Mauricio Galindo 116733
Rodrigo Higuera 117249

Finding the convex hull

Introduction

We analyze a concrete algorithm, the Graham's scan algorithm, for finding the convex hull of a set of points. Before doing so we define some concepts, which are important for the proposed problem of finding the convex hull.

Definitions

Polygon
A polygon is a piecewise-linear, closed curve in the plane. That is, it is a curve ending on itself that is formed by a sequence of straight-line segments, called the sides of the polygon. A point joining two consecutive sides is called a vertex of the polygon. If the polygon is simple, it does not cross itself. The set of points in the plane enclosed by a simple polygon forms the interior of the polygon, the set of points on the polygon itself forms its boundary, and the set of points surrounding the polygon forms its exterior.

Convex Polygon
A simple polygon is convex if, given any two points on its boundary or in its interior, all points on the line segment drawn between them are contained in the polygon's boundary.

Convex hull
The convex hull of a set of Q points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

Graham's scan

Graham's scan algorithm, as all the other algorithms that solve the convex hull problem, receives an input of n points and outputs a set of points in counterclockwise order that forms a polygon which constitutes the convex hull of the input set of points. From now on we shall refer to the convex hull of a set of points Q as CH(Q).
Graham's scan solves the convex-hull hull problem by maintaining a stack of S candidate points. Each point on the input set Q is pushed once into the stack. Eventually the points that are not vertices of CH(Q) are popped from the stack. When it terminates, the stack S contains exactly the vertices of CH(Q), in counterclockwise order of their appearance on the boundary.
The running time of this algorithm is nlog(n).

The pseudo code for the graham's scan procedures is given below.

```
def graham_scan(q):
	p0 = find_min_coordinate(q)
```

```
    # sort in order by polar angle

    q = sort_by_polar_angle_to_point(q, p0)
    q = discard_colinear_pints(q, p0)

    # initialize and set values for new stack
    s = new stack()
    s.push(q[0])
    s.push(q[1])
    s.push(q[2])

    for i=3 to m:
        while cross_product(s.next_to_top(), s.top(),  q[i])<= 0:
            # pop
            s.pop()
        s.push(q[i])

    return s
```

We first find the coordinate with the lowest y coordinate (ties are broken by x coordinate). We then proceed to sort the set of points by their angle with respect with the reference point p0, which we found in the previous step. The sort is done using the cross-product of the two points with respect to p0. If a point p1 is to the left of another point p2, the angle formed by this point and the reference point will be greater that the angle formed with p0 and p1. If the opposite true, then the angle will be smaller. We can compute the positions of the points using the cross-product, and thus avoiding the need for expensive trig functions.

Defining the cross-product for the 2-dimension space as the signed are of the parallelogram formed by the two vectors, as follows:

$$p_1 \times p_2 = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1$$

We can use this to compute the relative position of the point without computing its actual angle. In order to do this we first show that

$$p_1 \times p_2 = \|p_1\| \|p_2\| \sin\phi$$

$$\Rightarrow$$

$$\sin\phi = \frac{p_1 \times p_2}{\|p_1\| \|p_2\|}$$

And we also know that

$\|p_1\| > 0$
$\|p_2\| > 0$
$\Rightarrow$
$\|p_1\|\|p_2\| > 0$
$\therefore$
$\sin\phi = 0 \Leftrightarrow p_1 \times p_2 = 0$
$\sin\phi > 0 \Leftrightarrow p_1 \times p_2 > 0$
$\sin\phi < 0 \Leftrightarrow p_1 \times p_2 < 0$
therefore
$\theta = 0 \Rightarrow p_1 \times p_2 = 0$
$0 < \theta < \pi \Rightarrow p_1 \times p_2 > 0$
$\pi < \theta < 2\pi \Rightarrow p_1 \times p_2 < 0$

We proceed to initialize the stack with the first points in our ordered set, then for every other point we compute its position with respect two the top two points in the stack, again we do this using a cross-product, if we find that this point is to the right we pop an element from the stack and repeat this process until this condition no longer holds. After this, we push the current point into the stack.

When the procedure finishes, the stack will contain the vertex points of the polygon that forms the convex hull of our set of points Q, CH(Q), ordered in counter-clockwise order.

Proof of correctness

After sorting and removing all collinear points, except the farthest one, we end up with a sequence of p1 … pm.
Qi = {p0, p1, …, pi}.
Q – Qm the set of points that were removed because they were collinear relative to p0 with some as some point in Qm. These points are not in CH(Q) because all points that can be expressed as a convex combination of other point in the convex hull is not in the convex hull. Therefore CH(Qm) = CH(Q), and we only need to prove that when graham's scan terminates, the stack consists of the vertices of CH(Qm) in counterclockwise order.

Note that the points p0, p1 and pm are points of Qm, this are the points in the extremes of Qm and so they must belong to Qm, and p0, p1 and p1 are vertices of CH(Qi)

By induction

At the start of each iteration of the for loop of line 10 to 14 the stack s consists of, from bottom to top, the vertices of CH(Qi-1) in counterclockwise order.

At the beginning of each iteration of the for loop, the top point on the stack s is pi-1, which was pushed either at the end of the previous iteration or before the first iteration.
Let pj be the top point on s after executing the while loop but before we push pi onto the stack, and let pk be the point just below the top of the stack, pj. When pj is at the point of the stack s and we have not yet pushed pi, the stack s contains the same points it contained after iteration j of the for loop, By the loop invariant, then, s contains the points of the vertices of CH(Qj)

Before the first iteration of the for loop this holds, as the stack s consists of the vertices of Q2 = Qi-1, which for a triangle that form its own convex hull and appear in counterclockwise order from bottom to top.

When the for loop ends, we have i = m + 1, and so out initial condition implies that S consists of exactly the vertices of CH(Qm) = CH(Q), in counterclockwise order from bottom to top.
Before pushing pi, we know that pi's polar angle relative to p0 is greater tha pj's polar angle and that the angle <pk,pj,pi makes a left turn, otherwise we would have popped pj (Its either inside the triangle formed by the points p0, pj and pi or on one of the sides of the triangle. I.e. is a convex combination of the previous points, and by definition it can't be a vertex of CH(Q).) Then , because s contains exactly the vertices of CH(Qj), once we push pi, the stack will contain the vertices of CH(Qj U {pi}).

Now lets consider any point pt that was popped during the iteration I fi the for loop, and let pr be the point just below pt on the stack s at the time pt was popped (it might be the case that pr = pj). The angle <prptpi make a nonleft turn, and the angle of pt relative to p0 is greater than the angle of pr. Pt must be in the interior of the triangle formed by p0, pr, and pi or on a side of the triangle (but it is not a vertex), Since pt is within a triangle formed by three other points og Qi, it cannot ve a vertex of Qi. We then have that

CH(Qi – {pt}) = CH(Qi) -> eq 1

Let Pi be the set of points that were popped during the iteration I of the for loop. CH(Qi - Pi) = CH(Qi). But Qi – Pi = Qj U {pi}, and so

CH(Qj U {pi}) = CH(Qi - Pi) = CH(Qi)

Then once we push pi onto the stack, the stack s contains the vertices of CH(Qi) in counterclockwise order. Incrementing i will then cause the loop invariant to hold for the next iteration

Running time analysis

Finding the minimum coordinate take O(n) time.
Sorting the points takes ønlog(n) using an algorithm like merge-sort or heap-sort.
Pushing into the stack takes constant time.

A simple analysis of the order of the for and while loop would yield a running time of O(n^2), however if we take a closer look we see that this isn't the case, simply because the aggregate cost of the while loop cant be greater the O(m) fol all the iterations of the for loop.

The for loops executes m times.
$m < n$

Since we only push one time each point onto the stack, we cant pop the stack more than m times, therefore the while loop executes at most m times, so the combined time of the for loop and while loop is O(m).

The previous analysis yields a running time of nlog(n), which is the dominant factor of the algorithm (the sorting step). For the Graham's scan algorithm.