

**Graham's Scanning**



The first paper published in the field of computational geometry was on the construction of convex hull on the plane. And the honor goes to Graham. In the late 1960s, the best algorithm for convex hull was O(*n*2). At Bell Laboratories, they required the convex hull for about 10,000 points and they found out this O(*n*2) was too slow. In 1972, R. L. Graham developed his simple and efficient algorithm in response to this need. The algorithm is efficient in the sense that no matter how many points are on the boundary of the convex, it runs efficiently. Historically, this is the first publication that showed convex hull computation in O(*n* lg *n*) time in the worst case. In 1981, A. C. Yao proved that it is optimal in the worst-case sense. Note that O'Rouke reported that Toussaint (1985) found an earlier paper by Bass & Schubert (1967) that contained many ideas that appeared in later hull algorithms. The problem with Graham's algorithm is that it has no obvious extension to three dimensions. The reason is that the Grahams scan depends on angular sorting, which has no direct counterpart in three dimensions.

**Overall Strategy**

The Graham's algorithm first explicitly sorts the points in O(*n* lg *n*) and then applies a linear-time scanning algorithm to finish building the hull. The Graham's scan algorithm for computing the convex hull, CH, of a set Q of *n* points in the plane consists of the following three phases:

**Phase I**. First, select a anchor point (base point) *p*0 in Q, normally this is the point with minimum y-coordinate. In case of the tie, we select leftmost point (minimum x-coordinate) in the set.

**Phase II**. Next, sort the remaining points of Q (that is, Q − {*p*0}) lexicographically by polar angle, measured in radians. Interior points on the ray cannot be a convex hull points and remove these points during sort. Once the points are sorted, we connected them in counterclockwise order with respect to the anchor point *p*0. The result is a simple polygon as in figure below. Note that no explicit computation of angles is performed by the algorithm.



Note that a simple polygon formed in the sorting phase of Graham's scan.

**Phase III**. After pushing the anchor point *p*0 onto the stack S, we scan through the points in counterclockwise order, maintaining at each step a stack S containing a convex chain surrounding the points scanned so far. Each time we consider the new point *pi*, we perform the following test:

**a**. If *pi* forms a left turn with the last two points in the stack S, or if S contains fewer than two points, then push *pi* onto the stack S.

**b**. Otherwise, pop the last point from the stack S and repeat the test for *pi*.

We halt when we return to the anchor point *p*0, at which point stack S stores the vertices of the convex hull of Q in counterclockwise order.

**Formal Algorithm**

The details of the scan phase (Phase 3) are formalized in the following algorithm: [For the sake of completeness, we are including the other two phases too.]

GRAHAM\_SCAN(Q)

1.         Find *p*0 in Q with minimum y-coordinate (and minimum x-coordinate if there are ties).
2.         Sorted the remaining points of Q (that is, Q − {*p*0}) by polar angle in counterclockwise order with respect to *p*0.
3.         TOP [S] = 0                ▷ Lines 3-6 initialize the stack to contain, from bottom to top, first three points.
4.         PUSH (*p*0, S)
5.         PUSH (*p*1, S)
6.         PUSH (*p*2, S)
7. **for** *i* = 3 **to** *m*             ▷ Perform test for each point p3, ..., p*m*.
8.     **do while** the angle between NEXT\_TO\_TOP[S], TOP[S], and *pi* makes a nonleft turn  ▷ remove if not a vertex
9.                 **do** POP(S)
10.                 PUSH (S, *pi*)
11. **return** S

**Running Time Computation**

Line 1 requires O(*n*) time to find the anchor point *p*o (i.e., point with minimum y-coordinate). Since sorting based on polar angle takes O(*n* lg *n*) time, Line 2 requires O(*n* lg *n*) time (using merge or heap sort to sort polar angles). Moreover, removal of *n* − *m*points with duplicate angles takes O(*n*) time. Lines 3, 4, 5, and 6 require constant time, O(1), to initialize the stack to contain the first three points. **For-loop** in Line 7 is executed *m* − 2 times, so it takes O(*n*) time.

Now the **while-loop** in Line 8. This is a "problem". It may iterate as many as O(*n*) time, which leads to an over-estimate of O(*n*2). Note that each pass through the **while** statement, POP is executed.

We use the aggregate method of amortized analysis to analyze the **while-loop**. As in analysis of MULTIPOP, we observe there is at most one pop for each push operation. In case of GRAHMS SCAN, at least three points (*p*0, *p*1 and *p*m) are never popped from the stack. So in fact, at most *m* − 2 pop operations are performed. Therefore, the amortized cost of each Iteration of while loop is O(*n*)/*n* = O(1). It implies that the overall worst-case for for-loop is O(*n*). The worst case running time of GRAHAM\_SCAN is

T(*n*) = O(*n*) + O(*n* lg *n*) + O(1) + O(*n*) = O(*n* lg *n*),  where *n* = |Q|.

**Correctness of Grahams Scan**

The correctness of Graham scan lies in two situations.

Part 1 deals with **nonleft** turns.
Part 2 deals with **left** turns.

***Claim*1:**Each point popped from the stack is not a vertex of CH(Q) and lies inside new CH(S).



***Proof of Claim*1**

  Suppose that point p*j* is popped from the stack because angle  *pkpj pi* makes a nonleft turn as shown in the above figure. Since points are ordered in increasing polar angle around anchor *p*0, there exists a triangle ∆ *p*0 *pi* *pk* with *pj* either in the interior of the triangle or on the line segment joining *pi* and *pk* (geometric property). In either case, point *pj* cannot be a vertex of CH(Q).  And this completes the proof of claim 1.

***Claim*2**    The points on stack always form the vertices of a convex polygon.

***Proof of Claim*2**    The claim holds immediately after line 6, since points *p*0, *p*1, and *p*2 form a convex polygon (i.e. a triangle). Now examine how stack changes during the course of the algorithm. Stack changes in two ways:

Case i: When points are popped onto the stack S.
Case ii. When points are pushed onto the stack S.

**Case i**. Here we rely on the following geometric property:

If a vertex is removed from a convex polygon, then the resulting polygon is convex.

**Case ii**. To analyze this case, we rely on the following geometric property:



If we add point *ps* in the shaded regions to polygon P, then the resulting polygon is convex.

Now back to the case ii. Consider a point *pi* that is pushed onto the stack S. Keeping the above geometric property in mind, we claim the following:

***Subclaim***    If a point *pi* is pushed onto the stack, the resulting points form a convex polygon.



***Proof of Subclaim***    Suppose *pj* be the vertex on the top of the stack and also *pk* be the predecessor of *pj* on the stack.

* By the choice of *p*0 and *p*1, *pi* lies above the extension of the line *p*0*p*1. As a result, *pi* lies on the shaded region.
* By the fact, a left turn occurs at *pj*, *pi* lies below the extension of the line joining *pk* and *pj*. As a result *pi* lies on the shaded region.
* By the fact, *pi* occurs later in the sequence than *pj*, the polar angle of *p*0*pi* is greater than polar angle of *p*0*pj* thus *pi* of lies left of the linejoining points*p*0and*pj*. As a result *pi* lies in the shaded region.

So, the above three propositions imply that *pi* is in the shaded region. (And we know this corresponds directly to the shaded region of our geometric property.) Therefore, after *pi* has been pushed onto the stack, the points on stack form a convex polygon. And this completes the proof.

**Conclusion**

Since each point popped from the stack is not a vertex of CH(Q) ( by claim 1). And the points on stack always form the vertices of a convex polygon (by claim 2). Therefore, Grahams scan is correct.

**Applet**

Graham's scan starts with 25 random points, and then computes their convex hull.

The algorithm works in three phases (as mentioned above):

1. Find an extreme point. This point will be the anchor. The anchor is guaranteed to be on the hull, and is chosen to be the point with smallest y coordinate.
2. Sort the points in order of increasing angle around the anchor. We end up with a star-shaped polygon. Note that the anchor can "see" the whole polygon.
3. Build the hull, by marching around the star-shaped polygon, adding edges when we make a left turn, and back-tracking when we make a right turn.



Updated: May 1, 2010.