Pattern Recognition, Vol. 22, No. 5, pp. 561–565, 1989
Printed in Great Britain

0031-3203/89 $3.00 + .00
Pergamon Press plc

COMPUTING THE CONVEX HULL OF A SIMPLE POLYGON

CHERN-LIN CHEN
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.

(*Received* 15 *July* 1988; *received for publication* 28 *November* 1988)

Abstract—An algorithm for computing the convex hull of a simple polygon is presented. Its correctness and complexity are also shown.

Computational geometry Convex hull Simple polygon Algorithm

1. INTRODUCTION

The convex hull of a finite point set is the largest convex polygon generated by the set. The convex hull problem, particularly for a set of points in the plane, has been studied extensively and has applications in pattern recognition, image processing, computer graphics and operations research. There is a long list of articles[1–7] containing results on the convex hull of a planar point set. For n unstructured points the least available complexity of these algorithms is $O(n\log n)$.[8]

The complexity can be less than $O(n\log n)$ when the points are structured. An interesting case of the convex hull problem that occurs frequently in practice is when the points 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. In 1972 Sklansky[9] proposed a linear-time algorithm for computing the convex hull of a simple polygon. The algorithm is rather simple and elegant. Unfortunately, as has been pointed out by Bykat,[10] it does not always work. Another similar $O(n)$ algorithm proposed by Shamos[11] has also been discovered to fail for some types of simple polygons. A much more complicated linear-time algorithm that uses two stacks along with a proof of correctness was exhibited by McCallum and Avis.[12] Lee[13] later showed that only one stack is required. Graham and Yao[14] reported a compact algorithm that is similar in spirit to Lee's version. Both of Refs 13 and 14 included two types of pocket test. In the paper of Shin and Woo,[15] the Sklansky's original idea is adopted and only one pocket test is used.

The utter simplicity of Sklansky's algorithm compared to the later algorithms that work is a strong reason for investigations of Toussaint and Avis.[16] They have proved that Sklansky's algorithm can be used in the construction of the convex hull of simple polygons which possess a special property called weak external visibility. Orlowsky[17] enlarged the class of simple polygons on which Sklansky's algorithm can be applied. He introduced a notion called external left visibility and showed that this criterion is a necessary and sufficient condition for the success of Sklansky's algorithm. Sklansky,[18] in 1982 presented an amendation to his algorithm of 1972 for handling arbitrary simple polygons, not a subclass. However, counterexamples are given by Toussaint and Gindy.[19] Ghosh and Shyamasundar[20] propose a simple algorithm using only the Sklansky's convexity test and the comparison of the X coordinates of adjacent points. Though the statement of correctness is given, counterexamples similar to those in Ref. 19 can be found.

In this paper, a linear time algorithm for computing the convex hull of a simple polygon is described. With an augmented implicit pocket test for each candidate vertex of the input polygon, the convex hull can be found in a simple and elegant way. It is shown that the proposed algorithm is more compact than previous works.

2. NOTATION AND TERMINOLOGY

Let P be a simple polygon in the plane, which is represented by a sequence of vertices, i.e. $P = (v_0, v_1, \ldots, v_{n-1})$, such that two adjacent vertices v_i and v_{i+1} define an edge of the polygon. Each vertex v_i, $0 \le i \le n-1$, is represented by its X and Y coordinates. The convex hull of P, denoted by CH(P), is the smallest convex polygon that contains P in its interior. An extreme point of the polygon P is a vertex of P that lies on the boundary of CH(P). A simple polygon is clockwise oriented if its interior lies to the right as the polygon is traversed. The orientation of P can be easily tested and, if necessary, reversed in linear time. It is henceforth assumed in this paper that all polygons are clockwise oriented.

Let $L(v_i, v_j)$ denotes a directed line segment joining

two points v_i and v_j in the direction from v_i to v_j. A chain $\mathrm{CHN}(v_i, v_j)$ is a sequence of $L(v_i, v_{i+1})$, $L(v_{i+1}, v_{i+2}), \ldots, L(v_{j-1}, v_j)$. A closed region bounded by $L(v_i, v_j)$ and $\mathrm{CHN}(v_i, v_j)$ is called a pocket denoted by $\mathrm{PKT}(v_i, v_j)$ if all points in $\mathrm{CHN}(v_i, v_j)$ are on or to the right of $L(v_i, v_j)$.

Given three vertices $v_i = (x_i, y_i)$, $v_j = (x_j, y_j)$, and $v_k = (x_k, y_k)$, let $S(v_i, v_j, v_k) = (x_k - x_i)(y_j - y_i) - (y_k - y_i)(x_j - x_i)$. We have three cases, that is, $S(v_i, v_j, v_k)$.

(a) >0 implies v_k is strictly to the right of $L(v_i, v_j)$, that is, v_j is a convex vertex with respect to v_i and v_k.

(b) $=0$ implies three colinear points.

(c) <0 implies v_k is strictly to the left of $L(v_i, v_j)$, that is, v_j is a reflex vertex with respect to v_i and v_k.

The crux of the Sklansky's algorithm is a backtracking step which successively deletes non-convex vertices using the simple test stated above. However, as pointed out by Bykat,[10] a self-intersecting polygon may result during the deletion process. So the algorithm fails in some cases. In the following section, some observations are obtained by investigating the conditions which result in self-intersection. With these investigations we can avoid the intersection situation and then the convex hull can be found correctly.

3. PRELIMINARIES

Two interesting properties of a pocket due to Graham and Yao are cited first.

Lemma 3.1. No vertices of P that lies in $\mathrm{PKT}(v_i, v_j)$, with the possible exception of v_i and v_j, can be extreme.

Proof. Omitted.

Lemma 3.2. There is a successor of v_j in P that lies outside of $\mathrm{PKT}(v_i, v_j)$.

Proof. Omitted.

The successor of v_j lying outside of $\mathrm{PKT}(v_i, v_j)$ is called the emergence vertex of $\mathrm{PKT}(v_i, v_j)$.

Lemma 3.3. Reflex vertices of P lies in pockets.

Proof. If v_j is a reflex vertex of P, that is, $S(v_{j-1}, v_j, v_{j+1}) < 0$, then v_j is to the right of $L(v_{j-1}, v_{j+1})$ and on $\mathrm{CHN}(v_{j-1}, v_{j+1})$. So, v_j is in $\mathrm{PKT}(v_{j-1}, v_{j+1})$.

Now let us go for the reason why the Sklansky's algorithm fails. Figure 1 illustrates a case where that algorithm will not work. Processing from v_1 in a clockwise manner the algorithm discards vertex v_4 then backtracks to the triplet $v_2v_3v_5$ which is convex. Every subsequent vertex test is convex and thus the algorithm outputs the polygon $v_1v_2v_3v_5v_6v_7v_8$ which is not only non-convex but also non-simple. The mistaken step which leads to the undesirable result is that v_6 is accepted upon the convexity of the triplet $v_5v_6v_7$ being guaranteed. The vertex v_6 is in $\mathrm{PKT}(v_3, v_5)$. The erroneous acceptance of v_6 causes the intersection of $L(v_3, v_5)$ and $L(v_6, v_7)$.

By lemma 3.1, vertices in $\mathrm{PKT}(v_3, v_5)$ should be discarded. And by lemma 3.2, the emergence vertex exists for $\mathrm{PKT}(v_3, v_5)$. Thus, we have Fig. 2(a). A further convexity test deletes v_5 and the final output is shown in Fig. 2(b).

How to identify whether a vertex is in a pocket or not? The following lemmas give a simple way to compute it.

Lemma 3.4. The vertex v_{j+1} is in $\mathrm{PKT}(v_i, v_j)$ if $S(v_{j-1}, v_j, v_{j+1}) < 0$ and $S(v_i, v_j, v_{j+1}) > 0$.

Proof. The result is quite obvious as illustrated in Fig. 3. For v_{j+1} to be in $\mathrm{PKT}(v_i, v_j)$, it lies to the right of $L(v_i, v_j)$ and to the left of $\mathrm{CHN}(v_i, v_j)$.

Since v_{j+1} is in $\mathrm{PKT}(v_i, v_j)$, it should be discarded according to lemma 3.1. Then another vertex after v_{j+1} may become the immediate successor of v_j. This discarding process continues until the emergence vertex of $\mathrm{PKT}(v_i, v_j)$ appears. Points inside a pocket are deleted, hence self-intersection can be avoided. The generalization of lemma 3.4 is given as follows.

Lemma 3.5. For a vertex v_k which becomes an immediate successor of v_j, due to the discarding of its preceding vertices, is in $\mathrm{PKT}(v_i, v_j)$ if $S(v_{j-1}, v_j, v_{j+1}) < 0$ and $S(v_i, v_j, v_k) > 0$.

Proof. For $k = j + 1$, this is the case of lemma 3.4. For $k > j + 1$, v_k is the immediate successor of v_j means that $v_{j+1}, v_{j+2}, \ldots$, and v_{k-1} are deleted. They are all in $\mathrm{PKT}(v_i, v_j)$. Because v_{k-1} is in $\mathrm{PKT}(v_i, v_j)$ and $S(v_i, v_j, v_k) > 0$, v_k is to the right of $L(v_i, v_j)$, and must be left of $\mathrm{CHN}(v_i, v_j)$. Otherwise $L(v_{k-1}, v_k)$

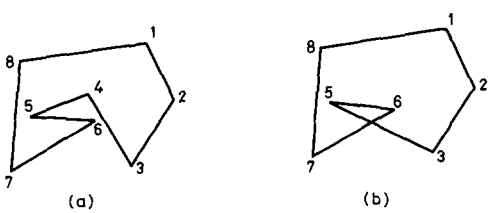

Fig. 1. A counterexample to Sklansky's algorithm.

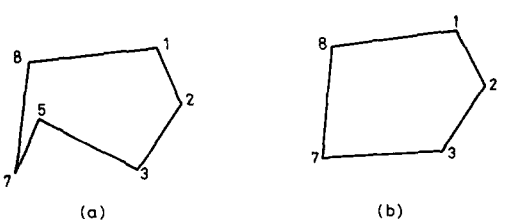

Fig. 2. A remedy to find the convex hull.

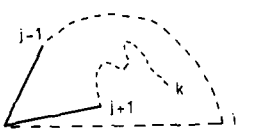

Fig. 3. Points inside $\mathrm{PKT}(v_i, v_j)$.

would intersect with CHN(v_i, v_j), which contradicts the definition of a simple polygon.

4. THE ALGORITHM

The proposed algorithm for finding the convex hull of a simple polygon is illustrated by the diagram in Fig. 4. The algorithm follows the original Sklanksy's algorithm with an implicit pocket test augmented. The main data structure used is an auxiliary stack $H = (h_j, h_{j-1}, \ldots, h_0)$, where h_0 denotes the bottom of the stack, and the variable j points to the stack top. The variable k refers to the current input vertex, and i refers to the vertex immediately preceding h_j. The input polygon is represented by a circular singly-linked list. The first vertex v_0 should be specified as one of the four extreme vertices: the leftmost, the rightmost, the top, and the bottom, to guarantee that v_0 is on CH(P). Other vertices of P then linked in clockwise order.

Box I initializes the stack by pushing the first two vertices into H. Box II tests the convexity of the vertex on the top of H with respect to its preceding vertex in H and the current input vertex. If non-convex, box III pops one vertex from H and then asks whether the elements in H are less than two. If the answer is 'NO', a backtracking step is taken. Due to lemma 3.4 and 3.5, box IV is the second convexity test before accepting a point and box V deletes all points inside. PKT(h_{j-1}, h_j). Box VI pushes the input vertex into H and gets another input. When the vertex v_1 is encountered again, this procedure stops. The contents of H, which are now $(h_0, h_j, \ldots, h_1, h_0)$ from top down, are the vertices of CH(P) in counter-clockwise order.

5. PROOF OF CORRECTNESS

The correctness of the proposed algorithm is proved by establishing three lemmas.

Lemma 5.1. Vertices removed do not belong to CH(P).

Proof. In the procedure (Fig. 4), vertices removed are due to being reflexive (box III) or being inside a pocket (box V). By lemma 3.3 a reflex vertex is in a pocket. Points in these two cases do not satisfy the definition of CH(P).

Let $H(P)$ denote the polygon represented by the elements in H after processing the input polygon P.

Lemma 5.2. $H(P)$ forms a convex polygon.

Proof. In the algorithm, box V deletes points inside PKT(h_{j-1}, h_j), thus CHN(h_0, h_j) will not stretch its next segment into PKT(h_{j-1}, h_j). Self-intersection is avoided, so $H(P)$ is a simple polygon. With the backtracking action (box III), reflexive vertices are popped from H. Points in H are currently convex vertices. Upon completion, each vertex h_i, $0 \le i \le j$, is convex with respect to h_{i-1} and h_{i+1}. $H(P)$ is a simple polygon and with convex vertices only, it is a convex polygon.

Lemma 5.3. $H(P) = \text{CH}(P)$.

Proof. By lemmas 5.1 and 5.2, vertices removed are not candidates for extreme points of P and the remaining points, $h_0, \ldots, h_j$, form a convex polygon. Thus $H(P)$ is the maximum convex polygon generated by the vertices of P, that is, the convex hull of P, CH(P).

The main theorem of this paper is the following.

Theorem 5.1. The proposed algorithm finds the convex hull of a simple polygon correctly and takes linear time.

Proof. The correctness is guaranteed by lemma 5.3. As the running time of this algorithm, it is seen that each input point can be pushed onto or popped from the stack at most once, if it is not rejected outright. Every popping or pushing consumes only constant time. Therefore, the algorithm takes time $O(n)$.

6. EXAMPLE

The simplicity and elegance of the proposed algorithm can be illustrated by an example. Consider the 17-vertex simple polygon shown in Fig. 5. Different

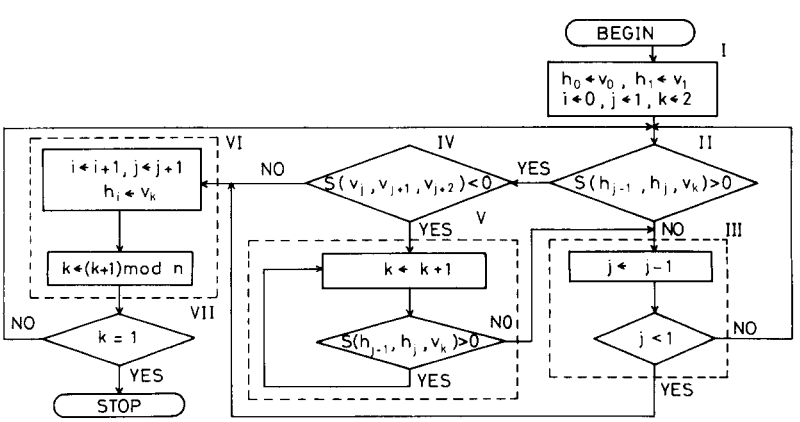

Fig. 4. Flowchart of the proposed algorithm.

Table 1. Stages of operations for the simple polygon in Fig. 5

Input	Stack	$S(h_{j-1}, h_j, v_k)$	$S(v_j, v_{j+1}, v_{j+2})$	Action
2	1, 0	−		Pop
3	2, 0	−		Pop
4	3, 0	+	+	Push
5	4, 3, 0	−		Pop
5	3, 0	+	+	Push
6	5, 3, 0	+	−	Reject
7	5, 3, 0	+		Reject
8	5, 3, 0	−		Pop
8	3, 0	+	+	Push
9	8, 3, 0	+	+	Push
10	9, 8, 3, 0	0		Pop
10	8, 3, 0	+	+	Push
11	10, 8, 3, 0	+	+	Push
12	11, 10, 8, 3, 0	+	+	Push
13	12, 11, 10, 8, 3, 0	+	+	Push
14	13, 12, 11, 10, 8, 3, 0	−		Pop
14	12, 11, 10, 8, 3, 0	+	+	Push
15	14, 12, 11, 10, 8, 3, 0	+	−	Reject
16	14, 12, 11, 10, 8, 3, 0	+		Reject
0	14, 12, 11, 10, 8, 3, 0	−		Pop
0	12, 11, 10, 8, 3, 0	−		Pop
0	11, 10, 8, 3, 0	0		Pop
0	10, 8, 3, 0	+	+	Push
1	0, 10, 8, 3, 0			Stop

stages of operation for computing the convex hull are given in Table 1. In the column of action, 'pop' means deleting the top element of the stack and 'push' means adding the input vertex to the stack and 'reject' means discarding the current input. The final content of the stack is the convex hull obtained in counterclockwise order.

7. CONCLUDING REMARKS

An algorithm has been presented for obtaining the convex hull of a simple polygon in linear time. Its correctness is shown and an example is given. In the algorithm of Fig. 4, omitting the box IV and V results in the original Sklansky's algorithm. The situation of three colinear points is treated in the proposed algorithm without any further testing. Due to its simplicity, it is the author's belief that the proposed algorithm is much easier to understand than previous works.

SUMMARY

Though linear-time algorithms for finding the convex hull of a simply-connected polygon have been reported, not all are concise and correct. In this paper, a new algorithm for computing the convex hull of a simple polygon on the plane, along with a proof of correctness, is presented. The central idea of this algorithm is the original concept of Sklansky's scan[9] augmented with an implicit pocket test. By exploiting some properties of pockets in simple polygons, the elegant structure of Sklansky's scan is preserved and its effectiveness is extended to general cases.

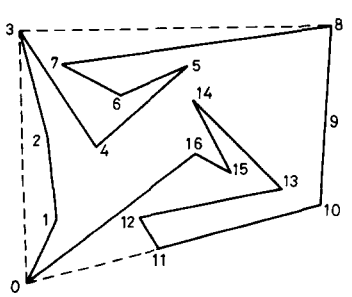

Fig. 5. A 17-vertex simple polygon and its convex hull.

REFERENCES

1. R. L. Graham, An efficient algorithm for determining the convex hull of a planar set, *Inf. Process. Lett.* **1**, 132–133 (1972).
2. R. A. Jarvis, On the identification of the convex hull of a finite set of points in the plane, *Inf. Process. Lett.* **2**, 18–21 (1973).
3. F. P. Preparata and S. J. Hong, Convex hulls of finite sets of points in two and three dimensions, *Communs ACM* **20**, 87–93 (1977).
4. S. G. Akl and G. T. Toussaint, A fast convex hull algorithm, *Inf. Process. Lett.* **7**, 219–222 (1978).
5. A. M. Andrew, Another efficient algorithm for convex hulls in two dimensions, *Inf. Process. Lett.* **9**, 216–219 (1979).
6. F. P. Preparata, An optimal real-time algorithm for planar convex hulls, *Communs ACM* **22**, 402–405 (1979).
7. D. G. Kirkpatrick and R. Seidel, The ultimate planar convex hull algorithm?, *SIAM J. Comput.* **15**, 287–299 (1986).
8. A. C. Yao, A lower bound to finding convex hulls, *J. ACM* **28**, 780–789 (1981).
9. J. Sklansky, Measuring concavity on a rectangular mosaic, *IEEE Trans. Comput.* **21**, 1355–1364 (1972).
10. A. Bykat, Convex hull of a finite set of points in two dimensions, *Inf. Process. Lett.* **7**, 296–298 (1978).
11. M. I. Shamos, Problems in computational geometry, Ph.D. diss., Yale Univ., New Haven, CT, 296–298 (1978).
12. D. McCallum and D. Avis, A linear algorithm for finding the convex hull of a simple polygon, *Inf. Process. Lett.*

9, 201–206 (1979).
13. D. T. Lee, On finding the convex hull of a simple polygon, *Int. J. Comput. Inf. Sci.* **12**, 87–98 (1983).
14. R. L. Graham and F. F. Yao, Finding the convex hull of a simple polygon, *J. Algorithms* **4**, 324–331 (1983).
15. S. Y. Shin and T. C. Woo, Finding the convex hull of a simple polygon in linear time, *Pattern Recognition* **6**, 453–458 (1986).
16. G. T. Toussaint and D. Avis, On a convex hull algorithm for polygons and its application to triangulation problems, *Pattern Recognition* **15**, 23–28 (1982).
17. M. Orlowski, On the conditions for success of Sklansky's convex hull algorithm, *Pattern Recognition* **16**, 579–586 (1983).
18. J. Sklansky, Finding the convex hull of a simple polygon, *Pattern Recognition Lett.* **1**, 79–83 (1982).
19. G. T. Toussaint and H. E. Gindy, A counterexample to an algorithm for computing monotone hulls of simple polygons, *Pattern Recognition Lett.* **1**, 219–222 (1983).
20. S. K. Ghosh and R. K. Shyamasundar, A linear time algorithm for obtaining the convex hull of a simple polygon, *Pattern Recognition* **16**, 587–592 (1983).

About the Author—CHERN-LIN CHEN received his B.S. and Ph.D. degrees in Electrical Engineering from the National Taiwan University in 1984 and 1987, respectively. In the summer of 1987 he joined the same school. He teaches courses in discrete mathematics and computer algorithms and his current research interests are directed at computational geometry and its applications to CAD and robotics.