Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A new algorithm for computing the convex hull of a planar point set*

LIU Guang-hui†, CHEN Chuan-bo

(*College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*)

†E-mail: aaalgh@263.net

Received Nov. 7, 2006; revision accepted May 14, 2007

Abstract: When the edges of a convex polygon are traversed along one direction, the interior of the convex polygon is always on the same side of the edges. Based on this characteristic of convex polygons, a new algorithm for computing the convex hull of a simple polygon is proposed in this paper, which is then extended to a new algorithm for computing the convex hull of a planar point set. First, the extreme points of the planar point set are found, and the subsets of point candidate for vertex of the convex hull between extreme points are obtained. Then, the ordered convex hull point sequences between extreme points are constructed separately and concatenated by removing redundant extreme points to get the convex hull. The time complexity of the new planar convex hull algorithm is $O(n\log h)$, which is equal to the time complexity of the best output-sensitive planar convex hull algorithms. Compared with the algorithm having the same complexity, the new algorithm is much faster.

Key words: Computational geometry, Convex hull, Extreme points, Ordered convex hull point sequence
doi:10.1631/jzus.2007.A1210 **Document code:** A **CLC number:** TP391

INTRODUCTION

Convex hull problem is one of the classical problems in computational geometry. The convex hull algorithm has wide practical use in areas such as computer graphics, image processing, design automation and pattern recognition (O′Rourke, 1998). Two-dimensional convex hull problems are divided into two categories, one is based on polygons, and the other on planar point sets. The convex hull of a simple polygon P is the smallest convex polygon containing P, whose vertices must be vertex of P. The convex hull of a planar point set S is the smallest convex polygon containing S, whose vertices must be point of S. The convex hull problem of a simple polygon is a special case of the convex hull problem of a planar point set.

Since Sklansky (1972) first proposed linearly convex hull algorithm of a simple polygon, many convex hull algorithms have been proposed (McCallum and Avis, 1979; Yao, 1981; Lee, 1983; Kirkpatrick and Seidel, 1986; Melkman, 1987; Bhattacharya and Sen, 1997; Levcopoulos *et al.*, 2002; Joswig and Ziegler, 2004). Yao (1981) showed that $\Omega(n\log n)$ is the lower bound of the convex hull problem for the worst-case input. Kirkpatrick and Seidel (1986) proved an $\Omega(n\log h)$ lower bound exists when both input and output sizes were considered, and proposed an $O(n\log h)$ optimal algorithm based on the prune-and-search technique. Although Sklansky was the first one to give the convex hull algorithm, unfortunately his algorithm has deficiencies. McCallum and Avis (1979) gave the first correct convex hull algorithm of a simple polygon. Melkman (1987) made a significant breakthrough by proposing an online convex hull algorithm of a 2D simple polyline which greatly simplifies the logic of the algorithm. The modified algorithm uses a double-ended queue to store an incremental hull for the vertices already processed. Convex hull algorithms are also investigated in China these years (Kong and Cai,

* Project (No. 2004AA420100) supported by the National Hi-Tech Research and Development Program (863) of China

1994; Cui *et al.*, 1997; Wu *et al.*, 1997; Jin *et al.*, 1998; Wang *et al.*, 1998; Liu, 2002).

Inspired by one of the characteristics of convex polygons, i.e., "edge same-side characteristic" (it would be explained later), we propose a new algorithm for computing the convex hull of a simple polygon, which is very simple and efficient. After that, we extend it to a new algorithm for computing the convex hull of a planar point set.

RELATED CONCEPTION

In this section, we give a description about related concepts on convex hull problem.

Definition 1 Let a point $p_i \in \mathbb{R}^2$ or $\mathbb{R}^3$ (i=1, 2, …, n), suppose the point set

$$S=\left\{s \mid s=\sum_{i=1}^{n}\lambda_i p_i, \sum_{i=1}^{n}\lambda_i \equiv 1, \lambda \geq 0, \lambda \in \mathbb{R}, i=1,2,...,n\right\}. \quad (1)$$

The smallest convex polygon containing S is called the convex hull of S.

Definition 2 Let $P_i(x_i, y_i)$ (i=1, 2, …, n) be n vertices of a polygon. If for any i, j, $i\neq j$, i, j=1,2, …, n, edge P_iP_{i+1} and edge P_jP_{j+1} are either neighbored with one shared end point or separated, then the polygon is called a simple polygon.

Definition 3 Let $A=(x_a, y_a)$, $B=(x_b, y_b)$ and $C=(x_c, y_c)$ be three different points in the XY plane, denote the directed straight line joining A and B (from A to B) by $L(AB)$, the orientation test function for discrimination of C with respect to $L(AB)$ is as follows:

$$S(A,B,C)=(x_b-x_a)(y_c-y_a)-(x_c-x_a)(y_b-y_a). \quad (2)$$

(1) If S>0, then C lies to the left of $L(AB)$;

(2) If S=0, then C lies on $L(AB)$;

(3) If S<0, then C lies to the right of $L(AB)$.

Definition 4 A point sequence composed of all the vertices of the convex hull of a simple polygon or a planar point set is called a convex hull point sequence. While the elements of the point sequence are arranged along one direction (clockwise or counterclockwise), it is called an ordered convex hull point sequence.

Definition 5 For a simple polygon P with a point sequence as $P_i(x_i, y_i)$ (i=1, 2, …, n), suppose x_{max}=max(x_i), x_{min}=min(x_i), y_{max}=max(y_i), y_{min}=min(y_i), and P_{LD}=(x_{min}, y_{min}), P_{LU}=(x_{min}, y_{max}), P_{RD}=(x_{max}, y_{min}), P_{RU}=(x_{max}, y_{max}), the former four points are called corner points of the simple polygon, the rectangle composed of the four corner points is called rectangular encasing box of the simple polygon. The eight extreme points of the simple polygon are defined as follows:

(a) Among points with x_i=x_{min}, denote the one with the maximum coordinate y by M_1;

(b) Among points with x_i=x_{min}, denote the one with the minimum coordinate y by M_8;

(c) Among points with x_i=x_{max}, denote the one with the maximum coordinate y by M_4;

(d) Among points with x_i=x_{max}, denote the one with the minimum coordinate y by M_5;

(e) Among points with y_i=y_{min}, denote the one with the maximum coordinate x by M_6;

(f) Among points with y_i=y_{min}, denote the one with the minimum coordinate x by M_7;

(g) Among points with y_i=y_{max}, denote the one with the maximum coordinate x by M_3;

(h) Among points with y_i=y_{max}, denote the one with the minimum coordinate x by M_2.

Definitions of the four corner points, rectangular encasing box and the eight extreme points of a simple polygon are also applicable to a planar point set. Line segments M_1M_2, M_3M_4, M_5M_6 and M_7M_8 divide the rectangular encasing box of a planar point set into five sub-regions, as shown in Fig.1. A point in one of the sub-regions from I to IV may be a vertex on the convex hull.

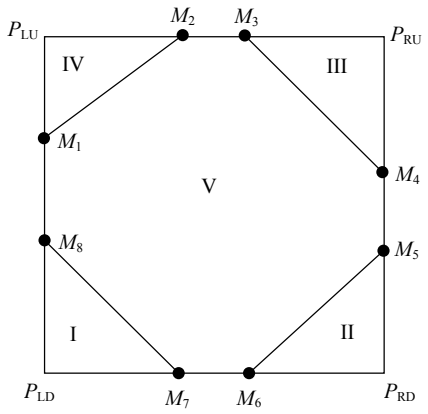

Fig.1 Sub-regions of the rectangular encasing box of a planar point set

NEW ALGORITHM FOR COMPUTING CONVEX HULL OF SIMPLE POLYGON

For convenience, we mention a polygon instead of a simple polygon. According to Definition 1 and Definition 5, it is obvious that M_i (i=1, 2, …, 8) of a polygon must be vertices of the convex hull of the polygon, and line segments M_8M_1, M_2M_3, M_4M_5 and M_6M_7 are edges of the convex hull of the polygon. Therefore, we could first compute the ordered convex hull point sequences between M_i and M_{i+1} (i=1, 3, 5, 7) separately, then we could concatenate them sequentially by removing redundant M_i to get the convex hull of the polygon. The convex hull problem is then transformed to the problem of computing the ordered convex hull point sequences between M_i and M_{i+1} (i=1, 3, 5, 7).

The integrated algorithm to compute the convex hull CH of a given polygon P is as follows:

Algorithm 1 *CHULL_SP*(CH, P)

Step 1: Find out extreme points M_i (i=1, 2, …, 8) of P, denote the point sequences between M_{2i-1} and M_{2i} by Q^i (i=1, 2, 3, 4), set i=1.

Step 2: Call *ORD_CHULL_SP*(M_{2i-1}, M_{2i}, Q^i, H^i) to compute the ordered convex hull point sequence H^i between M_i and M_{i+1}.

Step 3: If i<4, then i=i+1, goto Step 2; otherwise, continue.

Step 4: $CH=H^1 \cup H^2 \cup H^3 \cup H^4$.

ORD_CHULL_SP in Algorithm 1 is a procedure to compute the ordered convex hull point sequences between M_i and M_{i+1} (i=1, 3, 5, 7) of a simple polygon. For convenience of explanation, we give some denotations. For a point V, $V(x)$ and $V(y)$ denote its standard cartesian coordinates. For two different points A and B, denote the ray emanating from A to B by $R(AB)$, denote the directed straight line joining A and B by $L(AB)$. Denote the ordered convex hull point sequence between M_i and M_{i+1} (along clockwise direction) by $H=\{H_1, H_2, \ldots, H_r\}$ ($r\geq 2$), $H_1=M_i$, $H_r=M_{i+1}$. According to Definitions 4 and 5, it is known that only points which lie to the left side of the directed line joining M_i and M_{i+1} (i=1, 3, 5, 7) can possibly be points in the ordered convex hull point sequence of the polygon. So we just need to consider the convex vertices that lie to the left of the directed line joining M_i and M_{i+1} (i=1, 3, 5, 7), and we call them candidates.

As the topological structures of point sequences between M_i and M_{i+1} are the same, computation of the ordered convex hull point sequence between M_1 and M_2 is used as an illustration of the procedure. The ordered convex hull point sequences between other neighboring extreme points can be computed by analogy with it.

If M_1=M_2, then the result is simply M_1. So we would consider the general condition, i.e., M_1 and M_2 are two different points. Suppose the point sequence between M_1 and M_2 is Q. If Q is null, then $H=\{M_1, M_2\}$. If Q is not null, then for each point V in Q, first to check if it is a candidate: if it is, continue to judge if it could be a vertex on the convex hull and update H properly if necessary; otherwise, go ahead. Suppose V is the current candidate, if V is the first candidate in Q, just add it immediately after M_1 in H. A candidate that is not the first candidate is called a subsequent candidate. How to judge whether the subsequent candidates could be vertices on the convex hull or not and how to update H properly are the key issues of the algorithm.

We now give the important conceptions used in our algorithm. Assume $H=\{H_1, H_2, \ldots, H_r\}$ ($r\geq 2$) is current ordered convex hull point sequence between M_1 and M_2. The points in H have properties as follows: if $m<n$, then $H_m(x)<H_n(x)$, $H_m(y)<H_n(y)$. The directed polyline sequentially connecting the points in H and M_2 is called "Active-Polyline". The Active-Polyline divides the corresponding sub-region IV of the rectangular encasing box of P into two parts. The part that lies to the left of the Active-Polyline is called "Active-Region" (not including the borders). $R(H_jH_{j+1})$ (j=1, 2, …, r−1) divide the active region into many sub Active-Regions related to H_j (j=1, 2, …, r). The shadow region in Fig.2 is the sub Active-Region related to H_j, and the points in which all lie to the right of $R(H_{j-1}H_j)$ and on or to the left of $R(H_jH_{j+1})$. We call

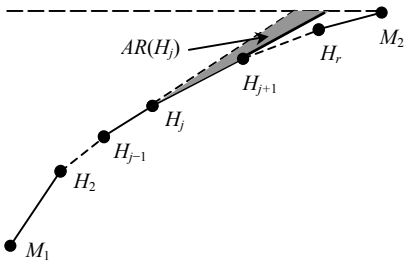

Fig.2 Sub Active-Region of H_j between M_1 and M_2

such sub Active-Region related to H_j "sub Active-Region of H_j", and denote it by $AR(H_j)$. $R(H_{j-1}H_j)$ and $R(H_jH_{j+1})$ are called "forward border" and "backward border" of $AR(H_j)$ respectively. Especially, the sub Active-Region lie on or to the left of $R(H_1H_2)$ is called "sub Active-Region of H_1", denoted by $AR(H_1)$; the sub Active-Region that lies to the right of $R(H_{r-1}H_r)$ and to the left of $R(H_rM_2)$ is called "sub Active-Region of H_r", denoted by $AR(H_r)$. According to the definition of $AR(H_j)$, given a point $V\in AR(H_i)$ ($l\le i\le u$) and H_j ($l\le j\le u$), we could have such conclusions: if V lies to the right of $R(H_{j-1}H_j)$ and to the left of or on $R(H_jH_{j+1})$, then $V\in AR(H_j)$; if $V(x)<H_j(x)$ or $V(x)\ge H_j(x)$ and V lies to the left of or on $R(H_{j-1}H_j)$, then $V\in AR(H_i)$ ($l\le i\le j-1$); otherwise, $V\in AR(H_i)$ ($j+1\le i\le u$).

The algorithm proposed in this paper is based on one of the characteristics of convex polygons, i.e., when the edges of a convex polygon are traversed along one direction, the interior of the convex polygon is always on the same side of the edges. We call this characteristic "edge same-side characteristic". In this paper, we assume that the edges of a convex polygon are traversed along clockwise direction, and the interior of the convex polygon is always on the right of these edges. If current candidate V lies in Active-Region, i.e., V lies to the left of the Active-Polyline, the edge same-side characteristic of the convex hull is not kept, which implies that V is one vertex on the convex hull, and that H needs to be updated to keep the edge same-side characteristic of the convex hull.

The algorithm to compute the ordered convex hull point sequence between M_1 and M_2 is given as follows:

Algorithm 2 *ORD_CHULL_SP*(M_1, M_2, Q, H)

Step 1: If $M_1=M_2$, then let $H=\{M_1\}$, end; otherwise, if $Q=\varnothing$, then let $H=\{M_1, M_2\}$, end; if $Q\ne\varnothing$, suppose $Q=\{Q_1, \ldots, Q_s\}$, $s\ge 1$, let $H=\{M_1\}$, $r=1$, $i=1$, $V=Q_1$, continue.

Step 2: If *IsCand*(V)=1, then call *DEAL_CAND_SP*(V); otherwise, continue.

Step 3: $i=i+1$, if $i\le s$, then let $V=Q_i$, go to Step 2; otherwise, continue.

Step 4: $r=r+1$, $H_r=M_2$.

IsCand(V) in Algorithm 2 is a function to judge whether V is a candidate or not. Suppose the direction to traverse a polygon is clockwise, V_1 and V_2 are neighboring vertices of V along the direction, according to the definition of a candidate: if $S(M_1, M_2, V)>0$ and $S(V_1, V, V_2)<0$, then *IsCand*(V)=1; otherwise, *IsCand*(V)=0. The procedure *DEAL_CAND_SP*(V, H) in Algorithm 2 is a procedure to deal with a candidate V (See Algorithm 3).

Algorithm 3 *DEAL_CAND_SP*(V, H)

Step 1: If $r=1$, then let $r=r+1$, $H_r=V$, quit; otherwise, continue.

Step 2: If *IN_AVR_SP*(V, H)=0, then keep H unchanged, quit; otherwise, let j=*IN_AVR_SP*(V, H), continue.

Step 3: If $V(x)\ge H_r(x)$, let $r=j+1$, $H_r=V$, quit; otherwise, continue.

Step 4: If $S(V, M_2, H_r)>0$, then let $W=H_r$, $r=j+1$, $H_r=V$, $r=r+1$, $H_r=W$, quit; otherwise, let $r=j+1$, $H_r=V$, quit.

IN_AVR_SP(V, H) in Step 2 of Algorithm 3 is a procedure to verify which sub Active-Region V lies in (See Algorithm 4). If V lies to the left of the Active-Polyline, then V lies in Active-Region. Let j=*IN_AVR_SP*(V, H): if $j=0$, then V does not lie in the Active-Region, thus there is no need to update H; otherwise, $AR(H_j)$ is the sub Active-Region where V lies in, and H needs to be updated.

Algorithm 4 *IN_AVR_SP*(V, H)

Step 1: If $V(x)=H_r(x)$, go to Step 2; If $H_r(x)<V(x)<M_2(x)$, go to Step 3; otherwise, go to Step 5.

Step 2: If $V(y)>H_r(y)$, go to Step 4; otherwise, return 0.

Step 3: If $S(H_{r-1}, H_r, V)\ge 0$, go to Step 4; if $S(H_{r-1}, H_r, V)<0$ and $S(H_r, M_2, V)>0$, return r; otherwise, return 0.

Step 4: Let $l=1$, $u=r-1$, j=*FIND_SAR*(V, l, u, H).

Step 5: If $S(H_{r-1}, H_r, V)>0$, go to Step 4; otherwise, return 0.

FIND_SAR(V, l, u, H) in Algorithm 4 is a function to find $AR(H_j)$ where V lies in and returns j, i.e., to find H_j between H_l and H_u, such that $S(H_{j-1}, H_j, V)<0$ and $S(H_j, H_{j+1}, V)\ge 0$ [Especially, if $S(H_1, H_2, V)\ge 0$, then $j=1$]. Note that if $V(x)<H_r(x)$ and V lies on or to the right of $L(H_{r-1}H_r)$, as Q is a partial point sequence of a simple polygon, it is not possible that V lies in Active-Region; otherwise, the line joining V and H_r would intersect with the polygon itself—this contradict with Definition 2, so return 0 under this condition in Step 5.

NEW ALGORITHM FOR COMPUTING CONVEX HULL OF PLANAR POINT SET

As a simple polygon is a special case of a planar point set, the algorithm in the former section could be extended to a new algorithm for computing the convex hull of a planar point set.

The main differences between a planar point set and a simple polygon are: (1) there is no convex vertex or concave vertex in a planar point set; (2) the subsets are not naturally separated by extreme points; (3) the points in a planar point set would not have all properties which are possessed by the vertices of a simple polygon. So the extension is not straightforward. The definition for a candidate is revised as a point lying to the left of $L(M_iM_{i+1})$ (i=1, 3, 5, 7). The subsets of candidates lying in sub-regions I to IV (excluding the borders) in Fig.1 are called candidate subsets between M_i and M_{i+1} (i=7, 5, 3, 1) respectively. Given a planar point set, we need first to find out the extreme points. For each remaining point V, if it satisfies one of the inequalities, i.e., $S(M_i, M_{i+1}, V)>0$ (i=1, 3, 5, 7), it is put into corresponding candidate subset between M_i and M_{i+1}.

The integrated algorithm to compute the convex hull CH of a planar point set PS is as follows:

Algorithm 5 *CHULL_PPS*(CH, PS)

Step 1: Find out extreme points M_i (i=1, 2, …, 8) of PS, denote the candidate subsets between M_{2i-1} and M_{2i} by Q^i (i=1, 2, 3, 4), set i=1.

Step 2: Call *ORD_CHULL_PPS*(M_{2i-1}, M_{2i}, Q^i, H^i) to compute the ordered convex hull point sequence H^i between M_i and M_{i+1}.

Step 3: If i<4, then i=i+1, goto Step 2; otherwise, continue.

Step 4: $CH=H^1 \cup H^2 \cup H^3 \cup H^4$.

ORD_CHULL_PPS in Algorithm 5 is a procedure to compute the ordered convex hull point sequences between M_i and M_{i+1} (i=1, 3, 5, 7) of a planar point set. Similarly, computing of the ordered convex hull point sequence between M_1 and M_2 is used as an illustration. Suppose the sub-candidate collection between M_1 and M_2 is Q. The algorithm to compute the ordered convex hull point sequence H between M_1 and M_2 is given as follows:

Algorithm 6 *ORD_CHULL_PPS*(M_1, M_2, Q, H)

Step 1: If M_1=M_2, then let H={M_1}, end; otherwise, if Q=∅, then let H={M_1, M_2}, end; if Q≠∅, suppose Q={Q_1, …, Q_s}, s≥1, let H={M_1}, r=1, i=1, V=Q_1, continue.

Step 2: Call *DEAL_CAND_PPS*(V, H), i=i+1.

Step 3: If i≤s, then let V=Q_i, go to Step 2; otherwise, continue.

Step 4: r=r+1, H_r=M_2.

DEAL_CAND_PPS(V, H) in Algorithm 6 is a procedure to deal with a point V in Q (see Algorithm 7). In order to explain this procedure, the concept of "the inverse Active-Region" needs to be introduced first. The definition of Active-Region is unchanged, and a sub Active-Region of H_j mentioned before is called "a sequential sub Active-Region of H_j" with the denotation still being $AR(H_j)$, and *FIND_SAR*(V, l, u, H), a function to find the sequential sub Active-Region where V lies in is unchanged. The shadow region in Fig.3 is an inverse sub Active-Region related to H_j (j=2, …, r−1), and the points in which lie to the right of $R(H_jH_{j-1})$ and lie on or to the left of $R(H_{j+1}H_j)$. We call such sub Active-Region related to H_j "inverse sub Active-Region of H_j", and denote it by $IAR(H_j)$. $R(H_{j+1}H_j)$ and $R(H_jH_{j-1})$ are called "forward border" and "backward border" of $IAR(H_j)$ respectively. According to the definition of $IAR(H_j)$, given a point $V \in IAR(H_i)$ ($l \le i \le u$) and H_j ($l \le j \le u$), we could conclude that if V lies to the right of $R(H_jH_{j-1})$ and to the left of or on $R(H_{j+1}H_j)$, then $V \in IAR(H_j)$; if $V(y) \ge H_j(y)$ or $V(y)<H_j(y)$ and V lies to the right of $R(H_{j+1}H_j)$, then $V \in IAR(H_i)$ (j+1≤i≤u); otherwise, $V \in IAR(H_i)$ ($l \le i \le j-1$). Here we adopt *FIND_ISAR*(V, l, u, t, H), a function to find the inverse sub Active-Region where V lies in, i.e., to find H_k between H_l and H_u, such that $S(H_k, H_{k-1}, V)<0$ and $S(H_{k+1}, H_k, V) \ge 0$, return k, $IAR(H_k)$ is the inverse sub Active-Region where V lies in. If $S(H_{k+1}, H_k, V)=0$, then V lies on the forward border of $IAR(H_k)$, let t=1; otherwise, let t=0.

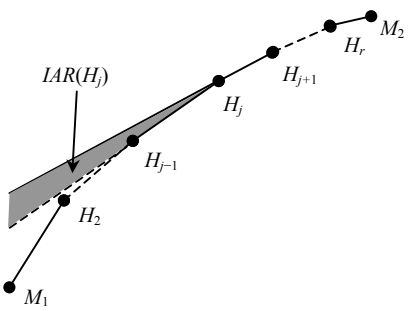

Fig.3 Inverse sub Active-Region of H_j between M_1 and M_2

Algorithm 7 *DEAL_CAND_PPS*(V, H)

Step 1: If r=1, then let r=r+1, H_r=V, quit; otherwise, call *IN_AVR_PPS*(V, m, n, t, H).

Step 2: If m=0, then keep H unchanged, quit; if n=0, go to Step 3; if n>0, go to Step 5.

Step 3: If $V(x)\geq H_r(x)$, then let r=m+1, H_r=V, quit; otherwise, continue.

Step 4: If $S(V, M_2, H_r)$>0, then let W=H_r, r=m+1, H_r=V, r=r+1, H_r=W, quit; otherwise, let r=m+1, H_r=V, quit.

Step 5: If t=0, let r=r−n+m+2, to delete points after H_m and before H_n in H, insert V immediately before H_n in H, quit; if t=1, let r=r−n+m+1, to delete points after H_m and before H_{n+1} in H, insert V immediately before H_{n+1} in H, quit.

IN_AVR_PPS(V, m, n, t, H) in Step 1 of Algorithm 7 is a procedure to verify which sub Active-Region V lies in (see Algorithm 8): if m=0, then V does not lie in Active-Region; if m>0, then $AR(H_m)$ is the sequential sub Active-Region where V lies in; if n>0, then $IAR(H_n)$ is the inverse sub Active-Region where V lies in; if t=1, then V lies on the forward border of $IAR(H_n)$; if t=0, then V does not lie on the forward border of $IAR(H_n)$.

For a point in a planar point set, there are two conditions to be discussed: (1) $V(x)\geq H_r(x)$ or $V(x)<H_r(x)$ and V lies to the left of $L(H_{r-1}H_r)$; (2) $V(x)<H_r(x)$ and V lies on or to the right of $L(H_{r-1}H_r)$. According to condition (1), if V lies in Active-Region, then we only need to find out the sequential sub Active-Region where V lies in, with the procedure just like Step 1 to Step 5 in *IN_AVR_SP*(V, H). According to condition (2), if V lies in Active-Region, then we need to find both sequential and inverse sub Active-Region where V lies in. In Step 5 of Algorithm 4, if $V(x)<H_r(x)$ and V lies on or to the right of $L(H_{r-1}H_r)$, owing to the definition of a simple polygon, the possibility of V being in Active-Region is excluded. However, it is not suitable for a planar point set. For a planar point set, we need first to find out H_j, such that $H_j(x)<V(x)\leq H_{j+1}(x)$; if V lies on or to the right of $L(H_{j-1}H_j)$, then V does not lie in Active-Region; otherwise, V lies in Active-Region. If we certify that V is in Active-Region, we need to further find both the sequential sub Active-Region and inverse sub Active-Region where V lies in, to update H properly.

Algorithm 8 *IN_AVR_PPS*(V, m, n, t, H)

Step 1: If $V(x)=H_r(x)$, let n=0, go to Step 2; if $H_r(x)<V(x)<M_2(x)$, let n=0, go to Step 3; otherwise, go to Step 5.

Step 2: If $V(y)>H_r(y)$, go to Step 4; otherwise, let m=0, quit.

Step 3: If $S(H_{r-1}, H_r, V)\geq 0$, go to Step 4; if $S(H_{r-1}, H_r, V)<0$ and $S(H_r, M_2, V)>0$, then let m=r, quit; otherwise, let m=0, quit.

Step 4: Let l=1, u=r−1, m=*FIND_SAR*(V, l, u, H), quit.

Step 5: If $S(H_{r-1}, H_r, V)>0$, let n=0, go to Step 4. otherwise, to find H_j such that $H_j(x)<V(x)\leq H_{j+1}(x)$: if $S(H_j, H_{j+1}, V)\leq 0$, then let m=0, n=0, quit; otherwise, call *FIND_AVR_PPS*(V, j, m, n, t, H).

FIND_AVR_PPS(V, j, m, n, t, H) in Algorithm 8 is a procedure to verify the sequential and inverse sub Active-Region where V lies in according to the location of V (See Algorithm 9). Assume that V lies in the sub Active-Region with shadow in Fig.4, which is divided into sub-regions I to IV (not including the segmental lines) by two straight lines $H_{j-1}H_j$ and $H_{j+1}H_{j+2}$. G is the common point of $H_{j-1}H_j$ and $H_{j+1}H_{j+2}$. Note that for j=r−2, only sub-regions I and III exist; for j=1, if V lies to the left of $R(H_3H_2)$, V lies in region I; if V lies to the right of $R(H_3H_2)$, V lies in region II.

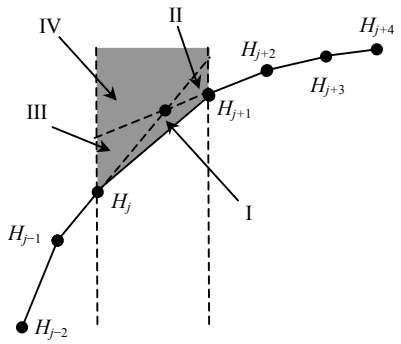

Fig.4 Four sub-regions divided by $H_{j-1}H_j$ and $H_{j+1}H_{j+2}$ in Active-Region of $H_j(x)<V(x)\leq H_{j+1}(x)$

Table 1 gives the values for m, n and t according to the location of V. For expressions like l=j+2, u=r−1: if they occur in the column of m, then m=*FIND_SAR*(V, l, u, H); if they occur in the column of n, then n=*FIND_ISAR*(V, l, u, t, H), and t is also computed.

Table 1 Values of *m*, *n*, *t*

Location of V	m	n	t
S_1=0, S_2=0	j−1	j+1	1
S_1=0, S_2<0	j−1	j+1	0
S_1=0, S_2>0	j−1	l=j+2, u=r−1	
S_1<0, S_2=0	j	j+1	1
S_1<0, S_2<0	j	j+1	0
S_1<0, S_2>0	j	l=j+2, u=r−1	
S_1>0, S_2=0	l=1, u=j−1	j+1	1
S_1>0, S_2<0	l=1, u=j−1	j+1	0
S_1>0, S_2>0	l=1, u=j−1	l=j+2, u=r−1	

S_1=S(H_{j-1}, H_j, V); S_2=S(H_{j+1}, H_{j+2}, V)

Algorithm 9 *FIND_AVR_PPS*(V, j, m, n, t, H)

Step 1: If j=1, then let m=1, go to Step 2; if j=r−2, then let n=r−1, if S(H_{r-1}, H_r, V)=0, let t=1, if S(H_{r-1},H_r, V)<0, let t=0, go to Step 3; otherwise, go to Step 4.

Step 2: If S(H_3, H_2, V)=0, then let n=2, t=1, quit; if S(H_3, H_2, V)>0, then let n=2, t=0, quit; otherwise, let l=3, u=r−1, n=*FIND_ISAR*(V, l, u, t, H), quit.

Step 3: If S(H_{r-3}, H_{r-2}, V)<0, then let m=r−2, quit; if S(H_{r-3}, H_{r-2}, V)=0, then let m=r−3, quit; otherwise, let l=1. u=r−3, m=*FIND_SAR*(V, l, u, H), quit.

Step 4: Get the values of m, n and t according to Table 1.

COMPUTING COMPLEXITY ANALYSIS

We analyze the time complexity of the algorithm for computing the convex hull of a planar point set. Suppose n is the number of points in a planar point set and h is the number of vertices on the convex hull of the planar point set. The total time complexity of Step 1, Step 3 and Step 4 in Algorithm 5 is $O(n)$. The time complexity of Step 2 in Algorithm 5 is mainly influenced by the update of H and the procedure of search for the sub Active-Regions where the candidates lie in. As a candidate can be added to the ordered convex hull point sequence at most once, so the deletion of it, the time complexity of update of H is $O(n)$. We now just analyze the time complexity of the search procedure for the sub Active-Region where one candidate lies in. The search steps in the search procedure are analyzed as follows: (1) Step 5 of Algorithm 8, to find H_j between H_l and H_u, such that $H_j(x)<V(x)\leq H_{j+1}(x)$, according to the properties that the points in the ordered convex hull point sequence have, a binary search method could be used; (2) The two functions *FIND_SAR*(V, l, u, H) and *FIND_ISAR*(V, l, u, t, H) in Algorithm 9 are remaining search steps, according to the conclusions drawn ahead, a binary search method could also be used. The time complexity of search steps is $O(\log h)$. The time complexity of the search procedure for the sub Active-Regions where the candidates lie in is $O(n\log h)$. So the time complexity of the algorithm for computing the convex hull of a planar point set proposed in this paper is $O(n\log h)$.

EXPERIMENTAL RESULTS

We have compared our strategy with the randomized hull algorithm proposed by Bhattacharya and Sen (1997). We ran experiments for different distributions which were adopted by Binay, and the averge is also computed on the basis of 15 trials. The experimented planar point sets have 50 000 points. The running CPU time (on IBM Thinkpad T40) statistics are shown in Table 2 (for upper hull). We mention our algorithm as ordered hull.

CONCLUSION

For the algorithms of planar convex hull problem, the central consideration is the location of a point in a planar subdivision. It is said in the bible of computational geometry (Preparata and Shamos, 1985) toward the problem: from the viewpoint of query time, the ability to use binary search is more important than the minimization of the size of the set to be searched. In other words, the fundamental idea is to create new geometric objects to permit binary

Table 2 Comparing the average performance of the ordered hull and randomized hull for various distributions

Strategy	Uniform in box (12)	Uniform in disc (33)	Uniform in annulus (45)	Uniform in circle (16 385)	Customized (9)
Randomized hull	5.939	6.269	5.238	1.816	28.546
Ordered hull	0.0582	0.0670	0.0700	0.5020	0.0500

The number in the parenthesis is the value of h(average)

search. The sequential sub Active-Regions and inverse sub Active-Regions are just such new geometric objects created by our algorithm. Comparison of our algorithm with those of a planar point set whose time complexity is also $O(n\log h)$ (Kirkpatrick and Seidel, 1986; Bhattacharya and Sen, 1997) shows that both need to separate the planar point set between extreme points into sub sets iteratively according to certain conditions, and that both need to compare the slopes of segments. While for the new algorithm proposed in this paper, there is no need to separate the sub candidate collections between extreme points, and only the orientation test function is used throughout the procedure. The new algorithm is simple and easy to implement. From the experimental results, it is obvious that our algorithm is much faster. Moreover, as the points in the ordered convex hull algorithm are computed sequentially in our algorithm, it is easier to implement the space-efficient planar convex hull algorithm (Brönnimann *et al.*, 2004; Brönnimann and Chan, 2006) based on the new planar convex hull algorithm than other algorithms with the same time complexity.

References

Bhattacharya, B.K., Sen, S., 1997. On a simple, practical, optimal, output-sensitive randomized planar convex hull algorithm. *Journal of Algorithms*, **25**(1):177-193. [doi:10.1006/jagm.1997.0869]

Brönnimann, H., Iacono, J., Katajainen, J., Morin, P., Morrison, J., Toussaint, G., 2004. Space-efficient planar convex hull algorithms. *Theor. Computer Sci.*, **321**(1):25-40. [doi:10.1016/j.tcs.2003.05.004]

Brönnimann, H., Chan, T.M., 2006. Space-efficient algorithms for computing the convex hull of a simple polygonal line in linear time. *Comput. Geom.*, **34**(2):75-82. [doi:10.1016/j.comgeo.2005.11.005]

Cui, G.H., Hong, F., Yu, X.X., 1997. A class of optimal algorithms for determine the convex hull of a set of nodes in a plane. *Chin. J. Computers*, **20**(4):330-334 (in Chinese).

Jin, W.H., He, T., Liu, X.P., Tang, W.Q., Tang, R.X., 1998. A fast convex hull algorithm of planar point set based on sorted simple polygon. *Chin. J. Computers*, **21**(6):533-539 (in Chinese).

Joswig, M., Ziegler, G.M., 2004. Convex hulls, oracles, and homology. *J. Symb. Comput.*, **38**:1247-1259. [doi:10.1016/j.jsc.2003.08.006]

Kirkpatrick, D.G., Seidel, R., 1986. The ultimate planar convex hull algorithm? *SIAM J. Computers*, **15**(1):287-299. [doi:10.1137/0215021]

Kong, X.S., Cai, H.X., 1994. An algorithm for finding the convex hull of a simple polygon using active double line testing. *Chin. J. Computers*, **17**(8):596-600 (in Chinese).

Lee, T.D., 1983. On finding the convex hull of a simple polygon. *Int. J. Comp. & Inf.*, **12**(2):87-98.

Levcopoulos, C., Lingas, A., Mitchell, J.S.B., 2002. Adaptive Algorithms for Constructing Convex Hulls and Triangulations of Polygonal Chains. 8th Scandinavian Workshop on Algorithm Theory. Turku, FL, p.80-89.

Liu, J.Y., 2002. Discussion on an $O(n)$ time algorithm for the convex hull of a planar point set. *Chin. J. Computers*, **25**(6):670-672 (in Chinese).

McCallum, D., Avis, D., 1979. A linear algorithm for finding the convex hull of a simple polygon. *Inf. Processing Lett.*, **9**:201-206. [doi:10.1016/0020-0190(79)90069-3]

Melkman, A.A., 1987. On-line construction of the convex hull of a simple polygon. *Inf. Processing Lett.*, **25**(1):11-12. [doi:10.1016/0020-0190(87)90086-X]

O′Rourke, J., 1998. Computational Geometry in C (2nd Ed.). Cambridge University Press, Cambridge.

Preparata, F.P., Shamos, M.I., 1985. Computational Geometry: An Introduction. Springer-Verlag, New York, p.45-46.

Sklansky, J., 1972. Measuring concavity on a rectangular mosaic. *IEEE Trans. on Comput.*, **C-21**(12):1355-1364. [doi:10.1109/T-C.1972.223507]

Wang, Z.Q., Hong, J.Z., Xiao, L.J., 1998. An optimal real time algorithm for determine the convex hull of a set of points in a plane. *Chin. J. Computers*, **21**(Suppl.):351-356 (in Chinese).

Wu, Z.H., Ye, C.Q., Pan, Y.H., 1997. An improved algorithm of convex hull computing. *J. Computer-Aided Design & Computer Graphics*, **9**(1):9-13 (in Chinese).

Yao, C.A., 1981. A lower bound to finding convex hulls. *J. ACM*, **28**(4):780-787. [doi:10.1145/322276.322289]