Computing Convex Hulls in 3D

Computational Geometry

Dan Halperin

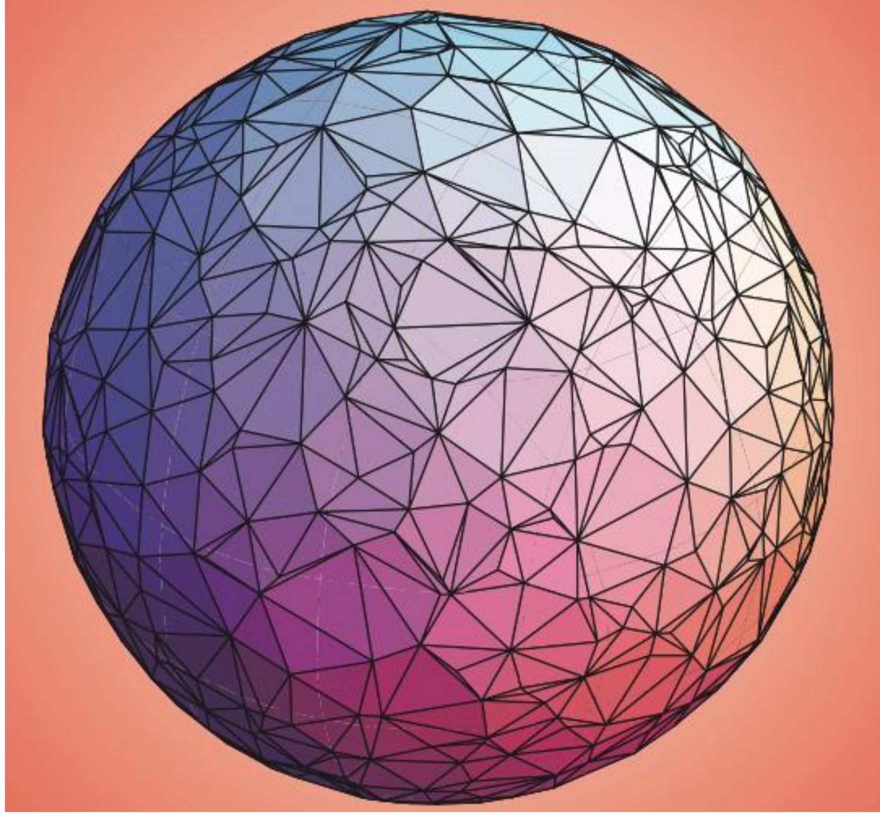
Tel Aviv University

Credits

- figures and pseudocode pieces are taken from Chapter 11, Convex Hulls, in Computational Geometry Algorithms and Applications by de Berg et al [CGAA]
- the original figures and pseudocode are available at the book's site: www.cs.uu.nl/geobook/

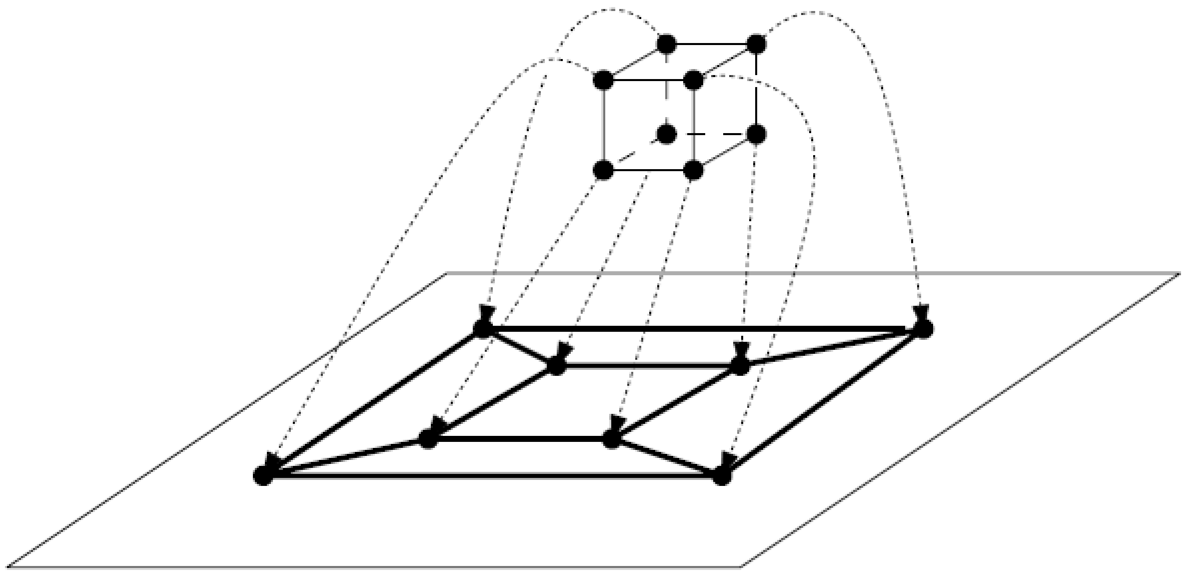
Convex hull in 3D

- the convex hull of a set P of n points in R^3 is a convex polytope whose vertices are points in P
- it therefore has at most n vertices
- its vertices and edges constitute a planar graph
- $CH(P)$ has at most $2n - 4$ faces and at most $3n - 6$ edges



[O'Rourke]

Convex polytopes and planar graphs



- the complexity bounds hold also for non-convex polytopes of *genus* zero with n vertices

Gift wrapping

- the convex hull of n points in R^3 can be computed in $O(nF)$ time, where F is the number of facets in the convex hull
- hence, the worst case running time of the gift-wrapping algorithm is $O(n^2)$

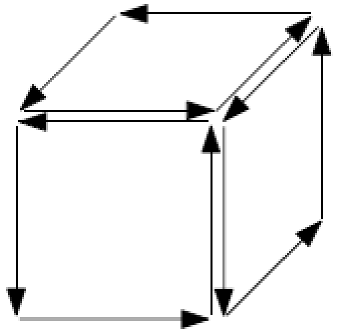
Randomized incremental construction

Outline of the algorithm

- the input: a set P of n points in R^3
- find four points not in a single plane, call them p_1, p_2, p_3, p_4
- randomly permute the remaining points: $p_5, p_6, \dots, p_n$
- let P_r denote the set $\{p_1, \dots, p_r\}$
- at stage $r = 5, \dots, n$ we add the point p_r and compute $CH(P_r)$
- the output: $CH(P_n)$

Representation: DCEL

- $CH(P_r)$ is a convex polytope, its boundary is a planar graph
- the vertices are points in 3-space
- the half-edges are oriented counterclockwise around the boundary of each face, when viewed from outside $CH(P_r)$

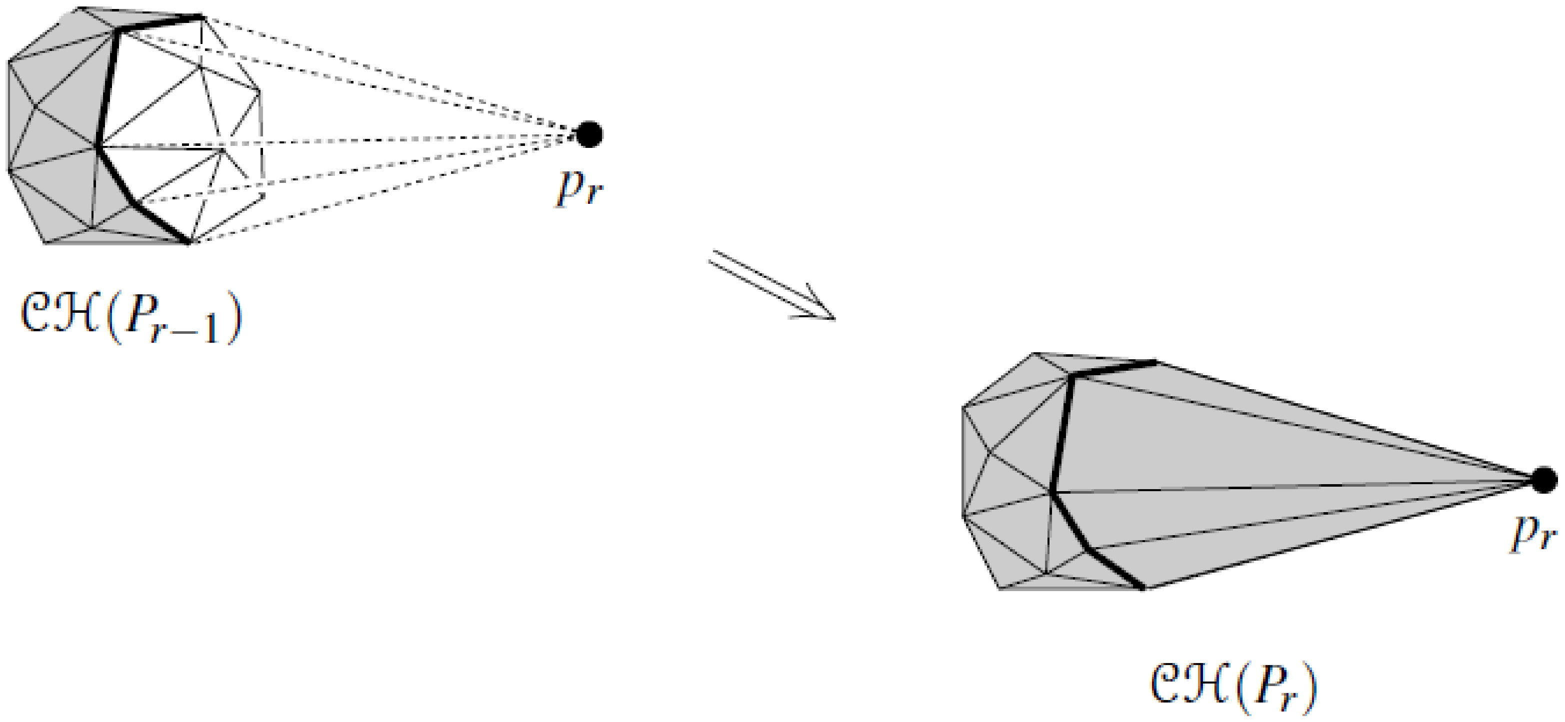


Choosing the first four points

- choose two points arbitrarily
- choose the third point not to lie on the line through the first two
  - if all the points lie on a line, report the segment that is their convex hull
- choose the forth point not to lie on the plane through the first three
  - if all the points lie on a plane, apply a two-dimensional CH algorithm

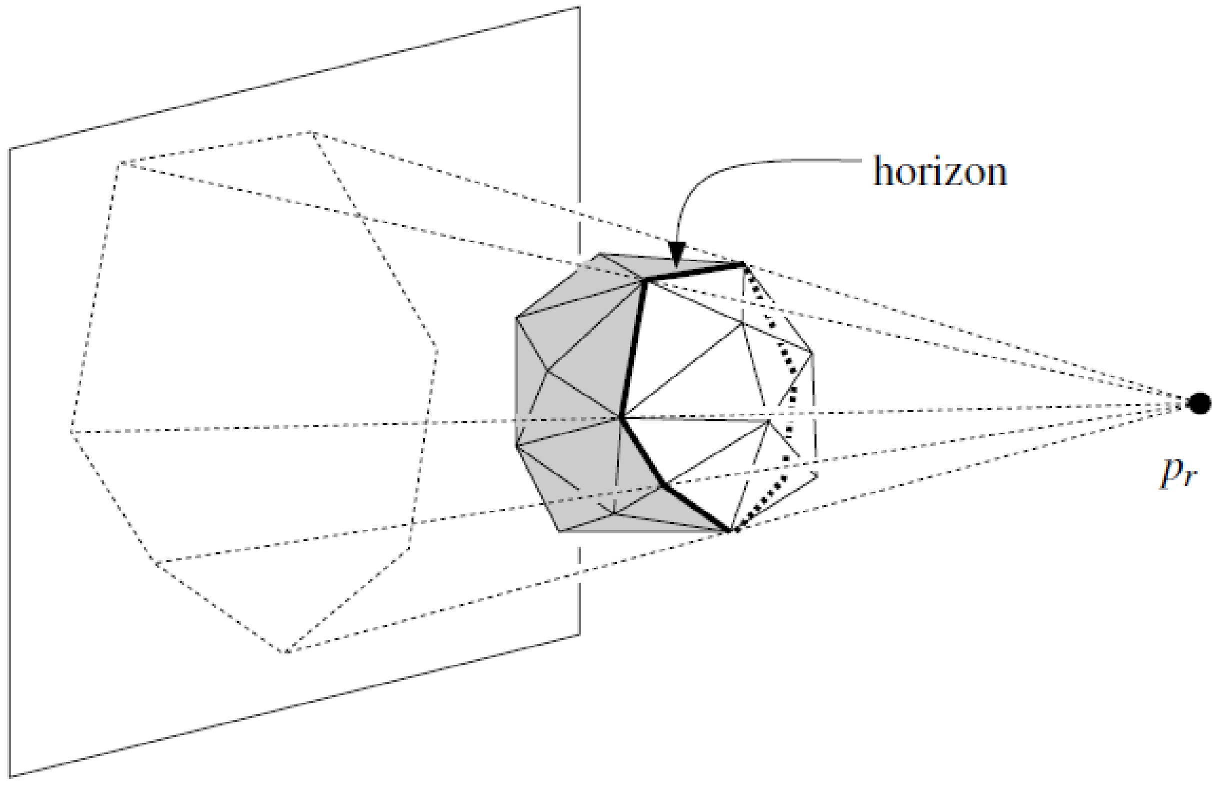
Adding the next point p_r to $CH(P_{r-1})$

- if $p_r \in CH(P_{r-1})$ we do nothing (also when p_r is on the boundary of $CH(P_{r-1})$)
- else



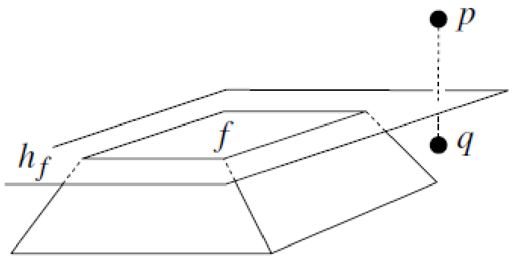
Adding the next point p_r, details

- if $p_r \notin CH(P_{r-1})$ we distinguish between visible facets of $CH(P_{r-1})$ and invisible facets (w.r.t. p_r)
- in-between visible and invisible facets lies the *horizon*
- the invisible facets will move on to $CH(P_r)$
- the visible facets will be replaced by triangles between p_r and edges of the horizon

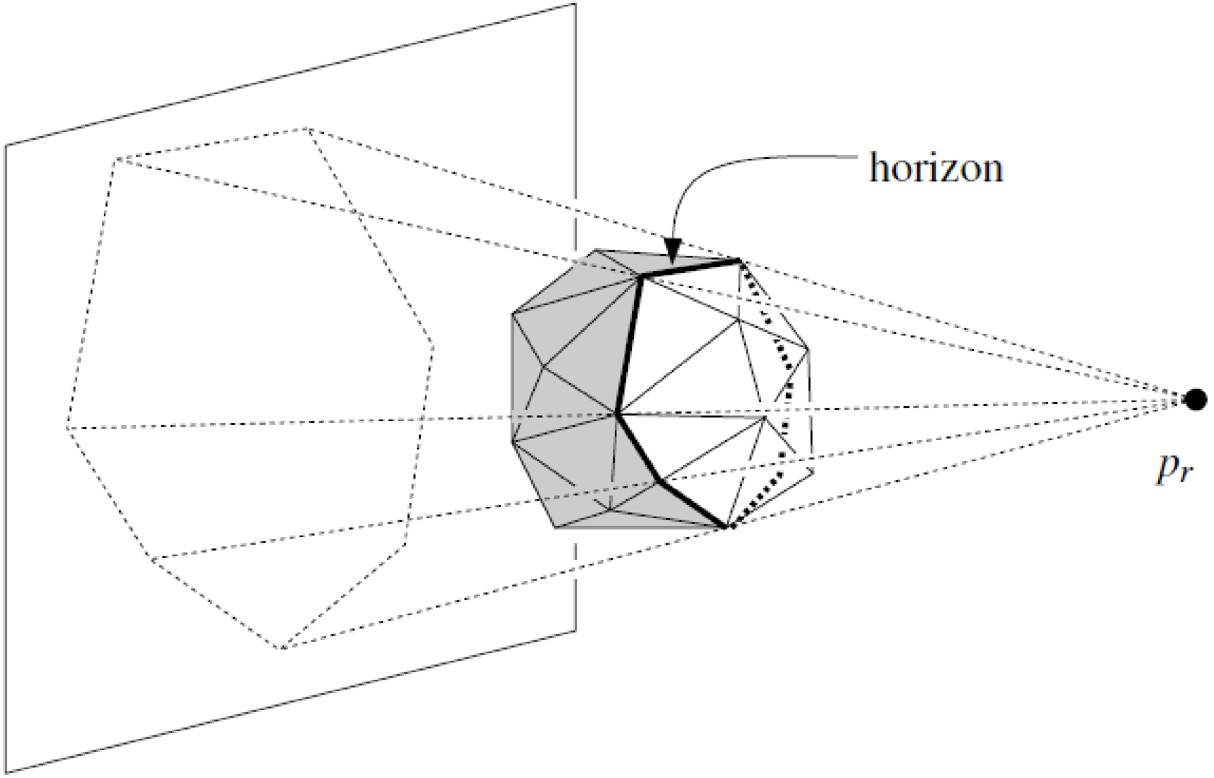


When is a face of $CH(P_{r-1})$ visible from p_r?

- let h_f be the plane through the facet f of $CH(P_{r-1})$
- $CH(P_{r-1})$ is on one side of h_f
- the facet f is visible from p_r if p_r is on the other side of h_f (not the side of $CH(P_{r-1})$)

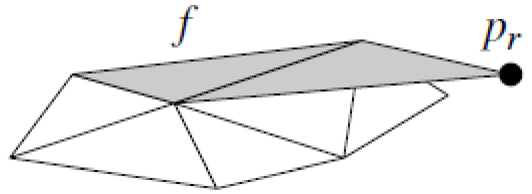


f is visible from p,
but not from q



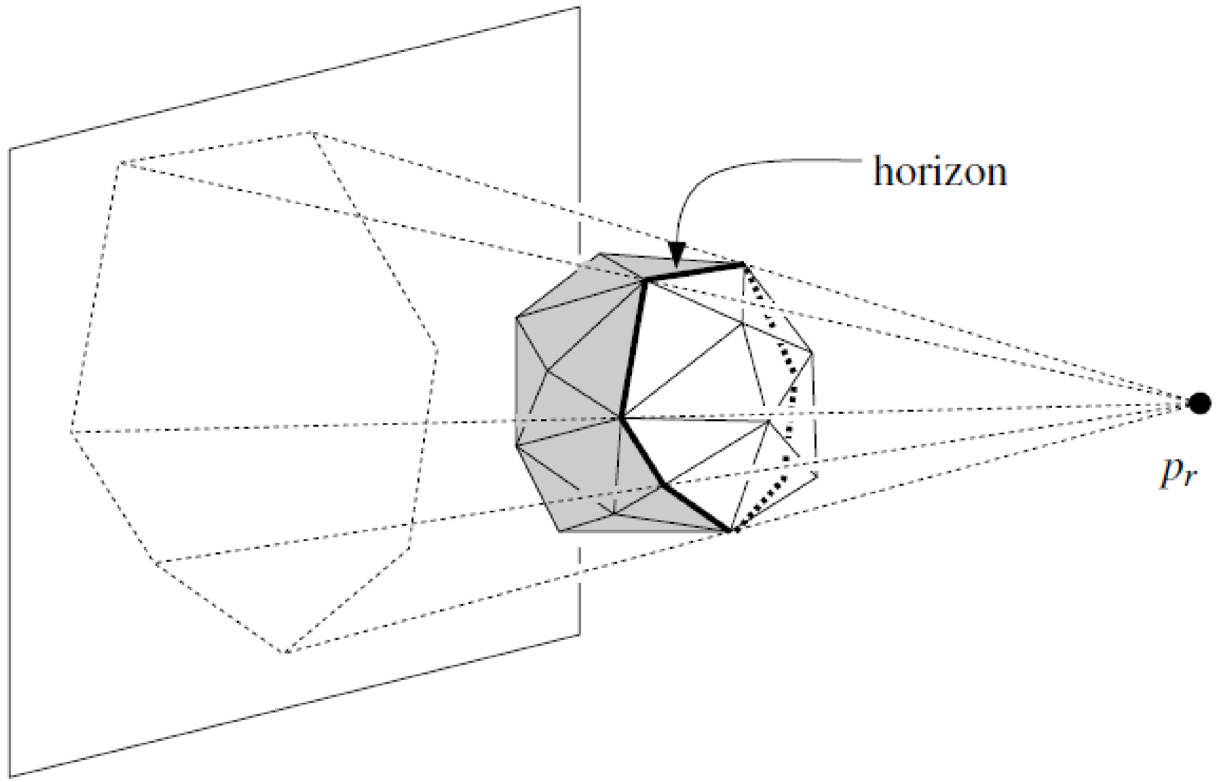
Co-planar facets

- if p_r lies on the plane h_f of an invisible facet f, we will add triangles from p_r to boundary edges of f that lie on the horizon: f needs to be merged with these triangles into a single facet



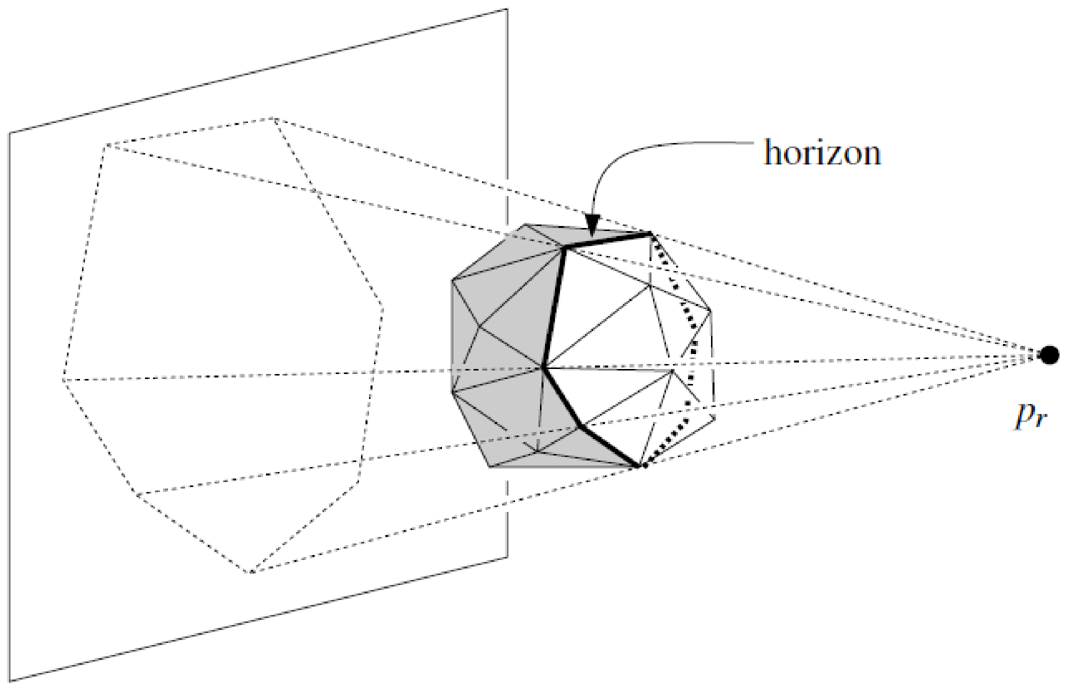
Cost of modifying $CH(P_{r-1})$ into $CH(P_r)$

- the visible facets are removed and triangles between p_r and edges of the horizon are added
- assuming we are given the horizon, this process takes time linear in the number of removed facets



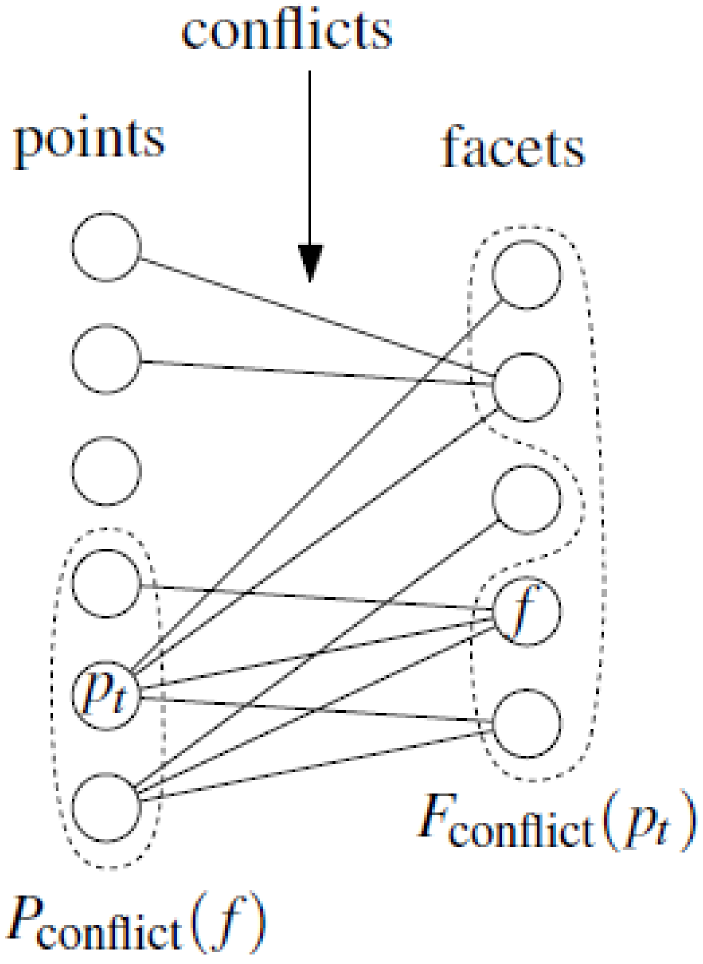
How to find the horizon relative to the next point?

- can be done naively in $O(r)$ time
- this will lead to an $O(n^2)$ time algorithm
- we will maintain *conflict lists*
- a point p_t is in conflict with a face f of $CH(P_r)$ iff f is visible from p_t
- for each point $p_t, \mathrm{t} > \mathrm{r}$, we will maintain the list of facets visible from p_t
- for each facet f of $CH(P_r)$ we will maintain the list of points visible from f



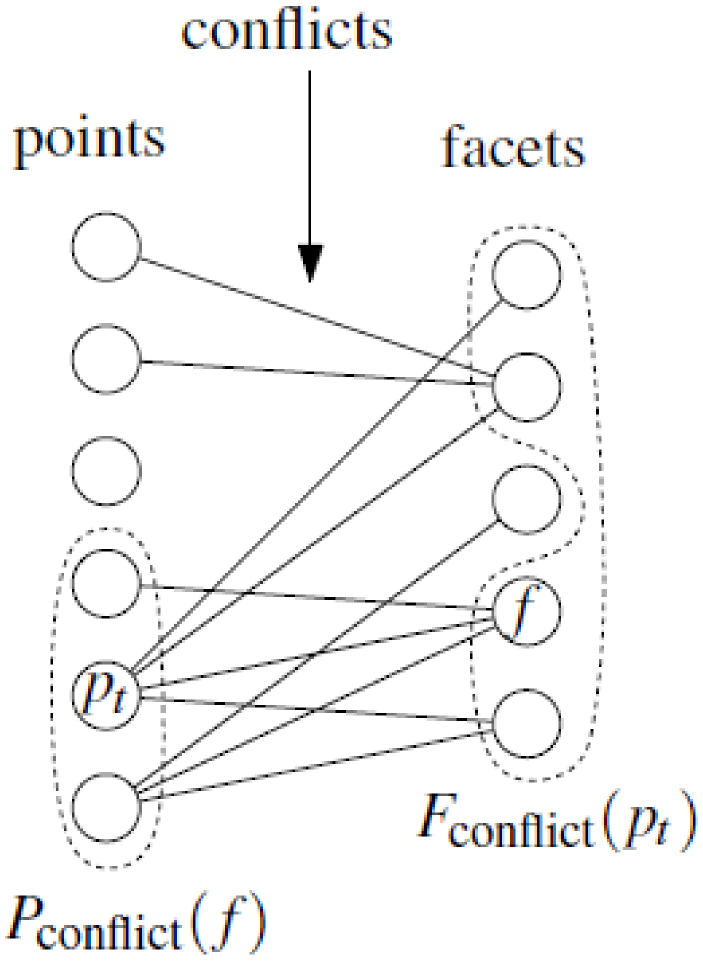
Conflict graph

- a point p_t is in conflict with a face f since they cannot co-exist in a convex hull
- we initialize the conflict graph for $CH(P_4)$ in linear time
- updating the graph I, removing visible facets when adding p_r: we remove their nodes from the graph as well as the arcs incident to these nodes; we also remove the node p_r
- the visible facets are the neighbors of p_r in the graph, and so this update is easy



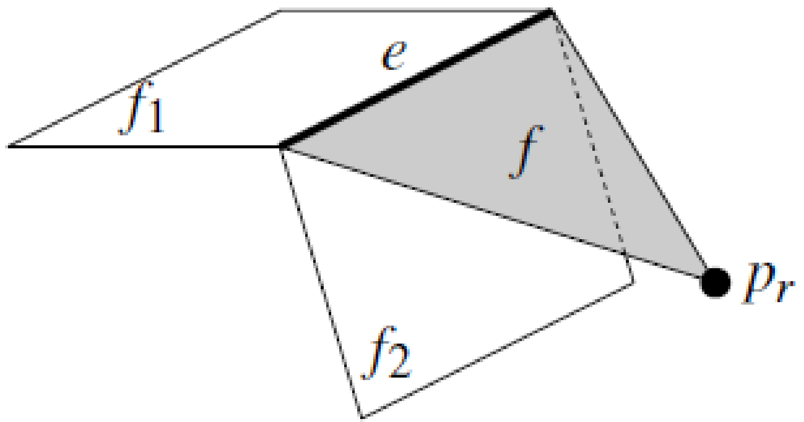
Updating the conflict graph, II

- we add new nodes for the newly created facets–those that connect p_r to the edges of the horizon
- it remains to find the conflicts of these new facets, and record them in the conflict graph–this is the tricky part of the algorithm!
- let f be one of the new facets–it is a triangle connecting p_r to an edge e of the horizon …



Updating the conflict graph, II, cont'd

- let f be one of the new facets–it is a triangle connecting p_r to an edge e of the horizon
- if f sees a point p_t, then the edge e sees it as well
- the edge e sees whatever the incident facets f_1 or f_2 see, so we only need to check the current conflicts of these two facets
- (if f merges with an existing invisible facet g, the new merged facet inherits g's conflicts)



The algorithm, initial steps

Algorithm CONVEXHULL(P)
Input. A set P of n points in three-space.
Output. The convex hull $\mathcal{CH}(P)$ of P.

1. Find four points p_1, p_2, p_3, p_4 in P that form a tetrahedron.
2. $\mathcal{C} \leftarrow \mathcal{CH}(\{p_1, p_2, p_3, p_4\})$
3. Compute a random permutation $p_5, p_6, \ldots, p_n$ of the remaining points.
4. Initialize the conflict graph $\mathcal{G}$ with all visible pairs (p_t, f), where f is a facet of $\mathcal{C}$ and $t > 4$.

The algorithm, inserting new points

for $r \leftarrow 5$ **to** n
  do (∗ Insert p_r into $\mathcal{C}$: ∗)
    if $F_{\text{conflict}}(p_r)$ is not empty (∗ that is, p_r lies outside $\mathcal{C}$ ∗)
      then Delete all facets in $F_{\text{conflict}}(p_r)$ from $\mathcal{C}$.
        Walk along the boundary of the visible region of p_r (which consists exactly of the facets in $F_{\text{conflict}}(p_r)$) and create a list $\mathcal{L}$ of horizon edges in order.
        for all $e \in \mathcal{L}$
          do Connect e to p_r by creating a triangular facet f.
            if f is coplanar with its neighbor facet f' along e
              then Merge f and f' into one facet, whose conflict list is the same as that of f'.
              else (∗ Determine conflicts for f: ∗)
                Create a node for f in $\mathcal{G}$.
                Let f_1 and f_2 be the facets incident to e in the old convex hull.
                $P(e) \leftarrow P_{\text{conflict}}(f_1) \cup P_{\text{conflict}}(f_2)$
                for all points $p \in P(e)$
                  do If f is visible from p, add (p, f) to $\mathcal{G}$.
        Delete the node corresponding to p_r and the nodes corresponding to the facets in $F_{\text{conflict}}(p_r)$ from $\mathcal{G}$, together with their incident arcs.
return $\mathcal{C}$

- DCEL operations are omitted–they are fairly straightforward

Analysis

- see Section 11.3 in [CGAA]
- the expected number of facets created during the algorithm for n input points is $6n - 20$ (proof uses backward analysis similarly to what we have seen in other algorithms)
- the overall expected size of the list of conflicts (visible points) of horizon edges is $O(n \log n)$ (proof has novel components)
- in summary: the convex hull of n points in R^3 can be computed in expected $O(n \log n)$ time

Remarks

- the algorithm is due to Clarkson and Shor, Test of Time Award, SoCG 2020 (together with Haussler and Welzl)
- the analysis as presented in [CGAA] is due to Mulmuley
- the algorithm extends to higher dimensions and runs in expected $\Theta(n^{\lfloor d/2 \rfloor})$ time in R^d, for $d > 3$

THE END

[Jeb Gaither, CGAL arrangements]