International Journal of Computer and Information Sciences, Vol. 12, No. 2, 1983

On Finding the Convex Hull of a Simple Polygon[1]

D. T. Lee[2]

Received January 10, 1983; revised February 26, 1983

In this paper we present a linear time algorithm for finding the convex hull of a simple polygon. Compared to the result of McCallum and Avis, our algorithm requires only one stack, instead of two, and runs more efficiently.

KEY WORDS: Convex hull; simple polygon; computational geometry; analysis of algorithms.

1. INTRODUCTION

One of the fundamental problems in computational geometry is that of finding the convex hull of a set of N points in k-dimensional space. Efficient and asymptotically optimal algorithms for the problem in two- and three-dimensional space are available.[1,2,4–6,11–14] In this note we shall consider a special case of the problem in two-dimensional space, i.e., the problem of finding the convex hull of a simple N edge polygon. The lower bound proof that determines the convex hull of N points requires $\Omega(N \log N)$ time[3,12,13] is no longer applicable in this case. In fact, a recent paper by McCallum and Avis[9] has shown that a linear time algorithm exists. Before this result was published, two algorithms proposed, respectively, by Sklansky[14] and Shamos,[13] claimed to run in linear time, have been recently proven incorrect. See Bykat[4] for a counterexample to Sklansky's algorithm. A slightly modified counterexample to Shamos' algorithm can also be constructed.[9] Sklansky proposed in a recent paper[15] an amendation to the algorithm of Ref. 14, but it was shown, unfortunately, to be incorrect by

[1] Supported in part by the National Science Foundation under Grants MCS 7916847 and MCS 8202359.

[2] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201.

0091-7036/83/0400-0087$03.00/0

Toussaint and El Gindy.[16] In this note we shall present and prove the validity of another linear time algorithm[3] for this problem. In comparing the result of Ref. 9 our algorithm requires only one stack, instead of two, and runs more efficiently. We maintain that the stack T given in Ref. 9 is not as essential as indicated by McCallum and Avis and can be eliminated entirely.

2. DEFINITIONS AND NOTATIONS

A polygon is a closed plane figure with straight-line edges. A polygon is said to be *simple* if no two nonadjacent edges of the polygon intersect. In what follows we shall denote a simple polygon P by a list of vertices, i.e., $P = v_0, v_1, \ldots, v_{N-1}$, such that (v_i, v_{i+1})[4] is an edge of P, for $i = 0, 1, \ldots, N-1$, and the interior of the polygon lies to the left as the polygon is traversed. Each vertex v_i is represented by its x- and y-coordinates. We assume that no interior angle of the polygon is equal to 180°. The convex hull of P, denoted HULL(P), is the smallest convex polygon that contains P in its interior and is also represented by a list of some subset of the vertices of P. The hull vertices, according to the Jordan curve theorem, occur in sorted order, i.e., if $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ are on the hull with $i_1 < i_2 < \cdots < i_k$, then $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ represents a convex polygon.

Given three vertices $v_i = \langle x_i, y_i \rangle$, $v_j = \langle x_j, y_j \rangle$, and $v_k = \langle x_k, y_k \rangle$, let $s = y_k(x_j - x_i) - x_k(y_j - y_i) + x_i y_j - y_i x_j$. We say that v_k is *strictly to the left* (or *strictly to the right*) of the directed line from v_i to v_j, denoted $\overrightarrow{v_i, v_j}$, if s is positive (or negative, respectively) and v_k is *to the left* (or *to the right*) of $\overrightarrow{v_i, v_j}$ if s is nonnegative (or nonpositive, respectively). We shall assume that the vertices of P, i.e., $v_0, v_1, \ldots, v_{N-1}$, are maintained as a doubly-linked list such that CCW(v_i) denotes the counterclockwise successor of v_i, i.e., v_{i+1}, and $CW(v_i)$ the clockwise successor, i.e., v_{i-1}, and that v_0 is the vertex with minimum y-coordinate. (If we have a tie, we take the one with the largest x-coordinate.) Using this notation, the polygon P can be represented as v_0, CCW(v_0), CCW(CCW(v_0)),... so that the interior is strictly to the left of the directed line $\overrightarrow{v_i, v_{i+1}}$, for $i = 0, 1, \ldots, N-1$. The convex hull of P, however, is represented by a circularly linked list such that PRED(v_i) denotes the predecessor of v_i and if $v_{i_1}, v_{i_2}, \ldots,$ and v_{i_k} are the hull vertices, then PRED(v_{i_j}) $= v_{i_{j-1}}$, for $j = 2, 3, \ldots, k$, and PRED(v_{i_1}) $= v_{i_k}$. If v_{i_j} and $v_{i_{j+1}}$ are not end vertices of an edge, the vertices v_{i_j}, CCW(v_{i_j}), CCW(CCW(v_{i_j})),..., and $v_{i_{j+1}}$, which define a closed region, are said to form a *lobe* with $(v_{i_j}, v_{i_{j+1}})$ being its *handle*. For convenience a lobe with (v_i, v_j) as a handle is denoted

[3] A similar algorithm was independently obtained by Graham and Yao.[7] But a counterexample to their algorithm was later provided by J. O'Rourke. O'Rourke and Toussaint[10] have given an easy fix on the algorithm.

[4] All index additions and subtractions are taken modulo N.

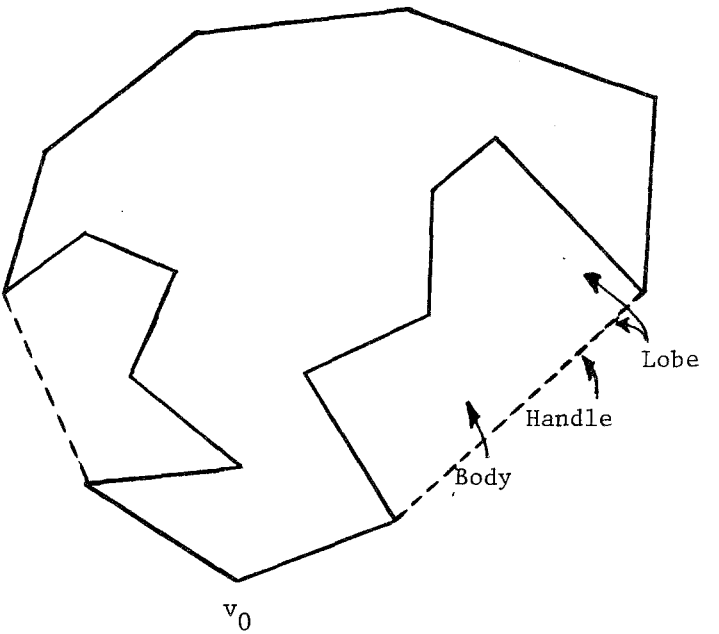

Fig. 1. Illustration of lobes, handles, and bodies.

as $L(v_i, v_j)$. The portion of a lobe $L(v_i, v_j)$ excluding the handle (v_i, v_j) is referred to as the *body* $B(v_i, v_j)$ of the lobe (Fig. 1). If v_{i_j} and $v_{i_{j+1}}$ are the two end vertices of an edge, then $(v_{i_j}, v_{i_{j+1}})$ is by itself a degenerate lobe with its body being empty.

3. THE ALGORITHM

We now give an informal description of the algorithm. As indicated earlier, we shall use an auxiliary stack that includes the vertices that are known to be on the convex hull among those vertices that we have examined so far, *less* those that belong to the body of some lobe and hence cannot be on the final convex hull. To be more precise, if the content of the stack, called *configuration* C of the stack, consists of vertices $v_{i_0}, v_{i_1},..., v_{i_k}$, $0 = i_0 < i_1 < \cdots < i_k$, then $C = v_{i_0}, v_{i_1},..., v_{i_k}$ is itself a convex polygon. When the last vertex v_{N-1} is determined to belong either to a configuration $C = v_{i_0},..., v_{i_k}$, $i_k = N - 1$ or to the lobe $L(v_{i_k}, v_{i_0})$, then $C = v_{i_0}, v_{i_1},..., v_{i_k}$ will be Hull(P).

Since the vertex v_0 is known to be a hull vertex, it is pushed into the stack initially. The next vertex $v_1 = \mathrm{CCW}(v_0)$ is also pushed into the stack

because vertices v_0 and v_1 so far satisfy the condition given earlier. In general, we have $C = v_{i_0}, v_{i_1}, \ldots, v_{i_k}$, $0 = i_0 < i_1 < \cdots < i_k$ and v_{i_k} is on top. Associated with each configuration there is a directed line $l(C)$ determined by the top two vertices, i.e., $l(C) = \overrightarrow{v_{i_{k-1}}, v_{i_k}}$. The directed line $l(C)$ is called the *current line* and the edge $(v_{i_{k-1}}, v_{i_k})$ the *current edge*.

Now suppose that the stack configuration is $C = v_{i_0}, v_{i_1}, \ldots, v_{i_k}$ and the next vertex to be examined, called the *active vertex*, is v_j. Then we have a convex polygon $v_{i_0}, v_{i_1}, \ldots, v_{i_k}$ as shown in Fig. 2. We shall distinguish two types of configurations. A *type 1* configuration C_1 is one where the current edge $(v_{i_{k-1}}, v_{i_k})$ define the handle of a nondegenerate lobe. The other type is called *type 2* and denote as C_2. In the following the symbol C without subscripts refers to either type of configurations. Given a pair $\langle C, v_j \rangle$, called a *state*, which consists of the current stack configuration and the active vertex, we have the following two cases to consider depending on which side of the current line $l(C)$ the active vertex v_j lies in order to obtain a new state, and repeat the entire process until v_{N-1} is visited.

(1) v_j is *not* strictly to the left of $l(C)$ (Fig. 3). In this case, the top vertex v_{i_k} and some vertices below it on the stack are no longer hull vertices.

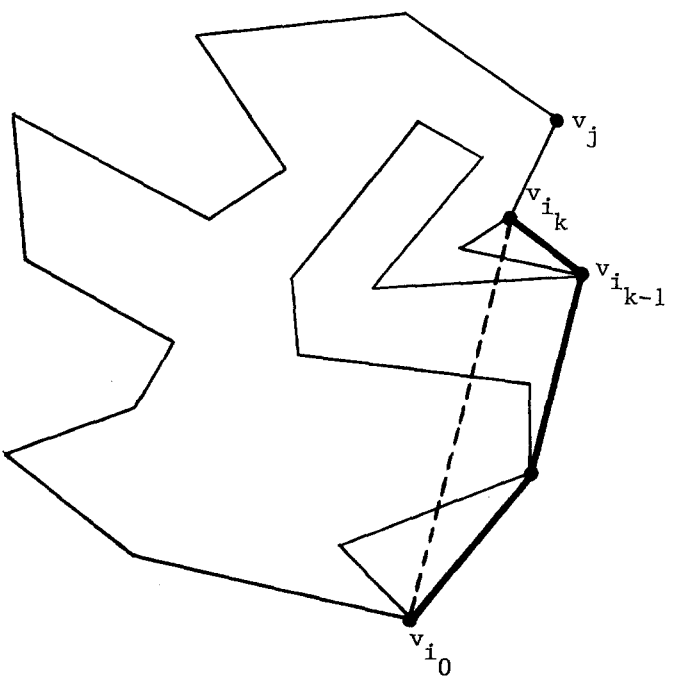

Fig. 2. Stack configuration and the active vertex.

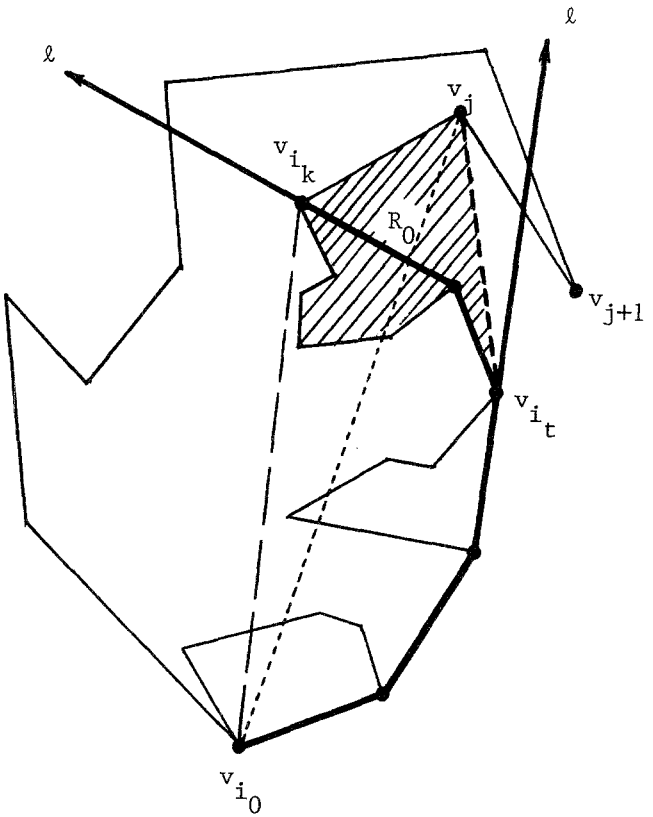

Fig. 3. The active vertex is on the right-hand side.

We therefore need to update the stack by deleting the vertices v_{i_k} and $v_{i_{k-1}}$,... until we either encounter v_0 or the vertex v_{i_t} such that v_j is strictly to the left of the directed line $\overrightarrow{v_{i_{t-1}}, v_{i_t}}$. After the update, we obtain a new configuration $C' = v_{i_0}, v_{i_1}, \ldots, v_{i_t}, v_j$ which is of type 1. Note that if v_j lies *on* $l(C)$, we still consider the resultant configuration as type 1 (with (v_{i_t}, v_j), $t = k-1$, as a handle) without affecting the algorithm. Thus, the next state we enter is $\langle C_1', v_{j+1} \rangle$.

(2) v_j is strictly to the left of $l(C)$ (Fig. 4). We further distinguish two subcases depending on the configuration type. (a) $C = C_1$. In this case we first check to see if v_i lies in the region R_0 defined by the lobe $L(v_{i_{k-1}}, v_{i_k})$ (Fig. 4a). If it does, vertices v_{i_k} and v_j are not hull vertices. In fact, the vertices of P that follow v_j and are in region R_0 are not hull vertices, and can be ignored. We can therefore search for the next vertex v_s of P such that v_s lies outside the region R_0. Now we have a pair $\langle C, v_s \rangle$ with v_s lying strictly

to the right of $l(C)$, a case similar to (1) and thus actions prescribed in (1) can be carried out. If v_j is not in R_0, we treat C as if it were of type 2 and go to case (b).

(b) $C = C_2$. In this case we have two subcases to consider. (b1) v_j is not strictly to the right of $\overrightarrow{v_{i_k}, v_{i_0}}$. That is, v_j lies inside the convex polygon represented by C and hence cannot be a hull vertex. In fact, all vertices of P that follow v_j and are inside the convex polygon (region R_1) can be ignored. So we search for the next vertex v_s of P such that v_s lies outside the region R_1. At that point we have a new state $\langle C, v_s \rangle$ and the entire process repeats. (b2) v_j is strictly to the right of $\overrightarrow{v_{i_k}, v_{i_0}}$. The new state we enter in this case is

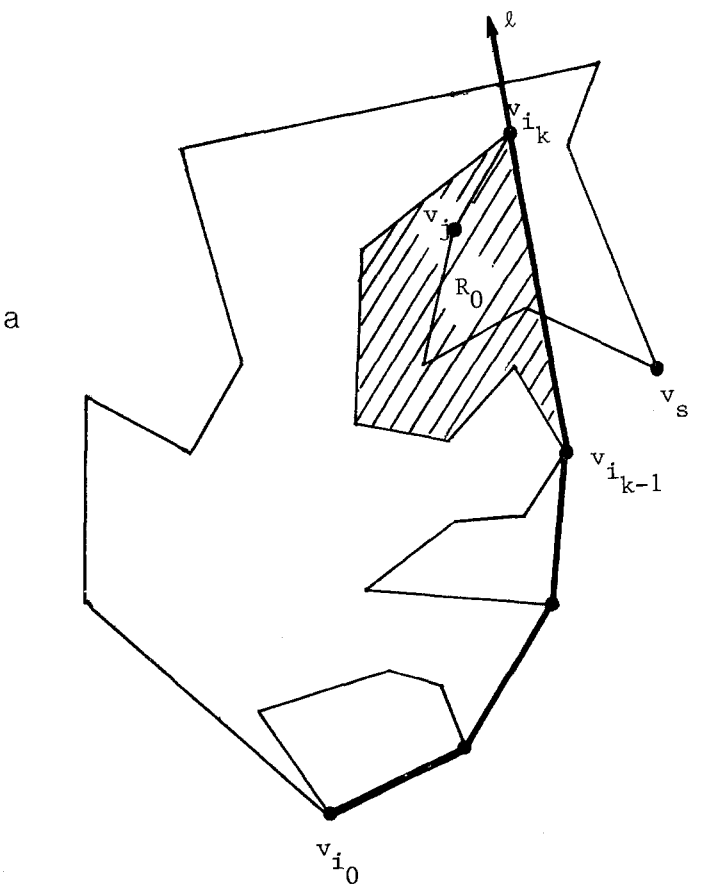

Fig. 4. Illustration of the active vertex on the left-hand side. (a) v_j in region R_0. (b) v_j in region R_1. (c) v_j in region R_2.

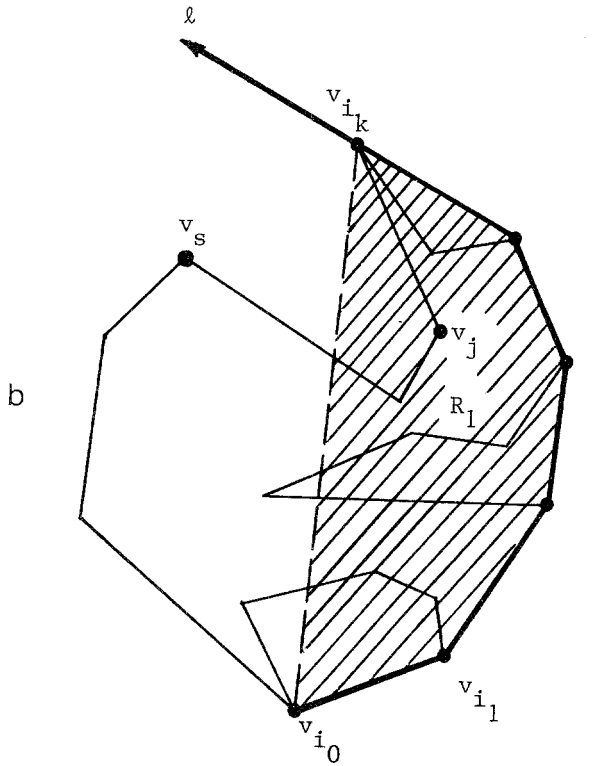

Fig. 4 (*continued*)

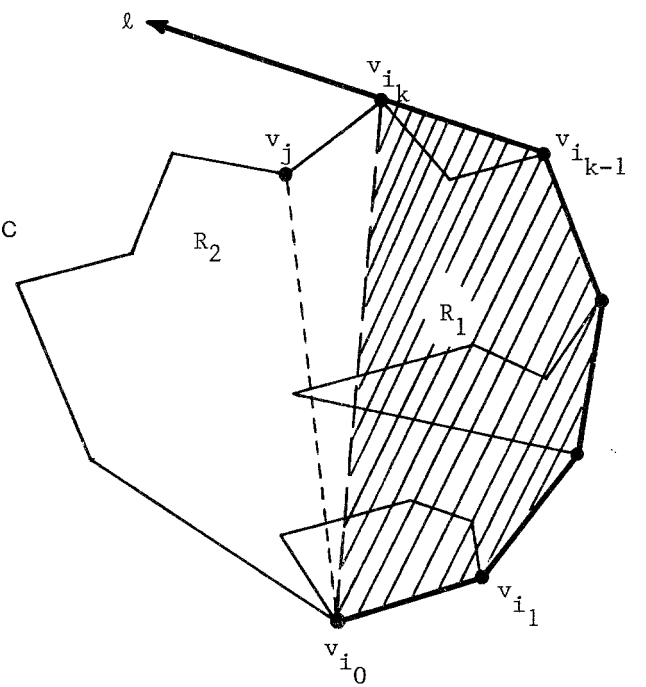

Fig. 4 (*continued*)

$\langle C', v_{j+1} \rangle$, where C' is obtained by pushing v_j into the stack. Note that the edge (v_j, v_{i_k}) may or may not be an edge of P. If it is not, as a result of case (b1), C' is of type 1, otherwise it is of type 2.

To summarize, each time after a lobe is "closed," i.e., a vertex is entered into the stack, the algorithm decides whether the *next* vertex is (1) inside the lobe, or (2) in the current hull represented by the stack, or is (3) a new hull vertex (in the latter case by strictly extending the hull sequence or by updating it). In cases (1) and (2) the boundary of the polygon must be traversed until a vertex emerges either from the lobe or from the current hull, respectively, at which point case (3) occurs.

From the above discussion the only difference in the actions taken between type 1 and type 2 configurations $C = v_{i_0}, v_{i_1}, \ldots, v_{i_{k-1}}, v_{i_k}$ is that of testing whether or not the active vertex is inside the lobe $L(v_{i_{k-1}}, v_{i_k})$. Therefore in the algorithm CH(P) below we perform the checking as soon as a new hull edge is created. The algorithm contains two subroutines UPDATE and CLOSELOBE. Subroutine UPDATE updates the stack as indicated in case (1). Subroutine CLOSELOBE is invoked whenever a new hull edge is created. It performs the checking prescribed in case (2a). Upon return of the subroutine call to CLOSELOBE we will enter a new state.

ALGORITHM CH(P)

Input: A simple polygon P, represented as $v_0, v_1, \ldots, v_{N-1}$, where v_0 is a vertex with the minimum y-coordinate and (v_i, v_{i+1}) is an edge of P. The vertices are arranged in counterclockwise direction such that the interior of the polygon lies to the left as the polygon is traversed. No interior angle is equal to 180°. CCW(v_i) denotes the counterclockwise successor of v_i and CW(v_i) the clockwise successor.

Output: The convex hull HULL(P) which is maintained as a circularly linked list, PRED(v_0), PRED(PRED(v_0)),...

Method: (*Comment*: The main program has two subroutines UPDATE and CLOSELOBE.)

SUBROUTINE UPDATE (u, v, l): (*Comment*: This routine has as input the top vertex u of the stack, the active vertex v, and the current line l. It updates the stack by deleting vertices from the stack that are not hull vertices due to the presence of v. Upon return, u contains the new top vertex of the stack and l the directed line determined by u and v.)

```
begin
  v' ← PRED(u)
  while v' ≠ v_0 and v is not strictly to the left of the directed line from PRED(v') to v'
      do v' ← PRED(v')
  u ← v'
  l ← the directed line from u to v
end UPDATE
```

SUBROUTINE CLOSELOBE (u, v, l): (*Comment*: This routine has as input the top vertex u of the stack, the active vertex v, and the directed line $l = \overrightarrow{u, v}$. It checks first to see if the next vertex v' lies in the lobe $L(u, v)$, i.e., region R_0. Upon return, v contains a new active vertex and l a new current line.)

```
begin v' ← CCW(v)
  if v' is not strictly to the right of l and v' is to the left of the directed line from v to CW(v)
  then (Comment: v' is in region R_0, case (2a)
    begin v' ← CCW(v')
      while v' is not strictly to the right of l do
          v' ← CCW(v')
      v ← v'
      l ← the directed line from PRED(u) to u
    end
  else (Comment: The edge (u, v) is either an edge of P or the next vertex v'
      to be examined is not in region R_0. We push v into the stack by
      linking v to u and enter a new state.)
    begin PRED(v) ← u
      u ← v
      v ← v'
    end
end CLOSELOBE
```

(*Comment*: Below is the main program.)

```
Begin
1. TOP ← CCW(v_0) (Comment: TOP is the top vertex of the stack).
2. NEXT ← CCW(TOP) (Comment: NEXT is the active vertex.)
3. PRED(TOP) ← v_0, PRED(v_0) ← v_0 (COMMENT: Initial configuration,
   C = v_0, TOP.)
4. l ← the directed line from v_0 to TOP (Comment: l is the current line.)
5. while NEXT ≠ v_0 do
   (Comment: At this point we have a new state ⟨C, NEXT⟩.)
6. if NEXT is not strictly to the left of l
   then (comment: case 1))
```

7. *begin* UPDATE (TOP, NEXT, l)
   CLOSELOBE (TOP, NEXT, l)
   (*Comment*: Upon exit, we have a new state $\langle C', \text{NEXT}\rangle$.)
   end
   else (*Comment*: case (2))
9. *if* NEXT is not strictly to the right of $\overrightarrow{\text{TOP}, v_0}$
   then (*Comment*: NEXT lies in region R_1, case (2b1.)
10. *begin* NEXT $\leftarrow$ CCW(NEXT)
11. *while* NEXT $\neq v_0$ *and* NEXT is not strictly to the right of $\overrightarrow{\text{TOP}, v_0}$
   do NEXT $\leftarrow$ CCW(NEXT)
   end
   else (*Comment*: NEXT is in region R_2, case (2b2.)
12. *begin* $l' \leftarrow \overrightarrow{\text{TOP, NEXT}}$
13. CLOSELOBE (TOP, NEXT, l')
14. $l \leftarrow l'$
   end
15. PRED(v_0) $\leftarrow$ TOP
   (*Comment*: This last step links the bottom and the top vertices of C.)
END CH.

4. ANALYSIS OF THE ALGORITHM

Step 1 through Step 4 are for the initialization phase in which the initial stack configuration $C_0 = v_0, v_1$ and the initial state $\langle C_0, v_2\rangle$ are obtained. In general, suppose that NEXT $= v_j$ and the current configuration $C = v_{i_0}, v_{i_1},..., v_{i_k}$. We shall prove by induction that each time Step 5 is executed, the pair $\langle C, v_j\rangle$ has the following properties, i.e., $0 = i_0 < i_1 < \cdots < i_k < j$ and for any previously examined vertex v_t, $t = 0, 1,..., j-1$, it satisfies one of the following three conditions, i.e., (1) v_t is in C, (2) v_t belongs to the body of some lobe, or (3) v_t is inside the convex polygon defined by C. (Note that the inclusion of condition (3) is for sake of the following inductive proof.) For the induction basis the initial state $\langle C_0, v_2\rangle$ obviously has the specified properties. Now let us assume that the claim is true for the state $\langle C, v_j\rangle$, where $C = v_{i_0}, v_{i_1},..., v_{i_k}$. Consider the following two cases.

(1) If v_j is not strictly to the left of $l(C)$, then subroutines UPDATE and CLOSELOBE will be executed (Steps 7 and 8). After UPDATE is executed, we will obtain a new stack configuration (Fig. 3) $C' = v_{i_0}, v_{i_1},..., v_{i_t}$ such that v_j is strictly to the left of $\overrightarrow{v_{i_{t-1}}, v_{i_t}}$. But after CLOSELOBE is executed, we will obtain a new state $\langle C', v_s\rangle$, where $s > j$ (Fig. 4a) or $\langle C'', v_{j+1}\rangle$, where $C'' = C', v_j$, i.e., v_j is pushd into the stack (Fig. 3). It is obvious that $\langle C'', v_{j+1}\rangle$ has the properties as claimed (Condition (1), v_j is in

C). When computing v_s, the vertices $v_{j+1}, v_{j+2}, \ldots, v_{s-1}$ are known to belong to the body of the lobe $L(v_{i_t}, v_j)$, satisfying condition (2). Also since C' is a subset of C, $i_0 < i_1 < \cdots < i_t < s$ holds. The claim follows.

(2) If v_j is strictly to the left of $l(C)$, then we check to see if v_j lies inside the convex polygon represented by C. If so, we will obtain a new active vertex v_s (Steps 10 and 11), where $s > j$ and the state $\langle C, v_s \rangle$. Obviously the claim is true for $\langle C, v_s \rangle$ since $v_j, v_{j+1}, \ldots, v_{s-1}$ satisfy condition (3). Otherwise, after Steps 12–14 are executed we will have either a state $\langle C, v_s \rangle$ for some v_s with $s > j$ or the state $\langle C', v_{j+1} \rangle$, where $C' = C, v_j$. By similar arguments given in (1), both $\langle C, v_s \rangle$ and $\langle C', v_{j+1} \rangle$ have the properties as claimed. Therefore we have the following theorem.

Theorem: The algorithm CH(P) for computing the convex hull of a simple polygon P with N vertices works correctly and takes $0(N)$ time.

Proof. The correctness of the algorithm follows from the previous arguments. All we need to show is that the algorithm takes linear time. Steps 7 and 8 take time proportional to the number of vertices deleted from the stack and the number of vertices examined, i.e., $s - j$. Steps 11 and 13 take time proportional to the number of vertices examined. All the other steps except Step 5 take constant time. Since each vertex is examined at most once and deleted from the stack or inserted into the stack at most once, Step 5 takes time proportional to the total number of vertices of the polygon, i.e., $0(N)$ time. Therefore, algorithm CH(P) takes $0(N)$ time.

5. CONCLUSION

We have presented an $0(N)$ time algorithm for finding the convex hull of a simple polygon with N vertices. This is an example which shows that some property (i.e., simplicity in this case) possessed by the input data indeed may help in terms of cutting down the complexity of the problem. As indicated earlier, the convex hull of an arbitrary N-gon requires $\Omega(N \log N)$ time. As another example, the closest pair problem for N points in the plane has been shown to require $\Omega(N \log N)$ time,[13] whereas the closest pair of N points which form a convex polygon can be found in $0(N)$ time.[8] We thus believe that there are problems in computational geometry, yet to be discovered, for which the geometric property of simplicity or any other property of the input data can be exploited.

ACKNOWLEDGMENT

The author wishes to thank Robin Nicholl for helpful discussions and for implementing the algorithm. Also thanks are due to G. Toussaint and H.

El Gindy for pointing out an error in the proof of correctness of the algorithm in an earlier version, and to F. P. Preparata for suggestions that help improve the presentation of the paper.

REFERENCES

1. S. G. Akl and G. T. Toussaint, "A fast convex hull algorithm," *Info. Proc. Lett.*, 7:219–222 (1978).
2. A. M. Andrew, "Another efficient algorithm for convex hull in 2-dimensions," *Info. Proc. Lett.*, **9**:216–219 (1979).
3. D. Avis, "On the complexity of finding the convex hull of a set of points," Tech. Rep., SOCS 79.2, School of Computer Science, McGill University, (1979).
4. A. Bykat, "Convex hull of a finite set of points in two dimensions," *Info. Proc. Lett.*, 7:296–298 (1978).
5. W. E. Eddy, "A new convew hull algorithm for planar sets," *ACM Trans. Math. Software*, 3:398–403 (1977).
6. R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," *Info. Proc. Lett.* **1**:132–133 (1972).
7. R. L. Graham and F. F. Yao, "Finding the convex hull of a simple polygon," *J. Algorithms* (to appear).
8. D. T. Lee and F. P. Preparata, "The all nearest neighbor problem for convex polygons," *Info. Proc. Lett.*, 7:189–192 (1978).
9. D. McCallum and D. Avis, "A linear time algorithm for finding the convex hull of a simple polygon," *Info. Proc. Lett.*, **9**:201–205 (1979).
10. J. O'Rourke and G. T. Toussaint, private communication.
11. F. P. Preparata, "An optimal real time algorithm for planar convex hulls," *Comm. ACM.* **22**:402–405 (1979).
12. F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," *Comm. ACM*, **20**:87–93 (1977).
13. M. I. Shamos, "Problems in computational geometry," Department of Computer Science, Yale University (1975 and 1977).
14. J. Sklansky, "Measuring concavity on a rectangular mosaic," *IEEE Trans. Comput.*, **C-21**:1355–1364 (1972).
15. J. Sklansky, "Finding the convex hull of a simple polygon," *Pattern Recognition Lett.*, **1**:79–83 (1982).
16. G. T. Toussaint and H. El Gindy, "An counterexample to an algorithm for computing monotone hulls of simple polygons," Tech. Rep. SOCS 83.1, School of Computer Science, McGill University (1983).