CS633 Lecture 12
Convex hulls

Jyh-Ming Lien

Department of Computer Science
George Mason University

Based on Jason C. Yang's note

Dec 03, 2008

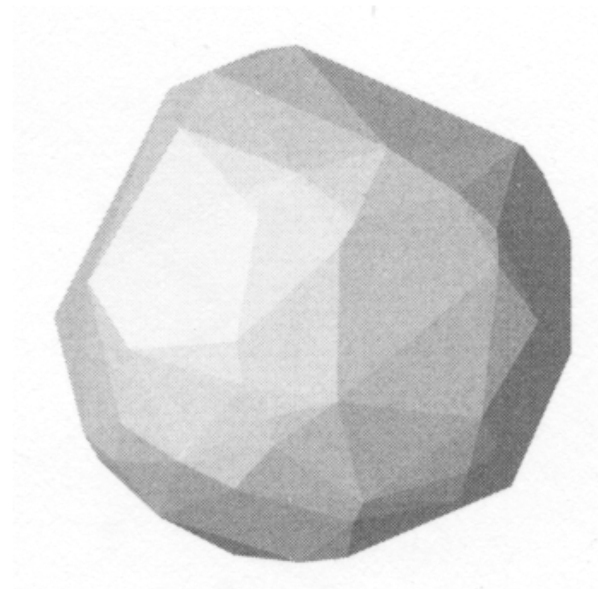
Outline

- 3D Convex hull
- Put it all together
  - Convex hull vs. half-space intersection
  - Convex hull vs. Delaunay & Voronoi diagrams
  - Voronoi diagram vs. Line arrangements
  - Convex hull vs. Line arrangements

Problem Statement

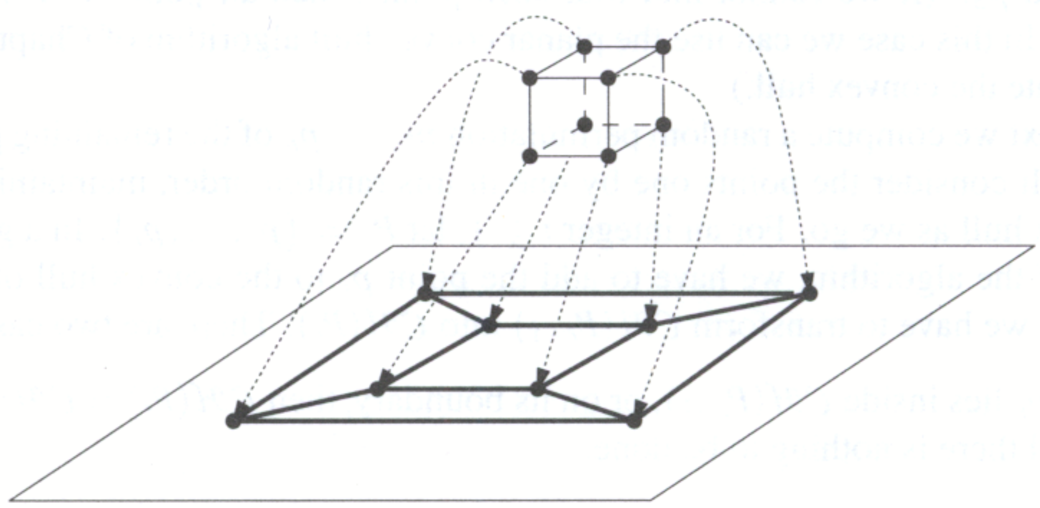
- Given P: set of n points in 3-space

- Return:
  - Convex hull of P: $\mathcal{CH}(P)$
  - Smallest convex object s.t. all elements of P on or in the interior of $\mathcal{CH}(P)$.



Complexity

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



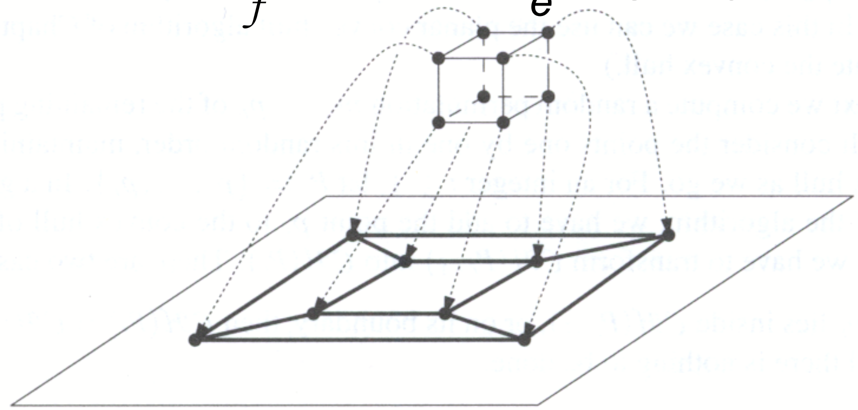
Complexity

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

$$(2/3)n_e \geq n_f$$

- Using Euler's formula (|V|-|E|+|F|=2)

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



This remains true for any polyhedron without handles

Algorithm

- Randomized incremental algorithm

- Steps:
  - Initialize the algorithm
  - Loop over remaining points
    Add p_r to the convex hull of P_{r-1} to transform
    $\mathcal{CH}(P_{r-1})$ 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 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
  - Walk 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?

- 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:

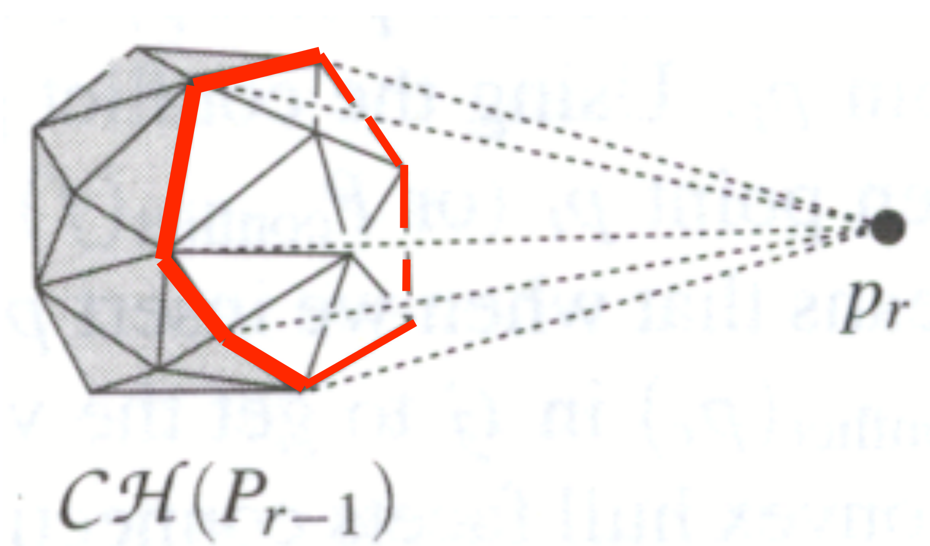
  1) P_r is inside or on the boundary of $\mathcal{CH}(P_{r\text{-}1})$

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

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

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

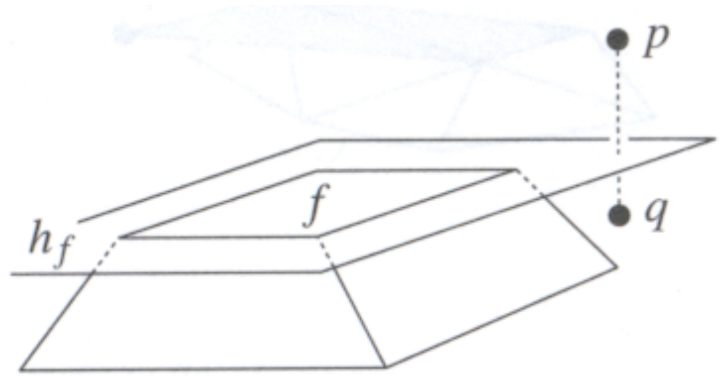
- Determine ***horizon*** of p_r on $\mathcal{CH}(P_{r\text{-}1})$
  - Closed curve of edges enclosing the ***visible*** region of p_r on $\mathcal{CH}(P_{r\text{-}1})$



Visibility

- Consider plane h_f containing a facet f of $\mathcal{CH}(P_{r-1})$

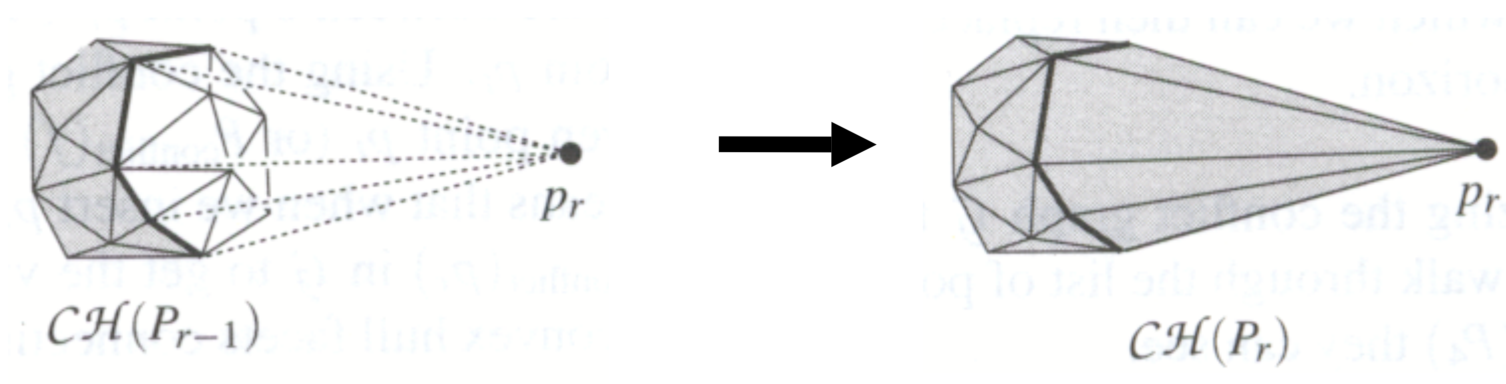
- f is ***visible*** from a point if that point lies in the open half-space on the other side of $\boldsymbol{h_f}$



f is visible from p, but not from q

$\mathcal{CH}(\boldsymbol{P}_{r\text{-}1}) \rightarrow \mathcal{CH}(\boldsymbol{P}_r)$

- Remove ***visible*** facets from $\mathcal{CH}(P_{r\text{-}1})$
- Find ***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 $\mathcal{CH}(P_{r-1})$ then ? ?
    - Determine visible region
    - Find horizon and remove visible facets
    - Add new facets by connecting each horizon edge to p_r

How to Find Visible Region

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

- 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

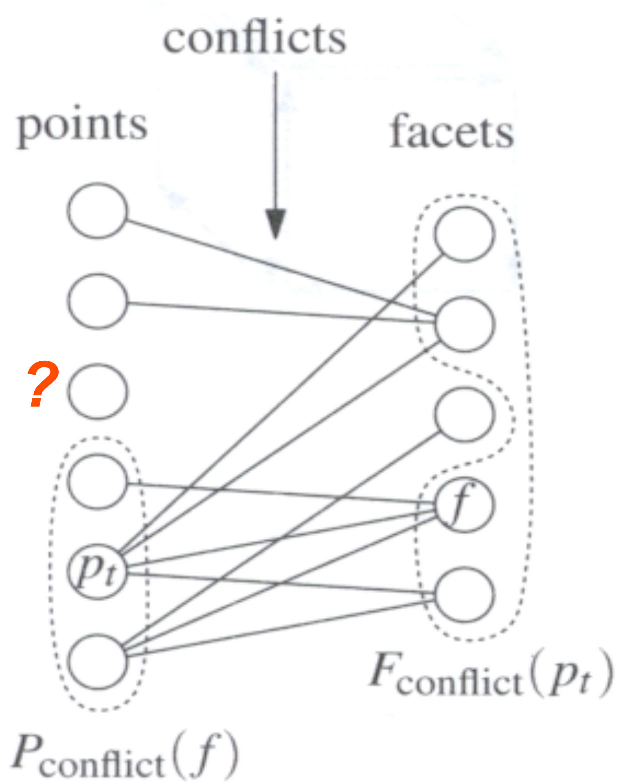
- For each p_t, where $t > r$, maintain

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

  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 $\mathcal{G}$

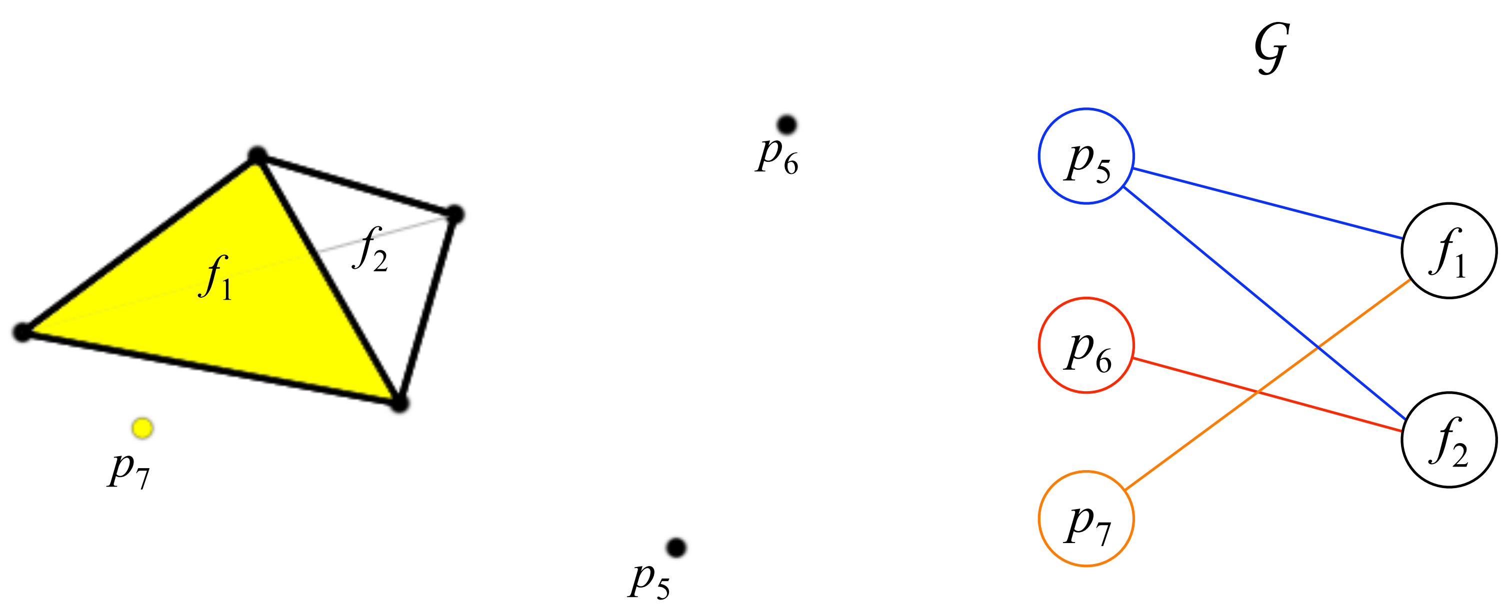


- Bipartite graph
  - pts not yet inserted
  - facets on $\mathcal{CH}(P_r)$
- Arc for every point-facet conflict
- Conflict sets for a point or facet can be returned in linear time

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

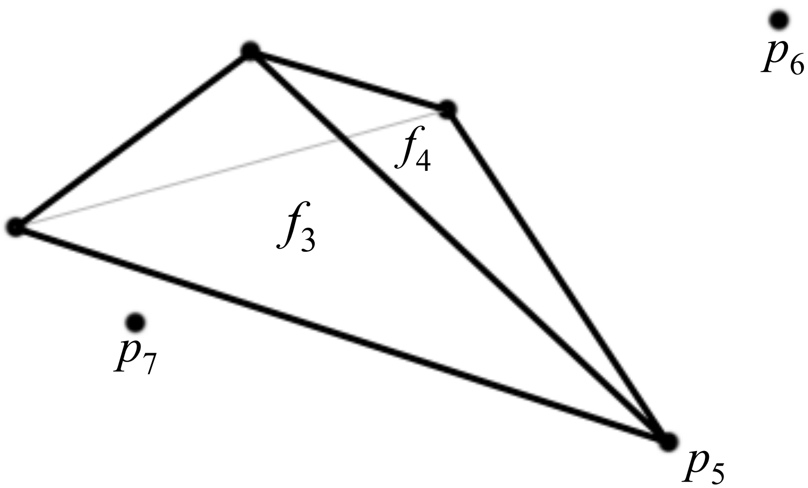
Initializing $\mathcal{G}$

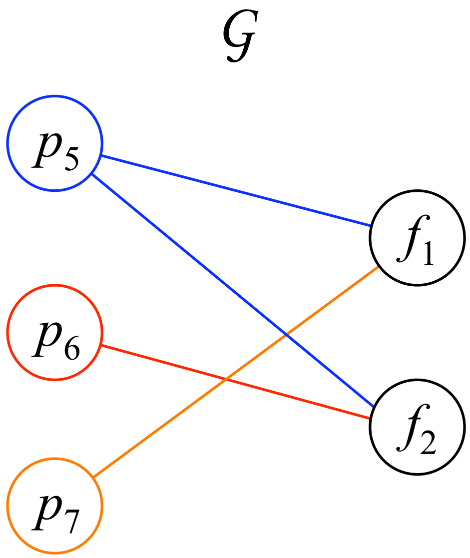
- Initialize $\mathcal{G}$ with $\mathcal{CH}(P_4)$ in linear time
- Walk through $P_{5\sim n}$ to determine which facet each point can see



Updating $\mathcal{G}$

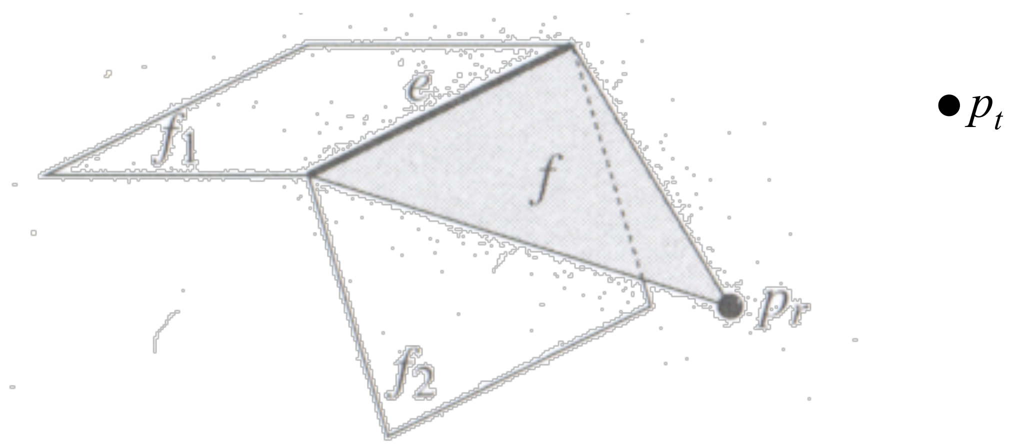
- Discard visible facets from p_r by removing neighbors of p_r in $\mathcal{G}$
- Remove p_r from $\mathcal{G}$
- Determine new conflicts





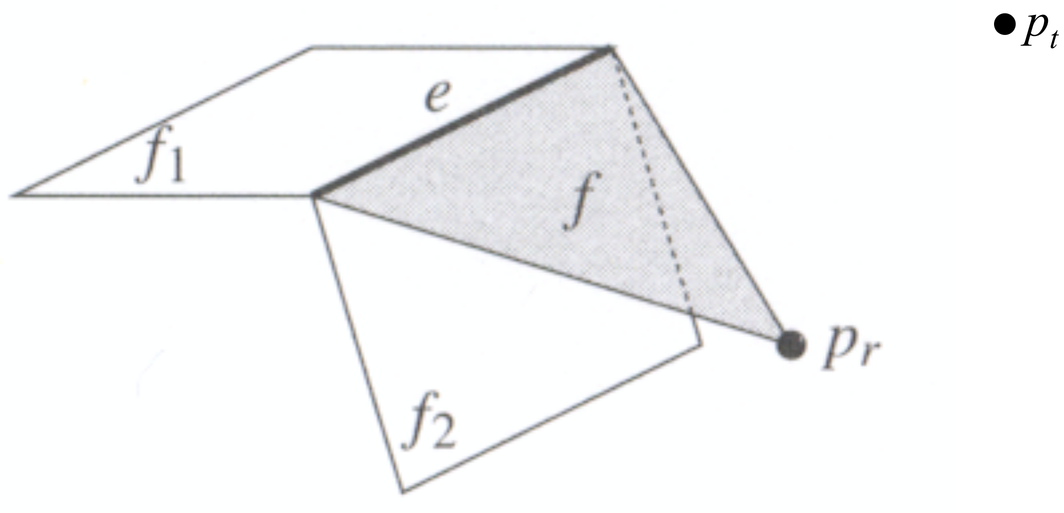
Determining New Conflicts

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



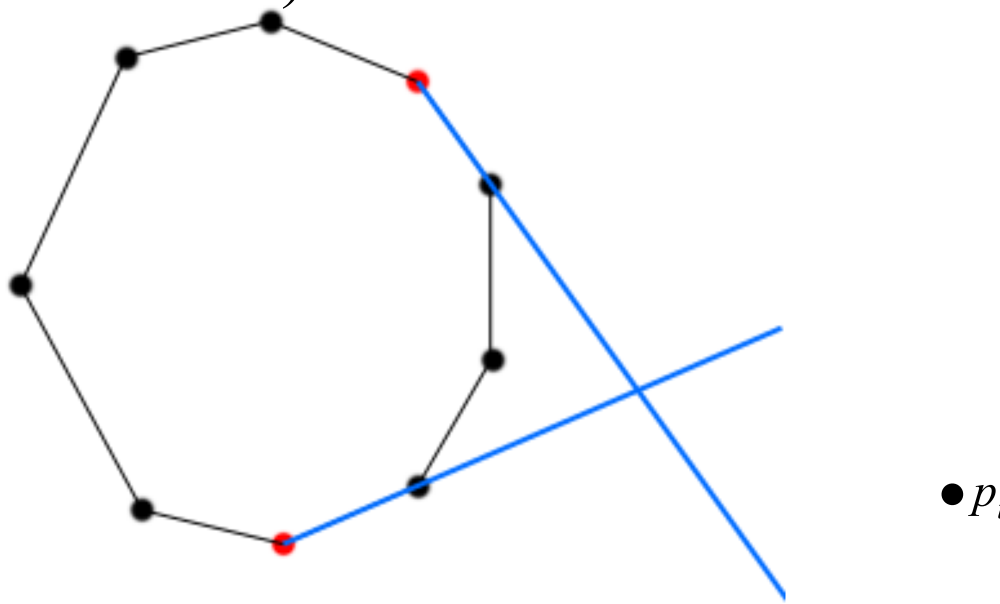
Determining New Conflicts

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



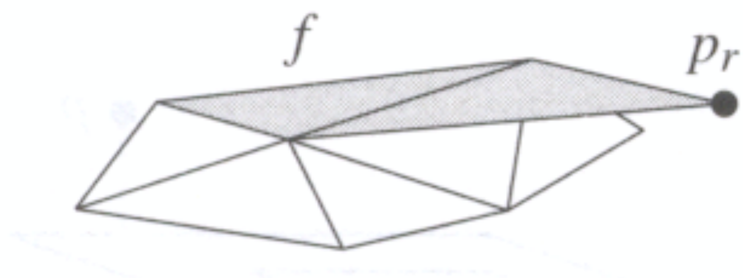
What About the Other Facets?

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



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

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 $\mathcal{CH}(P_{r-1})$ incident to e on the new facets
  - Delete p_r and $F_{\text{conflict}}(p_r)$ from $\mathcal{G}$

Expected Number of Facets Created

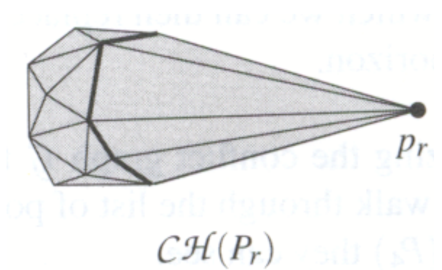
- Will show that expected number of facets created by our algorithm is at most $6n$-20

- 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 $p_5, \ldots, p_n$

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

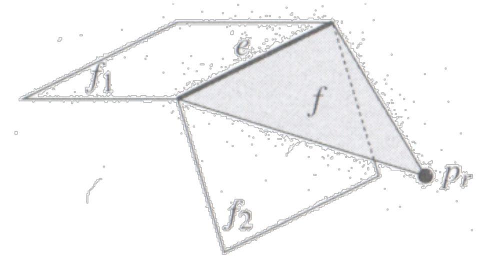
Running Time

- 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 $P(e) \leftarrow P_{\text{confict}}(f_1) \cup P_{\text{conflict}}(f_{12})$



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

Running Time

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

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

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 $\Theta(n^{\lfloor d/2 \rfloor})$

Outline

- 3D Convex hull

- Put it all together
  - Convex hull vs. half-space intersection
  - Convex hull vs. Delaunay & Voronoi diagrams
  - Voronoi diagram vs. Line arrangements
  - Convex hull vs. Line arrangements

Half-Plane Intersection

- Convex hulls and intersections of half planes are dual concepts
- To compute the intersection of half-planes, we can
  - Convert planes into points in dual space
  - Compute a convex hull in dual space
  - Convert the convex hull back to primal space
- Will this always work?

Half-Plane Intersection

- 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?

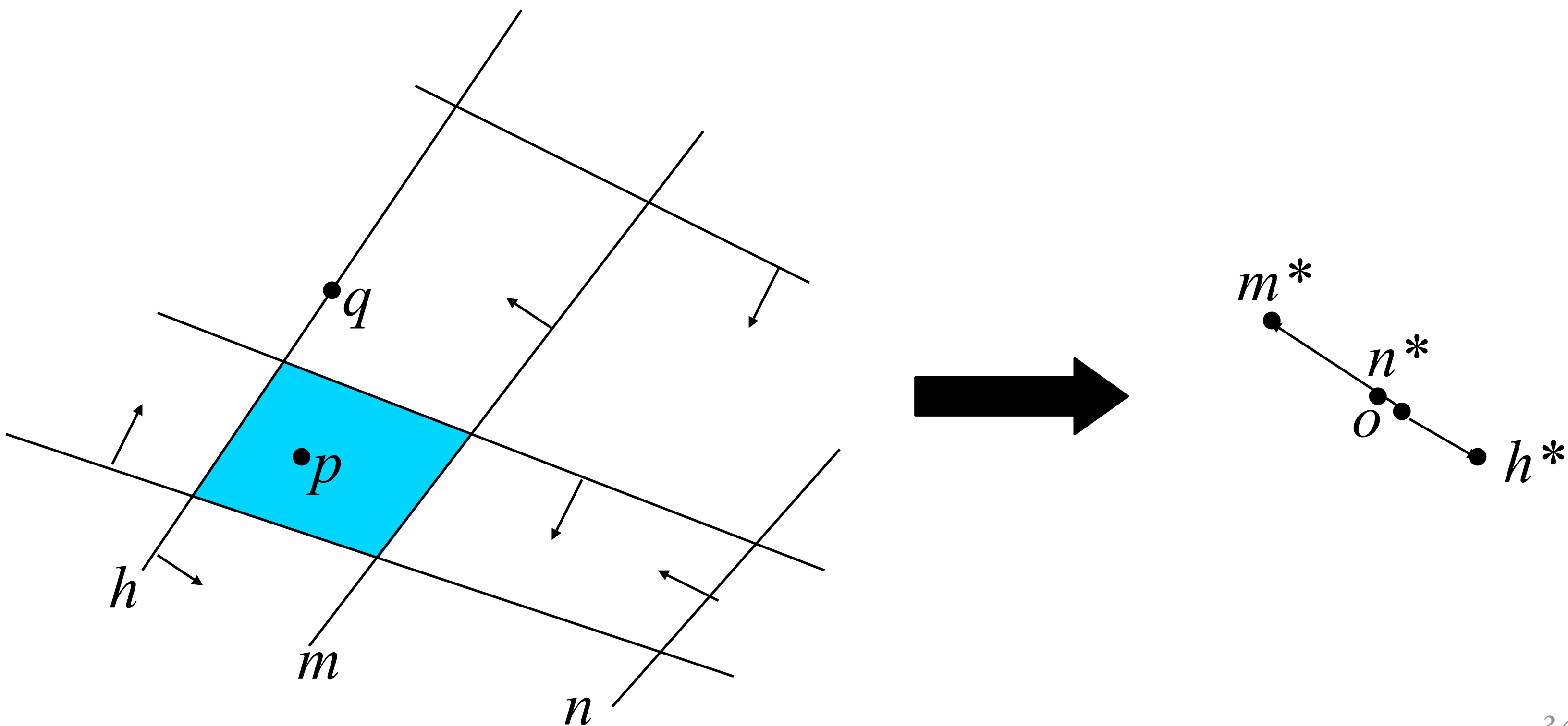
  Intersection of half-planes can be empty!

  And Convex hull is well defined.

- **Conditions for duality:**
  - Intersection is not empty
  - At least one point in the interior is known.

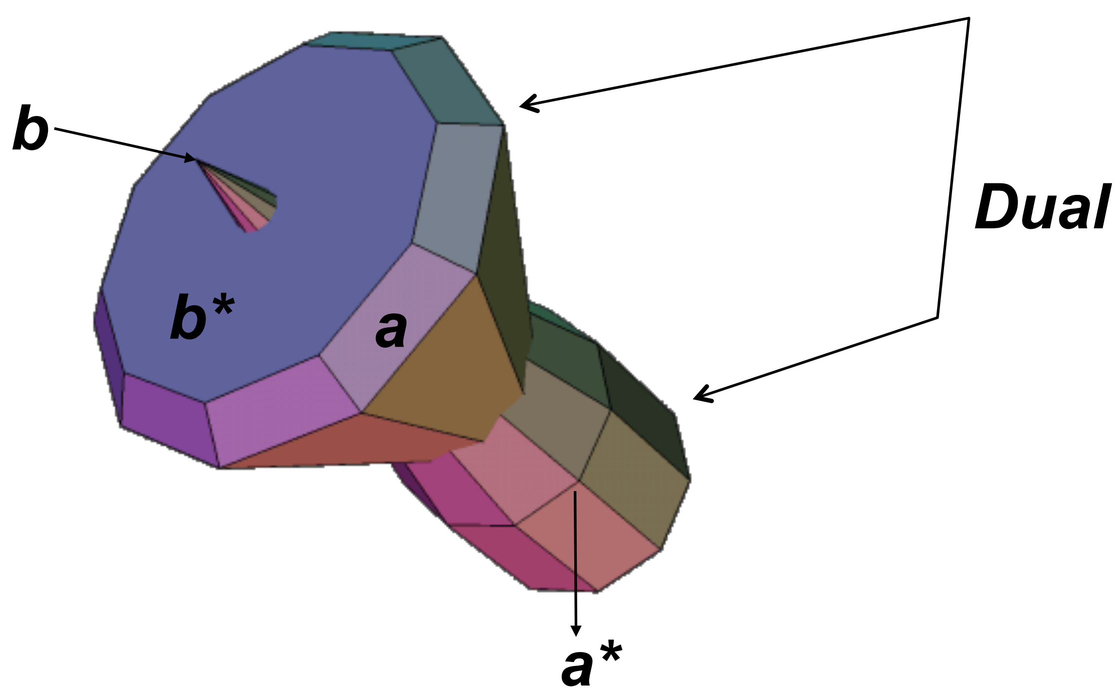
Example

- Given a halfspace $h=\{n_q, q\}$
- Given a point p in the intersection
- Point $h^* = n_q/((q-p) \bullet n_q)$



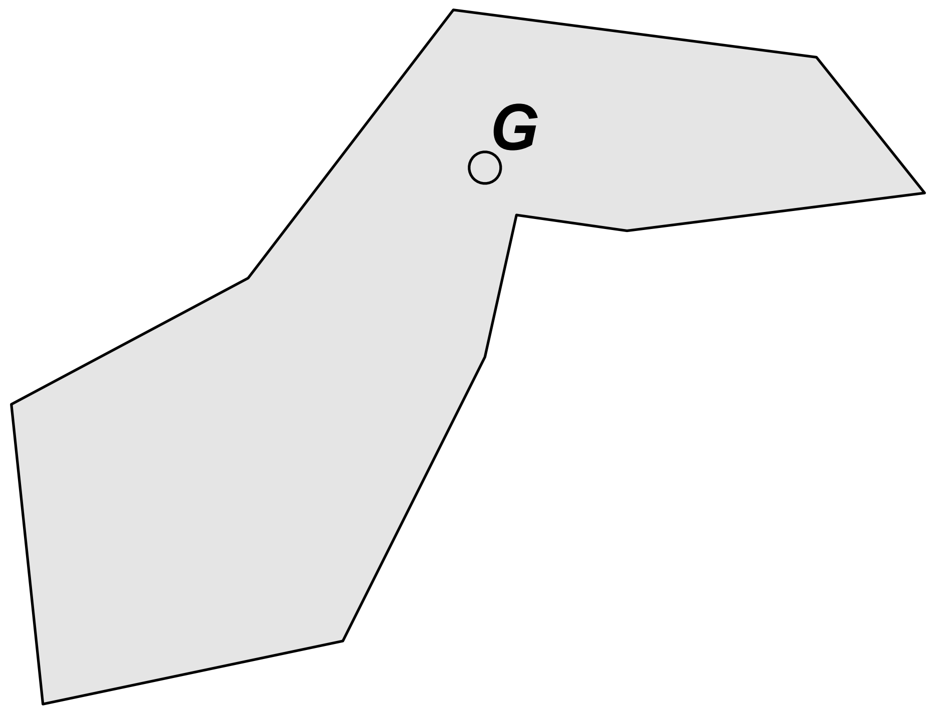
Example

- Image from *qhull*
  - Each facet becomes a vertex
  - Each vertex becomes a facet

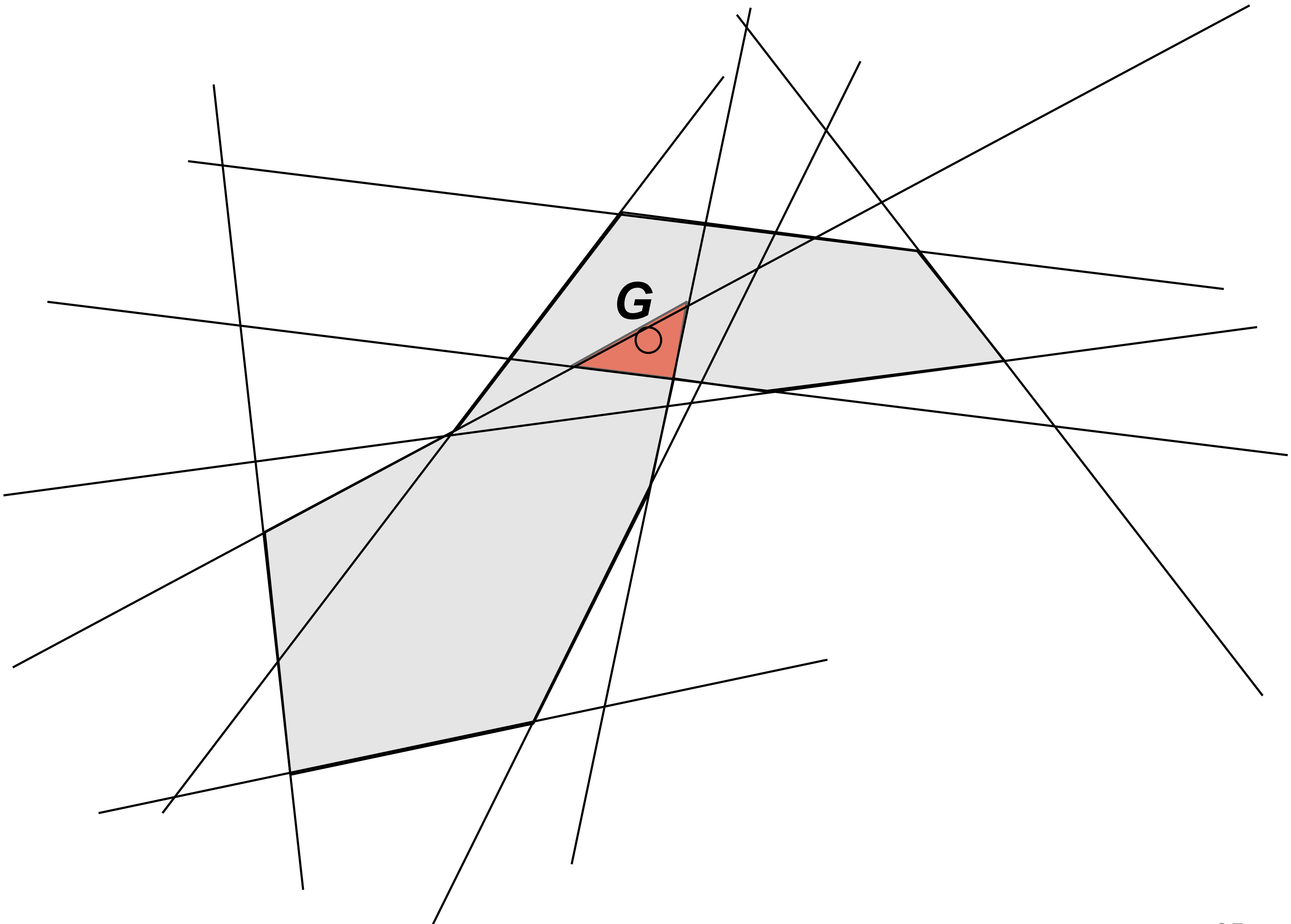


Kernel/Art Gallery Problem

- Kernel of a polygon is a set of points that can see very points in the polygon
  - Given a point G that is in the kernel, we can compute the kernel

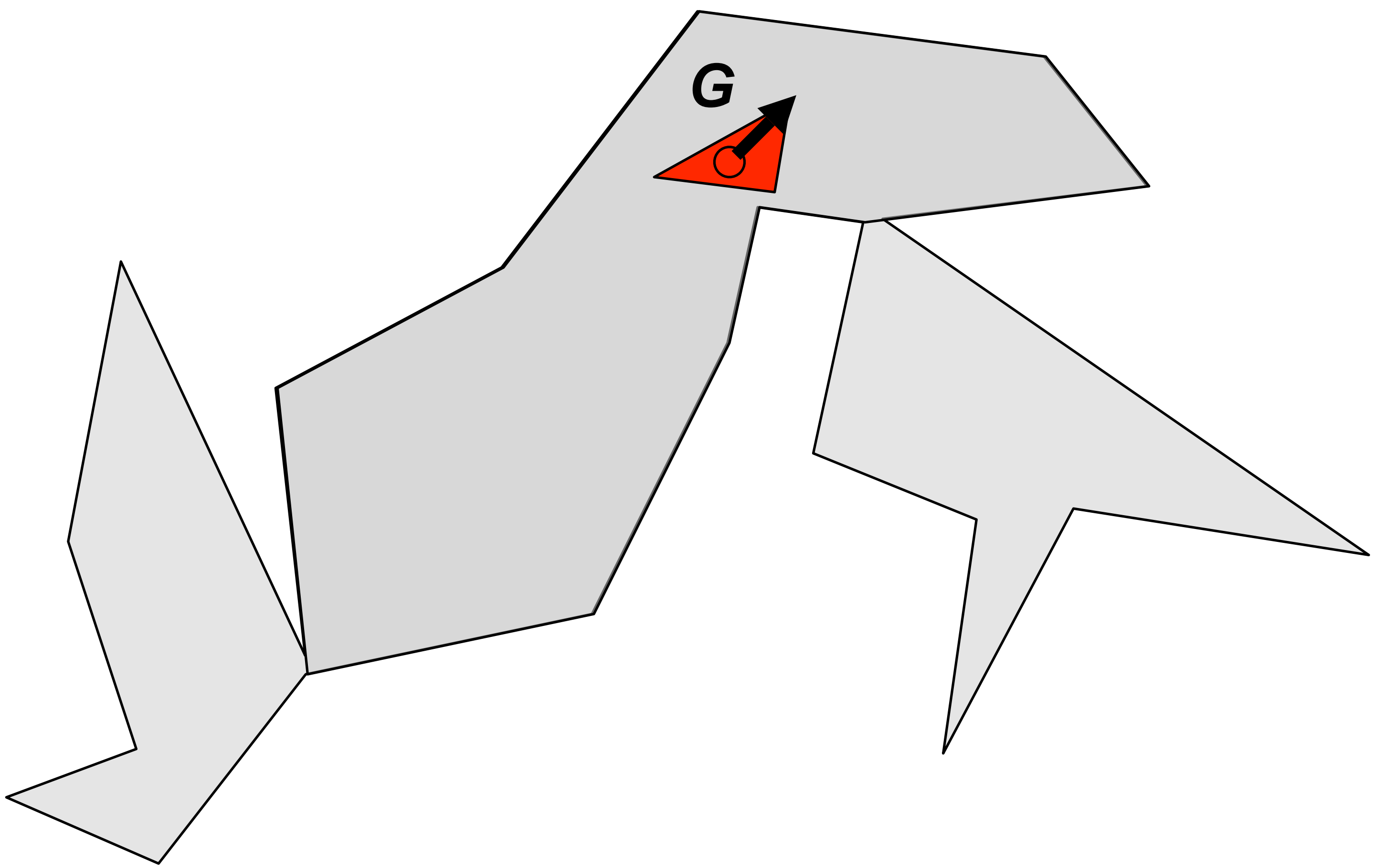


Kernel/Art Gallery Problem



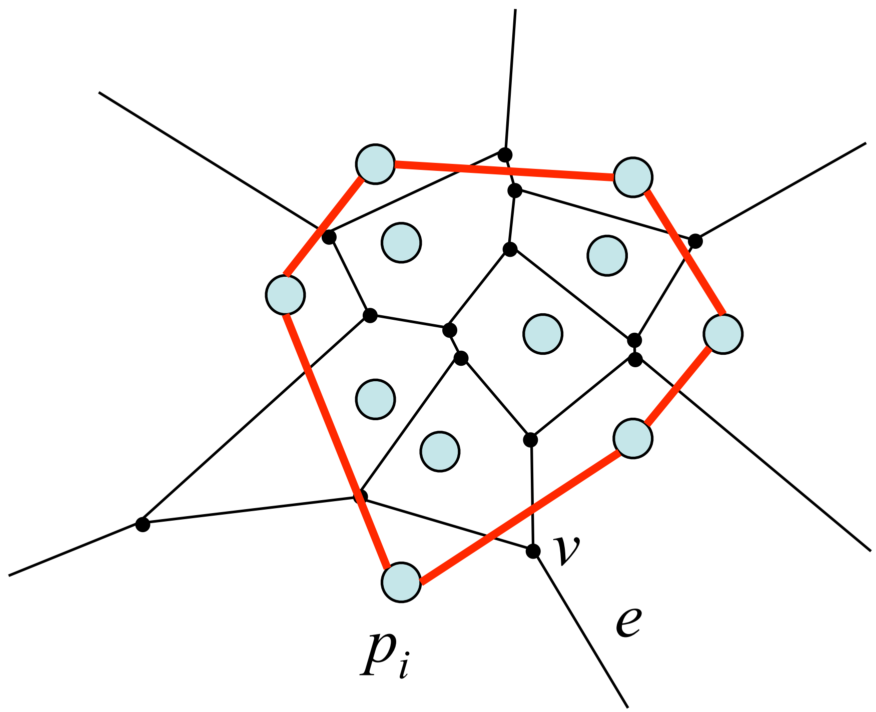
Kernel/Art Gallery Problem

Any point in the kernel can see the same or even more than G



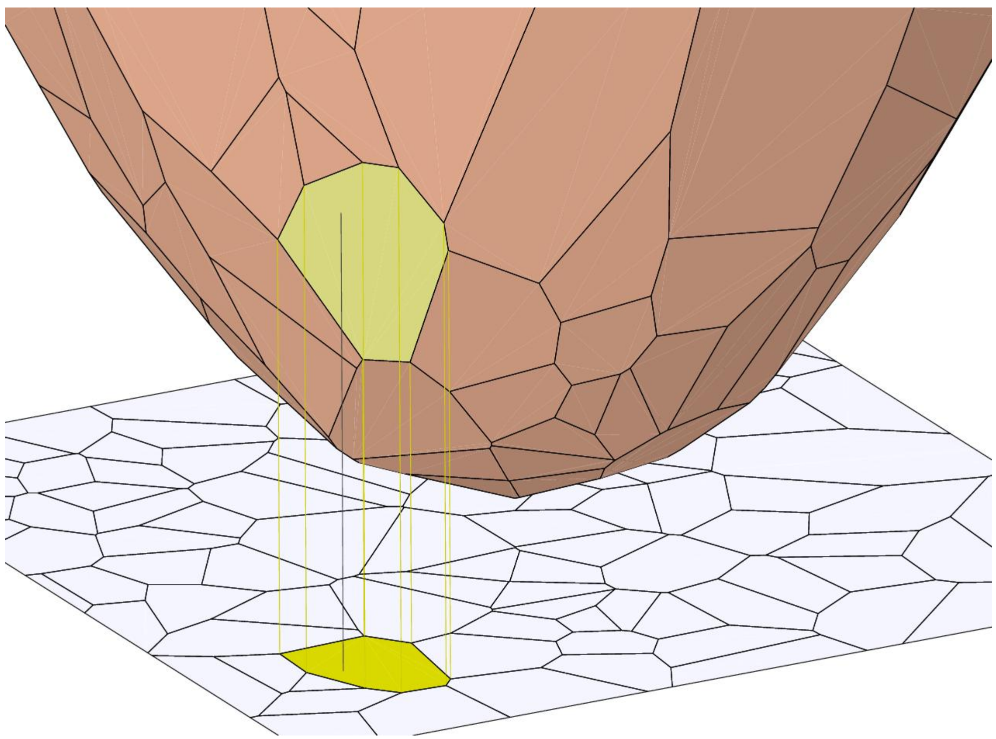
Convex hull vs. Voronoi/Delaunay

- Convex hull can be computed from Voronoi/Delaunay diagrams



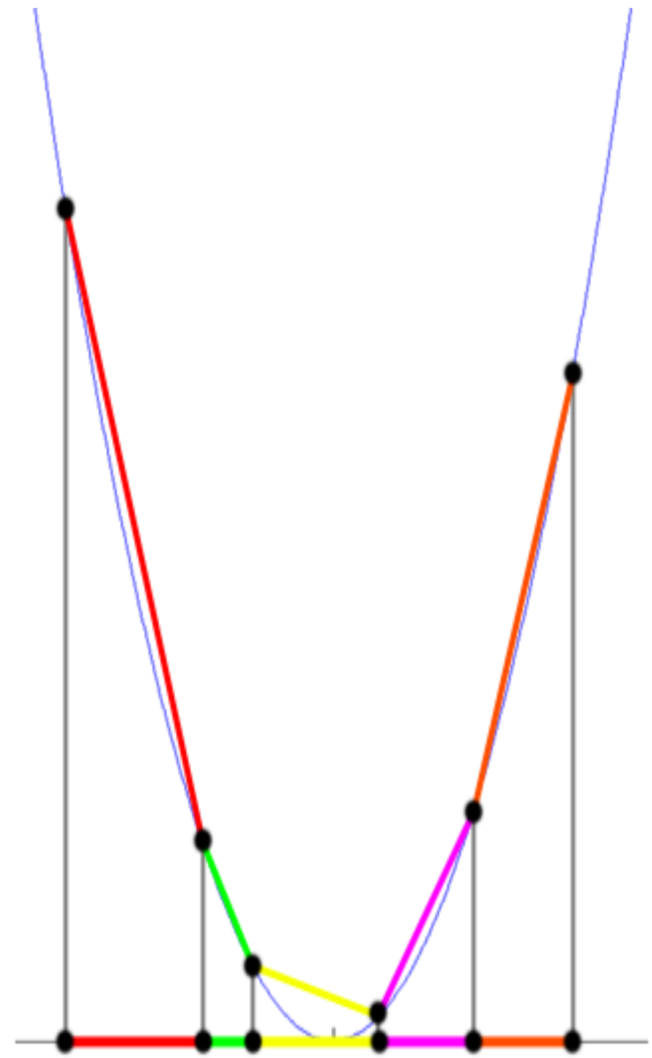
Convex hull vs. Voronoi/Delaunay

- k-d Voronoi/Delaunay diagrams can be computed from $(k+1)$-d convex hull



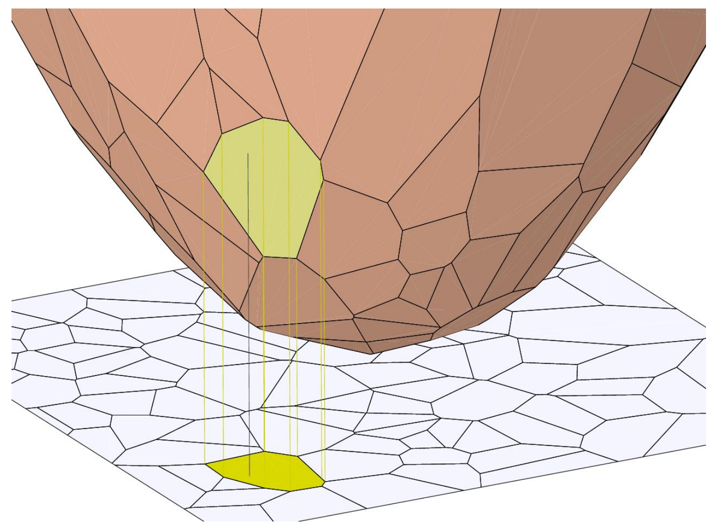
1D Delaunay Triangulation

- Input P=$\{x_1, x_2, \ldots x_n\}$
- U:=(y=x^2) a parabola
- Lift every point to U, i.e.,
  - $p^*_i=\{x_i, x_i^2\}$
- Compute the convex hull of P*
- Project the connectivity down



2D Delaunay Triangulation

- Input $P=\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- $U := (z = x^2 + y^2)$ a parabolid
- Lift every point to U, i.e.,
  - $p^*_i = \{x_i, y_i, x_i^2 + y_i^2\}$
- Compute the convex hull of P^*
- Project the connectivity down

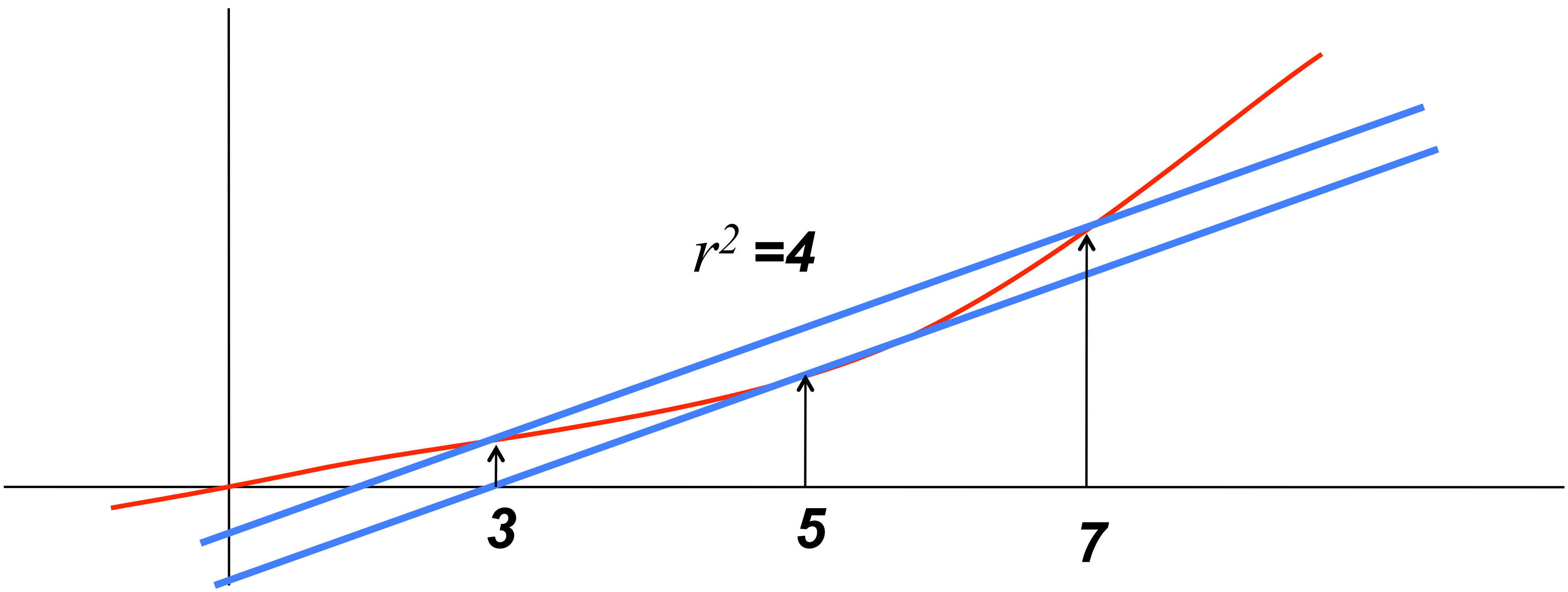


Why does it work?

- $p^*=\{p, p^2\}$
- Tangent line at p^*
  - Slop is $2p$
  - $y=2p(x-p)- p^2$
- Raise the tangent line by r^2
  - $y=2p(x-p)- p^2 +r^2$
  - What's the intersection between the line and the curve?
    - $x^2=2p(x-p)- p^2 +r^2$

    $\Rightarrow x=p\pm r$

Why does it work?



Why does it work?

- $p^*=\{a, b, a^2+b^2\}$
- Tangent plane at p^*
  - Slop is $(1, 1, 2a+2b)$
  - $z=2ax+2by-(a^2+b^2)$
- Raise the tangent plane by r^2
  - $z=2ax+2by-(a^2+b^2)+r^2$
  - What's the intersection between the line and the curve?
    - $x^2+y^2=2ax+2by-(a^2+b^2)+r^2$

    $\Rightarrow (x-a)^2+(y-b)^2=r^2$

Why does it work?

- Let h be a tangent plane for a facet of the convex hull
- Lower h so that h tangents paraboloid, call h^*
- Let the distance between h and h^* be r^2
- Now, we know
  - The projection of the intersection of h and the parabloid is a circle with radius r
- The circle is empty because
  - All other points are above h
  - All other points are more than r^2 distance away from h^*
- The project of the tangent point is a vertex of the Voronoi diagram

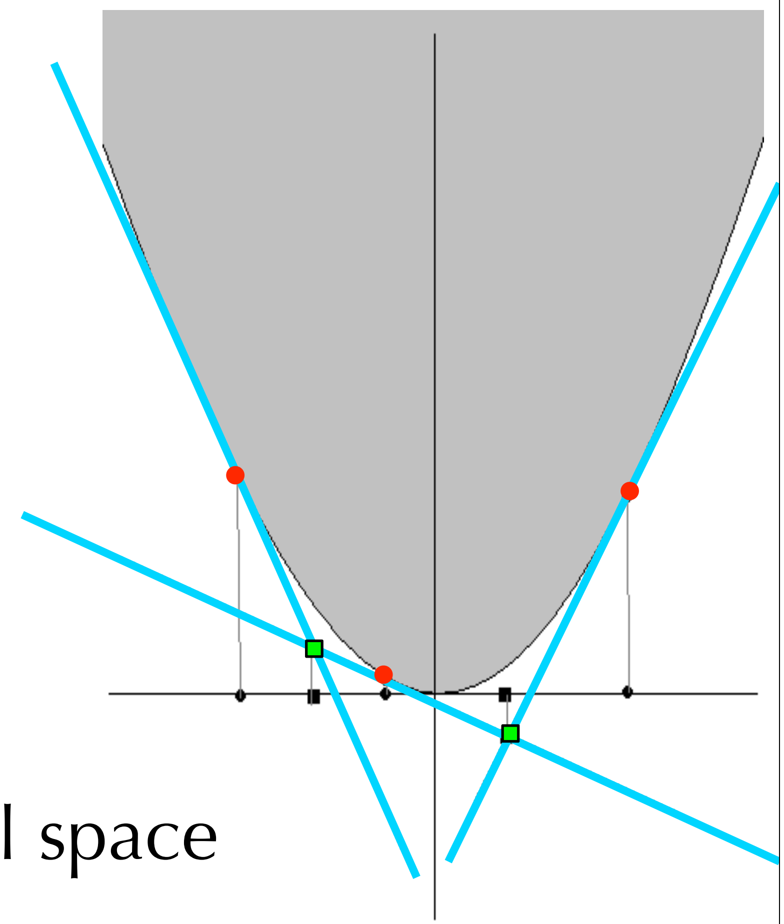
Voronoi vs. Arrangement

- x_1 and x_2

- Line x_1*: $y=2\,x_1 x - x_1^2$
- Line x_2*: $y=2\,x_2 x - x_2^2$

- Intersection
  - $2\,x_1 x - x_1^2 = 2\,x_2 x - x_2^2$

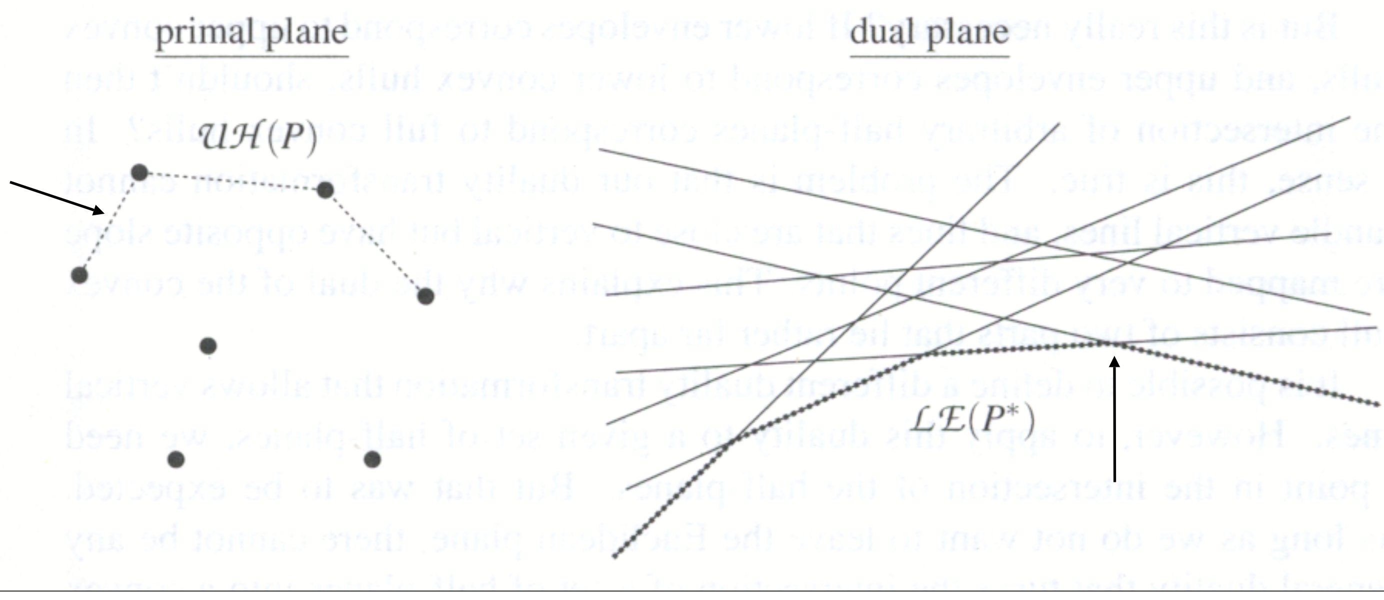
Why does it work?



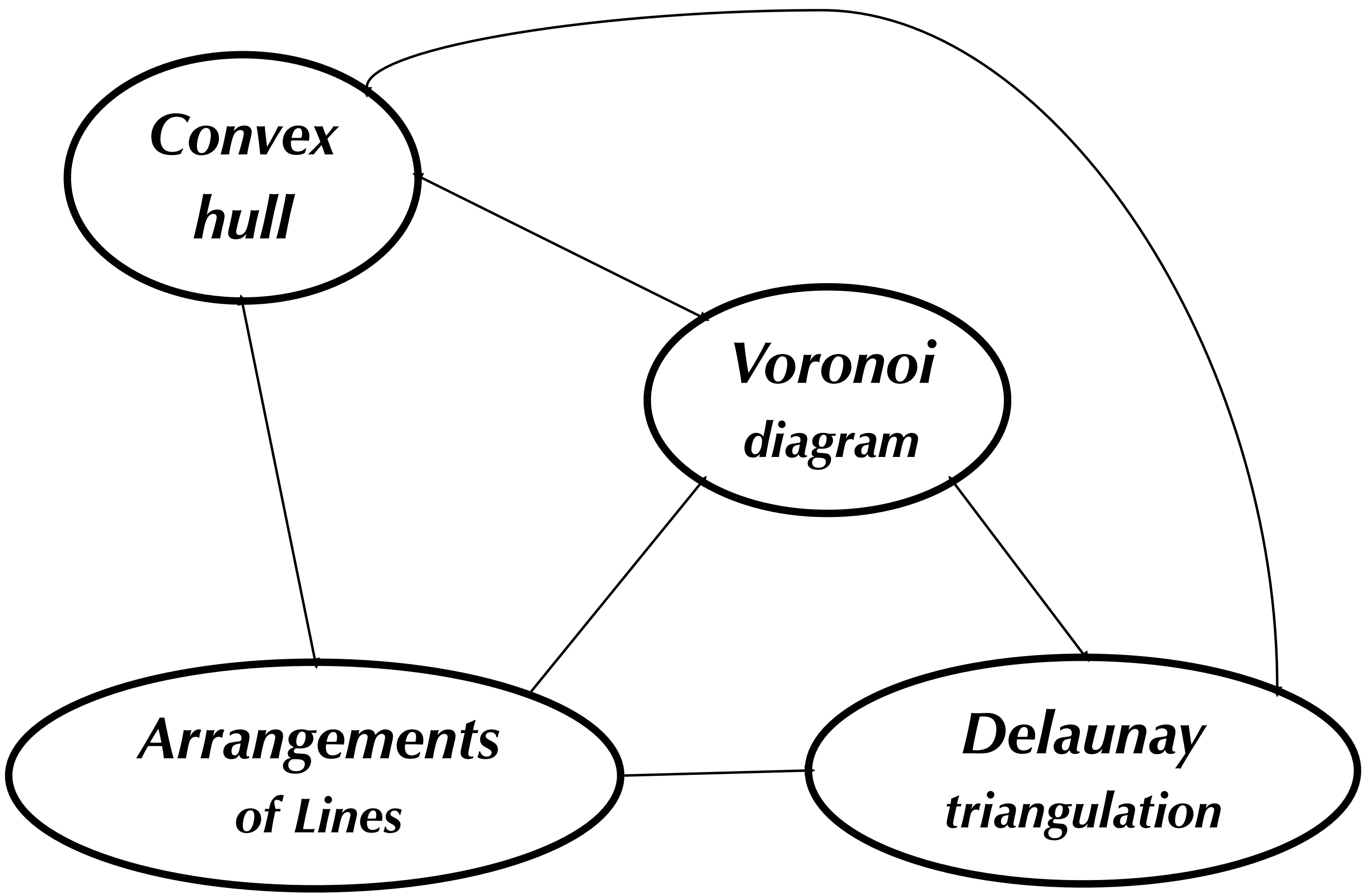
- Input P=$\{x_1, x_2, \ldots x_n\}$
- Lift
  - $a \Leftrightarrow a, a^2$
- Duality transform
  - $(a,b) \Leftrightarrow y=2ax-b$
- Arrangement of lines in dual space
- Project vertices with level=0

Convex Hulls vs. Arrangement

- 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



Conclusion


Convex hull
Voronoi diagram
Arrangements of Lines
Delaunay triangulation