DCG, Wintersemester 2024/25 Lecture 5 (4 pages)

Computing Convex Hulls I

Anne Driemel, Herman Haverkort, Elmar Langetepe updated: October 23, 2024

In one of the previous lectures we defined the convex hull of a set $A \subseteq \mathbb{R}^d$ as the set of all convex combinations of points in A. We also saw several characterizations of the convex hull of a finite number of points. We now want to discuss algorithms for computing such geometric structures. In particular, we are interested in computing a structured representation of the boundary of the convex hull. We will start by looking at convex hulls in the plane. In this case, the boundary can simply be represented as a polygonal chain.

1 Computational model and basic assumptions

When analyzing an algorithm, we are interested in the number of steps the algorithm needs to take and the amount of storage space that is used by the computations. In particular, as is standard in algorithmic analysis, we want to bound the worst-case complexity (number of steps/ amount of space) as a function of the input size. We will use the standard asymptotic analysis to simplify the expressions of the running time and space.

At the foundation of any such algorithmic analysis lies a computational model that determines what constitutes a basic step of the algorithm. In this course, if not stated otherwise, we will use the so-called *Real-RAM model of computation*. In this model, we use a random-access machine (RAM), that has multiple, uniquely addressed registers. Each of the registers can hold a real number and we can read and write any register of the machine directly by specifying its address. When analyzing the number of steps of an algorithm, we count the number of basic arithmetic operations on real numbers stored in the registers, such as addition, subtraction, multiplication, division, comparison. In some variants of the Real-RAM, even functions such as $\sqrt[k]{x}$, $\exp(x)$, $\log(x)$, $\sin(x), \cos(x)$ are counted as basic arithmetic operations, but we can usually avoid evaluating such expressions explicitly (which is good, because such expressions take more time to evaluate than other basic operations).

Moreover, we will make simplifying assumptions on the geometric configuration of the input. In particular, we will often make the assumption on a given finite set of points (set of lines, hyperplanes, etc), that no "unlikely" coincidences happen in the configuration. This is particularly helpful for simplifying the exposition of the algorithms as many special cases that do not increase the computational complexity can be ignored in the specification of the algorithm. In computational geometry, such an assumption is called a *general position* assumption. The most common assumptions are the following.

Assumption 5.1. *For a given set of points in the plane we may assume that no three points lie on a common line. More generally, for a given set of points in $\mathbb{R}^d$ we may assume that no $k \leq d+1$ points lie in a common $(k-2)$-flat.*

Assumption 5.2. *For a given set of lines in the plane we may assume that no three lines have a common point and no two lines are parallel.*

Assumption 5.3. *For points in the plane we may assume that no two points have the same x-coordinate or y-coordinate.*

2 A simple convex hull algorithm in the plane

Given a set of points P in the plane. Assume the points lie in general position, using Assumption 5.1 and 5.3. The algorithm given below computes the convex hull of P. The algorithm maintains a convex chain of a subset of points of P in counter-clockwise order. When adding a new point, the algorithm tests if the new point *makes a left turn* with the remainder of the chain. We say three points a, b, c make a left turn if the counter-clockwise angle from a to c centered at b is greater than π. If this is not the case, then the previously last point is repeatedly removed from the chain, until the current point makes a left turn with the remainder of the chain. It is important that the algorithm processes the points of P in the counter-clockwise order with respect to a fixed reference point in the convex hull. The algorithm is due to Ronald Graham.

Algorithm 5.1 Graham-Scan

```
1: procedure CONVEX-HULL(Set of points P in ℝ²)
2:     Let p_1 be the point in P with minimal y-coordinate
3:     Express the remaining points in P by their polar coordinates with origin p_1
4:     Let p_2, ..., p_n denote the points sorted by increasing polar angle (w.r.t. p_1)
5:     Let L be a list containing p_1, p_2 in this order
6:     for i ← 3 to n do
7:         Append p_i to L
8:         while the last three points in L do not make a left turn do
9:             Remove the middle point of the last three points from L
10:        end while
11:    end for
12:    return L
13: end procedure
```

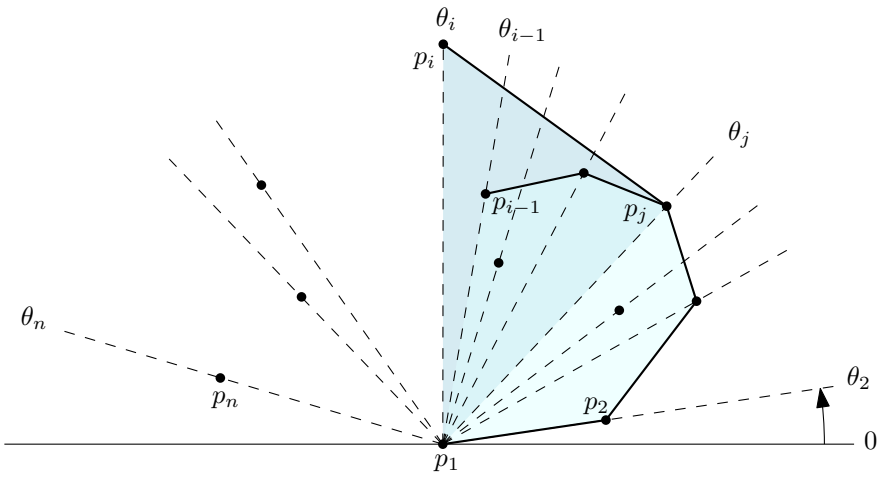

Figure 1: Illustration of the Graham-Scan algorithm for computing the convex hull.

Theorem 5.4. *The convex hull of n points in the plane can be computed in $O(n \log n)$ time and $O(n)$ space.*

Proof. We prove that Algorithm 5.1 computes the convex hull in $O(n \log n)$ time. We first prove that the algorithm computes the convex hull correctly. The proof is by induction on the

points treated in the main for-loop of the algorithm. We claim that the algorithm maintains the invariant that the polygonal chain of the points in the list L represents the boundary of the convex hull of the points $p_1, \ldots, p_i$ processed so far (in counter-clockwise order). The invariant clearly holds for $L = \{p_1, p_2\}$ before the for-loop starts and this is the base case of the induction. Assume $i > 2$ and consider the algorithm after the execution of the iteration of the for-loop with variable i. The last point in L is p_i. Let j be the index of the second-last point in L. The induction hypothesis applied to the points $P' = p_1, \ldots, p_j$ implies that the polygonal chain in L up to p_j represents the boundary of the convex hull of P'. Assume $i > j + 1$, then we claim that the points $p_{j+1}, \ldots, p_{i-1}$ lie in the triangle T spanned by p_1, p_j, and p_i. (Otherwise, if $i = j + 1$, then the triangle is empty.) This certainly holds for the sequence of those points that were removed from L during the current iteration of the for-loop. Let this sequence be denoted L'. By induction, L' together with p_1 and p_j forms the convex hull of the remaining points in $p_{j+1}, \ldots, p_{i-1}$ that were removed in previous iterations. Since the triangle T is convex, it also contains those points. Now, we can join the two convex hulls along the edge $\overline{p_1 p_j}$ and since the last three points x, p_j, p_i in L make a left turn, we obtain a convex chain that contains the points $p_1, \ldots, p_j$ as well as the points $p_{j+1}, \ldots, p_i$. Figure 1 shows an example. By induction, this proves correctness of the algorithm for all of P.

We now want to analyze the running time. Finding p_1 with smallest y-coordinate takes $O(n)$ time. Sorting the points by polar angle takes $O(n \log n)$ time. There are n iterations of the for-loop in line 6 and the running time of a single iteration of the for-loop depends on the number of iterations of the while loop in line 8 that removes points from the list L, which can be up to $n - 3$ steps in the worst case. However, note that each points of P is added to L at most once and therefore it can only be removed from the list L at most once. Therefore, the total number of executions of the line 9 of the algorithm is bounded by $O(n)$, which is much better than $O(n^2)$. Therefore, lines 5-12 take $O(n)$ time and the total running time is dominated by the sorting in line 3. □

3 Lower bounds

Given the algorithm for computing the convex hull for n points in the plane with running time $O(n \log n)$ in the worst case, one may wonder if this is the best we can do in the worst case. We can show that this is indeed optimal by reduction from sorting.

Theorem 5.5. *Computing the convex hull of n points in the plane takes $\Theta(n \log n)$ time in the basic Real-RAM model.*

Proof. (sketch) Theorem 5.4 implies the upper bound of $O(n \log n)$. We can show the lower bound of $\Omega(n \log n)$ as follows. It has been shown that any comparison-based sorting algorithm (such as quicksort, mergesort, etc) needs $\Omega(n \log n)$ comparisons for sorting n numbers in the worst case. This bound holds even if we allow comparisons between polynomial functions of the numbers to be sorted—this is the type of comparisons offered by the Real-RAM model.

Given n real numbers $x_1, \ldots, x_n$, we create a set of points by lifting to the unit parabola as follows (see also Figure 2)

$$P = \{(x_1, x_1^2), \ldots, (x_n, x_n^2)\}$$

If we now compute the convex hull of P, then we can extract the correct sorted order of the x_i in time linear in n from the ordering of the points in P along the convex hull. Therefore, if there would be an algorithm for computing the convex hull that has worst-case running time in $o(n \log n)$ in the Real-RAM, then this would also imply a sorting algorithm in $o(n \log n)$ time. However, this contradicts the lower bound of $\Omega(n \log n)$. □

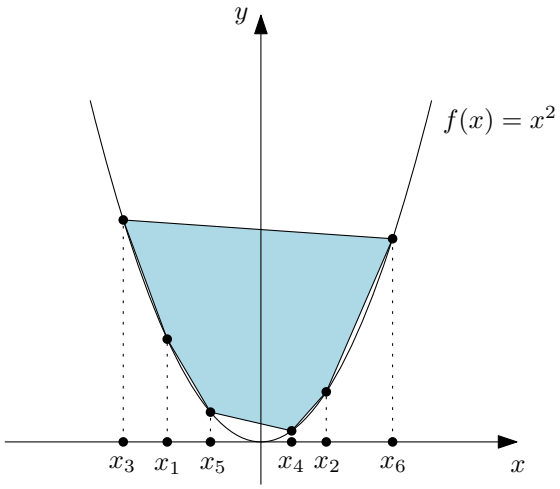

Figure 2: Reduction from sorting to convex hull

4 Another algorithm for the convex hull in the plane

We discuss another algorithm for computing the convex hull. The algorithm is incremental, which means that it constructs the convex hull for an increasing subset of the input. In each step, a point is added to the set and the convex hull is updated. In this sense, the algorithm is similar to the previous algorithm, however, it does not rely on the points being processed in any particular order. We will see later in the course that a random ordering yields a good running time in the average case.

Algorithm 5.2 Incremental algorithm

```
1: procedure CONVEX-HULL(Set of points P in ℝ²)
2:     Let p_1, ..., p_n be the points of P in any given order
3:     Let C be the (counter-clockwise) chain of the convex hull of {p_1, p_2, p_3}
4:     for i ← 4 to n do
5:         if p_i is not contained in the convex hull C then
6:             Find the common tangents of p_i with C
7:             Let a and b be the vertices of C where the tangents touch C
8:             Replace the chain a, ..., b in C with a, p_i, b
9:         end if
10:    end for
11:    return C
12: end procedure
```

References

- Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. Computational Geometry— Algorithms and Applications. Third Edition. Springer. Chapter 1.