DCG, Wintersemester 2024/25 Lecture 7 (5 pages)

Computing Convex Hulls in $\mathbb{R}^3$

Anne Driemel and Herman Haverkort updated: November 1, 2024

In this lecture we want to adapt Algorithm 5.1 from Lecture 5, which computes the convex hull for points in $\mathbb{R}^2$, to the setting of points in $\mathbb{R}^3$. We assume the points lie in general position (see Lecture 4) and we use the DCEL data structure (see Lecture 7) to store the convex hull C of the points processed so far.

1 The algorithm

Just like in Lecture 5, we maintain a conflict graph G to keep track of the facets of the convex hull that are in conflict with the points still to be added to the convex hull. We say a facet is in conflict with a point, if the facet would be removed if the point is added in the next step. The conflict graph consists in maintaining, for each facet and for each point, a list with all points or facets that they are in conflict with. We use $\mathit{Conflicts}(X)$ to denote the set of neighbours of X in G, that is, the points or facets that X is in conflict with. As before, we will make sure that the points in the conflict list $\mathit{Conflicts}(f)$ of a facet f are always stored in sorted order.

Let us now discuss what needs to be done when updating the convex hull of points $p_1, \ldots, p_{i-1}$ with the next point p_i. If p_i lies inside the convex hull, then its neighborhood in the conflict graph is empty and we do not need to do anything. Otherwise, the neighborhood represents the facets that have a direct line of sight with p_i and they need to be removed from the convex hull. The set of facets to be removed forms a connected set with a boundary that forms a closed cycle of edges on the convex hull, see Figure 1; we consider the cycle directed such that the facets that will remain are to the left and the facets to be removed are to the right. We call the edges of this cycle *horizon edges*. Each of these edges forms a new facet with p_i, see Figure 2.

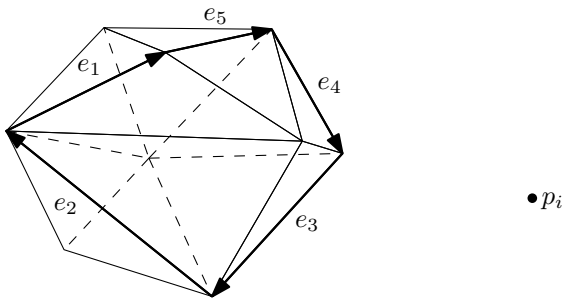


Figure 1: Horizon edges $e_1, \ldots, e_5$ computed when updating the convex hull with p_i.

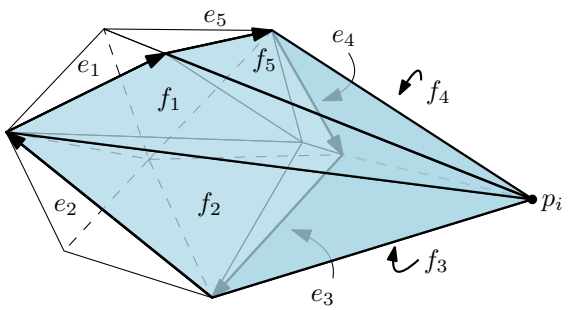


Figure 2: New facets $f_1, \ldots, f_5$ created when updating the convex hull with p_i.

Definition 7.1 (Horizon edge)**.** *Let C be a convex polytope stored in a DCEL and let p_i be a point. Let e be a half-edge of C. If the facet $IncidentFacet(e)$ is not in conflict with p_i whereas the facet $IncidentFacet(Twin(e))$ is in conflict with e, then we call e a horizon edge.*

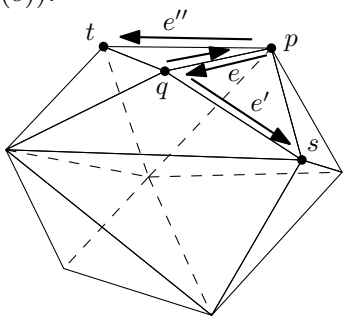
Algorithm 7.1

```
1: procedure CONVEX-HULL(Set of points P in ℝ³)
2:     Let p_1, ..., p_n be a random permutation of the points of P
3:     Let C be a DCEL storing the convex hull of {p_1, p_2, p_3, p_4}
4:     // Initialise the conflict graph:
5:     for each facet f of C and each j ∈ {5, ..., n} such that p_j conflicts with f do
6:         Add f and p_j to each other's conflict lists
7:     end for
8:     // Incremental construction:
9:     for i ← 5 to n do
10:        if Conflicts(p_i) ≠ ∅ then
11:            H ← cycle of half-edges clockwise along boundary of union of Conflicts(p_i)
12:            for e ∈ H do
13:                // Create new triangular facet:
14:                Create face f with boundary cycle of new half-edges Start(e) → p_i → End(e)
15:                  → Start(e), setting all attributes except the twin pointers;
16:                // Collect points that f may conflict with from the old facets incident on e:
17:                T ← Conflicts(IncidentFacet(e)) ∪ Conflicts(IncidentFacet(Twin(e)))
18:                // Connect up f in the conflict graph G:
19:                for p ∈ T do
20:                    if f is in conflict with p then
21:                        add f and p to each other's conflict lists
22:                    end if
23:                end for
24:                // Insert f in the DCEL of C:
25:                Set twin pointers between e and its newly created twin
26:                // (the twin pointers of the other new edges will be set on Line 30)
27:                // Make sure IncidentEdge(Start(e)) is an edge that is not about to be deleted:
28:                Let IncidentEdge(Start(e)) point to e
29:            end for
30:            Go around H again and set twin pointers between half-edges to/from p_i
31:            // Discard old facets:
32:            for f ∈ Conflicts(p_i) do
33:                Remove f from the conflict lists of all points it is in conflict with
34:                Remove f and all incident half-edges and now-isolated incident vertices
35:            end for
36:            Clear the conflict list Conflicts(p_i) of p_i
37:        end if
38:    end for
39:    return C
40: end procedure
```

2 Analysis

We define *configurations* and *killing sets* in much the same way as before, in Lecture 5. Let P be the set of n points which is the input of the algorithm.

Definition 7.2 (Configuration)**.** *A configuration Δ is an ordered tuple (s, q, p, t) of four different points from P. A configuration is present on the convex hull, if the four points appear next to each other on the boundary of the convex hull, in the following way: there is a half-edge e from p to q, and there is a half-edge e' from q to s with $e' = Next(e)$, and there is a half-edge e'' from p to t with $e'' = Next(Twin(e))$.*



Definition 7.3 (Killing-Set)**.** *For $a, b, c \in \mathbb{R}^3$ define*

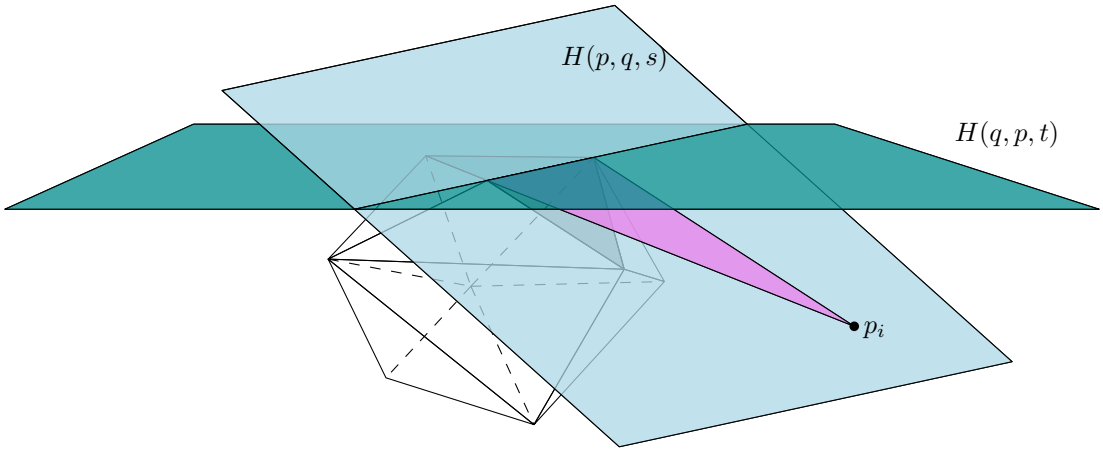
$$H(a,b,c) = \left\{ x \in \mathbb{R}^3 \;\middle|\; \langle v, x - b \rangle \geq 0 \right\} \qquad \text{with } v = (c-b) \times (a-b)$$

Assume we use a right-handed *coordinate system. If a, b and c appear in counterclockwise order along the boundary of a face f, then v is orthogonal to the hyperplane containing f and v points away from the polytope.*[1]

The killing-set $K(\Delta)$ of a configuration $\Delta = (s, q, p, t)$ is defined as follows

$$K(\Delta) = (H(p,q,s) \cup H(q,p,t)) \cap (P \setminus \{s,p,q,t\})$$

We think of the killing-set as the set of points that would lead to one of the two facets of the configuration being removed from the convex hull. If one of the points of the killing set is added to the convex hull in the next step, the configuration is destroyed. Note that a configuration cannot coexist together with any of the points in its killing set in the convex hull.



[1] In a right-handed coordinate system, the points $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$ trace out a triangle in counterclockwise order. If we would use a *left-handed* coordinate system instead, then the points $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$ would lie in *clockwise* order, and we would need a, b and c to lie in *clockwise* order along the boundary of f, otherwise v would not point *away* from the polytope.

Lemma 7.4. *Consider the execution of Algorithm 7.1 for computing the convex hull of n points. Let $\mathcal{D}_i$ be the set of configurations newly created in C when adding point p_i, for $5 \leq i \leq n$, and let $\mathcal{D}_4$ be the set configurations in C created in the beginning. It holds that*

$$\mathbf{E}\left[\sum_{i=4}^{n}\sum_{\Delta\in\mathcal{D}_i}\big|K(\Delta)\big|\right] \in O(n\log n)$$

Proof. We adapt the proof of Lemma 5.6 from Lecture 5. Let $\mathcal{W}_i$ denote the set of configurations that are destroyed in C during the addition of point p_i. The first part of the proof is the same, except that there are 4 points (instead of 3) in a configuration. So, we obtain

$$\mathbf{E}\left[\sum_{i=4}^{n-1}\sum_{\Delta\in\mathcal{D}_i}\big|K(\Delta)\big|\right] \leq 4n\cdot\mathbf{E}\left[\sum_{i=4}^{n-1}\frac{\big|\mathcal{W}_{i+1}\big|}{i}\right] \tag{1}$$

Just like in the other proof, we obtain, by the properties of the sets $\mathcal{D}_i$ and $\mathcal{W}_{i+1}$

$$\sum_{i=4}^{n-1}\frac{\big|\mathcal{W}_{i+1}\big|}{i} \leq \sum_{i=4}^{n-1}\frac{\big|\mathcal{D}_i\big|}{i} \tag{2}$$

Now, this time, we do not have the property that $\big|\mathcal{D}_i\big|$, the number of new configurations created when adding p_i is constant. Instead, we claim that the *expected* size of $\mathcal{D}_i$ is constant for any fixed i. We argue as follows. Consider the last $i-1$ points $p_{i+1},\ldots,p_n$ of the permutation to be fixed, and let the remaining i points of P be the set S. Let $\mathcal{C}_i$ be the set of configurations that are present in the convex hull of S. Consider a fixed configuration $\Delta\in\mathcal{C}_i$ and consider the event that Δ is in the set $\mathcal{D}_i$. In order for this event to happen, one of the 4 points defining the configuration Δ must be p_i. The point p_i is uniformly distributed in S. Therefore, the probability of this event is $4/|S|$, which is equal to $4/i$. Therefore, we get for fixed S (and thus for fixed $\mathcal{C}_i$) that

$$\mathbf{E}\left[\big|\mathcal{D}_i\big|\right] = \sum_{\Delta\in\mathcal{C}_i}\frac{4}{i} = \frac{4}{i}|\mathcal{C}_i| \tag{3}$$

The number of configurations in $\mathcal{C}_i$ is equal to twice the number of edges of the convex hull of S. By Theorem 7.2 from Lecture 7 the number of edges is at most $3(i-2)$. Plugging into the above we get

$$\mathbf{E}\left[\big|\mathcal{D}_i\big|\right] \leq \frac{4}{i}\cdot 6(i-2) \leq 24 \tag{4}$$

This upper bound holds for any choice of S, and therefore, it also holds as an upper bound on the expectation of $\big|\mathcal{D}_i\big|$ over all permutations of P. We can use this bound together with (1) and (2) and obtain by using linearity of expectation

$$\mathbf{E}\left[\sum_{i=4}^{n-1}\sum_{\Delta\in\mathcal{D}_i}\big|K(\Delta)\big|\right] \leq 4n\cdot\mathbf{E}\left[\sum_{i=4}^{n-1}\frac{\big|\mathcal{D}_i\big|}{i}\right] \leq 4n\sum_{i=4}^{n-1}\frac{\mathbf{E}\left[\big|\mathcal{D}_i\big|\right]}{i} \leq 4n\sum_{i=4}^{n-1}\frac{24}{i} < 96nH_{n-1} \tag{5}$$

where H_{n-1} denotes the $(n-1)$th Harmonic number, and this implies the theorem. □

Theorem 7.5. *Algorithm 7.1 computes the convex hull of a set P of n points in $\mathbb{R}^3$ in $O(n \log n)$ expected time, where the expectation is with respect to the random permutation used by the algorithm.*

Proof. We adapt the proof of Theorem 5.5 from Lecture 5. The correctness follows again by induction on the main for-loop, using the invariant, that, at the end of each iteration of the for-loop, the DCEL in C stores the convex hull of the points processed so far.

The main part of the proof is to show the running time. Again, the work done in the lines before the main for-loop and after the for-loop can be performed in $O(n)$ time in the worst case, just like in the algorithm in Lecture 5.

Finding the cycle of horizon edges on Line 11 can be done in time $O(|\mathit{Conflicts}(p_i)|)$ (→ Exercise); the same holds for the loops on Lines 30–36 that set the twin pointers and discard the old facets. Using similar arguments as in the proof of Lemma 7.4, one can show that (→ Exercise)

$$\mathbf{E}\left[\sum_{i=4}^{n} |\mathit{Conflicts}(p_i)|\right] \leq O(n)$$

Therefore, overall, the expected total time that is spent on the lines in the main for-loop, excluding Lines 17–23 (inserting f in the conflict graph) over the course of the entire algorithm is in $O(n)$.

It remains to analyze the expected total time spent on updating the conflict graph in Lines 17–23. Just like in the proof of Theorem 5.5, this is bounded by the total size of the killing sets of configurations created during the course of the algorithm. By Lemma 7.4 this is in $O(n \log n)$ □

References

- Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. Computational Geometry— Algorithms and Applications. Third Edition. Springer. Chapters 9 and 11.