Lecture 9: Computational geometry

Has applications in, for instance, computer graphics, robotics, VLSI design, and computer-aided design.

We consider finitely representable objects in the plane composed of: points $p = (x, y)$, lines (going through two points), line segments $\overline{p_0p_1}$, directed segments $\overrightarrow{p_0p_1}$.

A *polygon* is a closed curve of segments. It is *simple* if the curve does not intersect itself.

For a *convex polygon* a segment between two arbitrary points, internal or on the boundary of the polygon, has all its points internal or on the boundary.

Properties of line segments

We can answer the following questions in $O(1)$ time, using additions, subtractions, multiplications and comparisons.

1. Is the directed segment $\overrightarrow{p_0p_1}$ clockwise from $\overrightarrow{p_0p_2}$?
2. Given $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_1p_2}$ is there a left turn at p_1?
3. Do $\overrightarrow{p_1p_2}$ and $\overrightarrow{p_3p_4}$ intersect each other?

The **cross product** of vectors p_1 and p_2 can be seen as the signed area of the parallelogram formed by the four points $(0, 0), p_1, p_2$, and $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$, i.e. the determinant of a matrix:

$$p_1 \times p_2 = \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1y_2 - x_2y_1 = -p_2 \times p_1$$

If $p_1 \times p_2 > 0$ then p_1 is clockwise from p_2 wrt origo $(0, 0)$. If $p_1 \times p_2 < 0$ then p_1 is counterclockwise from p_2. When $p_1 \times p_2 = 0$ the vectors are collinear, i.e. pointing in the same or opposite directions.

So to decide whether $\overrightarrow{p_0p_1}$ is clockwise from $\overrightarrow{p_0p_2}$, compute the cross product: $(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$.

Decide if consecutive segments turn left or right

Given $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_1p_2}$ is there a left turn at p_1? Equivalently, we want to know which way an angle $\angle p_0p_1p_2$ turns. Just check if $\overrightarrow{p_0p_2}$ is clockwise or counterclockwise from $\overrightarrow{p_0p_1}$, by computing the cross product $(p_1 - p_0) \times (p_2 - p_0)$. If it is positive, there's a left turn at p_1.

Decide if two segments intersect

First try a quick rejection: the segments cannot intersect if their *bounding boxes* do not intersect. The bounding box is the smallest rectangle whose sides are parallel to the x- or y-axis and contains the segment.

If the bounding boxes intersect, investigate if each segment *straddles* the line containing the other segment; in which case the segments do intersect.

Can use the method with cross products to decide if $\overrightarrow{p_3p_4}$ straddles the line containing the points p_1 and p_2, and if $\overrightarrow{p_1p_2}$ straddles the line containing the points p_3 and p_4. The first holds if $\overrightarrow{p_1p_3}$ and $\overrightarrow{p_1p_4}$ have different orientations relative to $\overrightarrow{p_1p_2}$, the other holds if $\overrightarrow{p_3p_1}$ and $\overrightarrow{p_3p_2}$ have different orientations relative to $\overrightarrow{p_3p_4}$.

Determine relative orientations with cross product by checking if we get different signs for $(p_3 - p_1) \times (p_2 - p_1)$ and $(p_4 - p_1) \times (p_2 - p_1)$, and for $(p_1 - p_3) \times (p_4 - p_3)$ and $(p_2 - p_3) \times (p_4 - p_3)$.

In the 3rd edition of the course book the quick rejection step is skipped, but then intersection between $\overrightarrow{p_1p_2}$ and $\overrightarrow{p_3p_4}$ has to include a *boundary case* when the segments are collinear and overlapping.

Intersection between n segments

We just want to find one intersection; on tutorial 4 we consider the problem of finding all intersections. The segments can be arbitrarily oriented, but for simplicity we assume they are not vertical. Can then order the segments relatively in y direction wrt the x value of a sweep line (Figure 33.4 page 1023). We maintain two sets of data:

Sweep-line status: a red-black tree T of segments that the sweep line currently intersects, relatively ordered. At insertion and deletion from T, the usual comparisons between keys needed for tree traversal are replaced by cross products to determine the relative order between segments (tutorial 4).

Event list: the $2n$ segment endpoints sorted by x value.

1. If the event is a left endpoint of a segment ℓ then insert ℓ into T.

   If ℓ intersects a neighbor above or below it in T then we have found an intersection.

2. If the event is a right endpoint of a segment ℓ then delete ℓ from T.

   If ℓ's two neighbors intersect then we have found an intersection.

Example: Figure 33.5 page 1026.

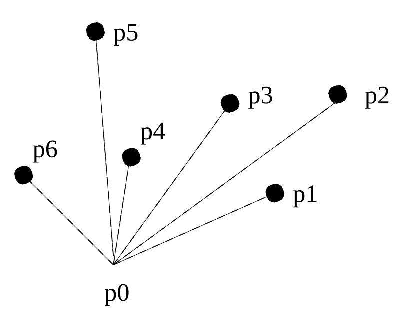
Sorting the event list takes $O(n \log n)$ time. Updating T takes $O(\log n)$ time for at most $2n$ events, i.e. $O(n \log n)$ total time.

Convex hulls

Given n points, compute the smallest convex polygon of segments that enclose the points.

Illustration: a tight rubber band that surrounds nails sticking out from a board.

Graham's scan: Choose lowest point p_0, sort other points by polar angle counterclockwise around p_0. Scan the points in that order, $\langle p_1, \ldots, p_{n-1} \rangle$. Note that p_1 and p_{n-1} must be vertices of the convex hull.



Let $\langle q_0, q_1, \ldots, q_k \rangle$ be the convex hull for $\langle p_0, p_1, \ldots, p_i \rangle$, where $q_0 \equiv p_0,\ q_1 \equiv p_1$.

For the next point p_{i+1} consider the angle between $\overrightarrow{q_{k-1}q_k}$ and $\overrightarrow{q_k p_{i+1}}$.

If there's a left turn then q_{k+1} is p_{i+1}, otherwise consider the angle between $\overrightarrow{q_{k-2}q_{k-1}}$ and $\overrightarrow{q_{k-1}p_{i+1}}$. Eliminate vertices on the q list until there's a left turn.

Sorting takes $O(n\ \log n)$ time. Thereafter a point may be included in a convex hull at most once, and be deleted at most once, i.e. the scan takes linear time. Hence, the total time is $O(n\ \log n)$.

Jarvis's march (gift wrapping): Start in the lowest point, p_0, and form right and left chains of the convex hull. The vertex q_1 is the one of smallest polar angle wrt p_0, q_2 has smallest angle wrt q_1, and so on, until the highest vertex, q_j, is reached. That completes the right chain. The left chain is computed similarly, by finding a vertex q_{j+1} of smallest angle wrt q_j from the negative x-axis. Continue until p_0.

A smallest polar angle is found in $O(n)$ time, so Jarvis's march takes time $O(n\,h)$, where h is the number of vertices on the convex hull. This gives a better time complexity than Graham's scan if $h = o(\log n)$.

There are also algorithms that runs in $O(n \log h)$ time. We can even get an $o(n \log n)$ algorithm by using *fusion tree* sorting.

Closest pair

Given n points, find two that are closest to each other. Two points may coincide, i.e. be at distance 0.

Naive solution: examine all $\binom{n}{2}$ point pairs, which takes $\Theta(n^2)$ time.

Divide-and-conquer algorithm: Divide the problem in P_L and P_R with half the points each (sorted by x value), and solve P_L and P_R recursively.

Let δ_L be the distance between two closest points in P_L, and δ_R be the distance between two closest points in P_R. Let $\delta = \min(\delta_L, \delta_R)$.

To see if there are closer points than the pairs within P_L and P_R, examine if any point in P_L has a point in P_R which is at distance less than δ. It suffices to look at the points in a strip of width 2δ centered between P_L and P_R.

Sort the points in the strip by y-coordinate. This gives a list Y'.

We need only consider distances from each point p in Y' to 7 points in Y' following p, since there can be at most 8 points in a rectangle of length δ and width 2δ within the strip.

Note that two points can coincide as there may be double points on the dividing line if n is even.

$$\text{Time: } T(n) = \begin{cases} O(1) & \text{if } n \leq 3 \\ 2\,T(n/2) + O(n\ \log n) & \text{if } n > 3 \end{cases}$$

with solution $T(n) = O(n \log^2 n)$.

Improvement by presorting all points by y-coordinate, in $O(n\ \log n)$ time. This gives the list Y.

We can then pick out the points in P_L and P_R (sorted by y value) for lists Y_L and Y_R, by traversing Y and ignore points whose x-coordinates are outside. This takes $O(n)$ time.

Similarly for the points in the middle strip list Y'.

Note that the lists Y_L and Y_R are passed on in the recursive calls.

Thereby the running time is $T(n) = 2\,T(n/2) + O(n) = O(n\ \log n)$, including the presorting cost.