第 6 卷第 5 期
1995 年 5 月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 6, No. 5
May 1995

确定任意多边形凸凹顶点的算法*

周培德

(北京理工大学计算机系,北京 100081)

摘要 本文提出一种确定任意多边形凸凹顶点的算法. 该算法的时间复杂性为 $O(n^2\log n)$次乘法和 $O(n^2)$次比较.

关键词 凸壳,多边形*,凸凹顶点*.

定义 1. 设 $P_1,P_2,\cdots,P_n$ 是给定多边形的 n 个顶点,$\overline{P_1P_2},\overline{P_2P_3},\cdots,\overline{P_nP_1}$是多边形的 n 条互不相交的边. 如果与点 $P_i(i=\overline{1,n})$相关联的两条边$\overline{P_{i-1}P_i}$与$\overline{P_iP_{i+1}}$所夹的角小于或等于 π,则称点 P_i 是凸的,否则称点 P_i 是凹的. 如果多边形的所有顶点都是凸的,则称该多边形是凸多边形,否则称该多边形是任意多边形.

在某些应用中,需要把任意的多边形分割成若干个凸多边形. 为此,如果事先能确定任意多边形各顶点是凸的,还是凹的,那么这将为分割任意多边形成凸多边形提供有利条件.

根据上述定义可以判定任意多边形顶点的凸凹性,所采用的基本算法是判断多边形所有相邻两边的夹角大于 π 还是小于 π. 该方法所需要的判夹角运算次数是 $O(n)$. 本文利用分治思想设计的算法,其基本想法是,先找出任意多边形各顶点组成的点集的凸包,该凸包的顶点必是原多边形的部分顶点,并且这些顶点是凸的. 设 P'_j 是凸包顶点,$j=\overline{1,m}$. 这 m 个点将多边形顶点序列 $P_1,P_2,\cdots,P_n$ 分成 m 个子序列,比如:

$$P_1=P'_1,\underbrace{P_2,\cdots}_{1},P_i=P'_2,\underbrace{P_{i+1},\cdots}_{2},\cdots,P_l=P'_m,\underbrace{\cdots,P_n}_{m}$$

然后确定这些子序列组成的点集 $S^k_{j_k,(j+1)_k}$,并对点集 $S^k_{j_k,(j+1)_k}\cup\{P_j,P_{j+1}\}$分别再运用求凸包的算法,新增的凸包顶点必是原多边形的凹点. 算法反复执行分割顶点序列与求凸包的工作,便可确定原多边形剩余顶点的凸凹性,直至所有点集 $S^k_{j_k,(j+1)_k}$为空.

算法描述

输入:按逆时针方向排列的任意多边形顶点的坐标 $P_i(x_i,y_i),(i=\overline{1,n})$.

输出:点 P_i 的凸凹性("+","−"分别表示凸点,凹点)

步 0:$k\leftarrow 1$

步 1:CONVEX HULL$(P_1,P_2,\cdots,P_n)$,设凸包的顶点集为 $A=\{P'_1,P'_2,\cdots,P'_m\}$,

* 本文 1993−05−06 收到,1993−08−24 定稿

作者**周培德**,1941 年生,副教授,主要研究领域为算法设计与分析,计算理论.

本文通讯联系人:周培德,北京 100081,北京理工大学计算机系

P'_j←"+"$(j=\overline{1,m})$

IF $m=n$ then 算法终止

ELSE GOTO 步 2

步 2:确定凸包顶点集 A 中相邻两点 P'_j 与 P'_{j+1}之间所包含的多边形顶点组成的子集 $S^k_{j_k,(j+1)_k}$,$(j=\overline{1,m}, P'_{m+1}=P'_1)$

IF $S^k_{j_k,(j+1)_k}=\varnothing$ THEN P'_j←"+",P'_{j+1}←"+"

ELSE GOTO 步 3

步 3:k←$k+1$,j←1

步 4:D←非空点集 $S^{k-1}_{j_{k-1},(j+1)_{k-1}}\cup\{P'_j,P'_{j+1}\}$

步 5:CONVEX HULL(D),得到凸包的顶点集 $B^l=\{B_{old},B_{new}\}$,其中 $Bold=\{P'_j,P'_{j+1}\}$,而 B_{new}是新增的凸包顶点集.

IF k 为偶数 $\wedge P\in B_{new}$ THEN P←"−"

ELSE k 为奇数 $\wedge P\in B_{new}$ THEN p←"+"

步 6:求 B^l 中相邻两点之间所包含的多边形顶点组成的子集 $S^k_{j_k,(j+1)_k}$.

步 7:j←$j+1$ GOTO 步 4,直至所有非空点集 $S^{k-1}_{j_{k-1},(j+1)_{k-1}}$均已处理过.

步 8:重复步 3 至步 7,直至所有 $S^k_{j_k,(j+1)_k}$为空.

CONVEX HULL$(P_1,P_2,\cdots,P_n)$

输入:平面上 n 个点的坐标 $P_1(x_1,y_1),\cdots,P_n(x_n,y_n)$,所有 n 个点存入 B:$array(1\cdots n)$ of $point$

输出:凸包顶点 $B_i(i=1,2,3,4)$:$array(1,\cdots,n_i)$ of $point$

步 0:IF $n=3$ THEN P_1,P_2,P_3(不在一直线上)均为凸包顶点

ELSE GOTO 步 1

步 1:求$\{x_1,x_2,\cdots,x_n\}$的最大、最小值,确定相应的点,记为 M_2、M_4.

求$\{y_1,y_2,\cdots,y_n\}$的最大、最小值,确定相应的点,记为 M_3、M_1.

步 2:连 M_1,M_2,M_3 和 M_4 成四边形,$\overrightarrow{M_iM_{i+1}}$($M_5=M_1$)右侧的点存入 $B_i(i=1,2,3,4)$.

点 $P(x,y)$在有向线段$\overrightarrow{P_1(x_1,y_1)P_2(x_2,y_2)}$的左、右侧及其线上分别决定于三阶行列式的值大于 0,小于 0 及等于 0.

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}$$

步 3:i←1

步 4:求 B_i 中各点(设 B_i 中有 n_i 个点,第 1 个和最后 1 个点分别是 M_i 和 M_{i+1})与$\overrightarrow{M_iM_{i+1}}$的距离,设为 $d_{i1},d_{i2},\cdots,d_{in_i}$,其中 $d_{i_1}=d_{in_i}=0$.

利用公式 $d=\dfrac{|(y_1-y_2)x+(x_2-x_1)y+(x_1y_2-x_2y_1)|}{\sqrt{(y_1-y_2)^2+(x_2-x_1)^2}}$可以计算点 $P(x,y)$与线段$\overrightarrow{P_1(x_1,y_1)P_2(x_2,y_2)}$的距离.

步 5:求$\{d_{i_1},d_{i_2},\cdots,d_{in_i}\}$的最大值及其对应的点,记为 $P'_i(x_i,y_i)$.因为是求最大值,只要进

行值的比较，所以可以考虑 d^2 的比较. 这样可以避免开方运算.

步 6：连接 $M_iP'_i$ 和 P'_iM_{i+1}. $\overrightarrow{M_iP'_i}$，$\overrightarrow{P'_iM_{i+1}}$右侧的点分别存入 B_{i_1} 和 B_{i_2}. 对 B_{i_1} 和 B_{i_2} 分别重复步 4(此时$\overrightarrow{M_iM_{i+1}}$分别改为$\overrightarrow{M_iP'_i}$和$\overrightarrow{P'_iM_{i+1}}$)与步 5 工作，并分别找到点 P_{i_1}'，P_{i_2}'. 连接 $M_iP'_{i_1}$，$P'_{i_1}P'_i$，$P'_iP'_{i_2}$，$P'_{i_2}M_{i+1}$递归地求点到线段的距离.

步 7：重复步 4、步 5 和步 6 的工作，直至所有 $B_{ij\cdots1m}$都成为空为止. i 由 1 增至 4，其它下标 j，…，1，m取值 0、1 或 2. 每次找到的点 $P'_{ij\cdots1}$必是凸包的一个顶点，而全部点 $P'_{ij\cdots1}$便组成凸包的顶点.

算法的正确性与复杂性

定理 1. 算法正确地确定了多边形顶点的凸凹性.

证明：算法中的步 1，求出的顶点显然是多边形的凸点. 经步 2 处理后，多边形的顶点序列被分成 m 个子序列. 设 P_i，…，P_{i+a}是其中的一个子序列(P_i 与 P_{i+a}已判定是相邻的两个凸点). P_{i+1}，…，P_{i+a-1}等 $a-1$ 个顶点均在$\overrightarrow{P_iP_{i+a}}$的左侧，它们的凸凹性待定. 对这些点再次利用求凸包的算法(步 5)，得到凸包顶点集 $B'=\{B_{old},B_{new}\}$，其中：$B_{old}=\{P_i,P_{i+a}\}$，不妨设：$B_{new}=\{P_{i+1},P_{i+4},P_{i+a-1}\}$. P_{i+1}，P_{i+4}，P_{i+a-1}是新增的凸包顶点. 凸包中与点 P_{i+4}相关联的两条边$\overline{P_{i+1}P_{i+4}}$，$\overline{P_{i+4}P_{i+a-1}}$的夹角和原多边形中与点 P_{i+4}相关联的两条边的夹角满足下列关系

$$\angle P_{i+1}P_{i+4}P_{i+a-1}\leqslant\angle P_{i+3}P_{i+4}P_{i+5} \tag{1}$$

又因为$\angle P_{i+1}P_{i+4}P_{i+a-1}$是第 2 次($k=2$)求凸包所得的顶角，该角(位于多边形内部部分)应大于或等于 π，所以$\angle P_{i+3}P_{i+4}P_{i+5}\geqslant\pi$. 因此顶点 P_{i+4}是凹点. 依据同样的理由，第 2 次求凸包所得的新顶点均为凹点. 对 m 个子序列同样考虑，求得的凸包新顶点也是凹点. 当再次循环执行步 3 至步 8 时，即第 3 次($k=3$)大循环，求凸包所得的新顶点对应的顶角必等于或大于原多边形顶点对应的顶角，基于类似的理由，这些顶点必是凸的. 依此类推，每次大循环中求得的凸包新顶点的凸凹性依其 k 值是奇数还是偶数而定. 相邻两次大循环中所求得的凸包新顶点的凸凹性恰好相反，这是由于式(1)中的不等号反向的原因.

点集 $S^k_{j_k,(j+1)_k}$表示了两种循环，一种是 j 由 1 增至 m，这是内循环，即处理 m 个子序列. 另一种是步 3 至步 8 的大循环($k\leftarrow k+1$)，处理 m 个子序列中的子序列. 每当进入一次新的大循环时，处理子序列的层次也增加 1. 直至所有的 $S^k_{j_k,(j+1)_k}$为空. 此时多边形的所有顶点均已判定是凸的还是凹的. 因此，算法正确地确定了多边形顶点的凸凹性.

定理 2. 在最坏情况下，算法的时间耗费是 $O(n^2\log n)$次乘法和 $O(n^2)$次比较.

证明：算法的步 2 和步 6 用性质相同的操作可以完成，该操作实际上是比较操作. 因为输入时，多边形顶点的下标都是按自然数顺序排列的，即 $P_1,P_2,\cdots,P_n$. 经步 1 处理后求得凸包顶点 $P'_1,P'_2,\cdots,P'_m$. $P'_j(j=\overline{1,m})$将序列 $P_1,P_2,\cdots,P_n$ 分成 m 个子序列，只要经过 n 次比较，便可构成所有的点集 $S^k_{j_k,(j+1)_k}(k=1)$. 同理，步 6 利用数目少于 n 的比较操作可以构成点集 $S^k_{j_k,(j+1)_k}(k\geqslant2)$. 而步 6 最多循环执行 $n-2$ 次，故步 2 和步 6 至多需要 $O(n^2)$次比较.

步 1 和步 5 求凸包是算法最耗费时间的步骤. 在最坏情况下，步 1 求得的凸包只包含 3 个顶点，而每次循环时，步 5 中求得的凸包顶点集 B_{new}只包含 1 个顶点. 这时需要求 $n-2$ 次

凸包.由文献[1]知,CONVEX HULL($P_1,P_2,\cdots,P_n$)至多耗费 $O(n\log n)$次乘法和 $O(n)$次比较.算法的其它步骤耗费常数时间.因此,算法在最坏情况下至多耗费 $O(n^2\log n)$次乘法和 $O(n^2)$次比较.

步 5 中的每次求凸包,由于点集 $S^{k-1}_{j_{k-1},(j+1)_{k-1}}$中所含点的数目总是小于 n,所以算法的时间复杂性低于上述的估计.

参考文献

1　周培德.算法设计与分析.北京:机械工业出版社,1992.

2　周培德.求凸壳顶点的一种算法.北京理工大学学报,1993,**13**(1):69—72.

3　Preparata F P. Advances in computing research. Computational Geometry, JAI PRESS INC., 1983.

4　Liu Hongchih, Srinath M D. Corner detection from chain—code. Pattern Recognition, 1990, **23**(1,2):51—68.

AN ALGORITHM FOR DETERMINING CONVEXO—CONCAVE VERTICES OF AN ARBITRARY POLYGON

Zhou Peide

(*Department of Computer Science, Beijing Institute of Technology, Beijing* 100081)

Abstract　This paper presents an algorithm for determining convexo—concave vertices of an arbitrary polygon. The algorithm requires $O(n^2\log n)$ multiplications and $O(n^2)$ comparisons.

key words　Convex hulls, polygon*, convexo—concave vertices*.