) complexity, can apply Euler's formula!

Thm) Steinitz Theorem: A graph the edge graph of some convex polyhedron iff it is planar and 3-connected (deleting any 2 vertices doesn't disconnect G)

Representations of polytopes, triangulations, convex subdivisions:

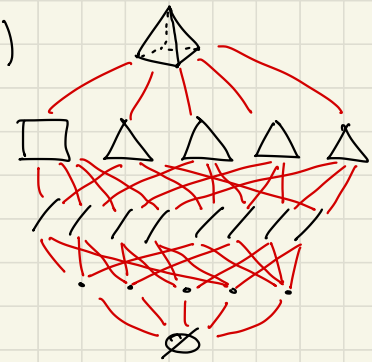
Double description: bipartite graph connecting vertices with facets (polytopes) / d-faces (triangulation/conv. subdiv.).

Facet graph: Double description + edges connecting any two facets/d-faces that share a ridge/facet.



Face lattice: A Hasse diagram, a DAG connecting each k-face with adjoining (k-1) and (k+1)-faces. (complexity is likely exploding)

All of these can be triangulated; divide each face into simplices representing them.



Warnings:

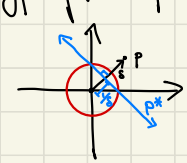
① If not triangulated, double description & facet graph are not complete representations; may need convex hull alg to reconstruct shape of a facet.

② Size of triangulated representation may be superpolynomial in size of DD of facet graph as d increases! e.g. hypercubes

③ Size of face lattice may be superpolynomial compared to DD or facet graph, even if they are triangulated. e.g. simplices have $\geq 2^{\lceil d/2 \rceil} \lfloor d/2 \rfloor$-faces.

↳ Not an issue for $d = 2 \sim 5$.

Polarity (Another Duality)

Primal	Dual
Point p	Hyperplane $p^* := \{x \in E^d \mid x \cdot \vec{p} = 1\}$ ($\vec{p}$: normal vector)
(origin has no dual)	 (hyperplane through origin has no dual)

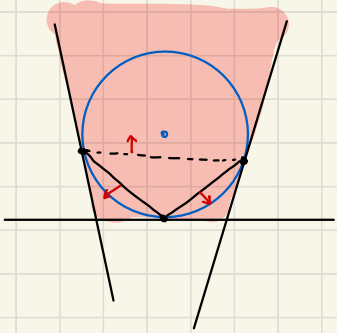
Point $p \in$ hyperplane h	$h^* \in p^*$ (preserves incidence)
p is on same (opposite) side of h as origin	h^* is on same (opposite) side of p^* as origin (preserves orientation)
k-flat • $\text{aff}(\overbrace{p_a, p_b, \ldots}^{\text{points or flats}})$ • $\overbrace{h_a \cap h_b \cap \ldots}^{\text{hp or flats}}$	$(d-1-k)$-flat • $p_a^* \cap p_b^* \cap \ldots$ • $\text{aff}(h_a^*, h_b^*, \ldots)$
k-flat f intersects $(d-1-k)$ flat g [overlaps from origin's viewpoint]	k-flat g^* intersects $(d-1-k)$-flat f^* (incidence, orientation)
V-polytope with origin inside • face lattice	H-polytope with origin inside • same lattice upside-down
Cross-polytope (2^d V, $2d$ F)	Hypercube ($2d$ V, 2^d F)

Convex hull from halfspace intersection alg:

- $c \leftarrow$ centroid(V)
- Translate V s.t. c is at origin
- $H \leftarrow$ dual(V) with all half spaces containing origin
- $P \leftarrow \bigcap_{h \in H} h$ (halfspace intersection black box alg)
- return P^*

Halfspae intersection from convex hull alg:

- Let p be a point inside $\bigcap_{h \in H} h$. (Find via LP)
- Translate H s.t. p is at origin
- $V \leftarrow \text{dual}(H)$
- $P \leftarrow \text{conv}(V)$
- Delete facets of P facing origin
- return P^*



Triangulations of point set $V \in E^d$: A simplicial complex T s.t.

- $\bigcup_{t \in T} t = \text{conv} V$, and
- V is the set of vertices in T.

A tetrahedralization of n points can have $\Omega(n^2)$ tetrahedra.

Each additional 2 dimensions → one more line s.t. orthogonal & offset from others

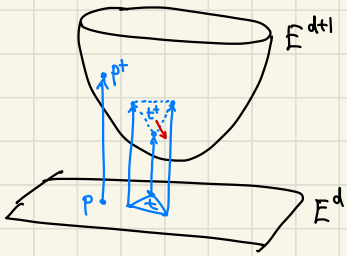
→ $\Omega(n^{\lceil d/2 \rceil})$ d-simplices.

Delaunay Triangulation: $\forall v \in V, t \in DT\, V$, v is not inside the hypersphere that circumscribes t.

(t: d-simplices)

The parabolic lifting map: Lifts points in E^d onto paraboloid in E^{d+1},

$p \mapsto \langle p, \|p\|^2 \rangle = p^+$

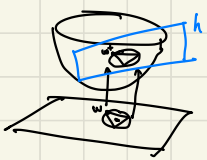


Thm) If $\text{conv}\, V^+$ is simplicial, the projection of its underside to E^d is DT V.

Proof: Let $t \in E^d$ be a d-simplex with vertices $\in V$, circumcenter c, radius r.

t is Delaunay iff $\forall v \in V, \|v-c\|^2 \geq r^2 \Leftrightarrow \|v\|^2 \geq 2c \cdot r + r^2 + \|c\|^2$.

For each vertex w of t, $w_{d+1} = \|w\|^2 = 2c \cdot r + r^2 - \|c\|^2$.



Let h be the hyperplane $x_{d+1} = 2c \cdot (x_1, \ldots, x_d) + r^2 - \|c\|^2$.

$\rightarrow$ each vertex of t^+ is on h.

t^+ lies on the underside of $\text{conv}\, V^+$ iff $\forall v^+ \in V^+$, v^+ is not below h.

$\Leftrightarrow v_{d+1} \geq 2c \cdot v + r^2 - \|c\|^2$.

But $v_{d+1} = \|v\|^2$, so t is Delaunay iff t^+ is an underside. //

Note: If $(d+2)$ points in V lie on a common empty hypersphere, $\text{conv}\, V^+$ is not simplicial, and DT is not unique. Triangulate facets any old way.

Corollary: Every finite point set has a DT.

Corollary: Every convex hull alg. for E^d is a DT alg. for E^{d+1}.

Corollary: The $\Omega(n^2)$ DT in E^3 implies existence of $\Omega(n^2)$-facet polytopes in E^4.

Upper-Bound Theorem for Triangulations (asymptotic)

An n-vertex triangulation T in E^d has $O(n^{\lceil d/2 \rceil})$ simplices.

Proof: Rotate T s.t. no facet is vertical.

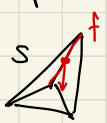
For each d-simplex s:

intersection of k facets of s is a $(d-k)$-face. s has $(d+1)$ facets.

At least $\lceil (d+1)/2 \rceil$ of them either face up or face down.

Let f be the intersection of $\lceil (d+1)/2 \rceil$ of them that face up (or down) (pick majority)

$\rightarrow f$ has dimension $\leq d - \lceil (d+1)/2 \rceil = \lceil d/2 \rceil - 1$.



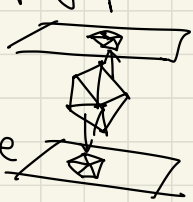
s is the unique simplex directly below (or above) f.

$\therefore$ # of d-simplices in $T \leq 2$(# of $(\lceil d/2 \rceil - 1)$-simplices of T)

$\leq 2 \cdot \binom{n}{\lceil d/2 \rceil} \in O(n^{\lceil d/2 \rceil})$

Number of k-simplices $\leq \begin{cases} 2\binom{d+1}{k+1} n^{\lceil d/2 \rceil} & \text{for } \lceil d/2 \rceil \leq k \leq d \\ n^{k+1} & \text{for } 0 \leq k \leq \lceil d/2 \rceil \end{cases}$

Upper Bound Theorem for Polytopes (asymptotic): an n-vertex polytope $P \subset E^d$ has $O(n^{\lfloor d/2 \rfloor})$ facets.



Proof: project facets to 2 subdivisions of E^{d-1}. Triangulate inner faces (doesn't decrease complexity). Apply $O(n^{\lceil (d+1)/2 \rceil})$ triangulation bound. Observe that $\lceil (d-1)/2 \rceil = \lfloor d/2 \rfloor$. //

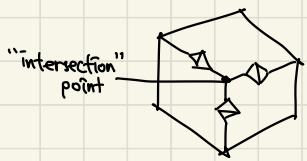
Exact Version [McMullen]: An n-facet polytope $P \subset E^d$ has

$$f_i(P) \leq \sum_{j=0}^{d} \binom{j}{i} \binom{n-1-\max(j,d-j)}{\min(j,d-j)} \quad i\text{-faces.}$$

By polarity, an n-vertex polytope has

$$f_i(P) \leq \sum_{j=0}^{d} \binom{j}{d-1-i} \binom{n-1-\max(j,d-j)}{\min(j,d-j)} \quad i\text{-faces.}$$

Small Dimension LP



Application: "Intersection" of planes that don't quite intersect.

Normal equations of planes: $n_i \cdot X = c_i, 1 \leq i \leq p$, n_i is the unit normal vector.

Idea: minimize distance of v from farthest plane, $d = \max_i |n_i \cdot V - c_i|$.

→ Rewrite as: min d s.t. linear constraints

$\{d \geq n_i \cdot V - c_i, d \geq c_i - n_i \cdot V\}$ $\forall i \in [p]$. This is an LP in 4D.

V has 3 dimensions, and d is a constraint variable.

Def) Linear Program: $X := (x_1, \ldots, x_d)$ variables, $C = (c_1, \ldots, c_d)$ objective vector.

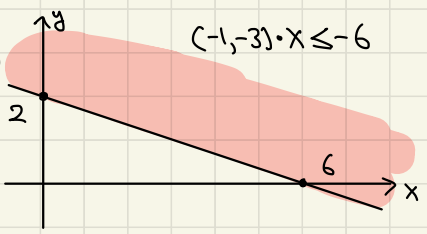
The goal is to maximize $X \cdot C = c_1 x_1 + \cdots + c_d x_d$, the objective function.

There are n constraints of the form $a_i \cdot X \leq b_i$ where $a_i = (a_{i1}, \ldots, a_{id})$, $b_i \in \mathbb{R}$.

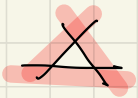
→ Each constraint defines a <u>halfspace</u>.

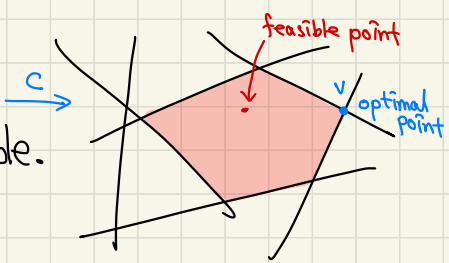
The intersection of all halfspaces is the <u>feasible region</u>, a convex H-polyhedron.

ex)



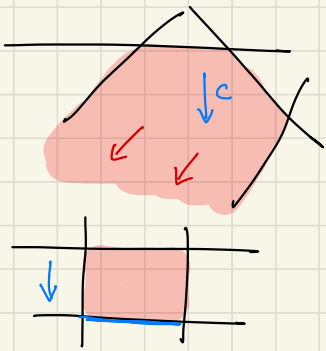
v is extreme in direction c.





If $\bigcap$ of halfspaces is empty, the LP is infeasible.

Feasible but unbounded LP is feasible but has no upper bound on $c \cdot x$. Note that the LP can be bounded even if the feasible region is unbounded.



There can exist multiple optima if $\exists$ a constraint $\perp$ to c.

↳ We will seek the lexicographically maximum solution.

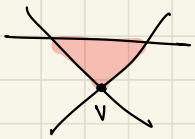
LP Algorithms:

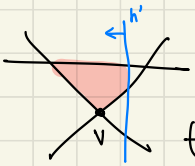
- Dantzig's "Simplex": works good in practice for large d, n, $O(\exp(n))$.
- Kachiyan, Karmarkar: polytime in bit complexity of input coefficients.
- Megiddo & other Gr'ers: algorithms in $O(f(d) \cdot n)$ where $f(d)$ is exponential, but oblivious to bit complexity.

Lemma: Let H be a set of halfspaces. Let c be the objective vector. Let v be the solution to $LP(c, H)$, the lex. maximum optima of $c \cdot v$ in $\bigcap_{h \in H} h$. Let h' be another halfspace. [assume adding $H \cup \{h'\}$ is feasible, for now]

(i) if $v \in h'$, then v solves $LP(c, H \cup \{h'\})$.

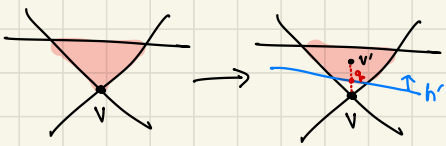
(ii) if $v \notin h'$, then solution to $LP(c, H \cup \{h'\})$ lies on the boundary of h'.

Proof Sketch: (i) 

→ 

$\bigcap_{h \in H \cup \{h'\}} h \subseteq \bigcap_{h \in H} h$, so v is still the best solution to $c \cdot v$. //

(ii) Suppose new optimum v' is <u>not</u> on the boundary of h'.

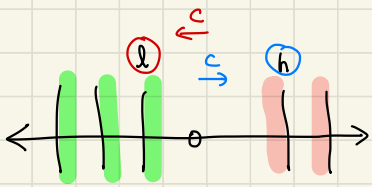


Let $q := \overline{vv'} \cap \text{boundary}(h')$. $q \neq v'$.

Either $c \cdot v > c \cdot v'$, so $c \cdot q > c \cdot v'$. #.

OR, $c \cdot v = c \cdot v'$ and $v \succ v'$, so $c \cdot q = c \cdot v'$ and $q \succ v'$, so v is not optimal. #//

Sidel's LP Alg: SolveLP(d, c, H).



If $d = 1$:

$h \leftarrow \min\{b/a \mid \overbrace{(a,b)}^{a \cdot x \leq b} \in H, a > 0\}$

$l \leftarrow \max\{b/a \mid (a,b) \in H, a < 0\}$

$z \leftarrow \min\{b \mid (a,b) \in H, a = 0\}$ $[0 \leq b]$

if $(h < l)$ or $(z < 0)$, output INFEASIBLE and terminate.

else if $c \geq 0$, return h. [also includes $c = 0$, then find lex. max, h]

else, return l.

ELSE, $d > 1$:

if $|H| = d$, return intersection of bounding hyperplanes in H.

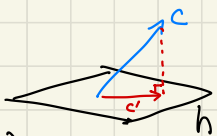
[in other words, solve $Ax=b$ where $A=\begin{bmatrix} - & a_1 & - \\ & \vdots & \\ - & a_d & - \end{bmatrix}$ and $b=\begin{bmatrix} b_1 \\ \vdots \\ b_d \end{bmatrix}$.]

else, choose random $h \in H$.

$v \leftarrow \text{SolveLP}(d, c, H \setminus \{h\})$.

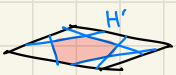
if $v \in h$, return v. [Lemma, case (i)]

else,



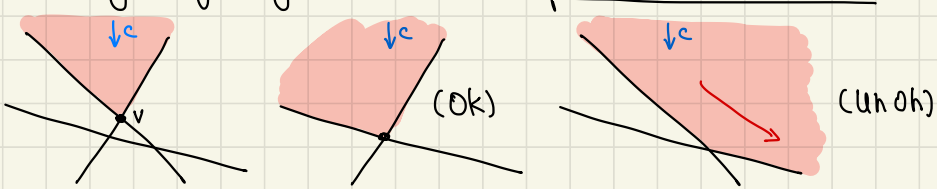
$c' \leftarrow$ orthogonal projection of c onto boundary(h).

$H' \leftarrow \{h' \cap \text{boundary}(h) \mid h' \in H \setminus \{h\}\}$.



return $\text{SolveLP}(d-1, c', H')$. [Lemma, case (ii)]

Warning: Alg only works if <u>no subproblem is unbounded</u>, very unlikely.



Big Idea: Most constraints don't affect the final answer; typically only d constraints are actively involved.

Backward analysis: Let $n=|H|$. Let $w \leftarrow \text{LP}(c, H)$, $v \leftarrow \text{LP}(c, H \setminus \{h\})$, with h chosen randomly. $\Pr[v \notin h]$?

Fact: $\exists$ subset $I \subseteq H$ with $|I|=d$ s.t. w solves $\text{LP}(c, I)$.

Such I is called the "basis" for w.

↳ (c is a linear combination of normal vectors of I) of nonneg. coefficients

If $h \notin I$, removing h from H cannot improve the optimum, so $w = v$.

If $h \in I$, removing h might improve the maximum.

$\Rightarrow \Pr[v \notin h] = \Pr[v \neq w] \leq \frac{d}{m}$.

Expected runtime for SolveLP(d, c, H):

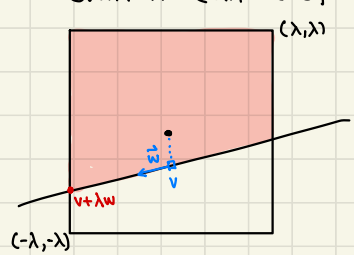
$$T(d,m) \leq \begin{cases} O(m) & \text{if } d=1 \\ O(d^3) & \text{if } m=d \\ \underbrace{T(d,m-1)}_{\text{first recursive call}} + \underbrace{O(d)}_{\text{test } v \in h} + \frac{d}{m}\left[\underbrace{O(dm)}_{\text{project } c,H \text{ s.t. } h \text{ down to } h} + \underbrace{T(d-1,m-1)}_{\text{otherwise}}\right] \end{cases}$$

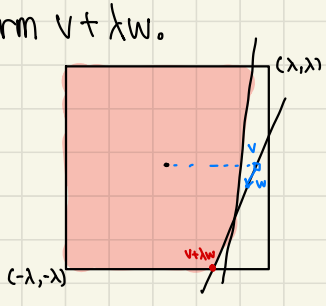
Verify by substitution that $T(d,m) \leq 2d! \cdot m \sum_{i=1}^{d} \frac{i}{(i-1)!}$ works.

The summation converges, so $T(d,m) \in O(d! \cdot m)$, linear in m. //

Unbounded LPs

- Add extra constraints $-\lambda \leq x_i \leq \lambda \ \forall i \in [d]$.
- λ is a "symbolic constant" (i.e. unknown value), some very large $\to \infty$
- Constraints are present in every subproblem.
- Solutions can be of the form $v + \lambda w$.

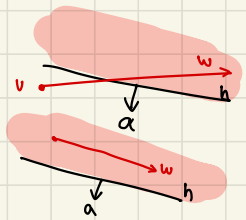




- Solution "$v+\lambda w$" means that for some λ_0, the ray $\{v+\lambda w \mid \lambda \geq \lambda_0\}$ is in the feasible region of the original LP (without the box).

- Solution $v+\lambda w$ is "in" halfspace h with normal $\vec{a}$ if

  - $w \cdot a < 0 \Rightarrow$ w "shoots into" the halfspace", good.
  - $w \cdot a = 0 \Rightarrow$ also requires that $v \in h$ as well.



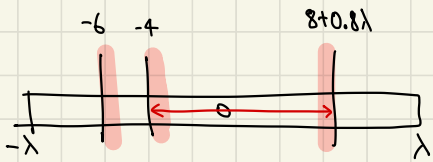
- Lexicographic comparison: $p+\lambda q <_L v+\lambda w \iff q \prec w$, or $q = w$ and $p \prec v$. [For this and above, assume $\lambda > 0$. If $\lambda = 0$, resort to original.]

Define $\min_L$, $\max_L$ lexicographically.

- Constraints: $a_i \cdot x \leq_L b_i + \lambda \cdot e_i$ where $a_i \in \mathbb{R}^d$, $b_i, e_i \in \mathbb{R}$.

All input constraints have $e_i = 0$, but subproblems...?

Solve LP(d, c, H): ← not the bounding box!



If $d = 1$: [Base Case]

$h+h'\lambda \leftarrow \min_L \{\{b_a+\lambda e_a \mid \overbrace{(a,b,e)}^{ax \leq b+\lambda e} \in H, a>0\} \cup \{0+\lambda \cdot 1\}\}$.

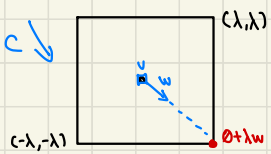
$l+l'\lambda \leftarrow \max_L \{\{b_a+\lambda e_a \mid (a,b,e) \in H, a<0\} \cup \{0-\lambda \cdot 1\}\}$.

$z+z'\lambda \leftarrow \min_L \{b+\lambda e \mid (a,b,e) \in H, a=0\}$

if $(z+z'\lambda <_L 0+0\cdot\lambda)$ or $(h+h'\lambda <_L l+l'\lambda)$, output INFEASIBLE and terminate.

else if $c \geq 0$ return $h+h'\lambda$, else return $l+l'h$.

If $d>1$: [General Case]



if $H=\emptyset$:

$v \leftarrow \vec{0}^d$, $w \leftarrow (\vec{w_i})^d$ where $w_i = \begin{cases} +1, & c_i \geq 0 \\ -1, & c_i < 0 \end{cases}$. return $v+\lambda w$.

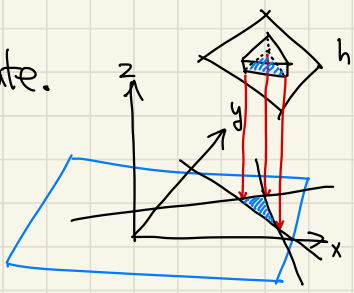
else, choose a random $h \in H$.

$v+\lambda w \leftarrow \text{SolveLP}(d, c, H \setminus \{h\})$.

Express h as $a \cdot x \leq b + \lambda e$.

If $a \cdot v + \lambda a \cdot w \leq_L b + \lambda e$, return $v+\lambda w$. [if $v+\lambda w \in h$]

Else, if $a = \vec{0}$, report INFEASIBLE and terminate.

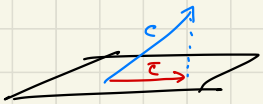


Else, solution lies on boundary(h)...

{Reduce the dimension/Gaussian Elimination:

- Project h orthogonally onto hyperplane defined by $(d-1)$ axes;
  i.e. just drop one coordinate somehow.
  [for lex. ordering, but robustness...?]
- Let a be normal to h; let k be the largest index s.t. $w/a_k \neq \emptyset$.
- Drop coordinate k.

$k \leftarrow \max_k \{k \mid w/a_k \neq \emptyset\}$.



$\overline{H} \leftarrow \{(a' - \frac{a'_k}{a_k} a \text{ with } k\text{-th coordinate removed}, b' - \frac{a'_k}{a_k} b, e' - \frac{a'_k}{a_k} e) \mid (a', b', e') \in H \setminus \{h\} \text{ where } h := (a, b, e)\}$. [Gaussian Elimination]

$\bar{c} \leftarrow c - \frac{c_k}{a_k} a$ with k-th coordinate removed.

{Incorporate constraints $-\lambda \le x_k \le \lambda$} $\Rightarrow a' = (0 \dots \overset{[k]}{1} \dots 0), b' = 0, e' = \pm 1$.

$\bar{H} \leftarrow \bar{H} \cup \{(\frac{\pm 1}{a_k} a$ with k-th coordinate removed$, \pm\frac{b}{a_k}, (\pm\frac{e}{a_k})\}$.

$\bar{v} + \lambda \bar{w} \leftarrow$ SolveLP$(d-1, \bar{c}, \bar{H})$.

{Lift solution}

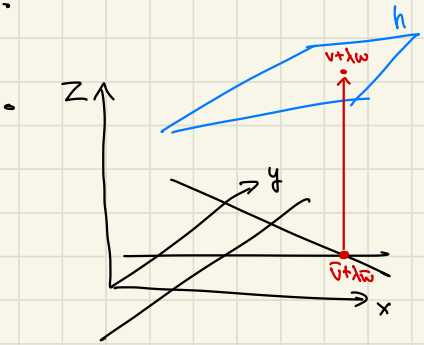
$v \leftarrow \bar{v}$ with zero inserted as k-th coordinate.

$w \leftarrow \bar{w}$ //

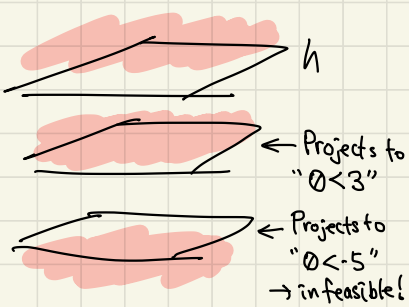
$v_k \leftarrow \frac{1}{a_k}(b - a \cdot v)$.

$w_k \leftarrow \frac{1}{a_k}(e - a \cdot v)$.

return $v + \lambda w$.



"Projecting down" parallel planes:
could make trivial constraints or conclude infeasability, depending on direction.



For numerical stability:

- Choose least distorting projection, $k \leftarrow \arg\max_k |a_i|$
- To resolve multiple solutions, use auxilliary objective vectors, $c_1 = (1, 0, \dots, 0), c_2 = (0, 1, 0, \dots, 0), \dots$ and map them down too. Use in BC.

High Dimensional Convex Hull

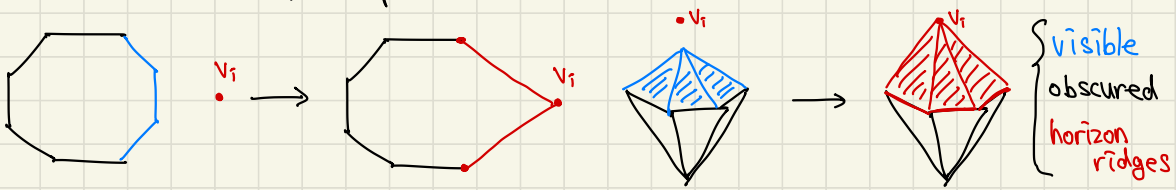
Let $V := \{v_1, \ldots, v_n\}$, $V_i := \{v_1, \ldots, v_i\}$, $P_i := \text{conv}(V_i)$.

Suppose no $d+1$ vertices are co-hyperplanar. $\Rightarrow P_i$ is simplicial.

Each P_i is represented by a facet graph.

Algorithm: Compute $P_{d+1} = \text{conv}(V_{d+1})$ [d-simplex, $O(1)$ time]

for $i \leftarrow d+2 \ldots n$, compute $P_i = \text{conv}(P_{i-1} \cup \{v_i\})$ [vertex insertion]



Computing $\text{conv}(P_{i-1} \cup \{v_i\})$:

(a) Find one facet of P_{i-1} visible from v_i. (If $v_i \in P_{i-1}$, we are done.)

(b) Find all facets visible from v_i by DFS in facet graph.

(c) Identify horizon ridges.

(d) Delete all visible facets.

} $O(\#\text{ deleted facets})$

(e) For each horizon ridge r, create facet $\text{conv}(r \cup \{v_i\})$. (node in facet graph)

(f) Generate new edges of facet graph (ridges of P_i).

$\text{conv}(r \cup \{v_i\})$ has d ridges, one shared with an obscured facet.

The other $(d-1)$ ridges are matched with other new facet ridges using a hash table. (Hash each ridge by its vertices)

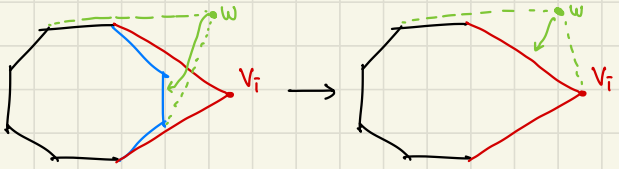
} $O(\#\text{ new facets created})$

Over all iterations: # facets deleted $\leq$ # facets created.

Step (a): Use conflict graph.

- Each uninserted vertex has a pointer to a visible facet.
- Each facet has a list of vertices that point to it.
- When a facet is deleted, redistribute its vertices to new facets.

↳ If vertex w can see a new facet, w can see its old ridge.



- To redistribute w, DFS facets visible from both v_i & w.

  Find a horizon ridge whose new facet is visible from w.

  If none exists, $w \in P_i$, so throw w away.

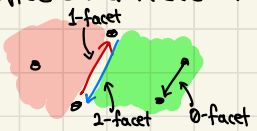
↳ time $\in O($(# of w's visibilities deleted$)$

Def) Visibility: $\langle w, f \rangle$ pair with facet f visible from uninserted w.

Over all iterations: time $\in O($total # of visibilities deleted$)$

$= O($ // created$)$

Def) j-facet: oriented facet whose vertices are in V that has j vertices of V on its "front" side.



- All facets of $\text{conv}(V)$ are 0-facets.
- When a j-facet is deleted, j visibilities are deleted.
- The d vertices of a j-facet are its "triggers".
- The j vertices in front are its "stoppers". $\rightarrow$ $\langle$stopper, j-facet$\rangle$ is a visibility.
- Appears during incremental insertion iff all triggers inserted before any stoppers.
- If $\langle v_1, \ldots, v_n \rangle$ is a random permutation of V, it appears w.p. $p_j = \frac{1}{\binom{j+d}{d}} < \frac{d!}{j^d}$.

Def) $\leq j$-facet: facet that has $\leq j$ vertices on its front side.

Let $f_j := $ total # of j-facets, $F_j := $ total # of $\leq j$-facets $= \sum_{i=0}^{j} f_i$.

$f_0 = F_0 = |\text{conv}(V)| \in O(n^{\lfloor d/2 \rfloor})$.

Lemma: $F_j \in O(j^{\lceil d/2 \rceil} n^{\lfloor d/2 \rfloor})$ for $j \geq 1$.

Proof: Choose random subset $R \subseteq V$. Each vertex chosen w.p. $1/j$.

Let $r := |R|$ (random variable $\sim Bi(n, 1/j)$).

$\Pr[\text{particular } k\text{-facet appears in } \text{conv}(R)] = q_k = \overbrace{\left(\frac{1}{j}\right)^d}^{\text{choose } d \text{ triggers}} \overbrace{\left(1-\frac{1}{j}\right)^k}^{\text{no } k \text{ stoppers}}$.

If $k \leq j$, $q_k \geq \left(\frac{1}{j}\right)^d \left(1-\frac{1}{j}\right)^j \geq \left(\frac{1}{j}\right)^d \frac{1}{e}$. errata?

$E[\text{size}(\text{conv}(R))] = \sum_{k=0}^{n} q_k f_k \geq \sum_{k=0}^{j} q_k f_k \geq \left(\frac{1}{j}\right)^d \frac{1}{e} \sum_{k=0}^{j} f_k = \left(\frac{1}{j}\right)^d \frac{1}{e} F_j$.

But also, $\in O(E[r^{\lfloor d/2 \rfloor}]) = O(E[r]^{\lfloor d/2 \rfloor}) = O((n/j)^{\lfloor d/2 \rfloor}) \Rightarrow F_j \in O(j^{\lceil d/2 \rceil} n^{\lfloor d/2 \rfloor})$. //

$\rightarrow E[\text{runtime}] \in O(E[\text{total \# facets created}] + E[\text{total \# visibilities created}])$

$= O\left(\sum_{j=0}^{n} p_j f_j + \sum_{j=1}^{n} j\, p_j f_j\right) \subseteq O\left(n^{\lfloor d/2 \rfloor} + \sum_{j=1}^{n} j\, p_j f_j\right)$. [separating $j=0$ case]

$\sum_{j=1}^{n} j p_j f_j < \sum_{j=1}^{n} \frac{d!}{j^{(d-1)}}\left(F_j - F_{j-1}\right) = d!\left[\sum_{j=1}^{n-1} \frac{F_j}{j^{(d-1)}} + \frac{F_n}{n^{d-1}} - \sum_{k=2}^{n} \frac{F_{k-1}}{k^{(d-1)}} - F_0\right]$ $\quad \binom{n}{d}$

$= d!\left[\sum_{j=1}^{n-1}\left(\frac{F_j}{j^{(d-1)}} - \frac{F_j}{(j+1)^{(d-1)}}\right) + \frac{F_n}{n^{(d-1)}} + F_0\right] \in O\left(\sum_{j=1}^{n-1} \frac{F_j}{j^{(d-1)}(j+1)^{(d-1)}} \overbrace{\left[(j+1)^{(d-1)} - j^{(d-1)}\right]}^{O(j^{(d-2)})} + n\right)$ $\quad [F_n \in O(n^d)]$

$= O\left(\sum_{j=1}^{n-1} \frac{j^{\lceil d/2 \rceil} n^{\lfloor d/2 \rfloor}}{j^{2d-2)}} \cdot j^{(d-2)} + n\right) = O\left(n^{\lfloor d/2 \rfloor} \sum_{j=1}^{n-1} \frac{1}{j^{\lfloor d/2 \rfloor}}\right) = \begin{cases} O(n^{\lfloor d/2 \rfloor}) \text{ if } d \geq 2, \\ O(n \log n) \text{ if } d = 2, 3. \end{cases}$

* Worst-case optimal for fixed dimension d!... but exponential in d.

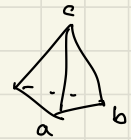
Special case: suppose $\forall R \subseteq V$, $\text{conv}(R)$ has $O(|R|)$ facets. (common in practice)

Then, $F_j \in O(j^{(d-1)} n)$, so exp. runtime $\in O(n \sum_{j=1}^{n} \frac{1}{j}) = O(n \log n)$ $\forall$ dimension!

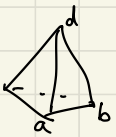
3D Orientation

Two ways to orient tetrahedron vertex labels:

positive orientation



negative orientation



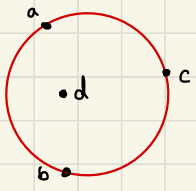
Orientation test: $\text{Orient3D}(a,b,c,d) = \det \begin{bmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ d_x & d_y & d_z & 1 \end{bmatrix} = \det \begin{bmatrix} a_x - d_x & a_y - d_y & a_z - d_z \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{bmatrix}$ better numerically

↳ + sign $\Rightarrow$ pos. orientation, − sign $\Rightarrow$ neg. orientation

↳ 0 $\Rightarrow$ points are coplanar.

pos. ori. $\rightarrow$ facet $\triangle bcd$ is visible from a, where bcd are in ccw order to observer outside the polyhedron.

$\text{InCircle}(a,b,c,d) = \text{Orient3D}(a^+, b^+, c^+, d^+)$ (recall $a^+ := \langle a, \|a\|^2 \rangle$)



$$= \det \begin{bmatrix} a_x - d_x & a_y - d_y & (a_x-d_x)^2 + (a_y-d_y)^2 \\ b_x - d_x & b_y - d_y & (b_x-d_x)^2 + (b_y-d_y)^2 \\ c_x - d_x & c_y - d_y & (c_x-d_x)^2 + (c_y-d_y)^2 \end{bmatrix}$$

→ extra column transmormation for better accuracy

*(a,b,c) must be in ccw order, else sign is reversed!

+ sign ⇒ d inside, − sign ⇒ d outside, 0 ⇒ d on circle.

Voronoi Diagrams in E^d

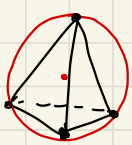
Voronoi Cells: $\text{Vor } w = \{p \in E^d \mid \overline{|pw|} < |pv| \ \forall v \in V\}$. Each cell is convex.

(length of line segment)

Voronoi Diagram: Vor V is a polyhedron complex in E^d (aka convex subdivision of E^d) containing every Vor w and all of its faces.

Constructing Vor V:

- Construct DT(V).
- For each d-simplex $s \in DT(V)$, compute its circumcenter, the center of its circumscribing hypersphere.
- Construct face lattice of DT(V); turn it upside down.



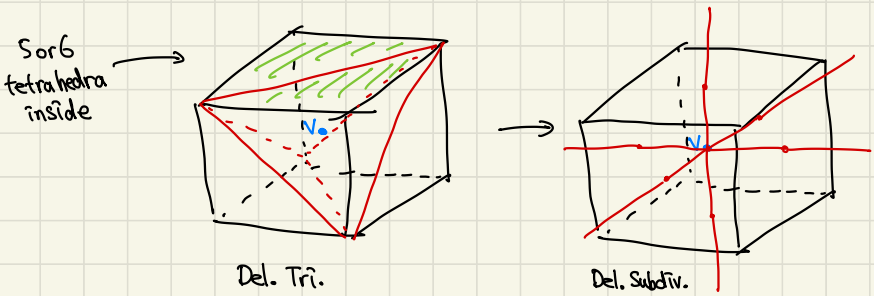
↳ d-simplex dualizes to circumcenter = vertex of Vor V.

↳ facet of DT(V) dualizes to edge.

↳ edge // facet.

↳ vertex // cell Vor(V). (sort of, need clean up)

- Merge equal Voronoi vertices. [really use InSphere to test]
- Merge any two faces if the faces of same dimension (1...(d-1))
  if they are faces of exactly the same subset of Delaunay cells.

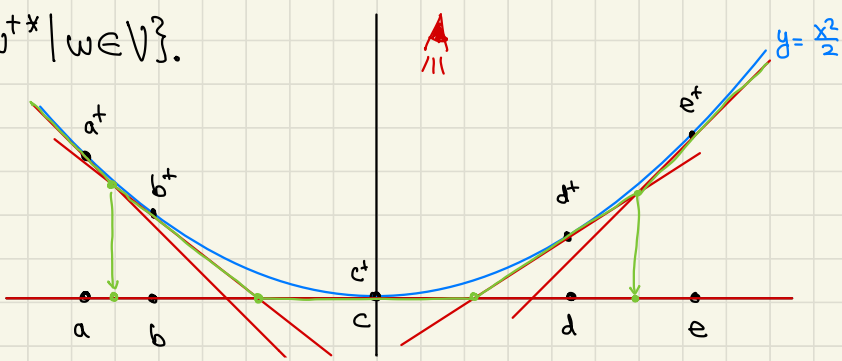


Voronoi Diagrams & The Lifting Map

Given vertex set V, let V^+ be the vertices lifted onto paraboloid $x_{d+1} = \frac{1}{2}\|v\|^2$.

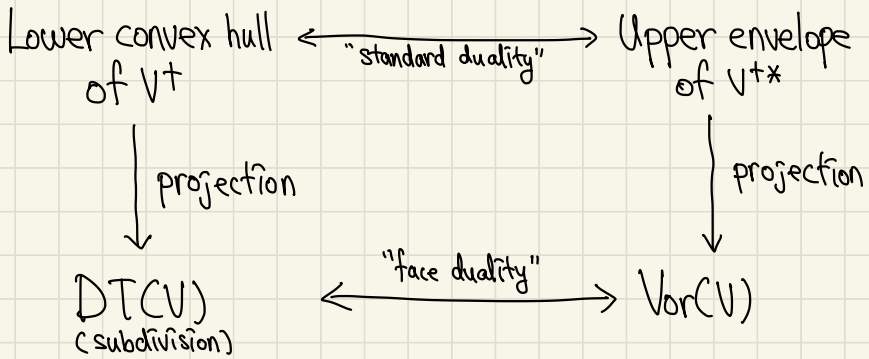
Duality: vertex $w = (w_1, \ldots, w_k)$ dualizes to hyperplane $x_k = w_1 x_1 + \cdots + w_{k-1} x_{k-1} - w_k$.

Let $H := \{w^{+*} \mid w \in V\}$.



Let the upper envelope U of H be the faces of arrangement A(H) of level 0 (i.e. no hyperplane above them). U projected down to $E^d \to \text{Vor } V$.

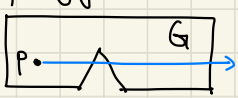
Connections between objects:



Point in Polygon

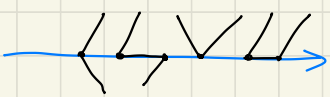
Point location without preprocessing?

Query: Is point p in polygon G? [inside, outside, on boundary]

Shoot ray to right. 

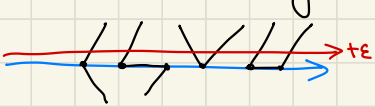
↳ Odd # crossings ⇒ inside, Even # ⇒ outside (if not on boundary)

Linear time (check every edge)... but degeneracy!

 Strategy: Symbolic Perturbation.

Shoot ray to the right of $(p_x, p_y+\varepsilon)$ where $\varepsilon > 0$ and infinitesimally small.

↳ for a sufficiently small ε, the combinatorial value doesn't change!



Then, the ray never passes a vertex.

Requires modification to the intersection test:

any vertex with y-coordinate p_y are now "below" $\vec{r}$.

→ degenerate cases go away. * If p is on boundary, report it & halt.

Query: Is point p inside polyhedron H?

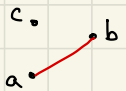
Shoot ray $\vec{r}$ to the right of $q := (p_x, p_y + \varepsilon^2, p_z + \varepsilon)$

Idea: break ties between competing coordinates through powers of ε.

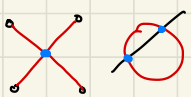
Numerical Robustness

2 Types of geometric computations.

① Predicates: discrete outputs. usually amenable to exact arithmetic.

↳ ex) ccw, inCircle 

② Constructors: create geometric objects. bit complexity may grow w/o bounds

↳ ex) compute intersections  → may need rationals / radicals

↳ products are (usually) used in predicates, so precision may matter!

Two paths to robustness:

① Exact arithmetic libraries: can be slow, not always applicable.

② Algorithm design for "lying": hard, very dependent on context, not general.

Path 1: discipline & exact arithmetic.

Slow down a program by a factor of $1.01\ldots \sim \infty$.

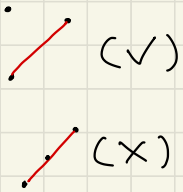
Integers vs Floating-Points? Integer operations are easy, and there

are various bignum packages. Floats have many software for it; we can retrofit. It can also be scaled & converted to Ints, using full bit range!

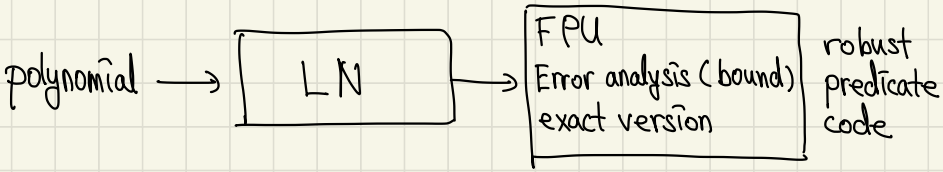
ex) Bailey's MPFUN (Fortran), Shewchuk's predicates (C)

Floating-point filters:

- Compute predicate in FPU with roundoff.
- Compute error bound using forward error analysis.
- If |error bound| ≥ |result|, resort to exact arithmetic.



LN expression compiler [Fortune & Von Wyk, '93, '96]:



Determinant tricks: [Clarkson '92], exact Gram-Schmidt, then $\det(A) = \pm 1$.

Radicals: $E \leftarrow$ expression using $+, -, \times, /, \sqrt[k]{}$. V = exact value of E.

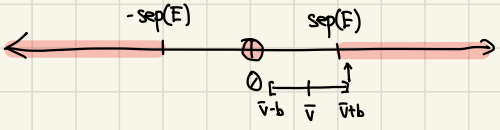
Compute approximation $\bar{V}$, error bound b.

If $|\bar{V}| \leq b$, increase precision, repeat.

↳ Issue: what if $V = 0$?

A <u>seperation bound</u> $\text{sep}(E)$ is a positive real # s.t. if $V \neq 0$, $|V| \geq \text{sep}(E)$.

Such bound can always be produced!



→ If $b < \frac{sep(E)}{2}$, the sign cannot be ambiguous.

Fancier methods for roots of polynomials, etc: algebraic geometry.

↳ Resultants, multivariate Stürm sequences, Sylvester determinants, Gröbner...

Constructors:

- Stored as expression trees, evaluated lazily to precision needed.
- Cascading constructors → increasing bit complexity and/or cascading trees.

Path 2: tolerate lies.

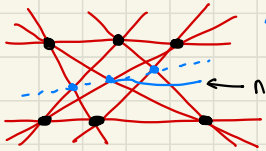
Failures occur because geometric predicates return mutually contradictory results. → there is no input s.t. it generates the predicates' results.

Parsimony? Algorithm is parsimonious if it never performs tests whose result is a formal consequence of previous tests.

Can we design a "parsimony engine"? Following says it's very hard.

Arrangement realizability problem: Given combinatorial line arrangement (such as DCEL w/o coordinates), do lines exist that realize it?

↳ Mnev '88: As hard as existential theory of reals ≥ NP-Hard.



Pappus' Thm: Blue points are colinear.

← not realizable!

↳ reduces to parsimony engine, so it's also ≥ NP-H.

Tolerances? Treat result $< \varepsilon$ as 0. Doesn't work, just changes when it crashes.

"Robust" algorithms: Output is correct for some perturbation of the output.

[Fortune '89] 2D CH correct for points perturbed by relative $O(\varepsilon)$ error.

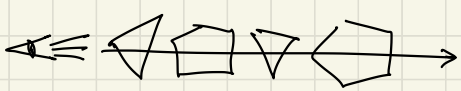
"Quasi-Robust": Output might not be realizable.

[Mirtenkovic '89] Pseudo-line arrangements.

Binary Space Partitions (BSP Trees)

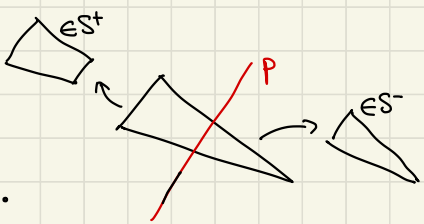
Motivation: Painter's Algorithm for object-space hidden surface removal.

↳ How to draw polygons in 3D space?



- Draw them in order farthest to nearest.
- Issue: Cycles → no total ordering! Split one into two pieces.

Input: Set S of non-overlapping objects.



BSP: Choose a line (in 2D) or plane (in 3D) p.

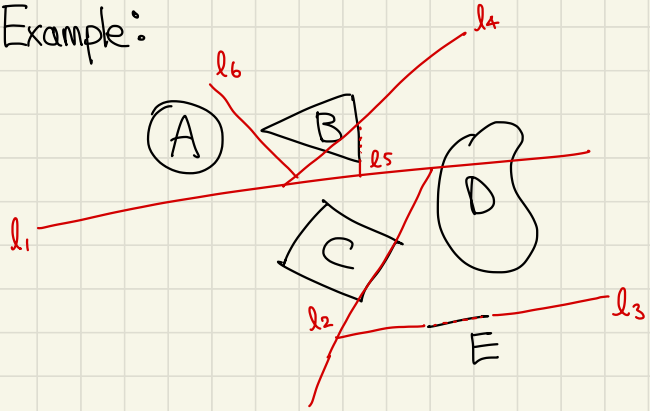
$S^\circ :=$ All objects of S included in P.

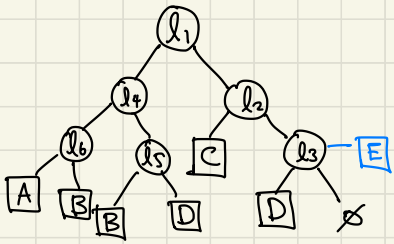
$S^+ :=$ All objects & fragments "left" of p.

$S^- :=$ // "right" of p.

Construct $BSP(S^+), BSP(S^-)$ recursively. Make root node with key p, contents S°. Left & right children are $BSP(S^+), BSP(S^-)$.

Example:





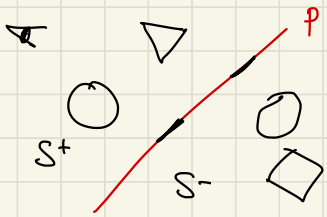
* Never choose a p for which one of S^+, S^- is empty & S^0 is empty (no progress)

Base case: $|S| \leq 1$. Make a leaf node.

Each node of BSP represents a convex cell.

Painter's Algorithm:

- Given a viewpoint v, paints (almost) all objects in correct order.
- Recursively visits all nodes of BSP tree.
- At node with splitting plane p:
  - If v is left of p, paint S^- recursively, then S^0, then S^+ recursively.
  - If v is right of p, " S^+ ", " , " " S^- " .
  - If $v \in p$, paint S^- (but skip S^0), then S^+.



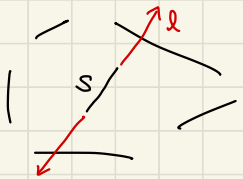
Tree size & traversal: Time wise, if rendering time & fragment size is $O(1)$, linear in # of fragments. Balance of BSP doesn't matter here though.

How to choose splitting planes? How many fragments created?

Some cases we can analyze:

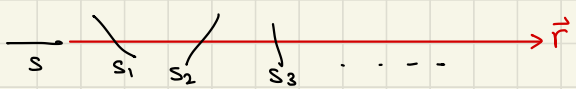
2D: S is set of n non-crossing segments.

- Use an auto-partition: every splitting line is affine hull of some $s \in S$.
- Choose $s \in S$ u.a.r., split with $\ell := \text{aff } s$.



Analysis: Naïve bound, $O(n^2)$ fragments.

$\forall s \in S$, let $s_1, s_2, \ldots$ be segments split by ray $\vec{r}$ from end of s, in order:



ℓ can only split s_i into fragments if s is chosen before $s_1, s_2, \ldots, s_i$ (and sometimes not even then).

$\to \Pr[\vec{r} \text{ cuts } s_i] \le \frac{1}{i+1}$. $\to E[\#s_i \text{ cut by } \vec{r}] \le \sum_{i=1}^{n-1} \frac{1}{i+1} = \sum_{j=2}^{n} \frac{1}{j} \le \int_1^n \frac{1}{j}\,dj = \ln n$.

$\to E[\#s_i \text{ cut by } \ell] \le 2 \cdot \ln n$. $\to E[\text{cuts generated by all splits}] \le 2n \ln n$.

$\to E[\#\text{fragments in BSP tree}] \le \underline{n + 2n \ln n}$. //

Corollary: $\exists$ a BSP tree with $\le n + 2n \ln n$ fragments.

Corollary: A randomized auto-partitioned BSP has $\le n + 4n \ln n$ fragments

w.p. $\geq 1/2$. [Markov argument]

Goal: Construct BSP with $\leq n + 4n \ln n$ fragments.

MC algorithm: Generate random BSP, $\geq 1/2$ success rate.

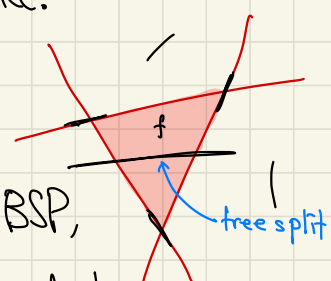
LV algorithm: Run MC algorithm until success. → Always succeeds w.p. 1.

↳ Expected runtime is $O(1)$ [Geometric, $E[\#\text{iter}] \leq 1 + 1/2 + 1/4 + \cdots \leq 2$].

Expected preprocessing time?

- $E[\#\text{ tree nodes}] \in O(n \log n)$.
- Time per node $\in O(n)$ [a bit pessimistic view, better if balanced BSP]

→ $O(n^2 \log n)$ exp. time, usually better in practice.



An optimization: free splits.

If some segment s completely crosses a face f of the BSP, choose s for the next split (in f only) instead of randomly.

↳ Never cuts other segments because S is non-crossing.

How to recognize free split: maintain two boolean flags for each segment, indicate whether each endpoint lies on a splitting line.

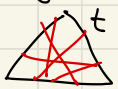
→ Better in practice, no asymptotic improvements.

3D: S is set of n non-crossing triangles (e.g. surface triangulation)

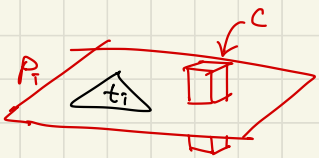
- Auto-partition: every splitting plane is $p := \text{aff } t$ for some $t \in S$.
- Naïve bound: If some triangle t is cut, each cut is a line segment.

  Cuts form line arrangement with $O(n^2)$ faces (fragments)

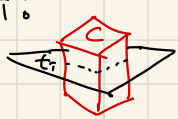
  n triangles $\to O(n^3)$ fragments!



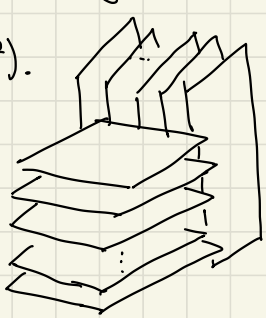
For better bound:



- Initially order S in random permutation.
- Use <u>same</u> permutation in every branch of tree, even if $\text{aff } t_i$ winds up splitting a BSP cell c that doesn't intersect t_i!
  - If a free split exists, always take it (in the cell c only).



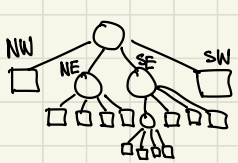
$\to E[\#\text{fragments}] \in O(n^2)$.



no autopartition can beat $\Omega(n^2)$ for worst-case input.

$\hookrightarrow \Theta(n^2)$ is a tight bound.

Quadtrees



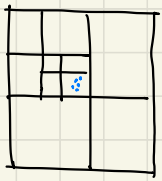
- Every treenode has 0 or 4 children.
- Refine any leaf with $> k$ points.

Advantages (quadtrees vs general BSP trees):

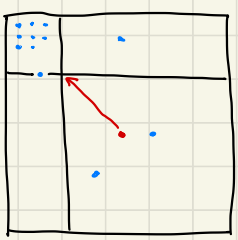
- Compact representation, stores no coordinates except root quadrant
- Good where aspect ratio matters:
  - k-nearest neighbors
  - n-body problems

Disadvantages:

- Close points → arbitrary degree of refinement



Shrinking step: If quadrant has a child with $>k$ points and other 3 children together have $<k$ points, "slide" the center diagonally until



$\geq k$ points are not in the interior of the small square. Then we still have 3 leaves, but the small square has better refining behavior.

Let $t :=$ total # treenodes, $i :=$ internal (non-leaves) treenodes,

$i_1 :=$ internal treenodes with 0 or 1 internal children,

$i_2 :=$ " " 2 or more " ".

Every internal node with 0 or 1 internal children has $\geq k$ points not inherited by an internal child. $\Rightarrow i_1 \leq \frac{n}{k}$.

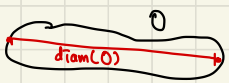
$i_1 + i_2 = i \geq 2i_2 \Rightarrow i_2 \leq i_1 \Rightarrow i \leq 2i_1 \leq 2\frac{n}{k}$.

$t = \text{children} + 1 = 4i + 1 \leq 8\frac{n}{k} + 1. \Rightarrow O(\frac{n}{k})$ treenodes. //

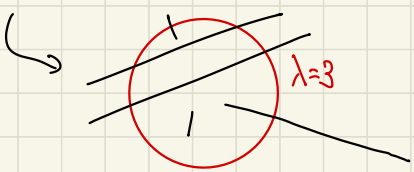
In 3D: Octrees have octants with 8 children each.

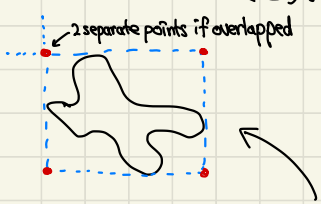
Model for Realistic Input BSPs

Diameter of object: $\text{diam}(O) := \max\{|pq| \mid p,q \in O\}$ (or sup)



A set S of objects in E^d has density λ if every ball $B \subset E^d$ intersects no more than λ objects of diameter $\geq$ diameter(B).





Each object has 2^d guards: corners of its axis-aligned bounding box.

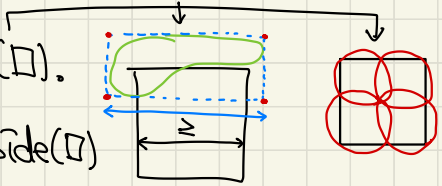
Given n objects, build quadtree on the multiset of $2^d n$ guards.

↳ can be multiple guards at the same position.

Lemma: An axis-parallel square/(hyper) cube $\square$ with k guards in its interior intersects at most $k + 2^d\lambda$ objects from S (if $d \leq 4$).

Proof: If an object o intersects $\square$ but doesn't have a guard in the interior of $\square$, then $\text{diam}(O) \geq \text{side}(\square)$.

Cover □ with 2^d balls of diameter $\leq$ side(□).



□ intersects at most λ objects of diameter $\geq$ side(□) per ball, plus k objects of diameter $<$ side(□). //

* For non-square quadrants, make a box covering the larger side.

→ Leaf quadrant interiors intersect at most $k + 2^d\lambda$ objects.

BSP algorithm for "realistic" inputs:

Choose k.

Phase 1: Build quadtree.

Phase 2: For each leaf with >1 fragment, build BSP subtree, e.g. autopartition.

→ Total # fragments?

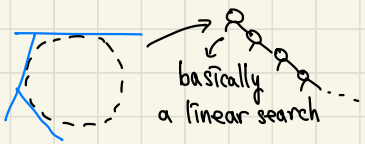
		choose $k \leftarrow 2^d\lambda$
Phase 1:	$O(\frac{n}{k})O(k+2^d\lambda)$	$O(n)$
Phase 2, 2D:	$O(\frac{n}{k})O((k+4\lambda)\log(k+4\lambda))$	$O(n \log \lambda)$
Phase 2, 3D:	$O(\frac{n}{k})O((k+8\lambda)^2)$	$O(n\lambda)$

Issue: We don't know λ...

Solution [Dutch Book]: $k \leftarrow$ something small, e.g. 2^d. Run Phase 1. If some leaf node has $>2k$ fragments, double k, repeat. Then run Phase 2.

Solution 2: $k \leftarrow 2^d$. Phase 1, but don't split a leaf with more unguarded fragments than guards. Then run Phase 2. [kind of adapts to local density]

BSP balance: Autopartitions can be really bad.



... but let $D := \text{diam}(\text{root quadrant})$, and let

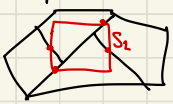
$d := \text{diam}(\text{smallest ball intersecting } k+2^d\lambda+1 \text{ objects}) \rightarrow \Delta = \frac{D}{d}$, the spread.

Depth of quadtree is $O(\log \Delta)$. Depth of P1+P2 tree is $O(\log(\Delta)+\lambda)$.

Preprocessing time: 2D, $O(\overbrace{n \log \Delta}^{P1} + \overbrace{n\lambda \log \lambda}^{P2})$. 3D, $O(n \log \Delta + n\lambda^2)$.

BSP Applications

Solid modeling: Union/Intersection. In 2D, polygonal subdivisions S_1, S_2. Compute overlay, then classify $S_1 \cap S_2$, $\neg S_1 \cap S_2$, $S_1 \cap \neg S_2$, $\neg S_1 \cap \neg S_2$. Merging (deleting edges between) appropriate faces gives the operation.

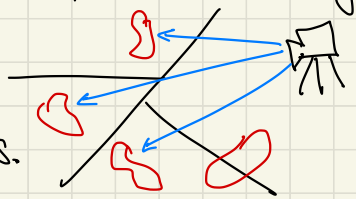


$\rightarrow$ Worst-case runtime $\Theta((n+I)\log n)$.

In 3D, polyhedral subdivisions S_1, S_2. Compute BSP of facets of S_1. Then, facets of BSP subdivide facets of S_2. $\rightarrow$ Each of these cells is in one of $S_1 \cap S_2$, $S_1 \cap \neg S_2$, $\neg S_1 \cap S_2$, $\neg S_1 \cap \neg S_2$.

Ray tracing: Version 1: no reflections/refractions.

- Traverse BSP tree from nearest to farthest, ONCE for all rays.
- No need to compute fragments; can store entire objects in multiple leaves.

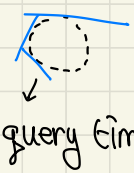


Version 2: reflections & refractions.

(3D analogue of DCEL)

- Build a convex subdivision from the BSP [Winged-edge data structure]
- Use Version 1 for the original rays; reflected/refracted rays are traced through the data structure. [good in practice, no asymptotics]
  * having good aspect ratio is a nice heuristic for performance.

Point location: Preprocess planar/spatial subdivision S of complexity $\Theta(n)$ to find query points. In 2D, trapezoidal map takes $O(n)$ space, $O(n \log n)$ preprocessing time, $O(\log n)$ query time. In 3D, two ways:



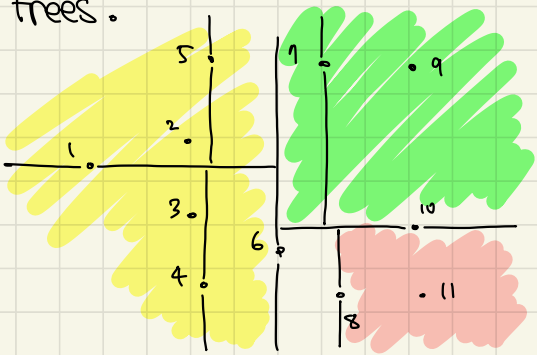
- Randomized autopartition → $O(n^2)$ space, $O(n^3)$ pre-time, $O(n)$ query time
- Octree BSP "realistic" case → $O(n\lambda)$ space, $O(\log \Delta + \lambda)$ query time
  (λ: density of S; Δ: spread of S)
  ↳ preprocessing takes $O(n\lambda^2 + n \log \Delta)$. Phase 1 octree has $O(\frac{\Delta}{\lambda})$ treenodes and $O(n)$ fragments, $O(\log \Delta)$ depth. → $O(n \log \Delta)$ [$O(n)$ per level]
  Phase 2 autopartition: $O(\frac{\Delta}{\lambda})$ leaves $\times$ $O(\lambda^3)$ pre-time/leaf $\in O(n\lambda^2)$.

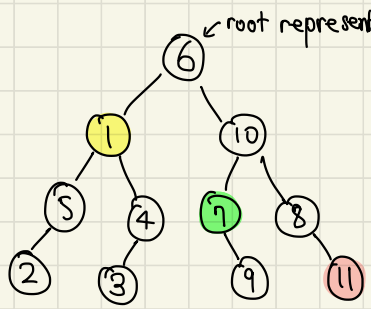
Nearest neighbors problems (aka post office problem): Given n-point set V, find point $p \in V$ nearest to query point q. In 2D, Voronoi + trap. map gives $O(n)$ space, $O(n \log n)$ pre-time, $O(\log n)$ query time.

dD: Voronoi + BSP tree. "Realistic" case gives $O(n\lambda)$ space, $O(\log \Delta + \lambda)$ query time. (λ ← density of VorV; Δ ← spread of VorV)

"Special" cases: query points are $\in V$. → Delaunay Tri. connects every vertex to its nearest neighbor. Worst case $O(n^{\lceil d/2 \rceil} + n \log n)$ time, but in "realistic" case, $O(n \log n)$ with appropriate incremental algorithm.

k-d trees:

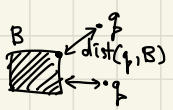




Goal: given a query point q, find point p s.t. $|qp| \leq (1+\varepsilon)|qs|$ where s is the true nearest neighbor to q. $\varepsilon = 0 \rightarrow$ exact, $\varepsilon > 0 \rightarrow$ approximate n.n.

Maintain:

- Nearest neighbor found so far (or k nearest) (goes down)
- Heap of unexplored subtrees, keyed by distance from q. (↑ goes up)



dist = 0 if $q \in B$.

$Q \leftarrow$ heap containing root node with key 0.

$r \leftarrow \infty$

while Q not empty $\&$ $(1+\varepsilon)\cdot \text{minkey}(Q) < r$:

  $B \leftarrow \text{removemin}(Q)$

  $p \leftarrow B$'s point

  if $|qp| < r$, then $\{w \leftarrow p;\ r \leftarrow |qp|\}$

  $B', B'' \leftarrow$ child boxes of B (ignore if null)

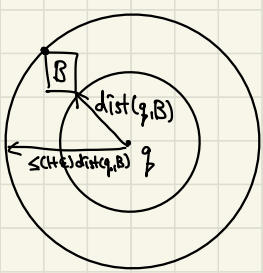
  if $(1+\varepsilon)\cdot \text{dist}(r, B') < r$, then $\text{insert}(Q, B', \text{dist}(q, B'))$

  if // $B'')$ // B'' $B')$

return w

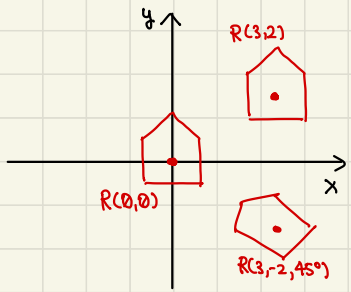
Query time: Let O be a circle of radius $|qs|$ centered at q.

- Algo. checks no box strictly outside O.
- Every box strictly inside O is empty.
- No box B of diagonal $< \varepsilon \cdot \text{dist}(q, B)$ is expanded.
- By a packing argument, $O\left(\frac{1}{\varepsilon^{d-1}}\right)$ such B exists
- # nodes visited $\in O\left(\underbrace{\frac{1}{\varepsilon^{d-1}} + \text{depth}}_{=: w}\right)$ ↓ for octree, $\log \Delta$ ↳ spread := $\frac{\text{farthest pair}}{\text{closest pair}}$



$\Rightarrow$ query time $O(w \log w)$. [kind of independent of n, fixed Δ?]

Robot Motion Planning



Let robot R be a simple polygon/polyhedron.

It navigates in a workspace (typically E^2 or E^3).

Translating 2D robot:

$R(x,y)$ is robot with its reference point translated to point (x,y).

Assume $R = R(0,0)$.

Translating/Rotating robot:

$R(x,y,\theta)$ is robot translated & rotated θ about its reference point.

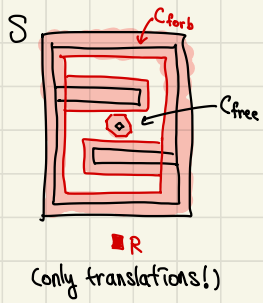
Configuration space C is the space of parameters (degree of freedom) that define a placement of the robot. e.g. trans/rot: $E^2 \times [0°, 360°)$ ← periodic parameter

Let point set S be polygonal obstacle(s).

Assume S is closed, R is open.

touching is allowed!

Requirement: $S \cap R(x,y,\theta) = \emptyset$.



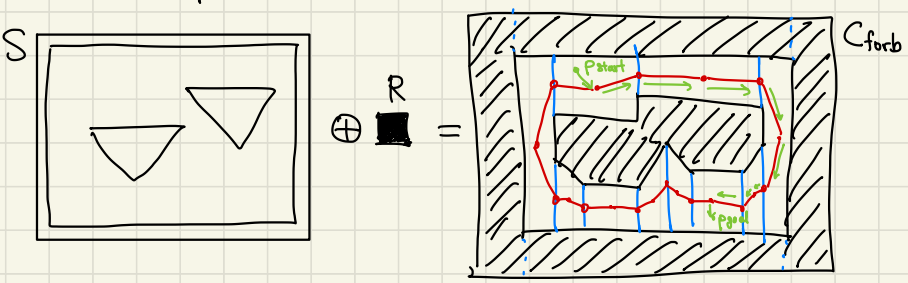
Def) Free Space: $C_{free}(R,S) := \{(x,y,\theta) \mid S \cap R(x,y,\theta) = \emptyset\}$.

Def) Forbidden space: $C_{forb}(R,S) = \overline{C_{free}(R,S)} = \{(x,y,\theta) \mid S \cap R(x,y,\theta) \neq \emptyset\}$.

Motion Planning: S := union of all obstacles, $O(n)$ complexity.

R := convex robot, $O(1)$ complexity. → $O(n \log^2 n)$ time if R is convex.

Forbidden space $C_{forb} \leftarrow S \oplus (-R)$ [Stored as DCEL] (Minkowski sum $\oplus$ is covered later!)



Free space $C_{free} \leftarrow C \setminus C_{forb}$

$E \leftarrow$ edges of C_{forb} → $O(n)$

Construct trapezoidal map $T(E)$ → $O(n \log n)$ time

Remove (or ignore) trapezoids included in C_{forb}

Construct roadmap: put one node in each trap, one on each vertical extension.

Put edges between nodes in traps & nodes on their sides.

Preprocessing ↑

Query: navigate from point p_{start} to p_{goal}.

Locate trapezoids containing p_{start}, p_{goal}. → $O(\log n)$

↳ If either is in C_{forb}, report IMPOSSIBLE.

Find path from p_{start} to p_{goal} in roadmap by BFS. → $O(n)$

↳ If $\nexists$ such path, report IMPOSSIBLE.

Report segence of line segments from p_{start} to p_{goal}. → $O(n)$

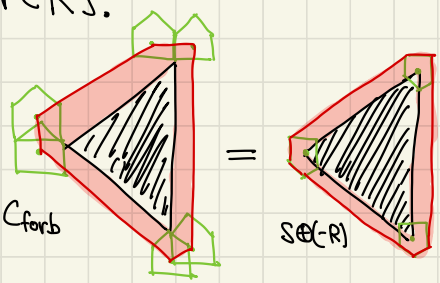
Runtime: $O(n \log^2 n)$ preprocessing time, $O(n)$ query time.

For a translating robot R, $C_{forb}(R,S) := \{p : \exists r \in R, s \in S \text{ s.t. } p+r=s\}$

$= \{s-r : r \in R, s \in S\}$.

Def) Minkowski Sum: $P \oplus Q := \{p+q \mid p \in P, q \in Q\}$.

→ C_{forb} is then $S \oplus (-R)$ where $-R := \{-r \mid r \in R\}$.

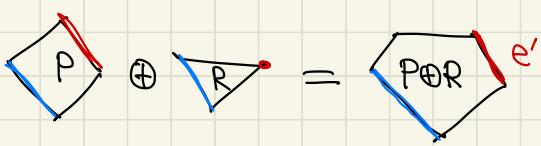


Thm) Let P, R be convex polygons with n & m edges each. $P \oplus R$ is also convex with at most $(n+m)$ edges.

Proof Sketch: Convexness: Let q_1, q_2 be 2 points in $P \oplus R$. Let $p_1 + r_1 = q_1$, $p_2 + r_2 = q_2$, $p_1, p_2 \in P$, $q_1, q_2 \in Q$.

Then $\alpha q_1 + (1-\alpha) q_2 = \overbrace{(\alpha p_1 + (1-\alpha) p_2)}^{\in P} + \overbrace{(\alpha r_1 + (1-\alpha) r_2)}^{\in R} \in P \oplus R$. //

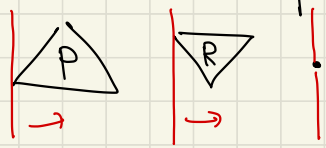
$(m+n)$ edges: let e be edge of $P \oplus R$. e is Mink. sum of some edge e' of one of P, R, and some vertex/edge f of the other.



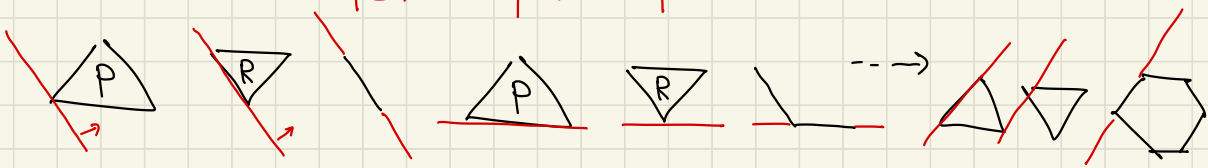
Charge each edge of $P \oplus R$ to an edge. Each edge of P & R charged at most once. //

Constructing P⊕R:

Walk around P&R. Maintain two parallel supporting lines. (1 for P, 1 for R)



start with sum of lexico. min points.



Let $p_1, \dots, p_n$ be vertices of P in ccw order with p_1 the lexico. least.

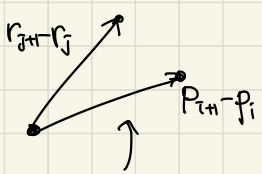
// $r_1, \dots, r_m$ // R // r_1 //

$i = j = 1$.

repeat:

add $p_i + r_j$ as vertex of P⊕R

$k \leftarrow \text{Orient2D}(\emptyset, p_{i+1} - p_i, r_{j+1} - r_j)$



if $k \geq 0$, $i{+}{+}$ // next turn hits p_{i+1}

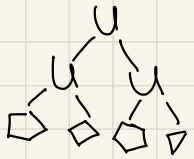
if $k \leq 0$, $j{+}{+}$

until $i = n+1$ and $j = n+1$

Minkowski sums of nonconvex polygons:

- Partition P, R into convex partitions, $\cup P_i$, $\cup R_j$.

- Compute $P_i \oplus R_j$ $\forall (i,j)$ pairs.
- Compute union $\bigcup_i \bigcup_j \{P_i \oplus R_j\}$ (by divide & conquer)
  - Use overlay of subdivisions algo for pairwise unions.



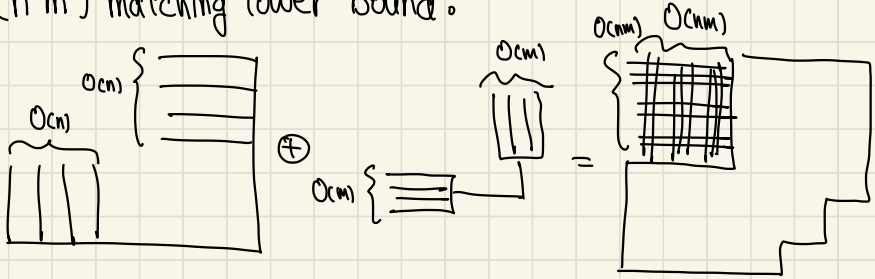
# of subpolygons $\leq (n-2)$ for P, $\leq (m-2)$ for R.

Complexity of $P \oplus R$:

- $P \oplus R$ is a union of $O(nm)$ hexagons.

  ↳ $P \oplus R$ // face of arrangement of $O(nm)$ lines → $\underline{O(n^2m^2)}$.

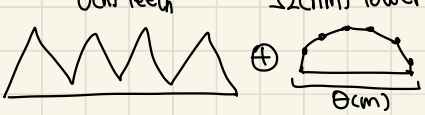
- $\Omega(n^2m^2)$ matching lower bound:



Runtime: $O(n^2m^2 \log(n+m))$ $\underline{\text{if}}$ tree is balanced "correctly".

Computing $P^{(n)} \oplus R^{(m)}$:	complexity	runtime
P, R convex	$O(n+m)$	$O(n+m)$
P, R nonconvex	$O(n^2m^2)$	$O(n^2m^2 \log(n+m))$
P nonconvex, R convex	$O(nm)$	$O(nm \log(n+m) \log n)$

$\Omega(nm)$ lower bound:



$O(nm \log(n+m))$: subdivision overlay time per level

$O(\log n)$ levels of div & conq. tree

[Flato suggests angle bisector decomposition]

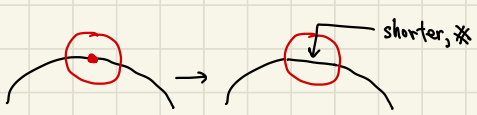
Shortest Paths for Translating Robots

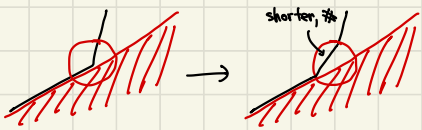
Roadmaps are not shortest paths.

Let C_{free} be polygonal free configuration space.

Lemma: shortest path in C_{free} from p_{start} to p_{goal} is polygonal chain that turns only at vertices of C_{free}.

Proof Sketch: [Full proof in Dutch Book, Ch. 15.1]





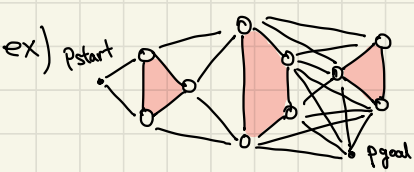
Visibility Graph: $G(V,E)$ where nodes $V :=$ vertices of $C_{free} + \{p_{start}, p_{goal}\}$, edges $E := \{(v,w) \mid v$ and w can "see" each other$\}$, and $w(v,w) := |\overline{vw}|$ ← Euclidean distance

✱ Ambiguity: 

Can u see w? → doesn't matter for length

$\forall x,y \in V$, shortest path between x & y in C_{free} is same as <u>shortest path in G</u>.

→ Find latter with Dijkstra's algorithm: $O(|V|\log|V|+|E|)$ time, $|E| \le |V|^2$.

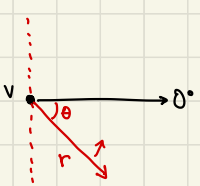
ex) 

How to compute the visibility graph?

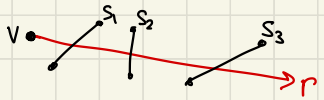
$\forall v \in V$, find every $w > v$ visible from v as follows. (no duplicate tests)

Sweep a ray r from v through angles $(-90°, 90°]$.

Maintain all segments that r intersects in a <u>status tree</u> T.



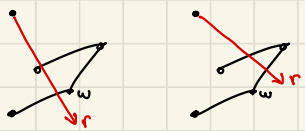
Segments in T are ordered by intersection with r.



When r passes over a vertex w:

- Use first segment in T to decide wheter $(v,w) \in E$ [visibility]
- Remove from T segments ending at w. ($\overline{uw}$ s.t. $Or2D(v,u,w) > 0$))
- Add to T segments beginning at w. (// < 0))

ex)



Algorithm:

- Sort vertices $> v$ in rotary order around v. $[O(n \log n)]$
- Find segments intersecting down ray $\downarrow r$ $[O(n)]$
- Add them to T. $[O(n \cdot \log n)]$
- For each w in rotary order, process w as above $[O(n \log n)]$

Runtime: $O(n \log n) \times O(n)$ vertices $\rightarrow O(n^2 \log n)$ total time

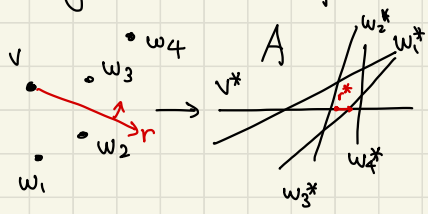
↳ This dominates Dijkstra's & computing C_{free}. Can we do better?

An $O(n^2)$ algorithm: Two reasons for $O(n^2 \log n)$ time.

① n sorts of n items each. ② n^2 events with $O(\log n)$ status tree updates

Problem 1: solved by duality.

- Form arrangement A of lines dual to V.



- For each $v \in V$, read its sorted list from A in $O(n)$ time.

  Each vertex of A on v^* represents an event for $\vec{r}$. $[O(n^2)]$

Problem 2: Don't maintain a status tree.

- We only need to maintain the segment closest to v on r.

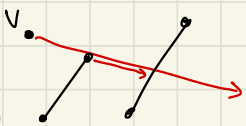
  3 situations:

① 

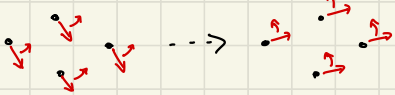
(easy) ② 

(easy) ③ 

(hard!)

Solution: Use whatever w points at for that angle. 

Question: How do we know which segment w maintains?

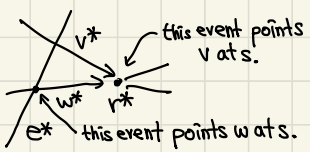
→ Sweep <u>all</u> rays in synchrony. 

Angle of ray corresponds to x-coordinate in dual.

Idea: Sweep all vertices of A from left to right. Each vertex is an event.

Problem: Sorting $O(n^2)$ vertices is $O(n^2 \log n)$ time...

Idea: Make A a DAG with all edges directed left to right. An event can be performed as long as all events that point to it has been performed.



⇒ Topological sort suffices! $O(n^2)$ time.

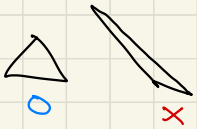
↳ Sorts a total order respecting partial orders.

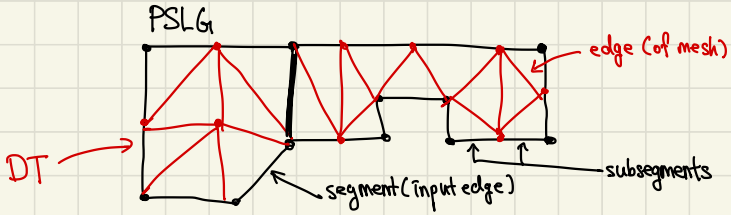
Process events in order of top. sort. Each event is now constant time. //

⇒ shortest translating path in $O(n^2)$ time!

Ruppert's Triangular Mesh Generation

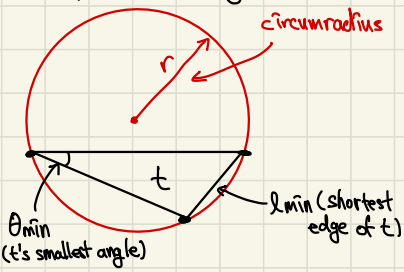
Motivation: We want triangles close to equilaterals.





Input: PSLG [no acute angles]

Output: Triangular Mesh, no angle $< 20.7°$, $\geq 138.6°$.



$$\text{Radius-Edge ratio} = \frac{r}{\ell_{min}} = \frac{1}{2\sin(\theta_{min})}$$ ← make this big!

→ minimize $\frac{r}{\ell_{min}}$ to maximize $\sin(\theta_{min})$, also θ_{min}.

A <u>subsegment</u> is any input segment or edge produced by subdividing a (sub)segment. A subsegment is <u>encroached</u> if:

- there is a vertex inside its diametric circle, or 
- it's missing from current DT.  [edge case]

Bisect each encroached subsegment, yielding 2 subsegments.

Algorithm: PLSG X, vertices V, segments S, upper bound B on radius-edge ratio

T ← DT(V) or CDT(V) [your choice]

while some $s \in S$ is encroached or some $t \in T$ is "bad" → radius-edge ratio $\geq B$, or → user specified size bound

  while some $s \in S$ is encroached: "Split" S. → Insert new vertex into T at midpoint of s. Update S.

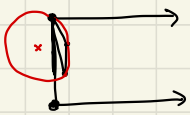
  if some $t \in T$ is bad:

    $p \leftarrow$ circumcenter of t

    if p encroaches upon some subsegments $s_1, s_2, \ldots \in S$: Split $s_1, s_2, \ldots$.

    else: Insert p into T. ↳ encroaching circumcenters are rejected to avoid creating small features.

Q: What if a circumcenter is outside domain?



↳ we can prove that ∃ an encroached subsegment → never need to insert it.

Q: How to detect encroachments & bad triangles fast?

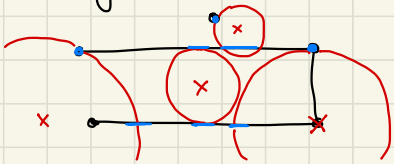
↳ Keep lists. Check for new ones (locally) after each insertion.

  * With CDT, s is encroached only by vertices visible from S.

Q: How to detect encroaching circumcenters?

↳ One way is to insert, check for encroachments, can be done locally.

Local feature size: $lfs(p) :=$ radius of smallest disk centered at p intersecting 2 nonintersecting features (vertex or segment) in X.



Lemma: $lfs(v) \leq lfs(u) + |uv|$. Proof:

→ lfs(·) is continuous & 1-Lipschitz.

In Ruppert meshes, no edge through any $p \in \Omega$ is shorter than $c \cdot \text{lfs}(p)$ for some $c = c(B)$, a function of radius-edge ratio bound.

↳ This gives guarantees of:

- Termination
- Good grading
- Size-optimality [# triangles is within constant factor of best possible mesh of Ω satisfying the bound B]

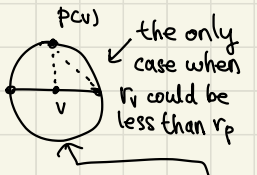
Termination: Let the insertion radius r_v of a vertex (including rejected circumcenters) v be the distance to v's nearest (visible) neighbor at the moment v is created. The parent $p(v)$ of:

- a subsegment midpoint v is the nearest encroaching vertex (might be a reject)
- the circumcenter of t is the most recently inserted endpoint of t's shortest edge.

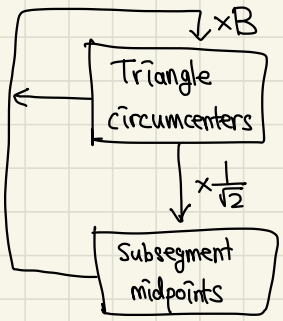
- input vertices is None.

Thm) Suppose no 2 segments in X adjoin at an acute angle. Let v be vertex (in mesh or rejected) with parent $p = p(v)$. Then either:

- $r_v \geq lfs(v)$, or
- $r_v \geq B r_p$ and v is a circumcenter, or
- $r_v \geq \frac{r_p}{\sqrt{2}}$ and v is a subsegment midpoint.



Proof Idea: Break into four cases, input, circumcenter, midpoint (parent rejected/not)



Arrows "give birth to" child, with worst-case insertion radius multiplier. If no cycle has product < 1, edge lengths cannot diminish. Therefore, set $B \geq \sqrt{2}$.

$\rightarrow \theta_{min} = \arcsin\frac{1}{2B} \geq 20.7°$.

Thm) If $B \geq \sqrt{2}$, algorithm terminates with no edge shorter than $\min_{p \in \Omega} lfs(p)$.

Proof: By induction.

* We can have acute angles down to 60° for termination.

Isosurface Stuffing (Tetrahedral Mesh Generation)

Input: Cut function $f: \mathbb{R}^3 \rightarrow \mathbb{R}$ (where $f(p) = 0$ is the boundary)

Output: tetrahedral mesh representing $\{p \mid f(p) = 0\}$, with all dihedral angles between 10.78° and 164.74°.

Pluses: Fast, numerically robust, simple.

Minuses: rounds off sharp edges & corners, no grading on isosurface.

Background grid is body centered cubic (BCC) lattice:

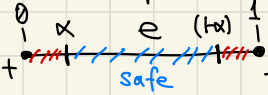
cubic lattice + cube centers, DT has identical tets, 90° & 60° dihedral angles.

Algorithm:

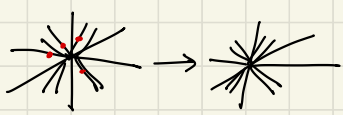
① Compute f at lattice points, signs: +, −, or 0.

② For each edge with one + and one − endpoint, compute cut point on edge where f=0. [iterated bisection in practice]

③ Warp the background grid. Cut point c on edge e violates endpoint v of e if $|cv| \le \alpha|e|$.



while some lattice point p is violated:



Warp (move) p onto a cut point that violates p. [snaps p onto isosurface]

Change p's sign to 0. $[\alpha_{blue} = 0.28511, \alpha_{green} = 0.39882]$

Discard cut points on all edges of background grid adjoining p.

④ For each background tet, create 0-3 output tets from stencil in lookup table that maps signs of vertices to stencils.

* Options for Quadruple-zero background tets:

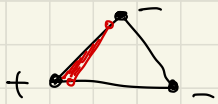
① No, always discard them.

② Heuristic: quad-zero becomes output tet if its quality is good enough,

and its barycenter has $f > 0$.

③ Change α values s.t. quad-zero tets are always good quality.

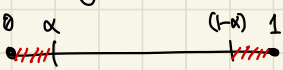
↳ This weakens angle bounds.

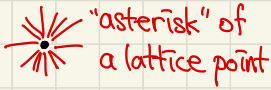


Main Idea: Cut points near lattice vertices → arbitrarily bad angles

Choose α big enough to eliminate these, small enough to not make new ones.

Computer Aided Proof of Angle Bounds: Enumerate all tets on all stencils, find worst angles.

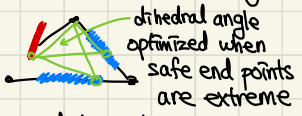




After warping, we know that

- A warped vertex lies on its asterisk & its sign is 0.
- +/− vertices weren't warped.
- No cut points on edge adjoining warped vertex.
- No 2 vertices can't warp toward each other on same edge.

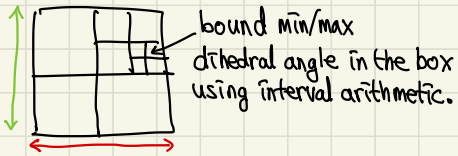
Treat each asterisk as 7 seperate cases.



Position of warped vertices & cut points described by 1 parameter each

→ At most 2 parameters to optimize: endpoints of edge of dihedral angle.

Discrete parameter space w/ quadtree.

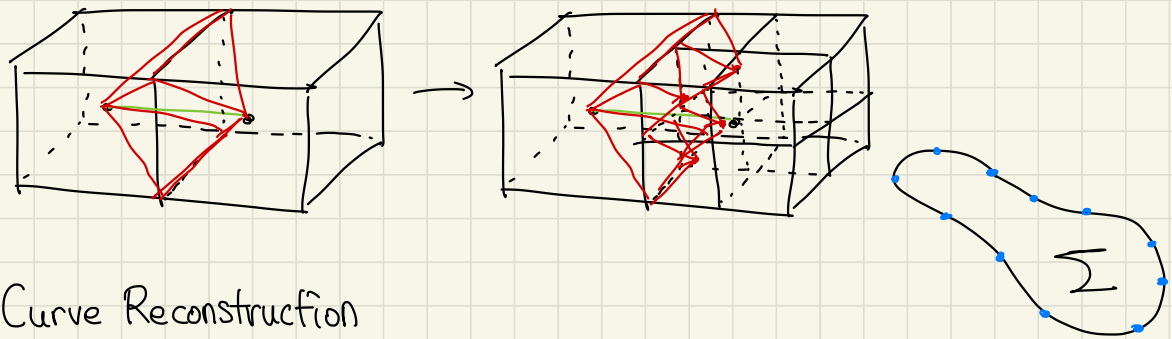


Use proof code as inner loop of optimizer to get best α values.

Graded Interior Tetrahedra: Use a modified octree that can have <8 children.

Use octree to build a graded background mesh. Balance adjoining octants.

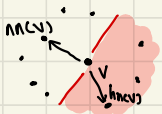
When 2 leaf octants of some size share facet, create 4 BCC tets.



Curve Reconstruction

Let P be a finite set of points sampled from smooth closed curve Σ.

↳ Σ is also called "1-manifold w/o boundary.



For $v \in P$, let $nn(v)$ be nearest neighbor of v in P, and let $hn(v)$ be v's half-neighbor: the nearest neighbor s.t. $\angle nn(v), v, hn(v) \geq 90°$.

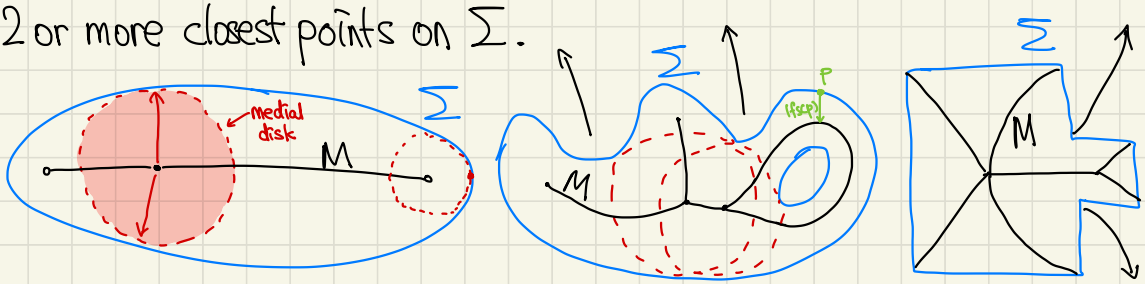
Algo: NN-Crust(P)

$E \leftarrow \emptyset$. for each $v \in P$, $y \leftarrow nn(v)$, $z \leftarrow hn(v)$. $E \leftarrow E \cup \{vy, vz\}$.

Output E. ["Works" in any E^d, $d \geq 2$]

When does it work?

Def) Medial Axis of Σ: The closure of the set of points that have 2 or more closest points on Σ.

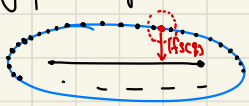


The local feature size $lfs(p)$ of a point $p \in \Sigma$ is distance from p to the point nearest to p on M. * not necessarily medial axis point touching p!

↳ Zero at sharp corners, so we exclude this input.

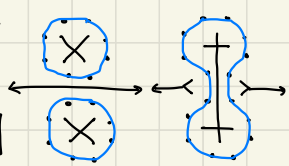
↳ $lfs(\cdot)$ is continuous & 1-Lipschitz.

"Σ is smooth": $\exists c > 0$ s.t. $lfs(p) \geq c \ \forall p \in \Sigma$.

P is an <u>ε-sample</u> of Σ if every point $q \in \Sigma$ has a sample point $p \in P$ within a distance of $\varepsilon \cdot lfs(q)$. 

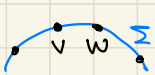
(0.5-sample)

[SOTA]

NN-Crust is guaranteed to work when $\varepsilon < 1/3$. [$\varepsilon < 0.66$] 

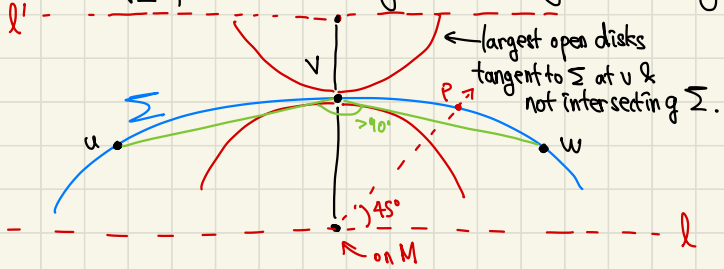
Nothing " " $\varepsilon \geq 1$. [$\varepsilon \geq 0.12$]

Analysis: An edge vw ($v, w \in P$) is <u>correct</u> if v & w are adjacent on Σ.

Every $v \in P$ adjoins exactly 2 correct edges. 

Lemma: If $\varepsilon \le \frac{1}{\sqrt{2}}$, correct edges will adjoin at angles $> 90°$.

Proof:

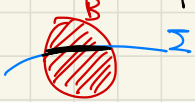


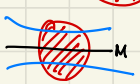
$< \frac{1}{\sqrt{2}}$-sample $\to \min\{|pv|, |pw|\} < \frac{|pq|}{\sqrt{2}}$. But $|pv| \ge \frac{|pq|}{\sqrt{2}}$.

If w on/below $l \to |pw| \ge \frac{|pq|}{\sqrt{2}}$. #, so w must be above l.

Likewise for u and $l' \Rightarrow \angle uvw > 90°$. //

Lemma: Let B be a closed disk. If $B \cap \Sigma$ contains >1 point, either

- $B \cap \Sigma$ is homeomorphic to a segment, or 

- there is a point of M in B's interior. 

Proof: Suppose $B \cap \Sigma$ is not homeo. to segment. Then either

- $B \cap \Sigma$ includes a closed loop → 

→ point of $M \subseteq B$
- $B \cap \Sigma$ has 2^+ connected components.  → shrink B around center →  → shrink at point of tangency → 

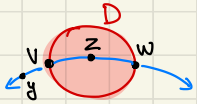
//

Thm) If $\varepsilon \le 1/3$, edges $(v, nn(v))$ and $(v, hn(v))$ are correct.

Proof: Let vw be any incorrect edge. We shall show $w \neq nn(v), hn(v)$.

Let vy, vz be the correct edges. Let D be a closed disk w/diam. vw.

Case 1: $D \cap \Sigma$ is topological segment.

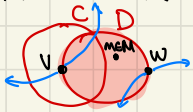


↳ $|vz| < |vw| \to w \neq nn(v)$. If $nn(v) = z$, $w \neq hn(v)$ b/c $\angle zvw \leq 90°$.

If $nn(v) = y$, $w \neq hn(v)$ because $|vz| < |vw|$ and $\overbrace{\angle zvy > 90°}^{\text{lemma 1}}$.

Case 2: Otherwise, by Lemma 2, some point $m \in M$ is in D's interior.

Let C be circle, center v, diameter $|vw|$.



Let p, q be points of $C \cap \Sigma$, nearest v along Σ.

- If p, q don't exist, y & z are inside C. $\Rightarrow w \neq nn(v)$.

  Assuming $(v, nn(v))$ is correct $\to w \neq hn(v)$.

- Otherwise, $|pm| \leq |pv| + |vm| \leq \frac{|vw|}{2} + |vw| = \frac{3}{2}|vw|$.

  Sample point x nearest p satisfies $|px| \leq \varepsilon \, lfs(p) \leq \frac{|pm|}{3} < \frac{|vw|}{2}$

  $= |pv| \leq |pw|$. $\to x \neq v$, $x = w$, and $|vx| \leq |pv| + |px| < |vw| \Rightarrow w \neq nn(v)$.

  [Not complete, but believe that $w \neq hn(v)$.]

Speed: Use $DT(P)$ to improve from $\Theta(n^2)$ to $\Theta(n \log n)$ time (in 2D).

Thm) Every correct edge $\in DT(P)$ if $\varepsilon \leq 1$.

Surface Reconstruction

Restricted DT: $DT|_\Sigma P$ is a subcomplex of $DT(P)$ containing every simplex

whose Voronoi dual face intersects Σ.

Thm) If P is a 0.32-sample of Σ, $|DT|_\Sigma P|$ is homeomorphic to Σ.

Uses: 1) Surface Meshing: Σ is known; place vertices for good RDT/mesh

2) Surface Reconstruction: Σ is unknown; choose triangles from $DT(P)$.

↳ Let P be a finite set of points sampled from a smooth closed 2-manifold Σ.

Medial axis M of Σ: Still the closure of set of points that have non-unique "closest point on Σ".

$lfs(p)$ for $p \in \Sigma$: distance from p to nearest point on M.

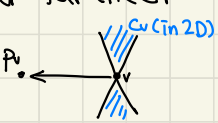
P is ε-sample of Σ if every $q \in \Sigma$ has a sample point $p \in P$ within $\varepsilon \cdot lfs(q)$. $|pq| \leq$

Cocone Algorithm:

For each tet $s \in DT(P)$, let c_s be its circumcenter.

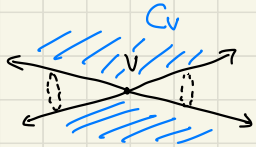
For each tri $t \in DT(P)$, its Voronoi dual edge is $e_t = c_s c_{s'}$, where s & s' are the tets with face t.

For each vertex $v \in DT(P)$, its pole p_v is the circumcenter farthest from v of all tets adjoining v.



[The pole vector $\overrightarrow{vp_v}$ estimates the normal to Σ at v.]

The cocone C_v of v is $\{q \mid 67.5° \leq \angle qvp_v \leq 112.5°\}$.



A tri $t \in DT(P)$ is a <u>cocone triangle</u> if e_t intersects the cocones of all three vertices of t.

If $\varepsilon \leq 0.05$,

- every cocone tri t has empty circumsphere of radius $\leq \frac{1.18\varepsilon}{1-\varepsilon}$ lfs(p).
- every // is nearly orthogonal to the normals of Σ.
- every tri in DT(P) is a cocone tri. [sadly, others are cocone as well]

Manifold Extraction: An edge is <u>sharp</u> if 2 successive triangles around it form angle $> 270°$.  [include edge with just one tri]

Label all tets of DT(P) "inside", all outside tets "outside" via DFS, never walking through surviving tris.

Output tris where "inside" tet meets "outside" tet.

→ If $\varepsilon \leq 0.05$,

- triangulation is homeomorphic to Σ,
- every point on tri is within $0.08\,\text{lfs}(q)$ from nearest point $q \in \Sigma$.