Programming Techniques

R. Rivest Editor

An Optimal Real-Time Algorithm for Planar Convex Hulls

F.P. Preparata
University of Illinois

An algorithm is described for the construction in real-time of the convex hull of a set of n points in the plane. Using an appropriate data structure, the algorithm constructs the convex hull by successive updates, each taking time $O(\log n)$, thereby achieving a total processing time $O(n \log n)$.

Key Words and Phrases: computational geometry, convex hull, planar set of points, real-time algorithms, on-line algorithms

CR Categories: 4.49, 5.25, 5.32

This work was supported in part by the National Science Foundation under Grant MCS76-17321 and in part by the Joint Services Electronics Program under Contract DAAB-07-72-C-0259.

Author's address: F.P. Preparata, Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

1. Introduction

Algorithms for finding the convex hull of a finite set of n points in the plane have been developed by several authors in recent years [1, 2, 4, 5]. Most of these algorithms are also optimal; that is, as pointed out in [6], they have worst-case running time $O(n \log n)$, which is also the best achievable performance for obtaining the ordered hull.

A common feature of the above mentioned algorithms is that they are all *off-line*, i.e. they operate on the data collectively. In other words, information about all points of the set must be available before any of those algorithms can be applied.

Instead, it is desirable to develop an algorithm which receives one point at a time and updates the convex hull accordingly, so that after points $p_1, p_2, \dots, p_i$ have been received, their convex hull is available. Such an algorithm is appropriately called *on-line*. A general feature of on-line algorithms is that no bound is placed on the update time, or equivalently, a new item (point) is input on request as soon as the update relative to the last item has been completed. We shall refer to the time interval between two consecutive inputs as the *interarrival delay*.

Frequently, known on-line algorithms are less efficient on the entire set than the corresponding off-line algorithms (some price must generally be paid to acquire the on-line property). For the planar convex hull problem, however, Shamos has designed an elegant on-line algorithm [7], which runs in time $O(n \log n)$, thereby matching the performance of off-line algorithms for the same problem.

A more demanding case of on-line applications occurs when the interarrival delay is outside the control of the algorithm. In this case the update must be completed in time no greater than the minimum interarrival delay. Algorithms for such applications are appropriately called in *real-time*. Shamos [7] points out that since any ordered hull algorithm on n points requires Ω $(n \log n)$ time, any real-time algorithm for this problem must be allowed $O(\log n)$ processing time between successive inputs.

Unfortunately, the algorithm described by Shamos exceeds this allowance, since its interarrival delay can be $O((\log n)^2)$. The algorithm works as follows. When the point p_i is supplied, assume inductively that the algorithm has available the convex hull H_{i-1} of the set of points $\{p_i, \dots, p_{i-1}\}$, a point 0 internal to H_{i-1}, and the polar angles of the vertices of H_{i-1}—a convex polygon—about 0. The vertices of H_{i-1} are arranged in a height-balanced tree (e.g. an AVL tree), in the order of their polar angles. Thus point p_i can be located between two consecutive vertices of H_{i-1} in time at most $O(\log i)$ and then tested for inclusion in H_{i-1}. If it is internal, it is discarded; otherwise, two vertices ℓ and r of H_{i-1} have to be located so that the segments $p_i\ell$ and p_ir belong to the lines of support of H_{i-1}. The points ℓ and r can each be located by performing a standard binary search on the vertex cycle of H_{i-1}; on the other hand, each probe of this search is itself a search in the AVL tree, thereby resulting in a worst-case running time $O((\log i)^2) = O((\log n)^2)$ for an update.

The intuitive reason why the algorithm sketched above fails to achieve an $O(\log n)$ update time is that a binary search is artificially forced on a search tree, rather than allowing the latter to be the guide of the search operation. This natural observation is the basis of the following convex hull algorithm, which runs in time $O(n \log n)$ and is therefore optimal, and has update time $O(\log n)$ and is therefore in real-time.

2. The Real-Time Algorithm

Let P be a polygon in the plane and let $(v_0, \ldots, v_{s-1})$ be the counterclockwise cycle of its vertices (indices are modulo s). The vertices of P are stored as an ordered sequence in a data structure $T(P)$ which is a height-balanced tree modified in a trivial way to be described later (see Appendix). Let min $T(P)$ denote the vertex stored in the leftmost node of $T(P)$, arbitrarily chosen in the vertex cycle. The convex-hull algorithms will make use of two procedures: TANGENTS and RESTRUCTURE.

Procedure TANGENTS (P, m, p) accepts as its inputs a point p, a convex polygon P, represented by the tree $T(P)$, and the vertex $m = \min T(P)$; this algorithm tests whether p is internal or external to P, and, in the latter case, it determines two vertices ℓ and r, previously defined (see Figure 1). Notice that ℓ and r are named so that $\sphericalangle (rp\ell) < \pi$.[1] If p is internal, the algorithm terminates without altering $T(P)$; otherwise the string of vertices comprised between ℓ and r is deleted, and the vertex p is then inserted between ℓ and r. This operation is performed by the procedure RESTRUCTURE (P, p, ℓ, r).

Less informally, we have (Λ is the empty symbol):

CONVEX-HULL UPDATE

```
Input:   $T(H_{i-1}), p_i$
Output:  $T(H_i)$
1. begin  $m \leftarrow \min T(H_{i-1})$
2.        $(\ell, r) \leftarrow$ TANGENTS $(H_{i-1}, m, p_i)$
3.        If $(\ell, r) \neq (\Lambda, \Lambda)$ then $H_i \leftarrow$ RESTRUCTURE $(H_{i-1}, p_i, \ell, r)$
            else $H_i \leftarrow H_{i-1}$
          (Comment: $(\ell, r) = (\Lambda, \Lambda)$ means that $p_i$ is internal to $H_{i-1}$)
   end
```

Obviously, Step 1 runs in time at most $O(\log i)$ (search in height balanced tree with at most i elements). We shall now show that both TANGENTS and RESTRUCTURE run in time at most $O(\log i)$.

We begin by considering TANGENTS. Let $T = T(H_{i-1})$, $m = \min(T)$, $M = \mathrm{ROOT}(T)$, and let $L(M)$, $R(M)$ be respectively the left and right subtrees of the root of T: Note that m is the leftmost member and M is some intermediate member of the vertex sequence. Given a point p_i and a vertex v of H_{i-1}, we shall say that v is

[1] $\sphericalangle (abc)$ denotes the counterclockwise angle formed by segments $\overrightarrow{ba}$ and $\overrightarrow{bc}$.

Fig. 1. Definition of vertices ℓ and r.

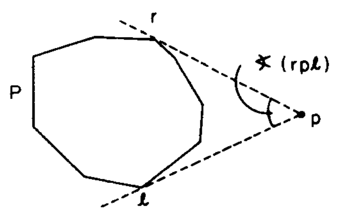
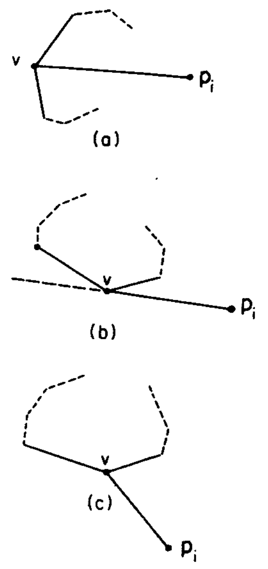


Fig. 2. Illustration of concave, supporting, and reflex vertices.

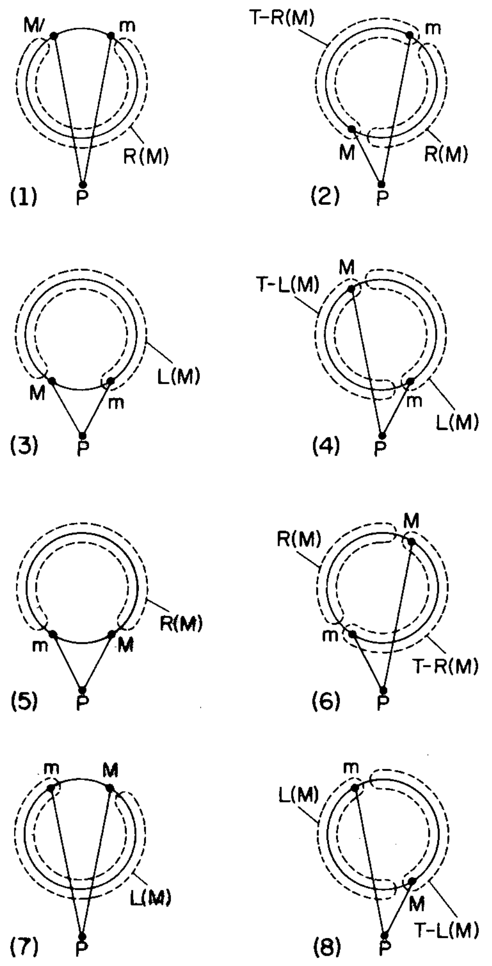


concave (with respect to p_i) if the segment p_iv intersects the interior of H_{i-1}; otherwise, if the two vertices adjacent to v lie on the same side of the line containing p_iv, v is *supporting*; in the remaining case, v is *reflex* (see Figure 2). We also denote as α the angle $\sphericalangle (mp_iM)$: obviously α is classified as *convex* ($\leq \pi$) or *reflex* ($> \pi$). Depending upon the classifications of the vertices m and M and of the angle α, we have a total of 18 possible cases: In fact, m and M can each be classified in three ways (concave, reflex, supporting) and α is either convex or reflex. However these cases can be conveniently reduced to eight new cases which cover all possibilities, as is summarized in Table I and illustrated in Figure 3. The diagrams of Figure 3 are to be read as follows: The "circle" on which "points" M and m lie stands for the polygon P; the sequence of vertices, as stored in $T(P)$, starts at m and runs counterclockwise on the circle; M is the vertex stored in the root of $T(P)$; here $L(M)$ and $R(M)$ refer respectively to the vertex sequences stored in the left and right subtrees of the root of $T(P)$. Each of these cases requires a distinct action. Specifically, in cases 1, 3, 5, and 7 (see Figure 3), the special vertices ℓ and r are to be found in a proper subsequence of the vertex sequence; this subsequence is either the sequence stored in $L(M)$ or that stored in $R(M)$. In cases 2, 4, 6, and 8 (Figure 3), there are two subsequences which respectively contain ℓ and r; as can be seen, each such subsequence is either the sequence stored in $L(M)$ or $R(M)$, or one of the latter extended with the vertex M.

Table I. Classification of α, m, and M for each of the cases of Figure 3.

Case	α	m	M
1	convex	concave	concave
2	convex	concave	nonconcave
3	convex	nonconcave	reflex
4	convex	nonconcave	nonreflex
5	reflex	reflex	reflex
6	reflex	reflex	nonreflex
7	reflex	nonreflex	concave
8	reflex	nonreflex	nonconcave

Fig. 3. The eight possible cases handled by algorithm TEST.



Thus the first task of the procedure is the classification of the situation according to Table I; subsequently, depending upon the outcome of this classification, the procedure either recursively calls itself on the shorter subsequence or it calls two other simple procedures, LEFTSEARCH and RIGHTSEARCH, which respectively determine ℓ and r. It is worth pointing out that when p_i is internal to H_{i-1}, cases 1 or 7 or Figure 3 will occur repeatedly; that is, the algorithm will examine a nested family of subtrees and will terminate when the subtree consists of only one leaf, i.e. when $m = M$.

The arguments of LEFTSEARCH and RIGHTSEARCH are search trees, although not necessarily AVL trees; in fact, the trees $T - R(M)$ and $T - L(M)$, whose root is M, are not AVL but have depth $O(\log i)$. Both procedures are quite straightforward: basically, each of them traces a path of T searching for a vertex v such that the line containing pv is supporting for the polygon. As an example, LEFTSEARCH is sketched below, while a detailed description of TANGENTS is given as an appendix. Below, $L(c)$ and $R(c)$ are respectively the left and right subtrees of a node c.

Procedure LEFTSEARCH

```
Input:   a tree $T$, describing a sequence of vertices.
Output:  a vertex $\ell$.
1.  begin  $c \leftarrow$ ROOT($T$)
2.          If $pc$ is supporting then $\ell \leftarrow c$
3.          else begin If $c$ is reflex then $T \leftarrow L(c)$ else $T \leftarrow R(c)$
4.                  $\ell \leftarrow$ LEFTSEARCH($T$)
                end
5.          return $\ell$
    end
```

It is obvious that LEFTSEARCH involves tracing a path of the tree T, spending a bounded time at each node. Since T is a balanced tree with at most $(i - 1)$ nodes, the running time is $O(\log i)$. Referring now to the time performance of TANGENTS, we notice that the case classification task is completed in time bounded by a constant; thus the bulk of the work is done either on the recursive call or in the calls of the auxiliary procedures LEFTSEARCH and RIGHTSEARCH. Typically, algorithm TANGENTS could be viewed as tracing a path from the root to some node c of $T(H_{i-1})$, recursively calling itself. If p_i is internal to H_{i-1}, then c is a leaf of $T(H_{i-1})$; otherwise, starting at node c, two paths of $T(H_{i-1})$ are traced by LEFTSEARCH and RIGHTSEARCH, respectively, until ℓ and r are found. Since the amount of work expended at each node is bounded by a constant, TANGENTS runs in time proportional to the depth of $T(H_{i-1})$, i.e. $O(\log i)$).

Finally, we consider the procedure RESTRUCTURE, which is invoked only when p_i is external to H_{i-1}. Let n_{i-1} be the number of vertices H_{i-1}. As mentioned earlier, the vertices comprised between ℓ and r must be deleted and p_i inserted. With regard to the deletion, slightly different actions will be taken depending upon whether ℓ precedes r in $T(H_{i-1})$ or not. In the first case, we have to split twice and splice once AVL trees with at most $i - 1$ elements; in the second case, only two splittings occur. But split and splice of AVL trees are standard operations, known as Crane's algorithms [3, p. 465], which can be performed in time $O(\log i)$ and will not be further discussed. Similarly, insertion of p_i can be done in time $O(\log i)$.

Therefore, we conclude that the CONVEX-HULL UPDATE can be executed in time $O(\log i)$ after i points have been processed and can be used as an optimal real-time convex-hull algorithm.

Appendix

The procedure TANGENTS (P, m, p) accepts as its inputs a point p and a convex polygon P, given as a sequence of vertices beginning with m, also explicitly given. P is represented by means of an AVL tree $T(P)$ modified so that each vertex v_i stores a pointer NEXT$[v_i]$ to the address of its successor v_{i+1} on the boundary of P. This modification is prompted by the following considerations. The "case classification task" performed by TANGENTS uses the two vertices m and M; to ensure that this task be completed in constant time, both M (the root) and m (the first member) of $T(P)$ must be available. Thus when either the left or the right subtrees of $T(P)$ are chosen for a recursive call of TANGENTS, their first elements must be also supplied and this is expediently done by means of the pointer NEXT. (An update of NEXT occurs only when a vertex p_i is inserted by RESTRUCTURE and it is easily seen that it only involves two pointers, associated with ℓ and p_i respectively.) Below, u is a Boolean parameter which is set to 1 in cases 1, 3, 5, 7 (recursive calls of TANGENTS) and is set to 0 otherwise.

TANGENTS

Input: H, a polygon, given as a sequence of vertices stored in a modified AVL tree $T(H)$; p a point, m the first element in $T(H)$.

Output: Either a pair (ℓ, r) of integers or (Λ, Λ).

```
 1. begin $M \leftarrow$ ROOT($T(H)$), $T \leftarrow T(H)$
 2.         If $m = M$ then $r \leftarrow \ell \leftarrow \Lambda$ (Comment: $p_i$ is internal)
 3.         else begin If $\alpha \leq \pi$ then
 4.               If $m$ is concave then
 5.                  If $M$ is concave then $T \leftarrow R(M)$, $m \leftarrow$ NEXT($M$),
                        $u \leftarrow 1$ (case 1)
 6.                  else $T_1 \leftarrow T - R(M)$, $T_2 \leftarrow R(M)$, $u \leftarrow 0$ (case 2)
 7.               else If $M$ is reflex then $T \leftarrow L(M)$, $u \leftarrow 1$ (case 3)
 8.                    else $T_1 \leftarrow T - L(M)$, $T_2 \leftarrow L(M)$, $u \leftarrow 0$ (case 4)
 9.            else If $m$ is reflex then
10.                    If $M$ is reflex then $T \leftarrow R(M)$, $m \leftarrow$ NEXT($M$),
                        $u \leftarrow 1$ (case 5)
11.                    else $T_1 \leftarrow R(M)$, $T_2 \leftarrow T - R(M)$, $u \leftarrow 0$ (case 6)
12.                 else If $M$ is concave then $T \leftarrow L(M)$, $u \leftarrow 1$ (case
                        7)
13.                    else $T_1 \leftarrow L(M)$, $T_2 \leftarrow T - L(M)$, $u \leftarrow 0$ (case 8)
14.            If u = 1 then $(\ell, r) \leftarrow$ TANGENTS($T, m, p_i$)
15.            else $\ell \leftarrow$ LEFTSEARCH($T_1$), $r \leftarrow$ RIGHTSEARCH($T_2$)
            end
16.      return $(\ell, r)$
   end
```

Received October 1977

References

1. Graham, R.L. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Processing Letters 1* (1972), 132–133.
2. Jarvis, R.A. On the identification of the convex hull of a finite set of points in the plane. *Inform. Processing Letters 2* (1973), 18–21.
3. Knuth, D.E. *The Art of Computer Programming. Vol. 3: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.
4. Preparata, F.P., and Hong, S.J. Convex hulls of finite sets in two and three dimensions. *Comm. ACM 20*, 2 (Feb. 1977), 87–93.
5. Shamos, M.I. Problems in computational geometry. Dept. of Comptr. Sci., Yale U., New Haven, Conn., May 1975.
6. Shamos, M.I. Geometric complexity. Proc. Seventh Annual ACM Symp. on Theory of Computing, May 1975, pp. 224-233.
7. Shamos, M.I. Computational geometry. Dept. Comptr. Sci., Yale U., New Haven, Conn., 1977 (to be published by Springer Verlag).