Lecture 1

A First Algorithm: Planar Convex Hulls

We will start with a simple geometric problem, the computation of the convex hull of a finite set of points in the plane. We will formulate a basic algorithm that constructs the planar hull in quadratic time. It accesses the input points through a single predicate, the orientation predicate for three points. We will see how this predicate can be realized by a simple formula in the point coordinates. Next we discuss two techniques for improving the running time to $O(n \log n)$, where n is the number of input points. Collinear points require special care in convex hull algorithms and hence we call them a degeneracy. Finally, the algorithm would lead directly to an implementation if we had a Real-RAM to our disposal.

1.1 The Convex Hull Problem

A set is called *convex* if for any two points p and q in the set the entire line segment pq is contained in the set, see Figure 1.1. The *convex hull* $\operatorname{conv} S$ of a set S of points is the smallest (with respect to set inclusion) convex set containing S, see Figure 1.1. A point $p \in S$ is called an *extreme point* of S if there is a closed halfspace containing S such that p is the only point in S that lies in the boundary of the halfspace.

From now on we restrict our discussion to the plane. We define the convex hull problem as the problem of computing the extreme points of a finite set of points as a cyclically ordered list of point, see Figure 1.1. The cyclic order is the counter-clockwise order in which the extreme points appear on the hull.

1.2 A First Algorithm

The simplest method for constructing the convex hull works iteratively. We start with the convex hull of the first three points; we assume for simplicity that the first three points of S are not collinear and come back to this assumption in Section 1.5. For every point, we first determine whether it lies outside the current hull or not. If it is contained in the current hull, we do nothing. Otherwise, the point is an extreme point of the new hull and we update the hull by constructing the tangents from the new point to the old hull, see Figure 1.2a.

How can we determine whether a point r is contained in the current hull? Recall that the current hull is represented by its cyclic list of extreme points in counter-clockwise order, say $(v_0, v_1, \ldots, v_{k-1}, v_k = v_0)$. Consider a pair (v_i, v_{i+1}) of consecutive extreme points. Any point in the current hull lies on or to the left of the oriented line $\ell(v_i, v_{i+1})$ and every point to the right of $\ell(v_i, v_{i+1})$ lies outside the current hull, see Figure 1.2b. The geometric predicate of locating a point with respect to an oriented line is so important that we give it a name.

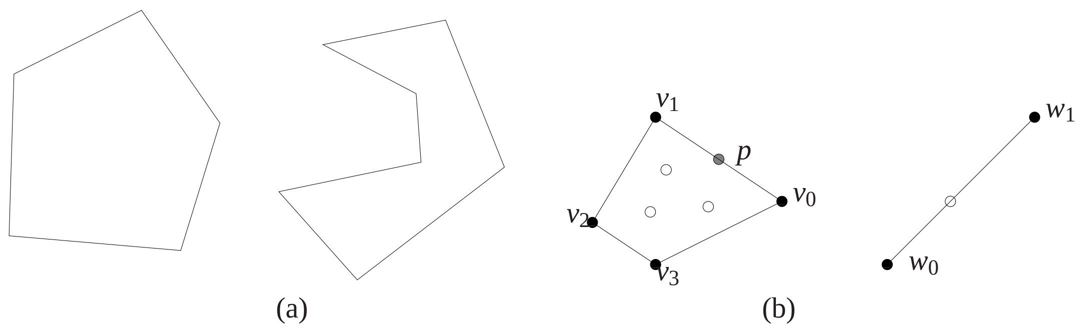

Figure 1.1: (a) shows a convex and a non-convex set, (b) shows two point sets and their convex hulls. The extreme points v_0, v_1, v_2, and v_3, respectively w_0 and w_1 are highlighted as solid disks. The point p lies on the boundary of the hull, but is not an extreme point. The cyclic clockwise list of extreme points is v_0, v_1, v_2, v_3 and w_0, w_1 (or any cycle shifts thereof), respectively.

Definition: Let p, q, and r be points in the plane (see Figure 1.3). If $p \neq q$, let $\ell(p,q)$ be the line passing through p and q and oriented from p to q. Then

$$\mathsf{Orientation}(p,q,r) = \begin{cases} +1 & \text{if } p \neq q \text{ and } r \text{ lies to the left of } \ell(p,q) \\ 0 & \text{if } p = q \text{ or } p \neq q \text{ and } r \text{ lies on } \ell(p,q) \\ -1 & \text{if } p \neq q \text{ and } r \text{ lies to the right of } \ell(p,q). \end{cases}$$

If $\mathsf{Orientation}(p,q,r) = +1$ (-1), we say that (p,q,r) form a left (right) turn, if $\mathsf{Orientation}(p,q,r) = +1$, the points are collinear. We next specialize to the convex hull problem. Assume that v_i and v_{i+1} are consecutive extreme points in the counter-clockwise order of extreme points. If r lies to the right of $\ell(v_i, v_{i+1})$, we also say that r *sees* the (counter-clockwise) hull edge v_iv_{i+1} and that this hull edge is *visible* from r.

THEOREM 1. *A point r lies outside* $\operatorname{conv} S$ *if and only if it can see at least one edge of* $\operatorname{conv} S$.

Proof. If r can see a hull edge, it is clearly outside $\operatorname{conv} S$. Assume next that $r \notin \operatorname{conv} S$ and let z be the point in $\operatorname{conv} S$ closest to r. If r lies in the interior of some hull edge then r can see this edge. So assume that z is an extreme point of S, say $z = v_i$. Then r sees at least one of the two hull edges incident to v_i. $\square$

We now know how to check whether a new point r lies outside the current hull. We simply check whether it can see some hull edge. We will see more efficient methods in Section **??**. We next turn to the update step. We need the notion of *weak visibility*. If r lies to the right of or on $\ell(v_i, v_{i+1})$, we say that r *weekly sees* the hull segment v_iv_{i+1} and that this segment is *weakly visible* from r.

THEOREM 2. *Let $(v_0, v_1, \ldots, v_{k-1})$ be the sequence of extreme points of* $\operatorname{conv} S$ *in counter-clockwise order and assume that $r \notin$* $\operatorname{conv} S$. *The hull edges weakly visible from r form a contiguous subsequence and so do the edges that are not weakly visible.*

If (v_i, v_{i+1}), …, (v_{j-1}, v_j) is the subsequence of weakly visible edges, the updated hull is obtained by replacing the subsequence $(v_{i+1}, \ldots, v_{j-1})$ by r. The subsequence $(v_i, \ldots, v_j)$ is taken in the circular sense, i.e., if $i > j$ then the subsequence is $(v_i, \ldots, v_{k-1}, v_0, \ldots, v_j)$.

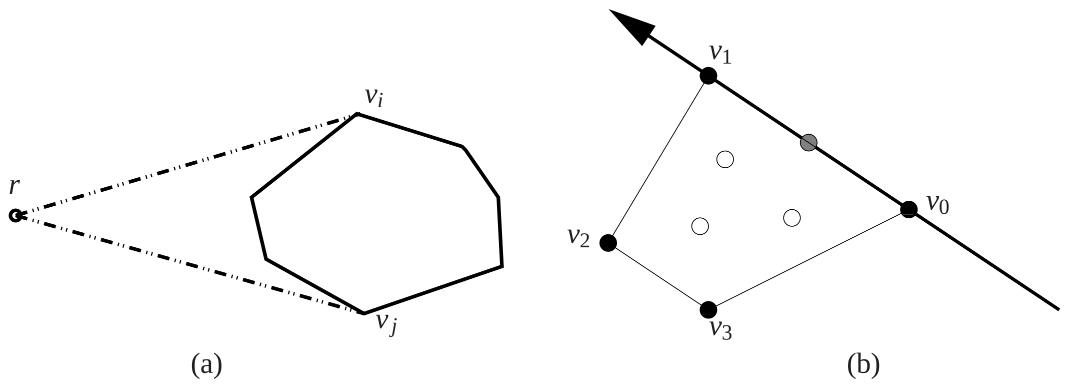

Figure 1.2: In (a) the current hull is shown as a polygon whose boundary is indicated by solid segments. The point r lies outside the current hull. The tangents from r to the current hull touch the hull in vertices v_i and v_j. The boundary of the new hull consists of the segment rv_j, followed by the part of the old hull from v_j to v_i, followed by the segment v_ir.
In (b) the oriented line $\ell(v_0, v_1)$ is highlighted. Every point to the right of this line lies outside the current hull.

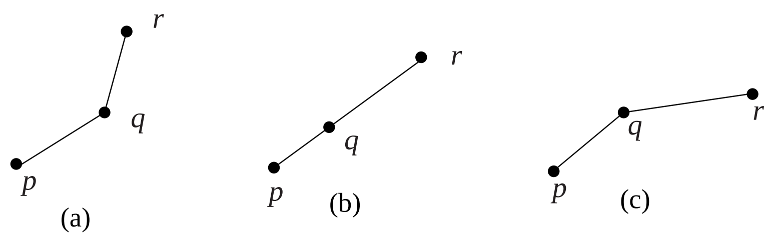

Figure 1.3: (a) shows a left turn, (b) shows collinear points, and (c) shows a right turn.

Proof. TODO □

Theorems 1 and 2 lead to the incremental convex hull algorithm shown as Algorithm 1. We still need to explain how we find all edges weakly visible from r and how we update L. Starting from the visible edge e, we move counter-clockwise along the boundary until a non-weakly-visible edge is encountered. Similarly, we move clockwise from e until a non-weakly-visible edge is encountered.

How to update the list L? We can delete the vertices in $(v_{i+1}, \ldots, v_{j-1})$ after all visible edges are found, as suggested in the above sketch ("the off-line strategy") or we can delete them concurrently with the search for weakly visible edges ("the on-line strategy").

We have now almost completed the description of our first geometric algorithm. We still need to discuss the implementation of the orientation predicate. We will see in the next section that the orientation predicate can be formulates as a simple arithmetic expression in point coordinates and hence orientation of three points can be determined in constant time.

Algorithm 1 computes the convex hull of n points in $O(n^2)$ time. For any point r, we check all edges of the current hull for visibility and maybe weak visibility. We also remove zero or more points from the current hull. Thus any point is processed in $O(n)$ time. The bound of $O(n^2)$ follows. In Section 1.4 we will

Algorithm 1 Incremental Convex Hull Algorithm

```
initialize $L$ to a counter-clockwise triangle $(a,b,c)$ with $a,b,c \in S$. Remove $a,b,c$ from $S$.
for all $r \in S$ do
    if there is an edge $e$ visible from $r$ then
        determine the sequence $((v_i,v_{i+1}),(v_{i+1},v_{i+2})\dots,(v_{j-1},v_j))$ of edges that are weakly visible from $r$.
        replace the subsequence $(v_{i+1},\dots,v_{j-1})$ in $L$ by $r$.
    end if
end for
```

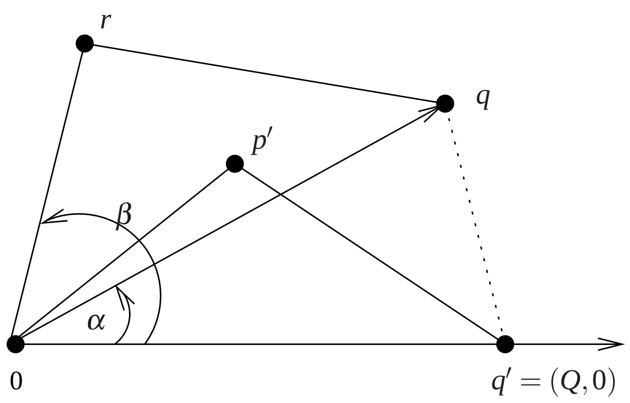

Figure 1.4: Proof of Lemma 3.

improve the running time to $O(n \log n)$.

1.3 The Orientation Predicate

LEMMA 3. *Let p, q, and r be points in the plane.*
(a) The signed area of the triangle $\triangle(p,q,r)$ is given by

$$\frac{1}{2}\begin{vmatrix} 1 & 1 & 1 \\ x_p & x_q & x_r \\ y_p & y_q & y_r \end{vmatrix}$$

(b) The orientation of (p,q,r) is equal to the sign of the determinant above.

Proof. Part(b) follows immediately from part (a) and the definition of signed area. So we only need to show part (a). We do so in two steps. We first verify the formula for the case that p is the origin and then extend it to arbitrary p. So let us assume that p is equal to the origin. We need to show that the signed area A of $\triangle(p,q,r)$ is equal to $(x_q y_r - x_r y_q)/2$.

Let α be the angle between the positive x-axis and the ray Oq and let Q be the length of the segment Oq, cf. Figure 1.4. Then $\cos\alpha = x_q/Q$ and $\sin\alpha = y_q/Q$. Rotating the triangle $\triangle(O,q,r)$ by $-\alpha$ degrees about the origin yields a triangle $\triangle(O,q',r')$ with $q' = (Q,0)$ and the same signed area. Thus, $A = Q \cdot y_{r'}/2$.

Next observe that $y'_r = R\sin(\beta - \alpha)$, where R is the length of the segment Or and β is the angle between the positive x-axis and the ray Or. Since $\sin(\beta-\alpha) = \sin\beta\cos\alpha - \cos\beta\sin\alpha$ and $R\cos\beta = x_r$ and $R\sin\beta =$

y_r we conclude that

$$\begin{aligned} A &= Q \cdot y_{r'}/2 = Q \cdot R \cdot \sin(\beta - \alpha)/2 \\ &= (Q\cos\alpha \cdot R\sin\beta - Q\sin\alpha \cdot R\cos\beta)/2 = (x_q y_r - x_r y_q)/2. \end{aligned}$$

This verifies the formula in the case where p is the origin.

Assume next that p is different from the origin. Translating p into the origin yields the triangle $\triangle(O, q', r')$ with $q' = q - p$ and $r' = r - p$[1] . On the other hand subtracting the first column from the other two columns of the determinant yields

$$\begin{vmatrix} 1 & 1 & 1 \\ x_p & x_q & x_r \\ y_p & y_q & y_r \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ x_p & x_q - x_p & x_r - x_p \\ y_q & y_q - y_p & y_r - y_p \end{vmatrix} = \begin{vmatrix} x_{q'} & x_{r'} \\ y_{q'} & y_{r'} \end{vmatrix}$$

which by the above is twice the area of the translated triangle. □

Part (b) of the lemma above is the analytical formula for the orientation predicate:

$$\mathsf{Orientation}(p,q,r) = sign(\det \begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix}) = sign((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x)). \qquad (1)$$

We have $\mathsf{Orientation}(p,q,r) = +1$ (resp., -1, 0) iff the polyline (p,q,r) represents a left turn (resp., right turn, collinearity). Interchanging two points in the triple changes the sign of the orientation.

1.4 Efficiency

Our incremental algorithm for convex hulls runs in $O(n^2)$ time on an input of n points. We show how to improve the running time to $O(n \log n)$. We first observe that the cost of updating the hull is $O(n)$, once it is know whether the new point r sees some edge.

Indeed, if r sees no edge, the old hull is the new hull and the cost of the update is zero. So assume that r sees some edge e of the current hull. We walk from e in both directions as long as edges are weakly visible. The cost of the walk is $O(1+x)$, where x is the number of edges weakly visible from r. We then delete x edges from the convex hull and add two new edges. We charge $O(1)$ to the update and to each edge removed. Since any edge can be removed only once and since at most $2n$ edges are every constructed, the total charge for the update is $O(n)$.

We next describe two techniques for finding a first visible edge or to decide that there is none.

1.4.1 A Sweep Algorithm

We simplify the search for a visible edge by processing the points in lexicographic order. A point p precedes a point q in lexicographic order if either p has the smaller x-coordinate or the x-coordinates are the same and p has the smaller y-coordinate. Sorting points according to lexicographic order takes $O(n \log n)$ time.

The advantage of processing the points in lexicographic order is twofold: First, any point is outside the convex hull of the preceding points and second, one of the edges incident to the lexicographic largest vertex of their hull is visible from it. Thus the search for a visible hull edge is trivial and takes $O(1)$ time.

THEOREM 4. *The sweep hull algorithm constructs the convex hull of n points in the plane in $O(n \log n)$ time.*

[1]Strictly speaking, we would have to write $q' = 0 + (q - p)$ and similarly for r'.

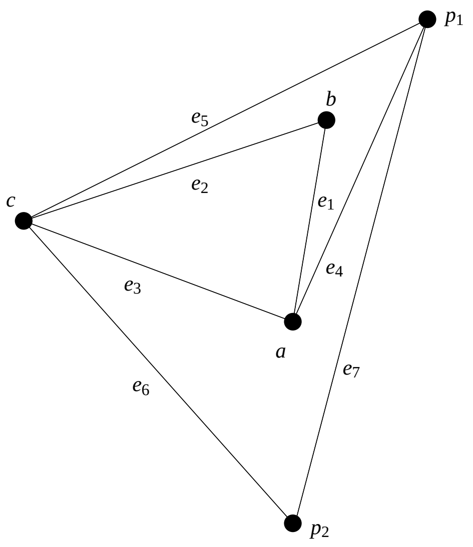

Figure 1.5: The initial convex hull consists of the points a, b, and c. When point p_1 is added the edges e_1 and e_2 are deleted from the hull and the edges e_4 and e_5 are added, and when p_2 is added to the hull the edges e_3 and e_4 are deleted from the hull and the edges e_6 and e_7 are added. The boundary of the current hull consists of edges e_7, e_5, and e_6 in counter-clockwise order. Every edge ever deleted from the hull points to the two edges that replaced it, e.g., e_3 and e_4 point to e_6 and e_7.

1.4.2 Incremental Construction

edit? Boissonnec?

We describe an alternative method for speeding up the search for a visible hull edge. The idea is to maintain the *history of the construction*. Again, we start with the counter-clockwise triangle formed by the first three points. The algorithm maintains the current hull as a cyclically linked list of edges and also keeps all edges that ever belonged to a hull. Every edge that is not on the current hull anymore points to the two edges that replaced it. More precisely, assume that S is the set of points already seen and that p is a point outside the current hull $CH(S)$. There is a chain C of edges of the boundary of $CH(S)$ that do not belong to the boundary of $CH(S \cup p)$. The chain is replaced by the two tangents from p to the previous hull. All edges in C are made to point to the two new edges, see Figure 1.5.

We are now ready to deal with the insertion of a point p. We proceed in two steps. We first determine whether p is outside the current hull and then update the hull (if p is outside).

In order to find out whether p lies outside the current hull, we walk through the history of hulls. We first determine whether p can see one of the edges of the initial triangle. If it can see no edge of the initial triangle, p lies inside the current hull and we are done. So assume that p can see an edge of the initial triangle, say e. If e is an edge of the current hull, p lies outside the current hull and e is a visible hull edge. If e is not an edge of the current hull, let r_0 and r_1 be the two edges that replaced e when $CH(S)$ was enlarged to $CH(S \cup q)$. p is outside $CH(S \cup q)$ if it sees either r_0 or r_1, see Figure 1.6. If p sees neither r_0 nor r_1, we stop. Otherwise, we set e to a visible edge among r_0 and r_1 and continue in the same fashion. In this way, the search either stops or finds a hull edge visible from p. Once we have found such an edge, we continue as in the basis algorithm.

What is the running time of the incremental construction of convex hulls? The worst case running time

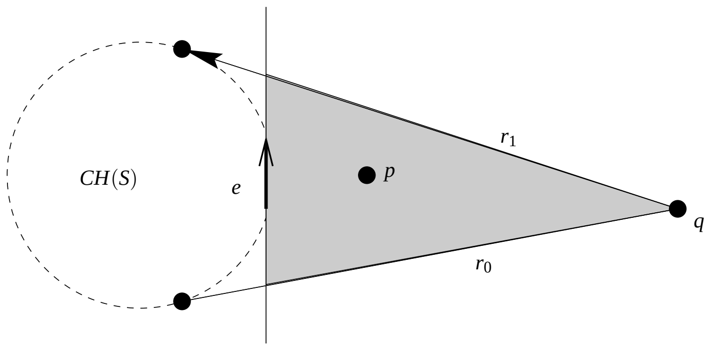

Figure 1.6: e is a (counter-clockwise) edge of the current hull and p lies to the right of it; e is replaced by r_0 and r_1 when the point q is added. If p lies neither to the right of r_0 nor to the right of r_1 then p lies in the shaded region and hence in $CH(S \cup q)$.

is $O(n^2)$ since the time to insert a point is $O(n)$. The time to insert a point is $O(n)$ since there are at most $2(k+1)$ edges after the insertion of k points and since every edge is looked at at most once in the insertion process.

The best case running time is $O(n)$. An example for the best case is when the points a, b, and c span the hull.

1.4.3 Randomized Incremental Construction*

The average case running time is $O(n \log n)$ as we will show next. What are we averaging over? We consider a fixed but arbitrary set S of n points and average over the $n!$ possible insertion orders. The following theorem is a special case of the by now famous *probabilistic analysis of incremental constructions* started by Clarkson and Shor [3]. The books [7, 2, 6, 4] contain detailed presentations of the method. The reader may skip the proof of Theorem 5. Why do we include a proof at all given the fact that the method is already well treated in textbooks? We give a proof because the cited references prove the theorem only for points in general position. We want to do without the general position assumption in this book.

THEOREM 5. *The average running time of the incremental construction method for convex hulls is $O(n \log n)$.*

Proof. We assume for simplicity that the points in S are pairwise distinct. The theorem is true without this assumption; however, the notation required in the proof is more clumsy.

The running time of the algorithm is linear iff all points in S are collinear. So let us assume that S contains three points that are not collinear. In this case we will first construct a triangle and then insert the remaining points. Let p be one of the remaining points. When p is inserted, we first determine the position of p with respect to the initial triangle (time $O(1)$), then search for a hull edge e visible by p, and finally update the hull. The time to update the hull is $O(1)$ plus some bounded amount of time for each edge that is removed from the hull. We conclude that the total time (= time summed over all insertions) spent outside the search for a visible hull edge is $O(n)$.

In the search for a visible hull edge we perform tests *rightturn*(x, y, p) where x and y are previously inserted points. We call a test *successful* if it returns true and observe that in each iteration of the while-loop at most two rightturn tests are performed and that in all iterations except the last at least one rightturn test is successful. It therefore suffices to bound the number of successful rightturn tests.

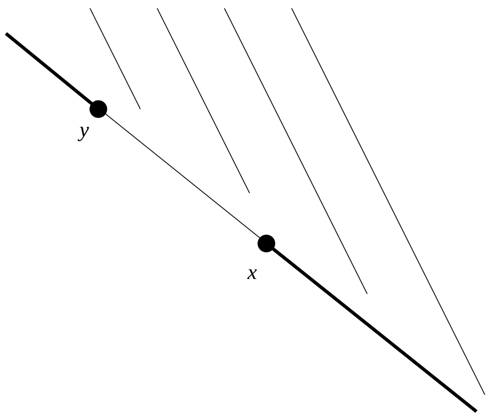

Figure 1.7: $K_{x,y}$ consists of all points in the shaded region plus the two solid rays.

For an ordered pair (x,y) of distinct points in S we use $K_{x,y}$ to denote the set of points z in S such that *rightturn*(x,y,z) is true plus[2] the set of points on the line through (x,y) but not between x and y, see Figure 1.7. We use k_{xy} to denote the cardinality of $K_{x,y}$, F_k to denote the set of pairs (x,y) with $k_{xy} = k$, $F_{\le k}$ to denote the set of pairs (x,y) with $k_{xy} \le k$, and f_k and $f_{\le k}$ to denote the cardinalities of F_k and $F_{\le k}$, respectively. We have

LEMMA 6. *The average number A of successful rightturn tests is bounded by* $\sum_{k\ge 1} 2 f_{\le k}/k^2$.

Proof. Consider a pair (x,y) with $k_{xy} = k$. If some point in $K_{x,y}$ is inserted before both x and y are inserted then (x,y) is never constructed as a hull edge and hence no rightturn tests $(x,y,-)$ are performed. However, if x and y are inserted before all points in $K_{x,y}$ then up to k successful rightturn tests (x,y,z) are performed.

The probability that x and y are inserted before all points in $K_{x,y}$ is

$$2!k!/(k+2)!$$

since there are $(k+2)!$ permutations of $k+2$ points out of which $2!k!$ have x and y as their first two elements. Thus the expected number of successful rightturn tests (x,y,z) is bounded by

$$2!k!/(k+2)! \cdot k = 2 \cdot k/(k+1)(k+2) < 2/(k+1).$$

The argument above applies to any pair (x,y) and hence the average number of successful rightturn tests is bounded by

$$\sum_{k\ge 1} 2 f_k/(k+1).$$

We next write $f_k = f_{\le k} - f_{\le k-1}$ and obtain

$$\begin{aligned} A &\le \sum_{k\ge 1} 2(f_{\le k} - f_{\le k-1})/(k+1) = \sum_{k\ge 1} 2 f_{\le k}(1/(k+1) - 1/(k+2)) \\ &= \sum_{k\ge 1} 2 f_{\le k}/((k+1)(k+2)). \end{aligned}$$

□

[2]The set to be defined next is empty if S is in general position. The probabilistic analysis of incremental constructions usually assumes general position. We do not want to assume it here and hence have to modify the proof somewhat.

It remains to bound $f_{\leq k}$. We use random sampling to derive a bound.

LEMMA 7. *$f_{\leq k} \leq 2e^2 n \cdot k$ for all k, $1 \leq k \leq n$.*

Proof. There are only n^2 pairs of points of S and hence we always have $f_{\leq k} \leq n^2$. Thus, the claim is certainly true for $n \leq 10$ or $k \geq n/4$.

So assume that $n \geq 10$ and $k \leq n/4$ and let R be a random subset of S of size r. We will fix r later. Clearly, the convex hull of R consists of at most r edges. On the other hand, if for some $(x,y) \in F_{\leq k}$, x and y are in R but none of the points in $K_{x,y}$ is in R, then (x,y) will be an edge of the convex hull of R. The probability of this event is

$$\frac{\binom{n-i-2}{r-2}}{\binom{n}{r}} \geq \frac{\binom{n-k-2}{r-2}}{\binom{n}{r}},$$

where $i = k_{x,y}$. Observe that the event occurs if x and y are chosen and the remaining $r-2$ points in R are chosen from $S \setminus \{x,y\} \setminus K_{x,y}$. The expected number of edges of the convex hull of R is therefore at least

$$f_{\leq k} \cdot \frac{\binom{n-k-2}{r-2}}{\binom{n}{r}}.$$

Since the number of edges is at most r we have

$$f_{\leq k} \cdot \binom{n-k-2}{r-2} \Big/ \binom{n}{r} \leq r$$

or

$$f_{\leq k} \leq r \cdot \binom{n}{r} \Big/ \binom{n-k-2}{r-2} = r \cdot \frac{n(n-1)}{r(r-1)} \cdot \frac{[n-2]_{r-2}}{[n-k-2]_{r-2}},$$

where $[n]_i = n(n-1)\cdots(n-i+1)$. Next observe that

$$\begin{aligned}\frac{[n-2]_{r-2}}{[n-k-2]_{r-2}} &\leq \frac{[n]_r}{[n-k]_r} = \prod_{i=0}^{r-1} \frac{n-i}{n-k-i} = \prod_{i=0}^{r-1} \left(1 + \frac{k}{n-k-i}\right) \\ &= \exp\left(\sum_{i=0}^{r-1} \ln(1 + k/(n-k-i))\right) \leq \exp\left(rk/(n-k-r)\right),\end{aligned}$$

where the last inequality follows from $\ln(1+x) \leq x$ for $x \geq 0$ and the fact that $k/(n-k-i) \leq k/(n-k-r)$ for $0 \leq i \leq r-1$. Setting $r = n/(2k)$ and using the fact that $n-k-r \geq n/4$ for $k \leq n/4$ and $n \geq 10$, we obtain

$$f_{\leq k} \leq e^2 n^2 / r = 2e^2 nk.$$

□

Putting our two lemmas together completes the proof of Theorem 5

$$A \leq 4e^2 \sum_{k \geq 1} nk/k^2 = O(n \log n).$$

□

There are two important situations when the assumptions of the theorem above are satisfied:

- When the points in S are generated according to a probability distribution for points in the plane.
- When the points are randomly permuted before the incremental construction process is started. We then speak about a *randomized incremental construction*.

1.5 Degeneracy

We assumed that the first three points in the input span a proper triangle. How can we remove this assumption?

In an off-line setting, i.e., all points are available at program start, we scan over the points once. Let p be the first point. We scan until we find a point q that is different from p. If all input points are equal to p, the convex hull is equal to the set consisting only of p. So assume we have two distinct points p and q. We continue scanning until we find a point r that is not collinear to p and q. If there is no such point, the convex hull is contained in the line passing through p and q and we simply need to find the two extreme points on the line. If there is such a point, we have found the initial triangle.

In an on-line setting, we have to work slightly harder. We initialize the hull to $\{p\}$. As long as input points are equal to p, there is nothing to do. As soon, as we encounter a point q different from p, we know that the hull is at least one-dimensional. The current hull is the line segment pq. As long as input points are collinear to p and q, the hull stays a segment and we update it accordingly. Once an input point r that is not collinear with p and q comes along, we know that the hull is two-dimensional and we switch to the algorithm discussed in the preceding sections.

If no three points are collinear, the assumption is trivially satisfied. Also, there is no need to distinguish between visible and weakly visible edges as there are no edges that are weakly visible but not visible. Collinear points make the formulation of convex hull algorithms more complex and therefore we call them a *degenerate configuration* for the convex hull problem.

Geometric algorithms are frequently formulated under the *non-degeneracy assumption* or *general position assumption*: The input contains no degenerate configuration. In Lecture **??** we will study perturbation as a general technique for ensuring general position.

1.6 The Real-RAM

We have an algorithm for planar convex hulls. *Do we have an implementation, i.e., is it straight-forward to convert the discussion into a running program in a popular programming language? The answer is No.*

In the formulation of the algorithm we have tacitly assumed the *Real-RAM* model of computation. A Real-RAM is a random access machine with the capability of handling real numbers. Of course, the operations on real numbers follow the laws of mathematics. The Real-RAM model is the natural computing model for geometric computing and numerical analysis. After all geometric objects are usually specified by real parameters: point coordinates are reals, the radius of a circle is a real, plane coefficients are reals, and so on.

Unfortunately, one cannot buy a Real-RAM. Real computers do not come with real arithmetic. They provide only floating point arithmetic and bounded integer arithmetic. We will study the effect of floating point arithmetic on geometry in the next lecture. We will see that we are far from an implementation.

In Lectures **??** to **??** we will then discuss the efficient realization of a Real-RAM to the extent needed by the convex hull algorithm and any other geometric algorithm that deal only with linear objects.

1.7 Historical Notes

The sweep hull algorithm was proposed by Andrew [1]; it refines an earlier algorithm of Graham [5].

randomized incremental algorithm [3]. Dimension jumps first in

1.8 Implementation Notes

1.9 Exercises

Bibliography

[1] A. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9:216–219, 1979.

[2] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, Cambridge, 1998.

[3] K. Clarkson and P. Shor. Applications of random sampling in computational geometry, II. *Journal of Discrete and Computational Geometry*, 4:387–421, 1989.

[4] M. de Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 1997.

[5] R. L. Graham. An efficient algorithm for determining the convex hulls of a finite point set. *Information Processing Letters*, 1:132–133, 1972.

[6] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[7] K. Mulmuley. *Computational Geometry*. Prentice Hall, 1994.