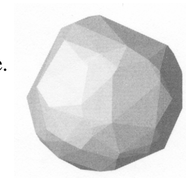
Convex Hulls in 3-space

(slides mostly by Piotr Indyk and Jason C. Yang)

Problem Statement

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



Complexity

- Complexity of $\mathcal{CH}$ for n points in 3D is $O(n)$
- ..because the number of edges of a convex polytope with n vertices is at most $3n-6$ and the number of facets is at most $2n-4$
- ..because the graph defined by vertices and edges of a convex polytope is planar
- Euler's formula: $n - n_e + n_f = 2$

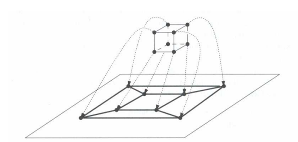
Complexity

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

$$2n_e \geq 3n_f$$

- Using Euler

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



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 $P_r := \{p_1, \ldots, p_r\}$]

Initialization

- Need a $\mathcal{CH}$ to start with
- Build a tetrahedron using 4 points in P
  - Start with two distinct points in P, say, 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? Planar case!
- Compute random permutation $p_5, \ldots, p_n$ of the remaining points

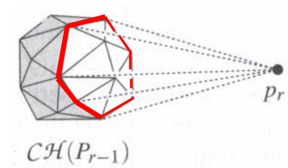
Inserting Points into $\mathcal{CH}$

- Add p_r to the convex hull of P_{r-1} to transform $\mathcal{CH}(P_{r-1})$ to $\mathcal{CH}(P_r)$
- Two Cases:
  1) P_r is inside or on the boundary of $\mathcal{CH}(P_{r-1})$
     – Simple: $\mathcal{CH}(P_r) = \mathcal{CH}(P_{r-1})$
  2) P_r is outside of $\mathcal{CH}(P_{r-1})$ – the hard case

7 / 41

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

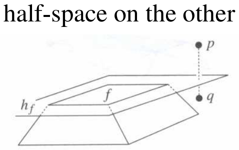
- Determine ***horizon*** of p_r on $\mathcal{CH}(P_{r-1})$
  – Closed curve of edges enclosing the ***visible*** region of p_r on $\mathcal{CH}(P_{r-1})$



8 / 41

Visibility

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

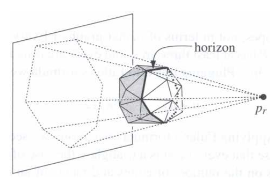


f is visible from p, but not from q

9 / 41

Rethinking the Horizon

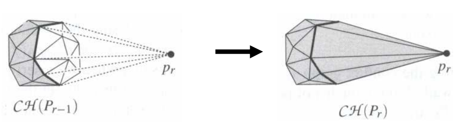
– Boundary of polygon obtained from projecting $\mathcal{CH}(P_{r-1})$ onto a plane with p_r as the center of projection



10 / 41

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

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



11 / 41

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 do we determine the visible region?

12 / 41

How to Find Visible Region

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

  Maintain information to aid in determining visible facets.

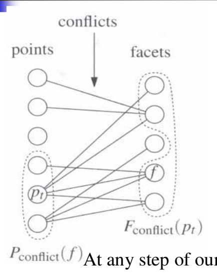
Conflict Lists

- 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 $\mathcal{G}$

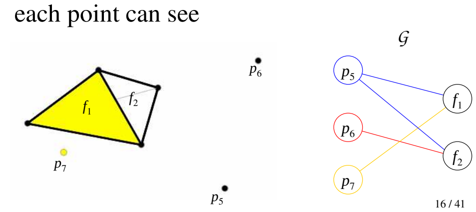


- Bipartite graph
  - points not yet inserted
  - facets on $\mathcal{CH}(P_r)$
- Arc for every point-facet conflict
- Conflict sets for a point or facet can be returned in linear time

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

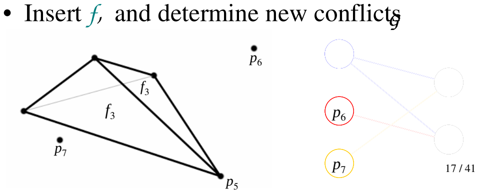
Initializing $\mathcal{G}$

- 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



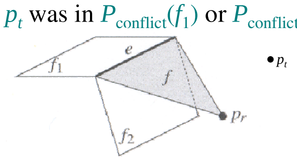
Updating $\mathcal{G}$

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



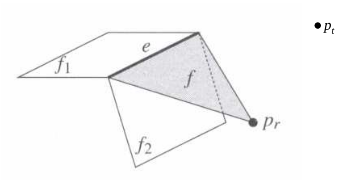
Determining New Conflicts

- 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

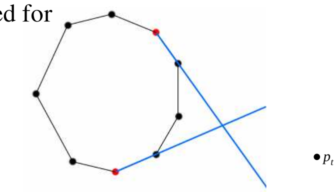
- Conflict list of f can be found by testing the points in the conflict lists of f_1 and f_2 incident to the horizon edge e in $\mathcal{CH}(P_{r-1})$



19 / 41

What About the Other Facets?

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



20 / 41

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 $F_{conflict}(p_r)$ 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 $F_{conflict}(p_r)$ from $\mathcal{G}$

21 / 41

Analysis

22 / 41

Expected Number of Facets Created

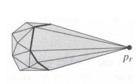
- Will show that expected number of facets created by our algorithm is at most $6n-20$
- Initialized with a tetrahedron = 4 facets

23 / 41

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:

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



24 / 41

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 $(6r$-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 $p_5,\ldots,p_n$

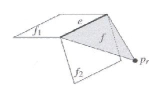
$$4+\sum_{r=5}^{n}\mathrm{E}[\deg(p_r, \mathcal{CH}(P_r))] \leqslant 4+6(n-4) = 6n-20.$$

Running Time

- Initialization ⇒ $O(n\log n)$
- Creating and deleting facets ⇒ $O(n)$
  - 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 ⇒ $O(n)$
- Finding new conflicts ⇒ **O(?)**

Total Time to Find New Conflicts

- For each edge e on horizon we spend
  $O(|P(e)|)$ time
  where $P(e) = P_{\text{conflict}}(f_1) \cup P_{\text{conflict}}(f_2)$
- Total time is $O(\Sigma_{e\in \mathcal{L}} |P(e)|)$



The sum is taken over all edges e created.

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

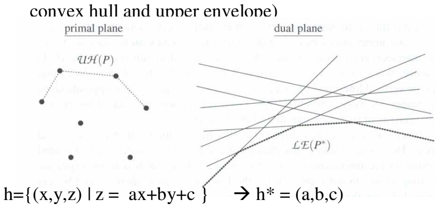
Running Time

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

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

Convex Hulls in Dual Space

- Upper convex hull of a set of points in 3D is essentially the lower envelope of a set of lines (similar with lower convex hull and upper envelope)



h={(x,y,z) | z = ax+by+c } → h* = (a,b,c)
p=(a,b,c) → p*={(s,t,r) | r = sa+bt+c }

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

31 / 41

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

32 / 41