求解简单多边形和平面点集凸包的新算法*)

刘光惠　陈传波
(华中科技大学计算机学院　武汉 430074)

摘　要　沿一定方向遍历凸多边形的边，其内部在边的同一侧。本文依据凸多边形的这一特性，提出求解简单多边形凸包的新算法，进而扩展得到求解平面点集凸包的新算法。新点集凸包算法先找到点集的极值点，得到极值点间的候选点子集，再求得相邻极值点间的有序凸包点列，最后顺序连接极值点间的有序凸包点列，得到凸包。新算法达到目前平面点集凸包问题的渐进最好算法的时间复杂度 $O(n \log h)$，其中，n 为平面点集的点数，h 为平面点集凸包的顶点数。相比相同复杂度的凸包算法，新算法简单、易于实现。又由于是顺序求得凸包上的点，新算法还具有更易于实现基于其上的有效空间算法的优点。

关键词　计算几何，凸包，极值点，有序凸包点列，有效空间

New Algorithms for Computing the Convex Hulls of a Simple Polygon and a Planar Point Set

LIU Guang-Hui　CHEN Chuan-Bo
(Huazhong University of Science and Technology, Wuhan 430074)

Abstract　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 these edges, a new algorithm for computing the convex hull of a simple polygon is proposed first, which is then extended to a new algorithm for computing the convex hull of a planar point set. To compute the convex hull of a planar point set, first to find the extreme points of the planar point set and get the sub collections of points candidate for vertices of the convex hull between them. Then the sorted convex hull point arrays 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(nlogh), which is equal to the time complexity of best output-sensitive planar convex hull algorithms. Compared with the same time complexity algorithms, the new algorithm is simple and easy to be implemented, and owing to its sequentially computing the vertices on the convex hull, it is easier to get a space-efficient planar convex hull algorithm based on the new planar convex hull algorithm than others.

Keywords　Computational geometry, Convex hull, Extreme points, Sorted convex hull point array, Space-efficient

1　引言

凸包问题是计算几何的一个基本问题，凸包算法在计算机图形学、图像处理、设计自动化、模式识别等领域应用较广[1]。二维凸包问题可以分为基于多边形和基于平面点集两类凸包问题。简单多边形的凸包是指包含简单多边形的最小凸多边形，平面点集的凸包是指包含平面点集内所有点并且顶点属于平面点集的最小凸多边形。简单多边形凸包问题是平面点集凸包问题的一个特例。

自 1972 年 Sklansky[2] 最早提出具有线性时间复杂度的简单多边形凸包算法以来，涌现了很多凸包算法[3~8]。Yao[4] 于 1981 年证明了 $\Omega(n\log n)$ 是所有平面点集凸包算法的时间复杂度下限，Kirkpatrick 和 Seidel[6] 于 1986 年证明了在同时考虑输入和输出的情况下，平面点集凸包算法的时间复杂度下限是 $\Omega(n \log h)$，并提出了基于分枝定界法的算法复杂度为 $O(n \log h)$ 的平面点集凸包算法。尽管 Sklansky 最先提出简单多边形的凸包算法，遗憾的是该算法是有缺陷的。McCallum[3] 于 1979 年提出了第一个正确的简单多边形凸包算法，而 Melkman[7] 于 1987 年提出了一个实时的可以用于多边线的凸包算法。在国内，对于凸包算法的研究也很活跃[9~13]。

根据凸多边形的构成特性，本文先提出简单多边形凸包的新算法，进而扩展提出平面点集凸包的新算法。新的点集凸包算法的时间复杂度达到 $O(n \log h)$，其中，n 为平面点集的点数，h 为平面点集凸包的顶点数。

2　概念定义

本节给出算法中用到的基本概念。

定义 1　设 $p_i \in R^2$ 或 R^3，$(i=1, 2, \cdots, n)$，设点集

$$S=\{s \mid s=\sum_{i=1}^{n}\lambda_i p_i, \sum_{i=1}^{n}\lambda_i \equiv 1, \lambda \geqslant 0, \lambda \in R, i=1,2,\cdots,n\} \tag{1}$$

称包含 S 的最小凸多边形为 S 的凸包。

定义 2　设 $P_i(x_i, y_i)$，$i=1,2,\cdots,n$，是给定多边形的 n 个顶点，若对任意 $i,j, i\neq j, i,j=1,2,\cdots,n$，线段 P_iP_{i+1} 与 P_jP_{j+1} 或是相邻且相交于一端点或是不相交，则称该多边形

*)国家 863 项目(2004AA420100)。**刘光惠**　博士研究生，主要研究方向为计算机图形学、图像处理和算法优化；**陈传波**　博士生导师，主要研究方向为计算机图形学、图像处理、模式识别与计算机网络及应用技术。

为简单多边形。

定义 3 设 $A=(x_a,y_a)$、$B=(x_b,y_b)$和 $C=(x_c,y_c)$为 XY 平面内任意不同的三点，记 $L(AB)$为 A 指向 B 的有向直线，用矢量叉积的方法判断点 C 与 $L(AB)$的关系的侧向判别函数为

$$S(A,B,C)=(x_b-x_a)\times(y_c-y_b)-(x_c-x_b)\times(y_c-y_a) \tag{2}$$

(1) 如果 $S>0$，点 C 在 $L(AB)$的左侧；

(2) 如果 $S=0$，点 C 在 $L(AB)$上；

(3) 如果 $S<0$，点 C 在 $L(AB)$的右侧。

定义 4 不论是简单多边形还是平面点集，称其凸包上的所有顶点构成的点列为凸包点列。沿一定的方向(顺时针或逆时针)连接而成的凸包点列是有序凸包点列。

定义 5 设简单多边形的顶点列为 $P_i(x_i,y_i)$，$i=1,2,\cdots,n$。设 $x_{max}=\max(x_i)$，$x_{min}=\min(x_i)$，$y_{max}=\max(y_i)$，$y_{min}=\min(y_i)$，记 $P_{LD}=(x_{min},y_{min})$，$P_{LU}=(x_{min},y_{max})$，$P_{RD}=(x_{max},y_{min})$，$P_{RU}=(x_{max},x_{max})$为简单多边形的四个角点，称以 P_{LD}、P_{LU}、P_{RU}和 P_{RD}为顶点的矩形为简单多边形的矩形包围盒。定义简单多边形的 8 个极值点如下：

(a)M_1 为 x 坐标等于 x_{min}的点中 y 坐标最大的点；

M_8 为 x 坐标等于 x_{min}的点中 y 坐标最小的点；

(b)M_4 为 x 坐标等于 x_{max}的点中 y 坐标最大的点；

M_5 为 x 坐标等于 x_{max}的点中 y 坐标最小的点；

(c)M_6 为 y 坐标等于 y_{min}的点中 x 坐标最大的点；

M_7 为 y 坐标等于 y_{min}的点中 x 坐标最小的点；

(d)M_3 为 y 坐标等于 y_{max}的点中 x 坐标最大的点；

M_2 为 y 坐标等于 y_{max}的点中 x 坐标最小的点。

以上四个角点、矩形包围盒和 8 个极值点的定义同样适用于平面点集。线段 M_1M_2、M_3M_4、M_5M_6 及 M_7M_8 将平面点集的矩形包围盒划分成五个子区域，见图 1。落入子区域 I、II、III 和 IV 内的点可能成为凸包上的点。

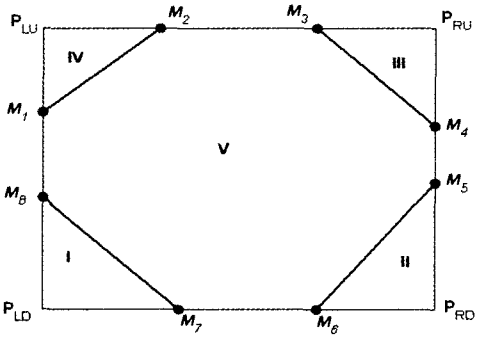

图 1　平面点集矩形包围盒的子区域划分

3　简单多边形凸包算法

为叙述方便，以下简称简单多边形为多边形。由点集凸包和极值点的定义知，M_i($i=1,2,\cdots,8$)必为多边形凸包的顶点，并且线段 M_8M_1，M_2M_3，M_4M_5 及 M_6M_7 为凸包边。因此，只需求解 M_i 与 M_{i+1}($i=1$，3，5，7)间的部分有序凸包点列即可。

首先，约定记法并给出相关定义。若 V 为点，记 $V(x)$、$V(y)$分别为点 V 的 x 坐标、y 坐标；若 A、B 为相异两点，记 $R(AB)$为由点 A 发出的指向点 B 的射线，记 $L(AB)$为经过点 A、B 的有向直线(A 指向 B)。设 M_i 与 M_{i+1}间的有序凸包点列(按顺时针方向连接)为 $H=\{H_1,H_2,\cdots,H_r\}$($r\geq2$)，$H_1=M_i$，$H_r=M_{i+1}$。由定义 4 和定义 5 知，只有在 M_iM_{i+1}连线(M_i 指向 M_{i+1}，$i=1$，3，5，7)左侧的点可能成为有序凸包点列中的点。又由于 M_i 与 M_{i+1}之间的点列取自多边形，故只需考虑点列中在 M_iM_{i+1}连线左侧的凸顶点，称满足以上条件的点为候选点。

由于 M_i 与 M_{i+1}($i=1$，3，5，7)之间的点列的拓扑结构是一样的，故以求解 M_1 到 M_2 之间的有序凸包点列 H 为例进行说明。对于其他相邻极值点间的有序凸包点列可以类似求解。设 M_1 与 M_2 之间的点列为 Q，若 Q 为空集，则 $H=\{M_1,M_2\}$。若 Q 非空，则令 $H=\{M_1\}$，$r=1$，再遍历 Q 中各点；若当前点为候选点，则进一步判断其能否成为凸包顶点。设 V 为当前候选点，若 V 是 Q 中的第一个候选点，则直接将 V 加到 M_1 之后。称 Q 中第一个候选点以外的候选点为后续候选点，则判断后续候选点是否为凸包顶点以及如何更新 H 是算法的关键。

下面，给出本文算法中用到的重要概念。设 $H=\{H_1,H_2,\cdots,H_r\}$($r\geq2$)为当前有序凸包点列，则 H 中的点有如下有序性质：若 $m<n$，则 $H_m(x)<H_n(x)$，$H_m(y)<H_n(y)$。称顺序连接 H 中的点及 M_2 的连线为活动线，它将多边形矩形包围盒中对应极值点间的子区域 IV 划分为左右两部分，称子区域 IV 中在活动线左侧的部分(不含边界)为活动区。$R(H_jH_{j+1})$($j=1$，2，$\cdots$，$r-1$)将活动区分割成许多与 H_j($j=1$，2，$\cdots$，r)对应的活动分区。图 2 中的阴影部分即是与 H_j 对应的活动分区，其中的点在 $R(H_{j-1}H_j)$的右侧，并在 $R(H_jH_{j+1})$的左侧或其上，称这样的活动分区为 H_j 的活动分区，记为 $AR(H_j)$。称 $R(H_{j-1}H_j)$、$R(H_jH_{j+1})$分别为 $AR(H_j)$的前、后边界。特别地，称在 $R(H_1H_2)$左侧或其上的活动分区为 H_1 的活动分区，记为 $AR(H_1)$；称在 $R(H_{r-1}H_r)$右侧并在 $R(H_rM_2)$左侧的活动分区为 H_r 的活动分区，记为 $AR(H_r)$。若 $AR(H_j)$内的点不在其边界上，则该点具有如下性质：在其他分区两边界的同一侧，同在左侧或同在右侧。划分活动分区的目的就是为了便于更新当前有序凸包点列 H。活动分区的定义及活动分区内点的性质为查找候选点所在的活动分区提供了依据。

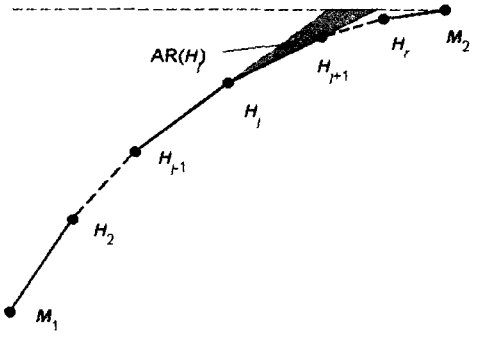

图 2　M_1 到 M_2 之间的活动分区示例

接下来，给出凸多边形的一个重要特性：沿一定方向(顺时针或逆时针)遍历凸多边形的边，凸多边形的内部位于遍历边的同一侧，我们称这一特性为凸多边形的边同侧特性。若顺时针遍历凸包的边，则凸包的内部在边的右侧。本文就是依据凸多边形的边同侧特性来判断后续候选点能否成为凸包顶点并更新 H 的。当后续候选点 V 位于活动区，即 V 位于活动线的左侧时，凸包的边同侧特性被破坏，说明 V 是新的

凸包顶点，需要更新 H。更新 H 的过程如下：先确定候选点 V 所在的活动分区，然后依据 V 所在的活动分区和候选点的位置信息，去除当前有序凸包点列中不再是凸包顶点的点，再把 V 放入到 H 中的合适位置使得凸包的边同侧特性得以满足。上述构造当前有序凸包点列的过程严格保证了凸包的边同侧特性，故该构造过程是正确的。以下是求解 $M1$ 到 $M2$ 之间的有序凸包点列 H 的算法。

算法 3.1 SQRT_CHARRAY_SP(M_1, M_2, Q)

Step1. 若 $M_1=M_2$，则 $H=\{M_1\}$，结束；否则，若 $Q=\varnothing$，则 $H=\{M_1, M_2\}$，结束；若 $Q\neq\varnothing$，设 $Q=\{Q_1, \cdots, Q_s\}$，$s\geqslant 1$，令 $H=\{M_1\}$，$r=1$，$i=1$，$V=Q_1$，继续；

Step2. 若 IsCandidate(V)=1，则 DEAL_CANDIDATE_SP(V, H)；否则，继续；

Step3. $i=i+1$，若 $i\leqslant s$，则 $V=Q_i$，转 Step2；否则，继续；

Step4. $r=r+1$，$H_r=M_2$。

算法 3.1 中的 IsCandidate(V)是判断点 V 是否为候选点的函数，设遍历多边形的方向为顺时针，V_1 和 V_2 分别为 V 的前、后相邻顶点，依据候选点的定义：若 $S(M_1, M_2, V)>0$ 且 $S(V_1, V, V_2)<0$，则 IsCandidate(V)=1；否则，IsCandidate(V)=0。DEAL_CANDIDATE_SP(V, H)是对候选点 V 进行处理并对 H 进行必要的更新的过程(见算法 3.2)。

算法 3.2 DEAL_CANDIDATE_SP(V, H)

Step1. 若 $r=1$，则 $r=r+1$，$H_r=V$，退出；否则，继续；

Step2. 若 INAVR_SP(V, H)=0，则 H 不变，退出；否则，j=INAVR_SP(V, H)，继续；

Step3. 若 $V(x)\geqslant H_r(x)$，则 $r=j+1$，$H_r=V$，退出；否则，继续；

Step4. 若 $S(V, M_2, H_r)>0$，则令 $W=H_r$，$r=j+1$，$H_r=V$，$r=r+1$，$H_r=W$，退出；否则，$r=j+1$，$H_r=V$，退出。

其中，INAVR_SP(V, H)过程判断 V 是否在活动区内，若是，则确定其所在的活动分区(见算法 3.3)。依据 V 与活动线的位置关系进行判断，若 V 位于活动线的左侧，则 V 在活动区内。设 j=INAVR_SP(V, H)：若 $j=0$，则点 V 不在活动区内；否则，$AR(H_j)$就是 V 所在活动分区。

算法 3.3 INAVR_SP(V)

Step1. 若 $V(x)=H_r(x)$，转 Step2；若 $H_r(x)<V(x)<M_2(x)$，转 Step3；否则，转 Step5；

Step2. 若 $V(y)>H_r(y)$，转 Step4；否则，返回 0；

Step3. 若 $S(H_{r-1}, H_r, V)\geqslant 0$，转 Step4；若 $S(H_{r-1}, H_r, V)<0$ 且 $S(H_r, M_2, V)>0$，返回 r；否则，返回 0；

Step4. 若 $S(H_1, H_2, V)\geqslant 0$，则返回 1；若 $S(H_{r-1}, H_r, V)<0$，则返回 r；否则，$l=2$，$u=r-1$，j=FIND_AR(V, l, u)；

Step5. 若 $S(H_{r-1}, H_r, V)>0$，转 Step4；否则，返回 0。

算法 3.3 中的 FIND_AR(V, l, u, H)是查找 V 所在活动分区的函数：在 H_l 与 H_u 之间查找 H_j，使得 $S(H_{j-1}, H_j, V)<0$ 且 $S(H_j, H_{j+1}, V)\geqslant 0$，返回 j，$AR(H_j)$就是 V 所在活动分区(特别地，当 $S(H_1, H_2, V)\geqslant 0$ 时，$j=1$)。注意，当 $V(x)<H_r(x)$ 且 V 在 $L(H_{r-1}H_r)$之上或右侧时，由于点列 Q 取自简单多边形，V 不可能在活动区内，否则，V 与 H_r 的连线就与多边形的边相交，这与定义 2 矛盾，故在 Step5 中返回 0。

求解多边形凸包的完整算法如下：先找到多边形的极值点 M_i(i=1, 2, …, 8)，再求解得出 M_i 到 M_{i+1}(i=1, 3, 5, 7)之间的有序凸包点列，最后将 M_i 到 M_{i+1}(i=1, 3, 5, 7)之间的有序凸包点列顺序连接起来并去掉冗余的 M_i 点就得出多边形的凸包。

下面，将新算法与已有算法进行比较。文[10]中的算法是针对有序简单多边形的，若采用本文算法，依据活动区的定义，可以排除不在 H_rM_{i+1} 连线左侧的候选点，且只要用到算法 3.2 的 Step1、Step2 和 Step3。而文[10]中的算法在前瞻过程中不能排除这样的点，只能在回溯过程中加以去除，故本文算法可以减少不必要的判断。而且，依据活动分区内的点的性质，对于候选点所在活动分区的查找可以使用二分查找法，而文[10]中的回溯过程是顺序进行的，假设当前有序凸包点列中有 h 个点，则文[10]的回溯过程需要 $O(h)$时间，而本文算法的回溯过程需要 $O(\log h)$时间。文[13]与本文算法依据的原理相同，而本文算法要优于其算法。其一，本文算法先淘汰了非候选点的遍历点，而文[13]的算法是求得可能成为凸包顶点的顶点列后，再淘汰不属于图 1 中子区域 I、II、III 和 IV 中的点，这样会增加对明显不能成为凸包顶点的点的多余判断，降低效率。其二，文[13]的算法存在问题。其算法中的 Step6 提出，若当前顶点在已形成的凸包顶点栈中栈顶两元素连线之右，且其 y 坐标大于栈顶元素的 y 坐标，就将其加入凸包顶点栈内。如图 3 所示，当算法执行到点 V，按照 Step6，点 V 的 y 坐标大于栈顶元素 H_r 的 y 坐标，应加入点 V 到 H 中，再到 Step7，发现点列中点遍历完毕，转 Step9，加入 M_2，结束。从图 3 中可以看出，在加入 M_2 之后，点 V 显然不再是凸包顶点。该算法的 Step8 中也存在类似问题，可能导致在构建凸包顶点列的过程中将明显不在凸包上的点(如图 3 中的 V)加入到 H 中，影响对后续点的判断，且最终可能导致在生成的凸包点列中存在不是凸包顶点的点。

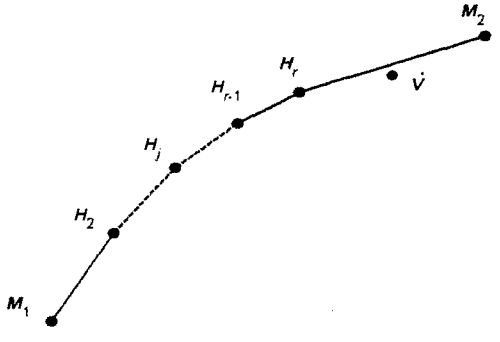

图 3 求解简单多边形凸包的算例分析

4 平面点集凸包算法

简单多边形是平面点集的特例，故扩展其凸包算法可以得到平面点集的凸包算法。

平面点集与多边形不同之处在于：一是平面点集中的点没有凹、凸顶点之分；二是平面点集不能依据极值点自然进行点集的子集划分，三是平面点集中的点并不具有多边形顶点之间的某些特殊性质。因而，平面点集的凸包算法不是多边形凸包算法的简单扩展。定义平面点集的候选点为在极值点 M_i 到 M_{i+1}(i=1, 3, 5, 7)连线左侧的点。称落入图 1 中子区域 I、II、III 和 IV(不含边界)中的候选点集分别为 M_i 到 M_{i+1}(i=7, 5, 3, 1)之间的候选点子集。候选点子集划分过程如下：先找到点集的极值点，然后遍历点集中的非极值点 V，若 V 满足 $S(M_i, M_{i+1}, V)>0$(i=1 或 3 或 5 或 7)，则 V 为 M_i 到 M_{i+1}之间的候选点子集中的点。同样，以求解 M_1 到 M_2 之间的有序凸包点列为例进行说明。设 M_1 到 M_2 之间

的候选点子集为 Q。以下是对给定 Q 进行求解 M_1 到 M_2 之间的有序凸包点列 H 的算法。

算法 4.1 SQRT_CHARRAY_PPS(M_1,M_2,Q)

Step1. 若 $M_1=M_2$,则 $H=\{M_1\}$,结束;否则,若 $Q=\varnothing$,则 $H=\{M_1,M_2\}$,结束;若 $Q\neq\varnothing$,设 $Q=\{Q_1,\cdots,Q_s\}$,$s\geqslant 1$,令 $H=\{M_1\}$,$r=1$,$i=1$,$V=Q_1$,继续;

Step2. DEAL_CANDIDATE _PPS(V),$i=i+1$;

Step3. 若 $i\leqslant s$,则 $V=Q_i$,转 Step2;否则,继续;

Step4. $r=r+1$,$H_r=M_2$。

算法 4.1 中的 DEAL_CANDIDATE_PPS(V)(见算法 4.2)是对 Q 中的点进行处理并对 H 进行必要的更新的过程。为说明该过程,需要先引入反向活动分区的概念。活动区定义仍然如前,称前面定义的关于 H_j 的活动分区为 H_j 的正向活动分区,仍然记为 $AR(H_j)$,FIND_AR(V,l,u)不变,是查找 V 所在正向活动分区的函数。图 4 中的阴影部分即是与 $H_j(j=2,\cdots,r-1)$所对应的反向活动分区,其中的点在 $R(H_jH_{j-1})$的右侧,并在 $R(H_{j+1}H_j)$的左侧或其上,称这样的活动分区为 H_j 的反向活动分区,记为 $IAR(H_j)$。称 $R(H_{j+1}H_j)$、$R(H_jH_{j-1})$分别为 $IAR(H_j)$的前、后边界。于是,给定点 $V\in IAR(H_i)$ $(l\leqslant i\leqslant u)$和点 $H_j(l\leqslant j\leqslant u)$,有如下结论:当 V 在 $R(H_jH_{j-1})$右侧且在 $R(H_{j+1}H_j)$左侧或其上,则 $V\in IAR(H_j)$;当 $V(y)\geqslant H_j(y)$或 $V(y)<H_j(y)$且 V 在 $R(H_{j+1}H_j)$右侧时,$V\in IAR(H_i)$ $(j+1\leqslant i\leqslant u)$;其他,$V\in IAR(H_i)$ $(l\leqslant i\leqslant j-1)$。在此,定义查找 V 所在反向活动分区的函数为 FIND_IAR(V,l,u,t,H):即在 H_l 与 H_u 之间查找 H_k,使得 $S(H_k,H_{k-1},V)<0$ 且 $S(H_{k+1},H_k,V)\geqslant 0$,返回 k,$IAR(H_k)$就是 V 所在的反向活动分区,若 $S(H_{k+1},H_k,V)=0$,即 V 在 $IAR(H_k)$的前边界上,则 $t=1$,否则,$t=0$。

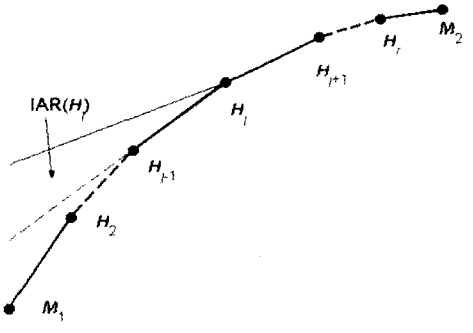

图 4 M_1 到 M_2 之间的反向活动分区示例

算法 4.2 DEAL_CANDIDATE_PPS(V)

Step1. 若 $r=1$,则 $r=r+1$,$H_r=V$,退出;否则,INAVR_PPS(V,m,n,t);

Step2. 若 $m=0$,则 H 不变,退出;若 $n=0$,转 Step3;若 $n>0$,转 Step5;

Step3. 若 $V(x)\geqslant H_r(x)$,则 $r=m+1$,$H_r=V$,退出;否则,继续;

Step4. 若 $S(V,M_2,H_r)>0$,则令 $W=H_r$,$r=m+1$,$H_r=V$,$r=r+1$,$H_r=W$,退出;否则,$r=m+1$,$H_r=V$,退出。

Step5. 若 $t=0$,则删除 H 中在 H_m 之后并在 H_n 之前的点,令 $H_{m+1}=V$,将原 H 中自 H_n 以后的点接续到 H_{m+1} 之后,再令 $r=r-n+m+2$,退出;若 $t=1$,则删除 H 中在 H_m 之后并在 H_{n+1}之前的点,令 $H_{m+1}=V$,将原 H 中自 H_{n+1} 以后的点接续到 H_{m+1} 之后,再令 $r=r-n+m+1$,退出。

算法 4.2 的 Step1 中的 INAVR_PPS(V,m,n,t) 过程判断点 V 是否在活动区内。若是,则确定 V 所在的活动分区(见算法 4.3)。若 $m=0$,则 V 不在活动区内;若 $m>0$,则 $AR(H_m)$就是 V 所在正向活动分区。若 $n>0$,则 $IAR(H_n)$就是 V 所在反向活动分区;若 $t=1$,则 V 在 $IAR(H_n)$的前边界上;若 $t=0$,则 V 不在 $IAR(H_n)$的前边界上。

对于平面点集中的点,则要分两种情况来分别确定点 V 所在活动分区:(a) $V(x)\geqslant H_r(x)$和 $V(x)<H_r(x)$且 V 在 $L(H_{r-1}H_r)$左侧;(b) $V(x)<H_r(x)$且 V 在 $L(H_{r-1}H_r)$之上或右侧。对情况(a),若确定 V 在活动区内,则只要确定 V 所在的正向活动分区就可以更新 H,具体确定过程与 INAVR_SP(V)中的 Step1 到 Step5 一样。对情况(b),在简单多边形凸包算法中,由于利用了简单多边形的性质,在算法 3.3 确定点 V 所在活动分区的 INAVR_SP(V) 过程中的 Step5 排除 V 在活动区内的可能性。而对于平面点集则不能简单加以排除,需要先查找 H_j,使得 $H_j(x)<V(x)\leqslant H_{j+1}(x)$,若 V 在 $S(H_{j-1}H_j)$之上或右侧,则 V 不在活动区,否则,V 在活动区内。若确定 V 在活动区内,则要如算法 4.3 确定点 V 所在的正、反向活动分区以便更新 H。

算法 4.3 INAVR_PPS(V,m,n,t)

Step1. 若 $V(x)=H_r(x)$,令 $n=0$,转 Step2;若 $H_r(x)<V(x)<M_2(x)$,令 $n=0$,转 Step3;否则,转 Step5;

Step2. 若 $V(y)>H_r(y)$,转 Step4;否则,则令 $m=0$,退出;

Step3. 若 $S(H_{r-1},H_r,V)\geqslant 0$,转 Step4;若 $S(H_{r-1},H_r,V)<0$ 且 $S(H_r,M_2,V)>0$,则令 $m=r$,$n=0$,退出;其他,令 $m=0$,退出;

Step4. 令 $l=1$,$u=r-1$,$m=$FIND_AR(V,l,u,H),退出;

Step5. 若 $S(H_{r-1},H_r,V)>0$,令 $n=0$,转 Step4;否则,查找 H_j,使得 $H_j(x)<V(x)\leqslant H_{j+1}(x)$,若 $S(H_j,H_{j+1},V)\leqslant 0$,则令 $m=0$,$n=0$,退出,若 $S(H_j,H_{j+1},V)>0$,则 FIND_AVR_PPS(V,j,m,n,t,H);

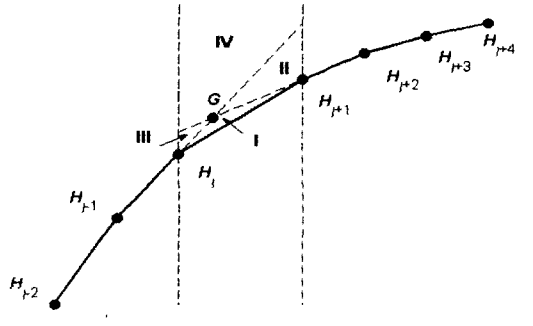

图 5 $H_j(x)<V(x)\leqslant H_{j+1}(x)$间的活动区的子区域划分

算法 4.3 中的 FIND_AVR_PPS(V,j,m,n,t,H)是依据 V 所在的位置来确定 V 所在的正、反向活动分区的过程(见算法 4.4)。假设点 V 在活动区内,则 V 在如图 5 中所示的阴影部分所示的活动分区中,该活动分区被两条直线 $H_{j-1}H_j$ 和 $H_{j+1}H_{j+2}$划分成子区域 I、II、III 和 IV(不含直线 $H_{j-1}H_j$ 和 $H_{j+1}H_{j+2}$)。设直线 $H_{j-1}H_j$ 与 $H_{j+1}H_{j+2}$的交点为 G。当 $j=r-2$ 时,只有子区域 I、III;当 $j=1$ 时,若 V 在 $R(H_3H_2)$左侧,则 V 在子区域 I 内,若 V 在 $R(H_3H_2)$右侧,则 V 在子区域 II 内。表 1 是依据 V 所在的位置来确定 m、n、t 的取值表,其中有类似 $l=j+2$,$u=r-1$ 的形式:若这样的形

式出现在 m 取值列中，则 m=FIND_AR(V,l,u,H)；若这样的形式出现 n 取值列中，则 n=FIND_IAR(V,l,u,t,H)，同时可以得出相应的 t 值。m、n、t 的取值确定下来之后，就可以对 H 进行必要的更新。

表 1 依据 V 的位置确定 m、n、t 的取值表

V 的位置	M	n	t
$S(H_{j-1},H_j,V)=0$ 且 $S(H_{j+1},H_{j+2},V)=0$	$j-1$	$j+1$	1
$S(H_{j-1},H_j,V)=0$ 且 $S(H_{j+1},H_{j+2},V)<0$	$j-1$	$j+1$	0
$S(H_{j-1},H_j,V)=0$ 且 $S(H_{j+1},H_{j+2},V)>0$	$j-1$	$l=j+2,u=r-1$	
$S(H_{j-1},H_j,V)<0$ 且 $S(H_{j+1},H_{j+2},V)=0$	j	$j+1$	1
$S(H_{j-1},H_j,V)<0$ 且 $S(H_{j+1},H_{j+2},V)<0$	j	$j+1$	0
$S(H_{j-1},H_j,V)<0$ 且 $S(H_{j+1},H_{j+2},V)>0$	j	$l=j+2,u=r-1$	
$S(H_{j-1},H_j,V)>0$ 且 $S(H_{j+1},H_{j+2},V)=0$	$l=1,u=j-1$	$j+1$	1
$S(H_{j-1},H_j,V)>0$ 且 $S(H_{j+1},H_{j+2},V)<0$	$l=1,u=j-1$	$j+1$	0
$S(H_{j-1},H_j,V)>0$ 且 $S(H_{j+1},H_{j+2},V)>0$	$l=1,u=j-1$	$l=j+2,u=r-1$	

算法 4.4 FIND_AVR_PPS(V,j,m,n,t,H)

Step1. 若 $j=1$，令 $m=1$，转 Step2；若 $j=r-2$，令 $n=r-1$，若 $S(H_{r-1},H_r,V)=0$，则令 $t=1$，若 $S(H_{r-1},H_r,V)<0$，则令 $t=0$，转 Step3；否则，转 Step4；

Step2. 若 $S(H_3,H_2,V)=0$，则令 $n=2,t=1$，退出；若 $S(H_3,H_2,V)>0$，则令 $n=2,t=0$，退出；否则，设 $l=3,u=r-1$，n=FIND_IAR(V,l,u,t,H)，退出；

Step3. 若 $S(H_{r-3},H_{r-2},V)<0$，则令 $m=r-2$，退出；若 $S(H_{r-3},H_{r-2},V)=0$，则令 $m=r-3$，退出；否则，设 $l=1,u=r-2$，m= FIND_AR(V,l,u,H)，退出；

Step4. 依据表 1 来确定 m、n、t。

求解平面点集凸包的完整算法如下：先找到极值点 M_i（$i=1,2,\cdots,8$）并划分得到 M_i 到 M_{i+1}（$i=1,3,5,7$）之间的候选点子集，再求解得出 M_i 到 M_{i+1}（$i=1,3,5,7$）之间的有序凸包点列，最后将 M_i 到 M_{i+1}（$i=1,3,5,7$）之间的有序凸包点列顺序连接起来并去掉冗余的 M_i 点就得出多边形的凸包。

5 算法复杂度分析

这里只分析平面点集凸包新算法的时间复杂度。在求解平面点集凸包的过程中，求解点集的极值点、划分极值点间的候选点子集和顺序连接极值点间的有序凸包点列并去除冗余极值点的时间复杂度为 $O(n)$。在求解极值点间的有序凸包点列的过程中，对候选点子集中的点是依次遍历，候选点至多加入有序凸包点列一次，而进入有序凸包点列中的点至多被删除一次。下面分析求解有序凸包点列过程中涉及的所有查找步骤：一是在算法 4.3 的 Step5 中，在 H_l 与 H_u 之间查找 H_j，使得 $H_j(x)<V(x)\leqslant H_{j+1}(x)$，由于有序凸包点列中点的有序性质，可以采用二分查找方法，设有序凸包点列中的点数为 h，则该查找步骤的时间复杂度为 $O(\log h)$；二是算法 4.4 中的 FIND_AR(V,l,u,A) 和 FIND_IAR(V,l,u,t,A) 两个函数，依据判定点所在活动分区的结论，同样可以采用二分查找方法，时间复杂度也为 $O(\log h)$。因此，设平面点集的点数为 n，平面点集的凸包顶点数为 h，则本文提出的平面点集凸包算法的时间复杂度为 $O(n\log h)$。

结论 本文依据凸多边形的特性，先提出了简单多边形凸包的新算法，并与相关算法进行比较，说明该算法更简单有效。然后，将该算法扩展成为平面点集凸包的新算法，其时间复杂度达到 $O(n\log h)$。关于平面点集的凸包算法，Yao[4] 于 1981 年证明了 $O(n\log n)$ 是所有平面点集凸包算法的时间复杂度的下限。文[10]提出的平面点集凸包快速求取算法的最坏时间复杂度为 $O(n\log n)$。文[11]提出平面点集凸包的最优实时算法，而在文[12]的分析中，在最坏情况下文[11]的凸包算法需要 $O(n^2)$ 时间。文[12]指出，目前平面点集凸包问题的渐进最好算法[6]的时间复杂度为 $O(n\log h)$。相比同样时间复杂度的算法[6,8]，本文算法的原理和实现要简单得多。文[6]和文[8]中的算法都要依据一定的条件对极值点间的点集进行递归划分处理，还要比较点间连线的斜率，过程复杂，不易于实现；而本文算法对极值点间的候选点子集无需进行再次划分，且在求解凸包的过程中只用到点与直线位置关系的侧向判别函数，因而本文算法简单、实现容易。由于本文算法无需再次划极值点间的候选点子集，且有序凸包点列中的点是顺序求得的，可以直接顺序存储，无需额外的空间，因而实现在本文提出的简单多边形和平面点集凸包算法的基础上的有效空间算法[14,15]要比实现基于文[6]或文[8]中算法的有效空间算法更简单，这也是本文算法的一个优点。

参 考 文 献

1 O'Rourke J. Computational Geometry in C . 2nd Edition. Cambridge: Cambridge University Press, 1998

2 Sklansky J. Measuring Concavity on a Rectangular Mosaic [J]. IEEE Transactions on Computing, 1972,C-21(12):1355～1364

3 McCallum D, Davis D. A linear algorithm for finding the convex hull of a simple polygon. In: Proc. Letters 9, 1979. 201～206

4 Yao C A. A lower bound to finding convex hulls [J]. Journal of the ACM, 1981,28(4): 780～787

5 Lee T D. On finding the convex hull of a simple polygon [J]. Int' l J Comp & Info, 1983,12(2): 87～98

6 Kirkpatrick D G, Seidel R. The ultimate planar convex hull algorithm [J]. SIAM Journal of Computers, 1986,15(1):287～299

7 Melkman A. On-line construction of the convex hull of a simple polygon [J]. Info Proc Letters 25, 1987. 11～12

8 Bhattacharya K, Sen S. On a Simple, Practical, Optimal, Output-sensitive Randomized Planar Convex Hull Algorithm [J]. Journal of Algorithms, 1997,25:177～193

9 崔国华，洪凡，余祥宣. 确定平面点集凸包的一类最优算法[J]. 计算机学报，1997,20(4):330～334

10 金文华，何涛，刘小平，等. 基于有序简单多边形的平面点集凸包快速求取算法[J]. 计算机学报，1998,21(6):533～539

11 王志强，洪嘉振，肖立瑾. 平面点集凸包的最优实时算法[J]. 计算机学报，1998,21(增刊):351～356

12 刘金义. 关于求平面点集凸包的一个 $O(n)$ 时间算法的商榷[J]. 计算机学报，2002,25(6):670～672

13 刘润涛. 一种简单多边形凸包的新线性算法[J]. 工程图学学报，2002,2:120～126

14 Brönnimann H, et al. Space-efficient planar convex hull algorithms [J]. Theoretical Computer Science, 2004,321:25～40

15 Brönnimann H, Timothy M. Chan. Space-efficient algorithms for computing the convex hull of a simple polygonal line in linear time [J]. Computational Geometry, 2006,34:75～82