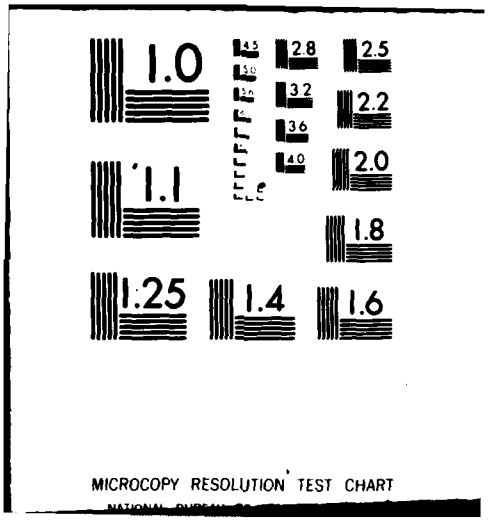
AD-A113 420 STANFORD UNIV CA DEPT OF COMPUTER SCIENCE F/G 12/1
FINDING THE CONVEX HULL OF A SIMPLE POLYGON,(U)
NOV 81 R L GRAHAM, F YAO N00014-81-K-0269
UNCLASSIFIED STAN-CS-81-887 NL

END
DATE FILMED
5-82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART

November 1981

Report. No. STAN-CS-81-887

13

Finding the Convex Hull of a Simple Polygon

by

Ronald L. Graham

Frances Yao

Department of Computer Science

Stanford University
Stanford, CA 94305

82

Finding the Convex Hull of a Simple Polygon

Ronald L. Graham*
Computer Science Department
Stanford University

and

F. Frances Yao
Xerox Palo Alto Research Center
Palo Alto, California

Abstract

It is well known that the convex hull of a set of n points in the (Euclidean) plane can be found by an algorithm having worst-case complexity $O(n \log n)$. In this note we give a short linear algorithm for finding the convex hull in the case that the (ordered) set of points from the vertices of a simple (i.e., non-self-intersecting) polygon.

* On leave from Bell Laboratories, Murray Hill, New Jersey.

** This research was supported in part by National Science Foundation grant MCS-77-23738 and by Office of Naval Research contract N00014-81-K-0269. Reproduction in whole or in part is permitted for any purpose of the United States government.

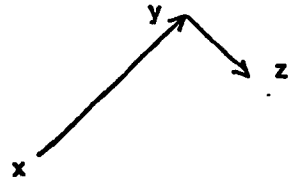
1. Introduction

The problem of finding the *convex hull* of a planar set of points P, that is, finding the smallest convex region enclosing P, arises frequently in computer graphics. For example, to fit P into a square or a circle, it is necessary and sufficient that H(P), the convex hull of P, fits; and since it is usually the case that H(P) has many fewer points than P has, it is a simpler object to manipulate. It is also the case that many fast graphics algorithms on polygons require that the input polygon be convex, thus making it a useful preprocessing step sometimes to first transform a general polygon into its convex hull. A number of algorithms exist for finding the convex hull of a set of points (e.g. [1][2][6]), with worst case complexity O(n log n) for $|P| = n$. It is also known that $\Omega(n \log n)$ is a lower bound just for determining H(P) -- that is, not necessarily rendering H(P) in, say, clockwise order [7]. This lower bound is proved for a decision tree model with quadratic tests, which accommodates all the known algorithms.

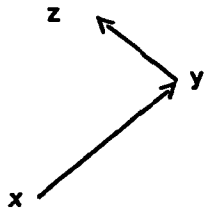
An interesting case of the convex hull problem that occurs frequently in practice is when the points of P form the vertices of a *simple polygon* (i.e., a polygon without self-intersections). Several authors have tried to find a fast algorithm for this problem. Sklansky [5] proposed an O(n) algorithm, which Bykat [1] later showed does not always work. It has also been noted that the algorithm that Shamos [4] suggested can sometimes fail. McCallum and Avis [3] recently published an O(n) algorithm which, being quite complicated and utilizing two stacks, entails rather intricate case analysis for the proof of its validity. In this note we provide a simple linear time algorithm for this problem and a short proof of its validity.

2. Preliminaries

Let P be a simple polygon in the plane with n vertices. We assume that each vertex v_i is represented by its X and Y coordinates. Without loss of generality, assume that $P = \langle v_1, v_2, \ldots, v_n \rangle$ is given by the sequence of its vertices as they are encountered in a clockwise traversal of the boundary. (The orientation of P can be easily tested and, if necessary, reversed.) The convex polygon, also clockwise oriented, whose vertices are the set of extreme points of P will be denoted by H(P). For a polygon P, we will use $P[v_i, v_j]$ to denote the path in P from v_i to v_j (following the orientation of P). The concatenation of two paths p and q will be written as $p \cdot q$. Given two points x and y, the notation L[x, y] will refer to the (directed) line segment from x to y. The primitive test used in our convex hull algorithm is a *right turn* test. A *turn* is an ordered triple of three points (x,y,z). We say that (x,y,z) is a *right, straight* or *left turn*, written $(x,y,z) = 1, 0$ or -1, if the Z component of the cross product $(z - y) \times (y - x)$ is positive, zero or negative, respectively. (See Figure 1). Alternatively, we say that z lies *to the right of, on* or *to the left of* L[x,y] if (x,y,z) is a right, straight or left turn, respectively. (One can test whether P is clockwise oriented by finding a vertex v_i with minimum X coordinate and checking that (v_{i-1}, v_i, v_{i+1}) is not a left turn.)



(x,y,z) is a right turn



(x,y,z) is a left turn

Figure 1

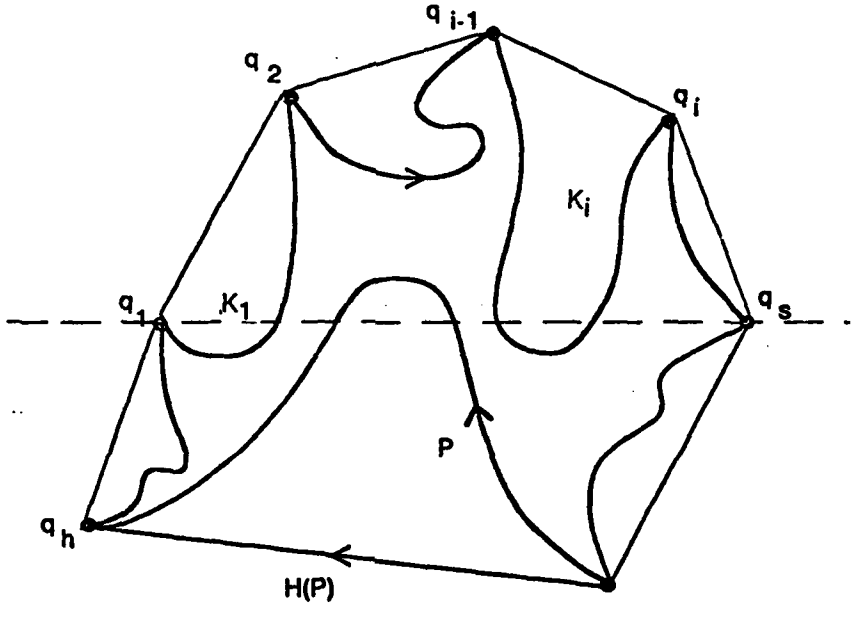


Figure 2. A simple polygon P and its convex hull H(P).

Let the vertices of the convex hull H(P) be $\langle q_1, ..., q_h \rangle$. It follows from the Jordan Curve Theorem that $\langle q_1, ..., q_h \rangle$ corresponds to a subsequence of the vertices in P, with the following characterization: (Figure 2)

(A1) $\langle q_1, ..., q_h \rangle$ forms a convex polygon (clockwise oriented).
(A2) No point of the path $P[q_{i-1}, q_i]$ lies to the left of $L[q_{i-1}, q_i]$ for $2 \leq i \leq h+1$. (Here q_{h+1} is identified with q_1).

We will call a polygon $L[q_{i-1}, q_i] \circ P[q_i, q_{i-1}]$ which satisfies (A2) a *pocket*, denoted by K_i, and edge $L[q_{i-1}, q_i]$ the *top* of the pocket. Notice that points of $P[q_i, q_{i-1}]$, other than its two end points q_i and q_{i-1}, may lie on $L[q_{i-1}, q_i]$. Thus, in general, each pocket K_i has an *interior* which is a disjoint union of a finite number of regions lying to the right of $L[q_{i-1}, q_i]$; the interior may also be empty in the degenerate case when $P[q_{i-1}, q_i] = L[q_{i-1}, q_i]$. The task of finding H(P) can then be specified as finding a sequence of pockets $R_1, R_2, ..., R_h$ so that their tops $L[q_1, q_2]$, $L[q_2, q_3]$, ... $L[q_h, q_1]$ form a convex polygon. Now, suppose we specify two vertices of H(P) -- say, q_1 and q_s. Then the chord $L[q_1, q_s]$ divides H(P) into two convex polygons with $L[q_1, q_s]$ as a common edge, while their other edges satisfy (A2). Indeed, (A1) and (A2) above imply that a characterization $H(P)[q_1, q_s] = \langle q_1, ..., q_s \rangle$ is as follows: (Figure 2)

(B1) $\langle q_1, ..., q_s \rangle$ forms a convex polygon (clockwise oriented).
(B2) $L[q_{i-1}, q_i] \circ P[q_i, q_{i-1}]$ forms a pocket for $2 \leq i \leq s$.

Note that $H(P)[q_1, q_s]$ depends only on $P[q_1, q_s]$. We will call $H(P)[q_1, q_s]$ the *left hull* of $P[q_1, q_s]$. In practice, a convenient choice for q_1 and q_s could be, say, two points of P with the minimum and the maximum X-coordinate, respectively. Without loss of generality, we now assume that in $P = \langle v_1, v_2, ..., v_m, v_{m+1}, ..., v_n \rangle$, vertices v_1 and v_m have the minimum and the maximum X-coordinate, respectively.

The problem of finding H(P) thus reduces to that of finding the left hulls of $P[v_1, v_m]$ and $P[v_m, v_1]$. We will describe an algorithm for the latter problem in the next section.

3. The Algorithm

Algorithm LeftHull is given by the diagram in Figure 3. The algorithm considers the points $\langle v_1, v_2, ..., v_m \rangle$ of $P[v_1, v_m]$ in order. The main data structure used is a stack $Q = \langle q_\$, q_1, ..., q_t \rangle$, with $q_\$$ being the element at the bottom, and the variable t pointing to the top of the stack. (The element $q_\$$ corresponds to q_s of the preceding discussion.) The main body of the algorithm is a loop, consisting of two boxes N and C.

Algorithm LeftHull

Input:

$P[v_1, v_m] = \langle v_1, ..., v_m \rangle$

Output:

$H(P)[v_1, v_m] = Q = \langle q_\$, q_1, ..., q_t \rangle$

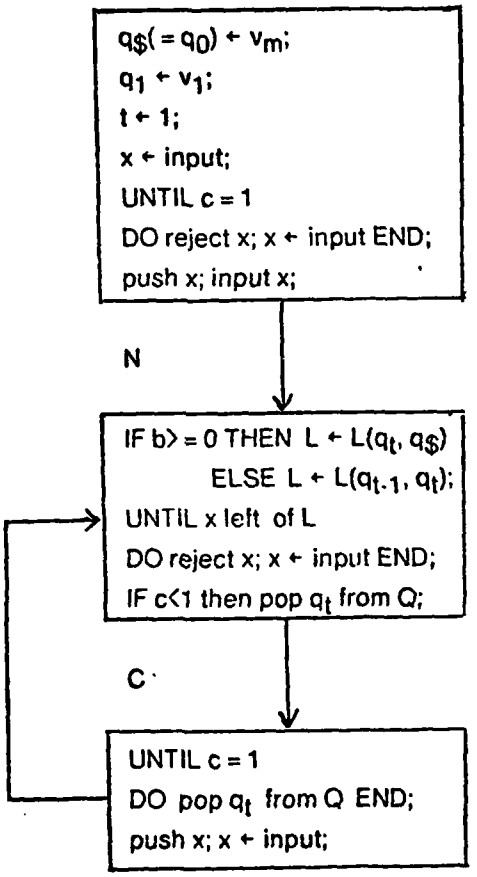


Figure 3. Algorithm LeftHull.

Each time we enter box N, the following induction hypotheses are true (see Figure 4).

(i) $Q = \langle q_\$, q_1, \ldots, q_t \rangle$ forms a convex polygon (clockwise oriented), with $q_\$ = v_m$, and $q_1 = v_1$.

(ii) $L[q_{i-1}, q_i] \circ P[q_i, q_{i-1}]$ forms a pocket for $2 \leq i \leq t$.

The only difference between these induction hypotheses and the characterization of $H(P)[q_1, q_\$]$ given by (B1) and (B2) is that, here, no stipulation is made about $L[q_t, q_\$]$ and $P[q_t, q_\$]$ in statement (ii). Of course, we must show that $L[q_t, q_\$] \circ P[q_\$, q_t]$ indeed forms a pocket when Algorithm LeftHull terminates. To explain the algorithm, we first define the notation used.

1. x refers to the input point v_i under consideration, and y refers to the input point v_j preceding the point q_t.
2. We use b to denote the turn (y, q_t, x); and c to denote the turn (q_{t-1}, q_t, x). For the current input x to be added to the stack, condition c = 1 is required by (i). The function of b will be explained later.
3. "Pushing x" means executing [$t \leftarrow t + 1$; $q_t \leftarrow x$; update y]; "popping x" means [$t \leftarrow t-1$]. No code is executed for "rejecting x".
4. By "x ← input" we mean [IF input is not exhausted THEN x ← nextPoint ELSE EXIT].

We will go through Algorithm LeftHull once, explaining the major steps and proving induction hypotheses (i) and (ii). In box I, we do the initialization and form K_1 by picking the first point after q_1 with c = 1 to be q_2. Thereafter, each iteration of boxes N and C creates a new pocket while maintaining overall convexity. Inductively, when we enter box N, pockets $K_1, \ldots, K_t$ satisfying (i) and (ii) have been found, with x being the first input point seen after q_t. We will consider two cases, corresponding to whether $b = (y, q_t, x) \geq 0$ or not.

Case (a). Suppose $b \geq 0$ (Figure 5a). The DO UNTIL loop in box N finds the first point x to the left of $L[q_t, q_\$]$. This defines a new pocket bounded by $L[q_t, x] \circ P[x, q_t]$.

Case (b). Suppose b = -1 (Figure 5b). Then x could be either in the pocket K_t, or outside of it. Note that in the latter case P must be left of $L[q_{t-1}, q_t]$. In the former case, P must also eventually leave K_t by moving left of $L[q_{t-1}, q_t]$ at some point x -- since this is the only way the simple polygon P could reach $q_\$$ without crossing $P[q_{t-1}, q_t]$. When such a point x is found, we will make a new pocket out of K_t by making $L[q_{t-1}, x]$ its top. This is accomplished by the last statement in box N which pops q_t from the stack. (In fact, this statement is included in box N purely for ease of discussion; it is clearly superfluous in view of the first statement of box C.)

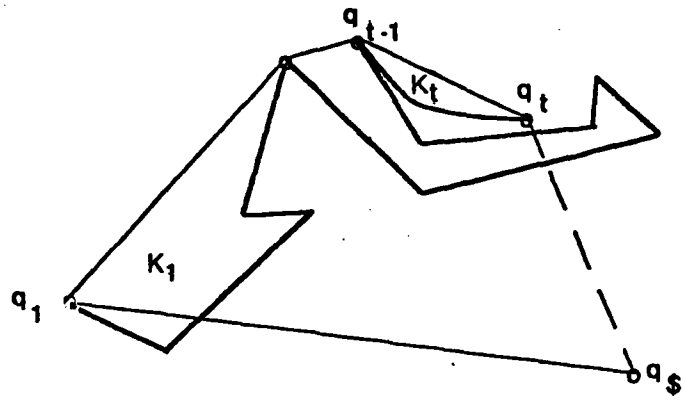
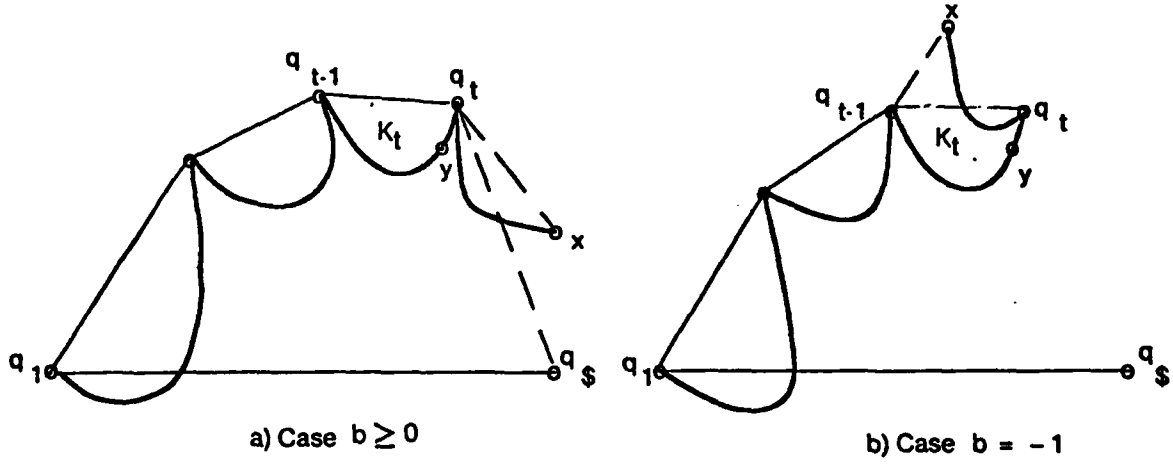


Figure 4. The induction hypotheses.



a) Case $b \geq 0$

b) Case $b = -1$

Figure 5. The result of executing box N.

Thus, in either case (a) or (b), after executing box N we will have t + 1 pockets lying between q_1 and x. They would satisfy the induction hypotheses if c = 1 were true. Box C is then executed to fulfill this convexity requirement, by merging the last pocket with as many pockets as necessary from the stack until the condition c = 1 is satisfied. (This will happen before q_1 is popped, since ($q_\$$, q_1, x) is a right turn as one can easily show.) At this time, x becomes the top element of Q, and the inductive step is complete.

The preceding discussion can easily be made rigorous. The correctness of the algorithm thus only depends on extending assertion (ii) to include the last segment $P[q_t, q_\$]$ when the algorithm terminates. If q_t itself is the last input, then (ii) is trivially true. Otherwise, the algorithm must terminate while executing the DO statement in box I or R. As we argued in the preceding paragraph, it is not possible that the algorithm terminates in box R after having entered it with b = 0. Thus it must terminate either in box I, or in box R after entering it with b = 1. In either case, the property "x lies to the right of $L[q_t, q_\$]$" will be true for all input points x seen after q_t. This completes the argument. The algorithm is obviously linear, since each input point can be pushed onto or popped from the stack at most once, if it is not rejected outright.

4. Conclusion

We present a simple linear time algorithm for finding the convex hull of a simple polygon. Note that our algorithm can actually be applied to non-simple polygons P as follows: First, at each point p of intersection of two (or more) edges of P, place a new vertex v(p). Viewed as a graph G(P), all vertices (old and new) have even degree, so that the graph G(P) is Eulerian. Choose a fixed Eulerian circuit in G(P). It is not difficult to see that each new vertex v(p) of degree 2d(p) may now be split into d(p) slightly perturbed vertices $v_i(P)$, $1 \leq i \leq d(P)$, of degree 2 so that each $v_i(P)$ is in the interior of H(P) and the resulting polygon P^* (= Eulerian circuit) is simple. Therefore $H(P^*) = H(P)$ and our algorithm can be used to compute $H(P^*)$. Observe, however, that if P has n vertices, P^* can have cn^2 vertices.

In the case that P is a simple path with n vertices, we can join the first and last points of P creating a simple new edge to form a (possibly non-simple) polygon $\overline{P}$. By applying the preceding transformation we can then find a simple polygon $\overline{P}^*$ on at most 3n vertices with $H(\overline{P}^*) = H(\overline{P}) = H(P)$ in time O(n).

References

[1] A. Bykat, Convex hull of a finite set of points in two dimensions, Information Processing Letters 7(1978) 296-298.

[2] R. L. Graham, An efficient algorithm for determining the convex hull of a planar set, Information Processing Letters 1(1972) 132-133.

[3] D. McCallum and D. Avis, A linear algorithm for finding the convex hull of a simple polygon, Information Processing Letters 9(1979) 201-206.

[4] M. Shamos, Problems in computational geometry, Doctoral Dissertation, Computer Science Department, Carnegie Mellon University (1975) revised (1977).

[5] J. Sklansky, Measuring concavity on a rectangular mosaic, IEEE Trans. Comput. 21(1972) 1355-1364.

[6] G. Toussaint, S. Akl and L. Devroye, Efficient convex hull algorithms for points in two and more dimensions, Technical Report No. 78.5, McGill University (1978).

[7] A. Yao, A lower bound for finding convex hulls, Technical Report, Computer Science Department (1979), to appear in J. of ACM.