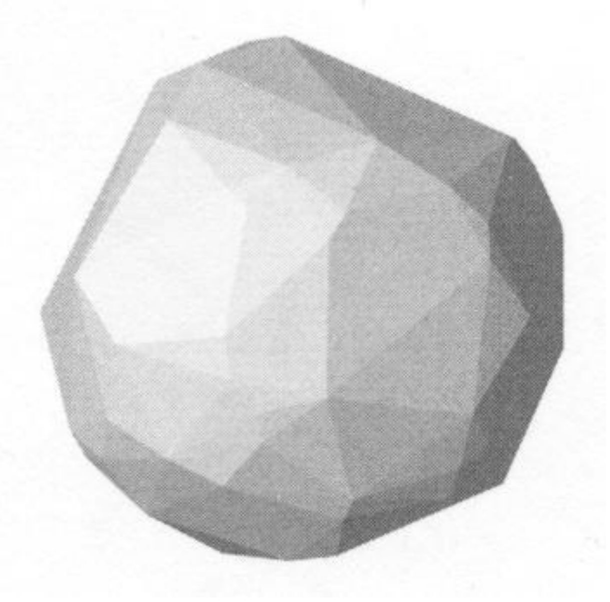
Convex Hulls in 3-space

Jason C. Yang

Problem Statement

- Given P: set of n points in 3-space
- Return:
  - Convex hull of P: $\mathcal{CH}(P)$
  - Smallest polyhedron s.t. all elements of P on or in the interior of $\mathcal{CH}(P)$.



Algorithm

- Randomized incremental algorithm

- Steps:
  - Initialize the algorithm
  - Loop over remaining points
    Add p_r to the convex hull of $P_{r\text{-}1}$ to transform $\mathcal{CH}(P_{r\text{-}1})$ to $\mathcal{CH}(P_r)$
    [for integer $r \geq 1$, let $P_r := \{p_1, \ldots, p_r\}$]

Main Idea:

Incrementally insert new points into the running/intermediate Convex Hull.

Initialization

- Need a $\mathcal{CH}$ to start with
- Build a tetrahedron using 4 points in P
  - Start with two distinct points in P: p_1 and p_2
  - W**alk through *P* to find p_3 that does not lie on the line** through p_1 and p_2
  - Find p_4 that does not lie on the plane through p_1, p_2, p_3
  - Special case: No such points exist?

    All points lie on a plane. Use planar $\mathcal{CH}$ algorithm!
- Compute random permutation $p_5,\ldots,p_n$ of the remaining points

Inserting Points into $\mathcal{CH}$

- Add p_r to the convex hull of $P_{r\text{-}1}$ to transform $\mathcal{CH}(P_{r\text{-}1})$ to $\mathcal{CH}(P_r)$

  [for integer $r \geq 1$, let $P_r := \{p_1, \ldots, p_r\}$]

- Two Cases:

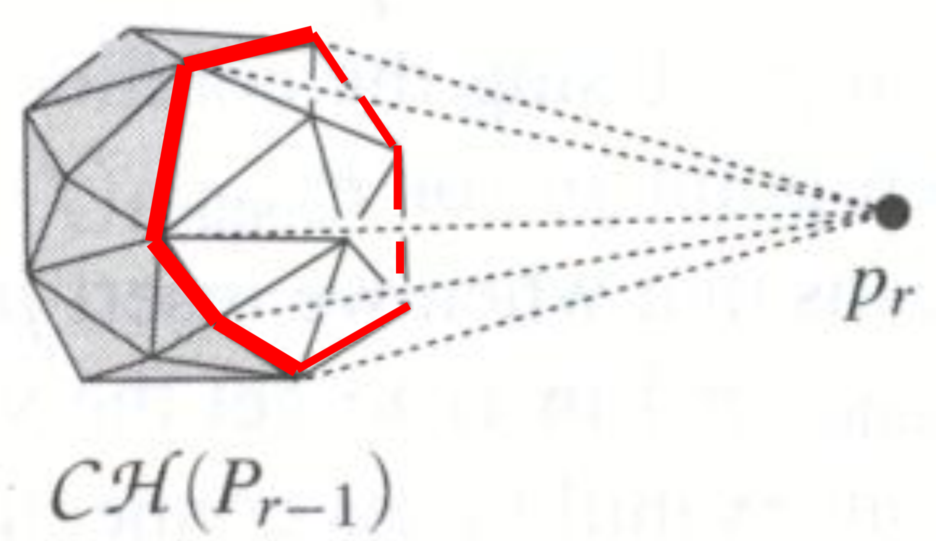
  1) P_r is inside or on the boundary of $\mathcal{CH}(P_{r\text{-}1})$

     Trivial: $\mathcal{CH}(P_r) = \mathcal{CH}(P_{r\text{-}1})$

  2) P_r is outside of $\mathcal{CH}(P_{r\text{-}1})$

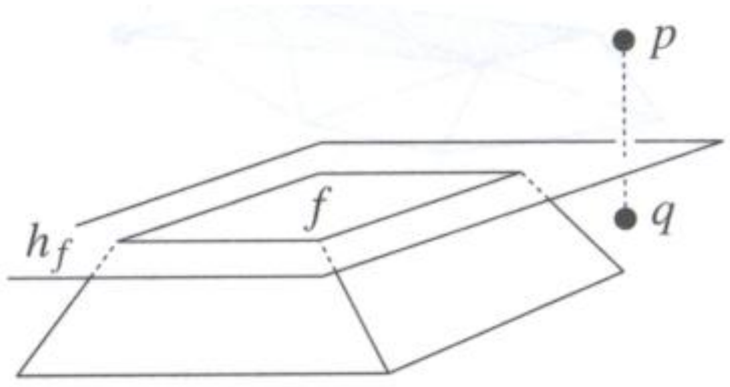
Case 2: P_r outside $\mathcal{CH}(P_{r\text{-}1})$

- Determine ***horizon*** of p_r on $\mathcal{CH}(P_{r\text{-}1})$
  - Closed curve of edges enclosing the ***visible*** region of p_r on $\mathcal{CH}(P_{r\text{-}1})$



Visibility

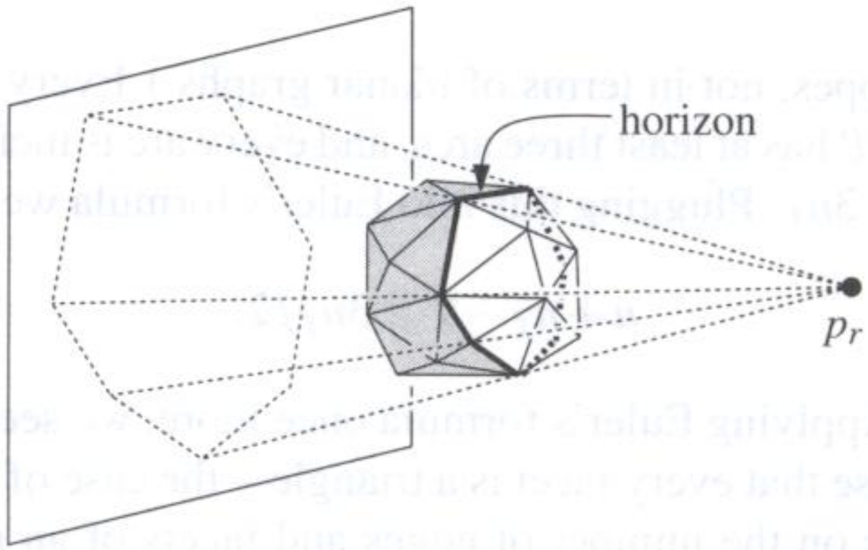
- Consider plane h_f containing a facet f of $\mathcal{CH}(P_{r\text{-}1})$
- f is ***visible*** from a point if that point lies in the open half-space on the other side of h_f



f is visible from p, but not from q

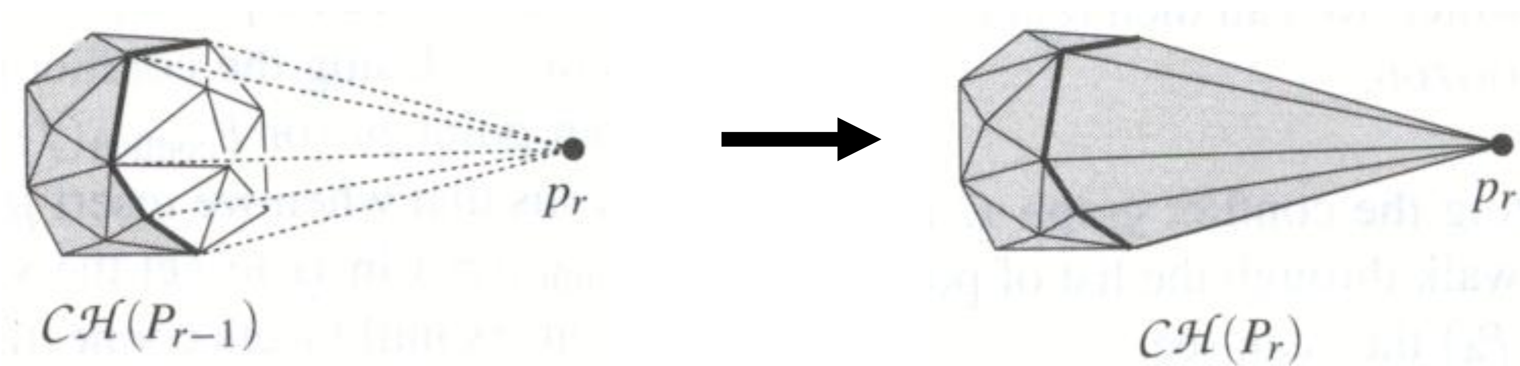
Rethinking the Horizon

– Boundary of polygon obtained from projecting $\mathcal{CH}(P_{r\text{-}1})$ onto a plane with p_r as the center of projection



$\mathcal{CH}(P_{r\text{-}1}) \rightarrow \mathcal{CH}(P_r)$

- Remove *visible* facets from $\mathcal{CH}(P_{r\text{-}1})$
- Found *horizon*: Closed curve of edges of $\mathcal{CH}(P_{r\text{-}1})$
- Form $\mathcal{CH}(P_r)$ by connecting each horizon edge to p_r to create a new triangular facet



Algorithm So Far…

- Initialization
  - Form tetrahedron $\mathcal{CH}(P_4)$ from 4 points in P
  - Compute random permutation of remaining pts.
- For each remaining point in P
  - p_r is point to be inserted
  - If p_r is outside $\mathcal{CH}(P_{r\text{-}1})$ then
    - Determine visible region
    - Find horizon and remove visible facets
    - Add new facets by connecting each horizon edge to p_r

How do we determine the visible region?

How to Find Visible Region

- Naïve approach:
  - Test every facet with respect to p_r
  - $O(n^2)$

- Trick is to work ahead:

  Maintain information to aid in determining visible facets.

Conflict Lists

- For each facet f maintain

  $P_{\text{conflict}}(f) \subseteq \{p_{r+1}, \ldots, p_n\}$

  containing points to be inserted that can see f

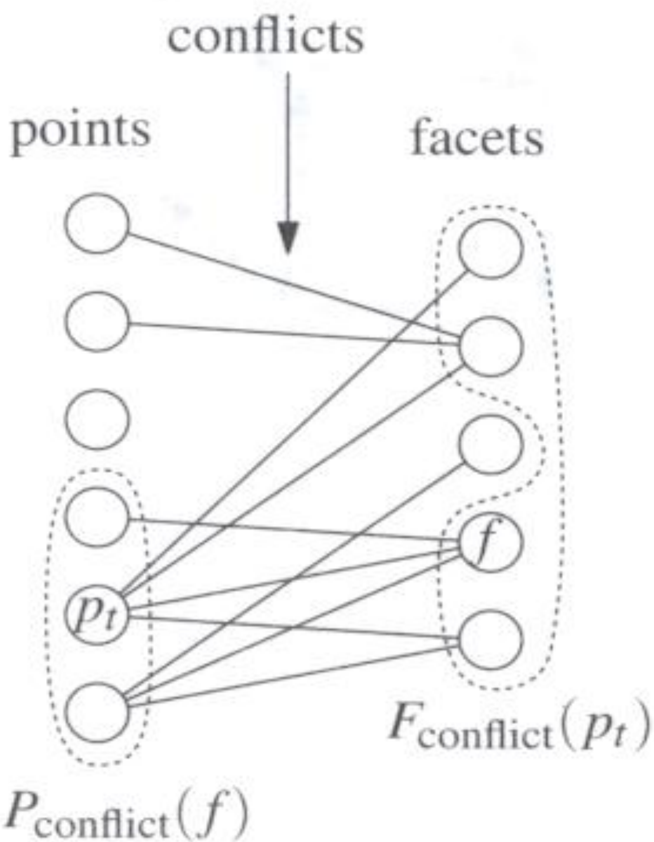
- For each p_t, where $t > r$, maintain

  $F_{\text{conflict}}(p_t)$

  containing facets of $\mathcal{CH}(P_r)$ visible from p_t

- p and f are in ***conflict*** because they cannot coexist on the same convex hull

Conflict Graph $\mathcal{G}$

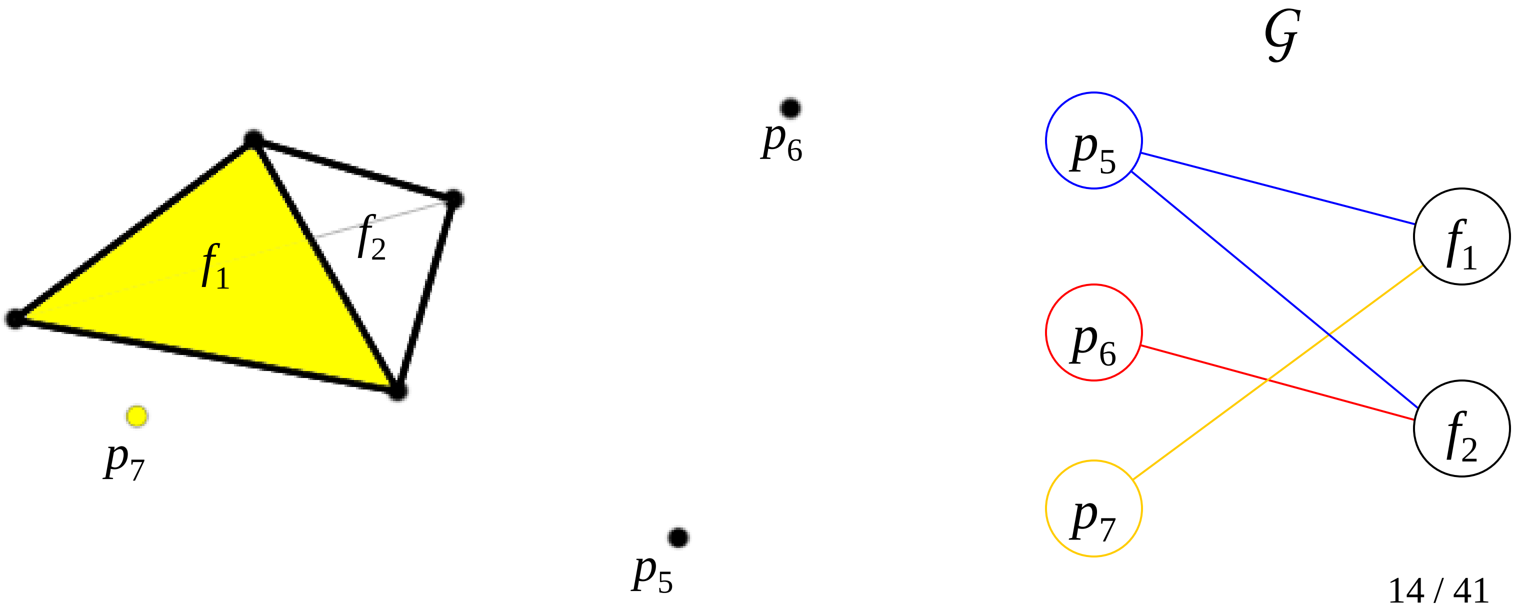


- Bipartite graph
  - pts not yet inserted
  - facets on $\mathcal{CH}(P_r)$
- Arc for every point-facet conflict
- Conflict sets for a point or facet can be returned in linear time

At any step of our algorithm, we know all conflicts between the remaining points and facets on the current $\mathcal{CH}$

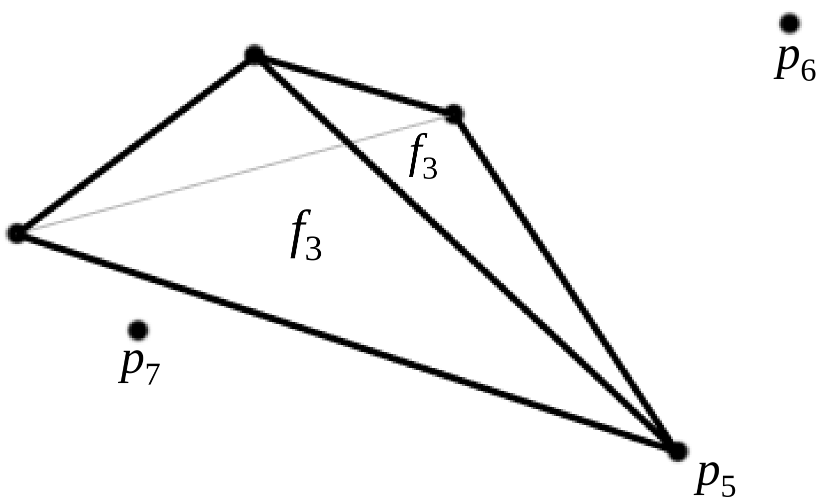
Initializing $\mathcal{G}$

- Initialize $\mathcal{G}$ with $\mathcal{CH}(P_4)$ in linear time
- Walk through $P_{5\text{-}n}$ to determine which facet each point can see

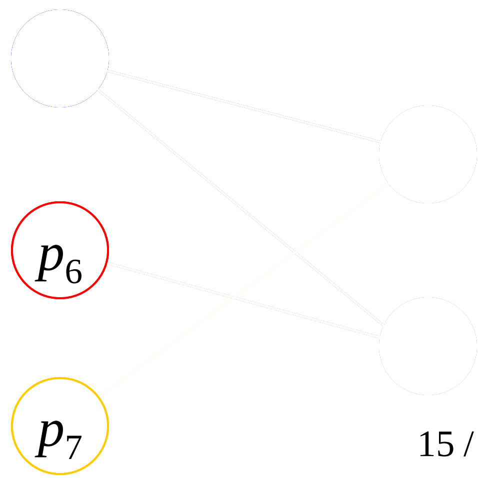


Updating $\mathcal{G}$

- Discard visible facets from p_r by removing neighbors of p_r in $\mathcal{G}$
- Remove p_r from $\mathcal{G}$
- Determine new conflicts

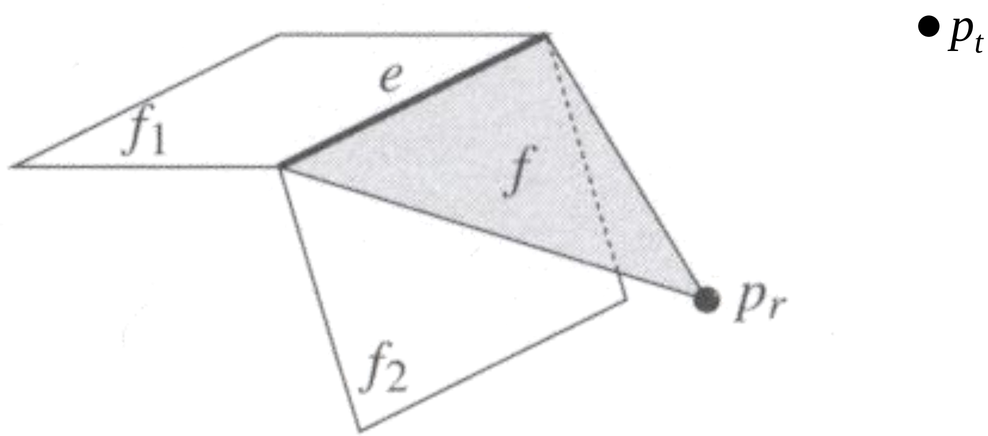


$\mathcal{G}$



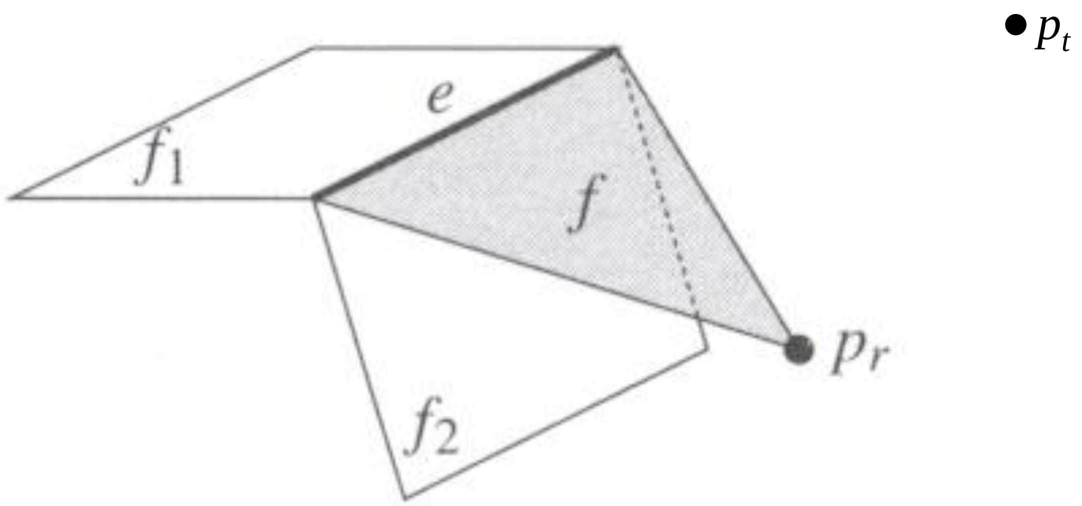
Determining New Conflicts

- If p_t can see new f, it can see edge e of f.
- e on horizon of p_r, so e was already in and visible from p_t in $\mathcal{CH}(P_{r\text{-}1})$
- If p_t sees e, it saw either f_1 or f_2 in $\mathcal{CH}(P_{r\text{-}1})$
- P_t was in $P_{\text{conflict}}(f_1)$ or $P_{\text{conflict}}(f_2)$ in $\mathcal{CH}(P_{r\text{-}1})$



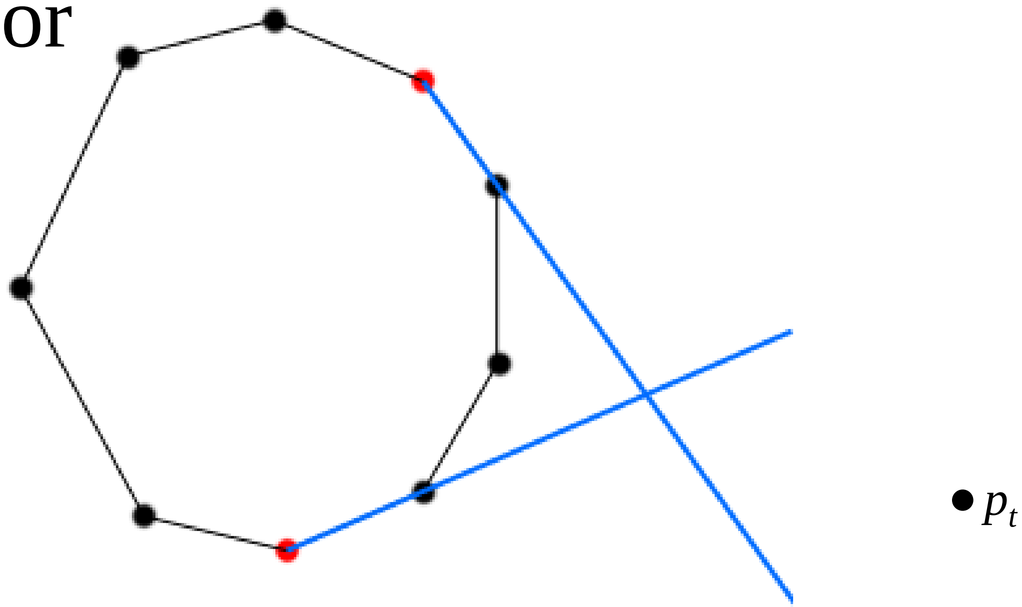
Determining New Conflicts

- Conflict list of f can be found by testing the points in the conflict lists of f_1 and f_2 incident to the horizon edge e in $\mathcal{CH}(P_{r\text{-}1})$



What About the Other Facets?

- $P_{\text{conflict}}(f)$ for any f unaffected by $\boldsymbol{p_r}$ **remains** unchanged
- Deleted facets not on horizon already accounted for

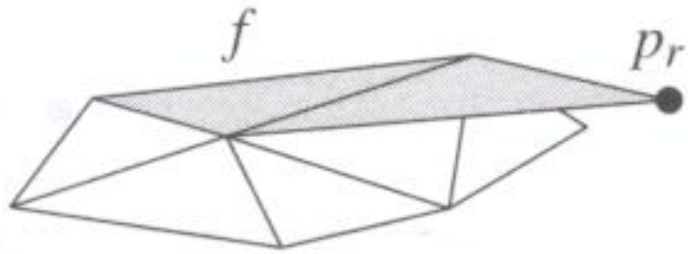


Final Algorithm

- Initialize $\mathcal{CH}(P_4)$ and $\mathcal{G}$
- For each remaining point
  - Determine visible facets for p_r by checking $\mathcal{G}$
  - Remove $F_{\text{conflict}}(p_r)$ from $\mathcal{CH}$
  - Find horizon and add new facets to $\mathcal{CH}$ and $\mathcal{G}$
  - Update $\mathcal{G}$ for new facets by testing the points in existing conflict lists for facets in $\mathcal{CH}(P_{\text{r-1}})$ incident to e on the new facets
  - Delete p_r and $F_{\text{conflict}}(p_r)$ from $\mathcal{G}$

Fine Point

- Coplanar facets
  - p_r lies in the plane of a face of $\mathcal{CH}(P_{r-1})$

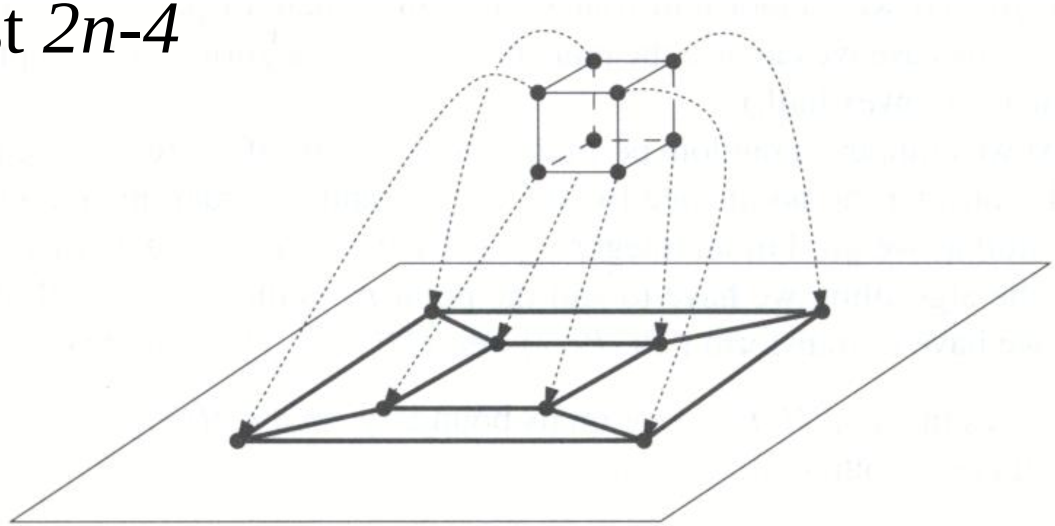


- f is not visible from p_r so we merge created triangles coplanar to f
- New facet has same conflict list as existing facet

Analysis

Complexity

- Complexity of $\mathcal{CH}$ for n points in 3-space is $O(n)$
- Number of edges of a convex polytope with n vertices is at most $3n$-6 and the number of facets is at most $2n$-4



- From Euler's formula: $n - n_e + n_f = 2$

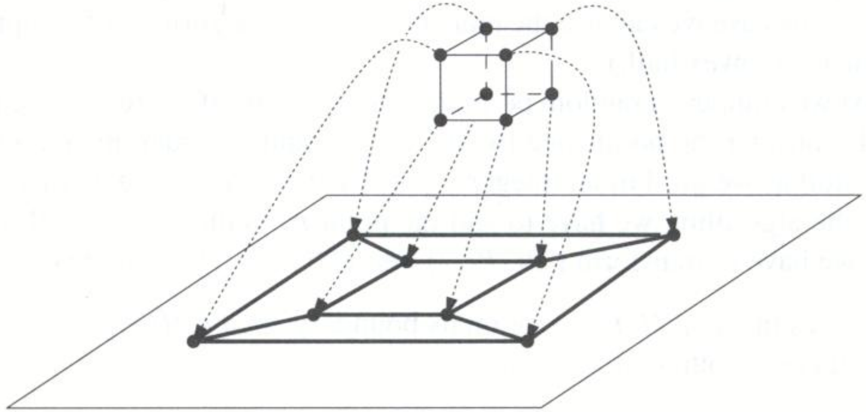
Complexity

- Each face has at least 3 arcs
- Each arc incident to two faces

$$2n_e \geq 3n_f$$

- Using Euler

$$n_f \leq 2n\text{-}4 \qquad\qquad n_e \leq 3n\text{-}6$$



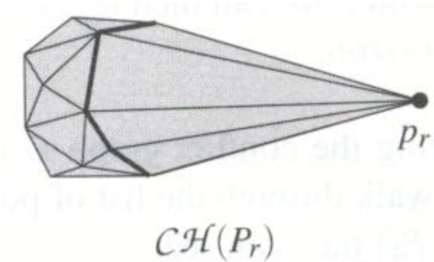
Expected Number of Facets Created

- Will show that expected number of facets created by our algorithm is at most $6n$-20
- Initialized with a tetrahedron = 4 facets

Expected Number of New Facets

- Backward analysis:
  - Remove p_r from $\mathcal{CH}(P_r)$
  - Number of facets removed same as those created by p_r
  - Number of edges incident to p_r in $\mathcal{CH}(P_r)$ is degree of p_r:

$$\deg(p_r, \mathcal{CH}(P_r))$$



Expected Degree of p_r

- Convex polytope of r vertices has at most $3r$-6 edges
- Sum of degrees of vertices of $\mathcal{CH}(P_r)$ is $6r$-12
- Expected degree of p_r bounded by $(6r\text{-}12)/r$

$$
\begin{aligned}
\mathrm{E}[\deg(p_r, \mathcal{CH}(P_r))] &= \frac{1}{r-4} \sum_{i=5}^{r} \deg(p_i, \mathcal{CH}(P_r)) \\
&\leqslant \frac{1}{r-4} \left(\left\{ \sum_{i=1}^{r} \deg(p_i, \mathcal{CH}(P_r)) \right\} - 12 \right) \\
&\leqslant \frac{6r - 12 - 12}{r-4} = 6.
\end{aligned}
$$

Expected Number of Created Facets

- 4 from initial tetrahedron
- Expected total number of facets created by adding $p_5,\ldots,p_n$

$$4+\sum_{r=5}^{n} \mathrm{E}[\deg(p_r, \mathcal{CH}(P_r))] \leqslant 4+6(n-4)=6n-20.$$

Running Time

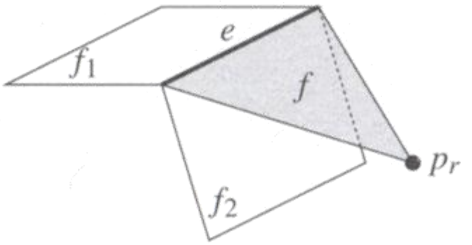
- Initialization $\Rightarrow O(n\log n)$
- Creating and deleting facets $\Rightarrow O(n)$
  - Expected number of facets created is $O(n)$
- Deleting p_r and facets in $F_{\text{conflict}}(p_r)$ from $\mathcal{G}$ along with incident arcs $\Rightarrow O(n)$
- Finding new conflicts $\Rightarrow$ **$O(?)$**

Total Time to Find New Conflicts

- For each edge e on horizon we spend

  $O(\text{card}(P(e))$ time

  where $P(e) \leftarrow P_{\text{confict}}(f_1) \cup P_{\text{conflict}}(f_2)$

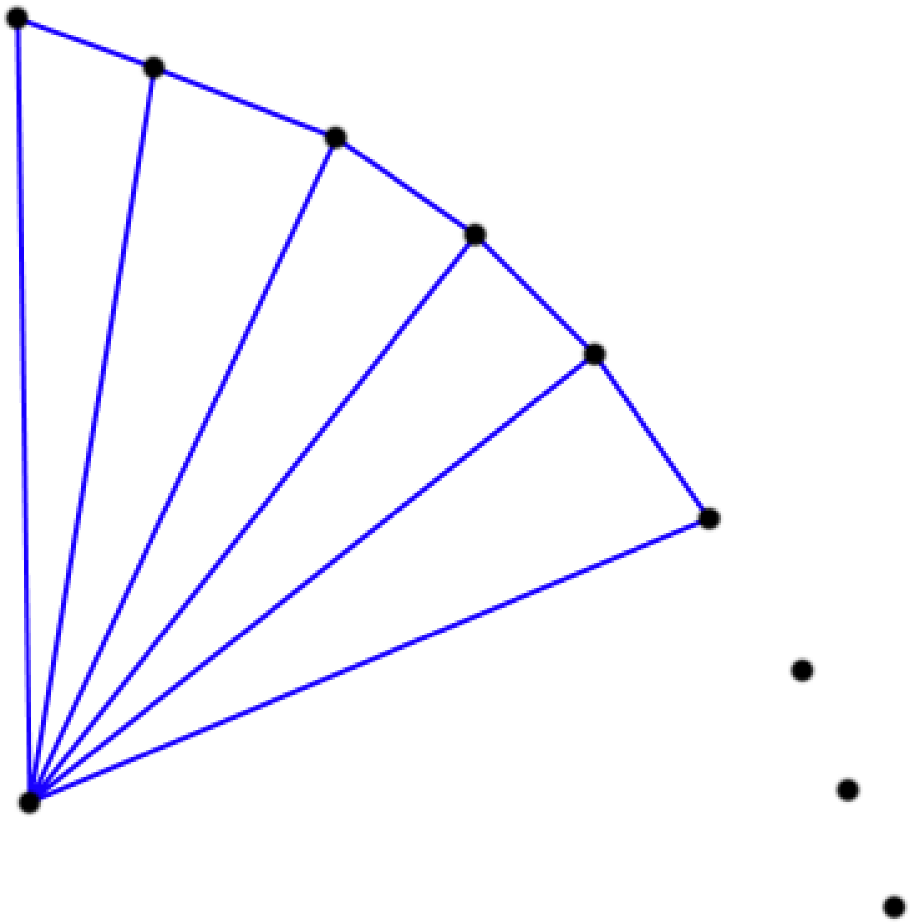


- Total time is $O(\Sigma_{e \in \mathcal{L}} \text{card}(P(e)))$

  bounded by expected value of $\Sigma \text{card}(P(e))$

- **Lemma 11.6** *The expected value of $\Sigma_e card(P(e))$, where the summation is over all horizon edges that appear at some stage of the algorithm is O(nlogn)*

Randomized Insertion Order



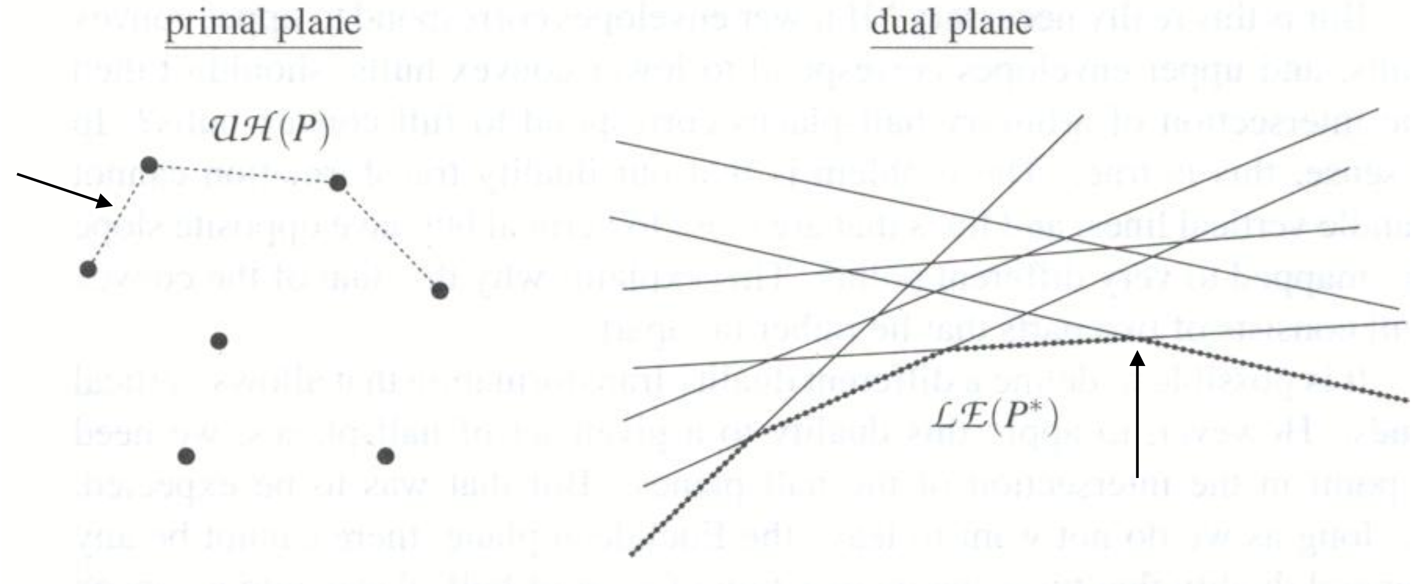
Running Time

- Initialization $\Rightarrow O(n\log n)$
- Creating and deleting facets $\Rightarrow O(n)$
- Updating $\mathcal{G} \Rightarrow O(n)$
- Finding new conflicts $\Rightarrow$ $\boldsymbol{O(n\log n)}$

- Total Running Time is O($n$log$n$)

Convex Hulls in Dual Space

- Upper convex hull of a set of points is essentially the lower envelope of a set of lines (similar with lower convex hull and upper envelope)



Half-Plane Intersection

- Convex hulls and intersections of half planes are dual concepts
- An algorithm to compute the intersection of half-planes can be given by dualizing a convex hull algorithm. *Is this true?*

Half-Plane Intersection

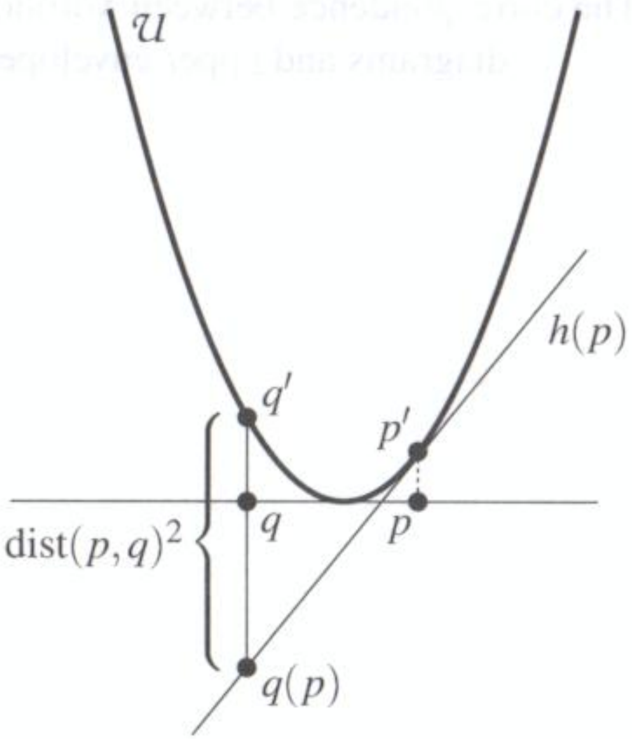
- Duality transform cannot handle vertical lines
- If we do not leave the Euclidean plane, there cannot be any general duality that turns the intersection of a set of half-planes into a convex hull. **Why?**

  Intersection of half-planes can be empty!

  And Convex hull is well defined.
- Conditions for duality:
  - Intersection is not empty
  - Point in the interior is known.

Voronoi Diagrams Revisited

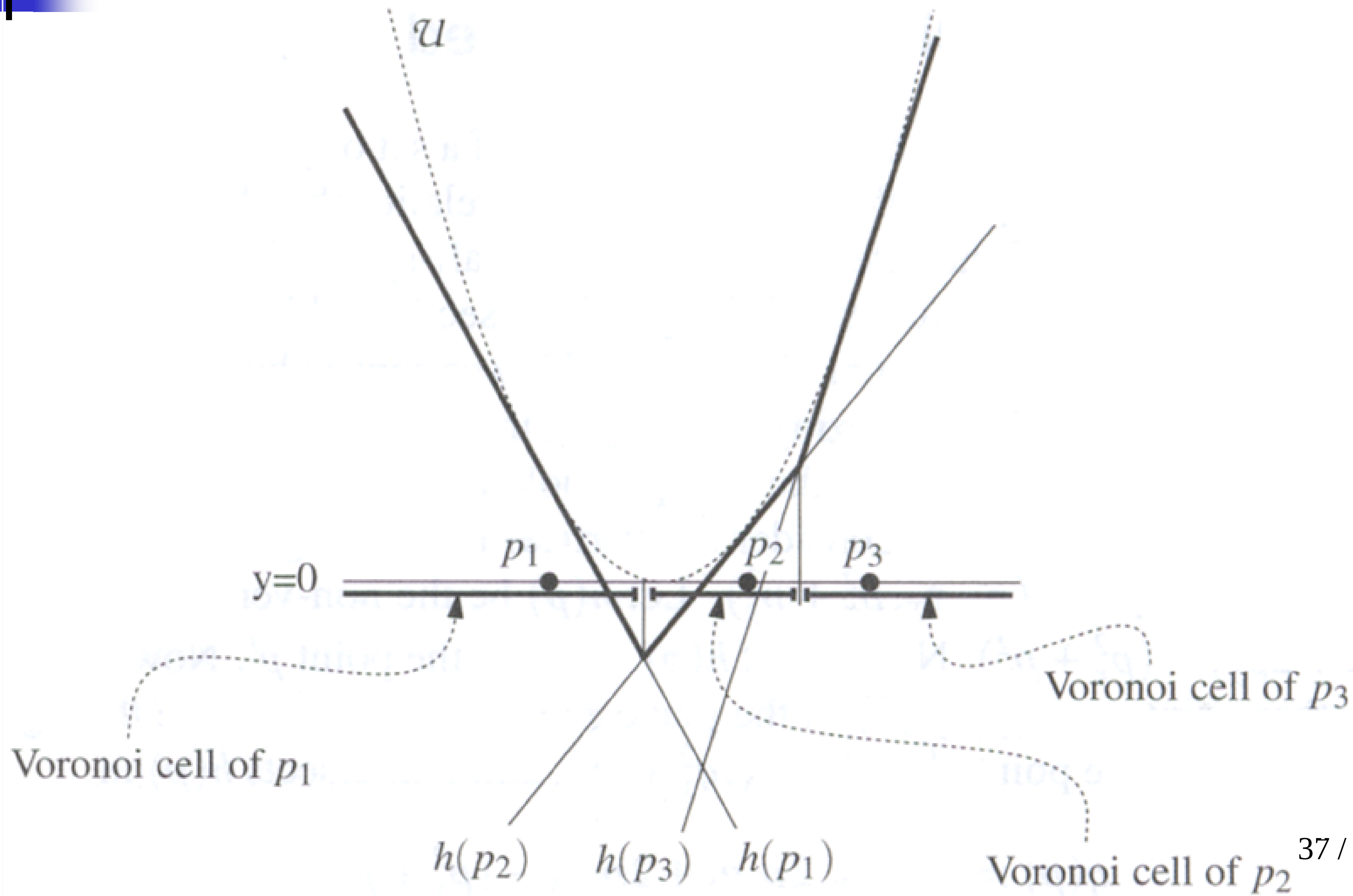
- $U := (z = x^2 + y^2)$
  a paraboloid
- p is point on plane $z=0$
- $h(p)$ is non-vert plane $z = 2p_x x + 2p_y y - (p^2_{x+} p^2_y)$
- q is any point on $z=0$
- $vdist(q', q(p)) = dist(p,q)^2$
- $h(p)$ and paraboloid encodes any distance p to any point on $z=0$



Voronoi Diagrams

- $H := \{h(p) \mid p \in P\}$
- $\mathcal{UE}(H)$ upper envelope of the planes in H
- Projection of $\mathcal{UE}(H)$ on plane z=0 is Voronoi diagram of P

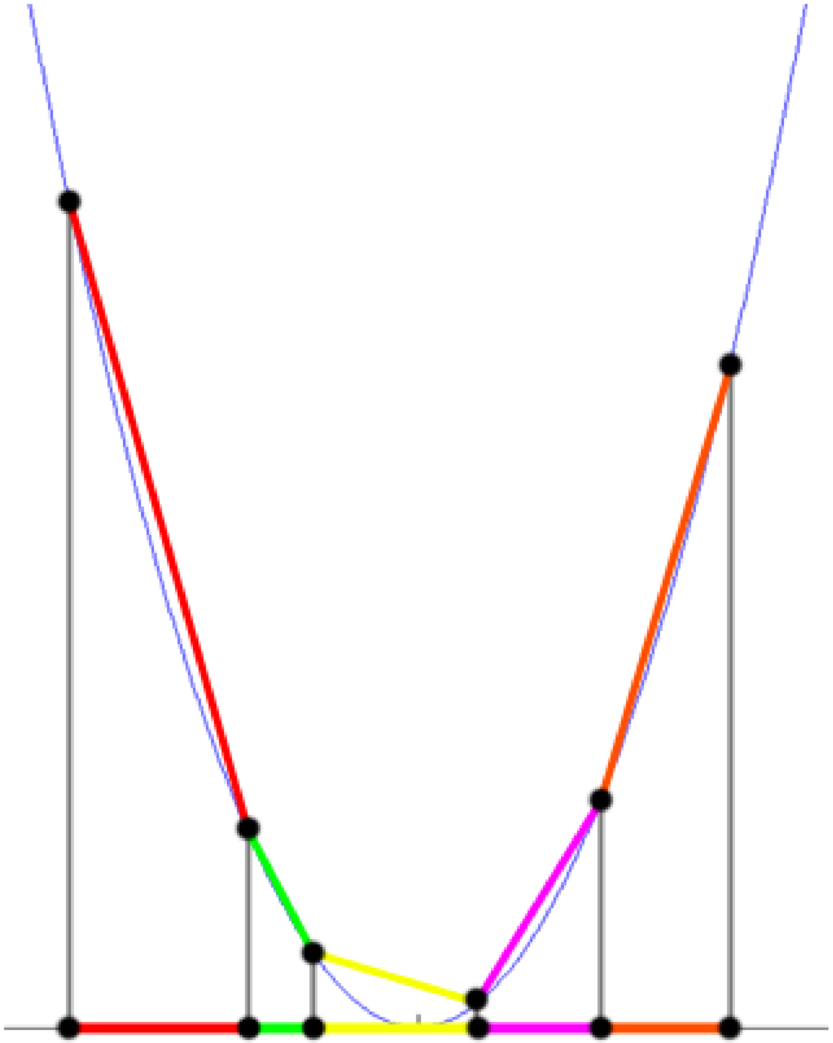
Simplified Case



Demo

- /mit/6.837/voronoi/voronoi

Delaunay Triangulations from $\mathcal{CH}$



Higher Dimensional Convex Hulls

- *Upper Bound Theorem*:
  The worst-case combinatorial complexity of the convex hull of n points in d-dimensional space is $\Theta(n^{\lfloor d/2 \rfloor})$.

- Our algorithm generalizes to higher dimensions with expected running time of $\Theta(n^{\lfloor d/2 \rfloor})$

Higher Dimensional Convex Hulls

- Best known output-sensitive algorithm for computing convex hulls in R^d is:

$$O(n\log k + (nk)^{1-1/(\lfloor d/2 \rfloor+1)}\log^{O(n)})$$

where k is complexity