第18卷 第4期 华侨大学学报(自然科学版) Vol.18 No.4
1997年10月 Journal of Huaqiao University (Natural Science) Oct.1997

任意多边形区域的裁剪*

范慧琳① 杨 幸②

(① 华侨大学计算机科学系, ② 华侨大学电子工程系, 泉州 362011)

摘要 讨论任意多边形区域的裁剪技术,给出一个任意多边形裁剪算法,并用TURBO PASCAL加以实现.

关键词 多边形区域,多边折线,裁剪,窗口,算法
分类号 TP 391.41

多边形裁剪是计算机图形技术的基本问题. 本文讨论的多边形裁剪是指多边形实面区域的裁剪,裁剪后的多边形边界除了原来多边形经裁剪后的线段外,还需要添上窗口区域的若干段边线以组成封闭的裁剪后的多边形区域. 多边形裁剪(这里选用矩形裁剪窗口)有两种方法:第一种方法是多边形的每一条边相对于整个窗口作裁剪,如Liang-Barsky算法和Weiler-Acherton算法[1];第二种方法是整个多边形相对于裁剪窗口的每一边界逐次进行裁剪,如Sutherland-Hodgman算法[2]. 这些算法在复杂多边形的处理上均存在一定的局限,如对自相交多边形无法作出正确的裁剪. 本文以Dan Cohen和Ivan Sutherland的区域编码方法[3]为基础讨论任意多边形的裁剪,并给出一个裁剪算法,能正确地实施凸、凹或自相交多边形,即任意多边形的裁剪. 该算法在486微机上用TURBO PASCAL实现.

1 任意多边形裁剪算法的设计思想

多边形裁剪是基于定义成多边折线的多边形轮廓线的裁剪[4],必须求出多边折线与窗口边线的交点(称为裁剪点),截取位于窗口内的可见部分. 多边折线的裁剪结果是生成可见多边折线表. 除此之外,为了输出封闭多边形,还必须确定窗口边线的可见部分. 每一窗口边线的可见线段的确定是基于与它相关的裁剪点表. 线段与窗口边线的裁剪点 $CP(x,y)$ 可用直线的两点式求得. 位于输入多边形内的角点被看成是2个(相重的)裁剪点,分别作为通过这一角点的两条窗口边线的裁剪点. 对每一条窗口边线,裁剪点表 $CP_1, CP_2, \cdots, CP_n$ 中的裁剪点数为偶数,按 $x_i < x_{i+1}$(对水平窗口边线)和 $y_i < y_{i+1}$(对垂直窗口边线)的顺序设置裁剪点在裁剪点表中的排列顺序,其中 (x_i, y_i) 为 CP_i 的坐标. 将表中的裁剪点成对组合成 $[CP_1, CP_2], [CP_3, CP_4], \cdots, [CP_{n-1}, CP_n]$,称为点对,每个点对表示窗口边线的一条可见线段,它是输出多边形的边. 每一窗口边线的所有点对便构成了该窗口边线的可见线段表. 最后,使用可见多边折线

* 本文1997-01-04收到

表和可见线段表收集整理出输出多边形的顶点表进而实现多边形的裁剪.

综上所述, 任意多边形裁剪算法可以通过以下三个步骤实现.

(1) 借助于多边折线(多边形轮廓线)裁剪算法求得可见多边折线表, 确定多边形轮廓的可见部分. 多边折线裁剪算法作为任意多边形裁剪算法的基础, 必须解决下列二个附加要求: (a) 当多边折线的结点或向量位于窗口边线上时, 如何正确确定裁剪点. 即要正确地解决当多边形的某一顶点或边位于窗口边线上时的退化情况. (b) 确定位于输入多边形内的窗口角点.

(2) 生成每一窗口边线的可见线段表, 确定窗口边线的可见部分.

(3) 利用可见多边折线表和可见线段表收集整理出输出多边形的顶点表. 而多边折线是由若干按顺序相连的向量构成的, 因此多边折线裁剪算法可由应用于每一向量(多边折线的一个组成部分)的线裁剪算法来实现[5]. 但必须注意的是, 这里的线裁剪算法必须满足以上两个附加要求, 并且在算法中必须存贮该向量在多边折线中的顺序的信息. 以下着重讨论这一线裁剪算法.

2　线裁剪算法

使用最为广泛的线裁剪算法是 Cohen-Sutherland 算法[3], 该算法的主要思想是对直线段的端点进行编码来测试线段是否需要裁剪进而完成裁剪. 本文的线裁剪算法采用了 Cohen-Sutherland 端点编码方法, 并作了进一步的工作, 使之满足作为任意多边形裁剪算法的一部分所需的2个附加要求.

2.1　线端点编码

窗口的4条边界把平面划分为9个区域, 每个区域设定四位二进制编码, 如图2所示, 线段的2个端点被赋予它们所在区域的相应编码. 四位编码 $C_tC_bC_rC_l$ 的编码规则如下:

(1) 若线段端点位于窗口左侧或在左边界上, 则 C_l= 1; (2) 若线段端点位于窗口右侧或在右边界上, 则 C_r= 1, ; (3) 若线段端点位于窗口下侧或在底边界上, 则 C_b= 1; (4) 若线段端点位于窗口上侧或在顶边界上, 则 C_t= 1. 显然, 如果端点 V 的编码 P-CODE(V) = 0000, 则 V 在窗口内.

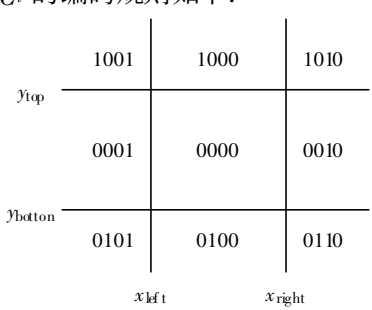

图1　区域编码

与 Cohen-Sutherland 编码不同的是, 这里设定窗口边界属于外部区域, 即左、右、底和顶边界的编码依次为0001, 0010, 0100和1000. 这样, 当多边折线的结点或向量位于窗口边界上时, 则认为它们位于窗口外, 因此以上第1节提出的第一个附加要求无需特别的处理便解决了.

2.2　线编码

借助于线端点编码, 建立两种线编码如下:

$$L\text{-}CODE1(V_1, V_2) = P\text{-}CODE(V_1).AND.P\text{-}CODE(V_2)$$

$$L\text{-}CODE2(V_1, V_2) = P\text{-}CODE(V_1).OR.P\text{-}CODE(V_2)$$

当第一种线编码 L-CODE1(V_1, V_2) ≠0时, 则线段完全不可见, 这是整条线不可见性的第一个判别条件. 当 L-CODE1(V_1, V_2) = 0时, 再确定 L-CODE2. 如果 L-CODE2(V_1, V_2) = 0, 则线

段位于窗口内,完全可见. 其余情况说明线段 V_1, V_2可能与窗口边线相交,必须进一步分析,确定可能存在的裁剪点.

2.3 基于端点的交点编码

对线段 V_1V_2,以其每一端点为出发点,确定线段与某一指定窗口边界的交点,这一交点的确定给出了线段相对于裁剪窗口的位置的有关信息,即线段是否穿过窗口或完全位于窗口外. 交点编码与相应的窗口边界编码一致. 当整条线不可见性的第一个条件不满足时,则须确定基于线段端点的交点编码.

以线段某一端点为基准,若该端点位于某一边界的外部中间区域,则考虑与该边界的交点,此时端点的交点编码与它的端点编码相同,如图2(a)所示. 当该端点位于角区域中,则其交点编码不同于端点编码,如图2(b)所示,若 V_1属于边 E,则 V_1的交点编码 IN-CODE(V_1)是1000;如果 V_1属于 E 则 V_1的交点编码是0001. 为了确定位于角区域中的端点 $V_e(x_e, y_e)$ 的交点编码,必须把 V_1V_2的斜率与用该端点和窗口角点 $P_c(x_c, y_c)$ 连成的线段斜率作比较.

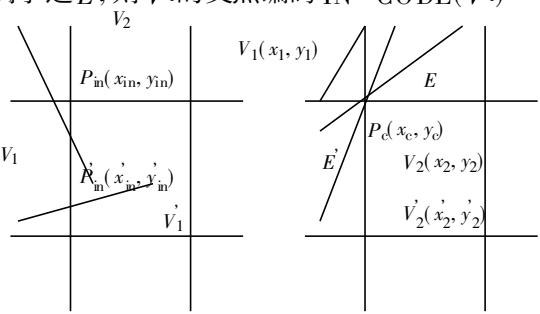

图2 交点编码的确定

下面考虑整条线不可见性的第2个判别条件. 对用线段端点的交点编码信息确定的交点 $P_{in}(x_{in}, y_{in})$,如果下列不等式

$$\begin{cases} x_{left} \le x_{in} \le x_{right} & \text{对水平边界} \\ y_{botton} \le y_{in} \le y_{top} & \text{对垂直边界} \end{cases}$$

成立,则 $P_{in}(x_{in}, y_{in})$ 是裁剪点,该线段穿越窗口. 如果以上不等式不成立,则线段完全位于窗外且它与其它窗口边界的其余交点均无需计算. 这是整条线不可见性的第2个判别条件.

该条件的设定使本算法的执行效率高于已有的裁剪算法. 从其构造过程可见,若多边折线上的向量与按上述方法确定的窗口边线无交点,则它与其它边线也必无交点,因而能快速地判别出多边折线中的不可见向量并予以舍弃. 特别是整个多边形或多边形的大部分在裁剪窗外且与其边界相交的情形,比已有算法(如 SH, SM 算法[3])更有效. 如图3,若按 SM 算法首先针对顶边界作裁剪,则需计算6个交点,而其中有4个交点的计算不是必需的,因为它们不是裁剪点. 若按本算法,通过使用交点编码和不可见性的两个条件,只需计算 V_2V_3, V_3V_4, V_4V_5与顶边界的交点,不必要的交点计算明显减少,仅当不可见性的第2条件成立时,才会产生不必要的交点计算,如图3中 V_4V_5与顶边界的交点.

2.4 确定线段与窗口边线之延长线的交点

这是为了判别窗口角点是否位于输入多边形内,即解决以上第2节给出的第2个附加要求. 窗口上的8条延长线被赋予8位二进制编码:

LEN-CODE= P-CODE(边界1) * 16+ P-CODE(边界2)

这里,延长线是位于边界2之外的边界1的部分,如图4所示,当延长线与多边折线的向量相交时,该延长线对应的数字计数器增1.

以下分析线段 V_1V_2与延长线相交的三种可能情况,进而确定相应的延长线编码.

(1) 根据整条线不可见性的第一个条件,线段 V_1V_2是完全不可见的且它的端点位于不同

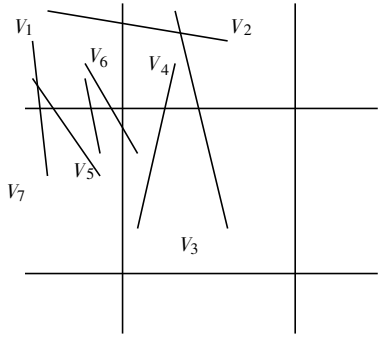

图3　图例

图4　延长线编码

的区域中, 如图5(a)所示, 即 L-CODE1(V_1, V_2) ≠0且 P-CODE(V_1) ≠P-CODE(V_2), 与线段相交的延长线用满足 P-CODE(V_e) ≠L-CODE1(V_1, V_2) 的位于角区域中的线端点 V_e 编码确定, 位于角区域中的每一端点 V_e 定义一个与延长线的交点, 此延长线的编码为

$$\text{LEN-CODE}=(\text{P-CODE}(V_e)-\text{L-CODE1}(V_1,V_1))*\ 16+\ \text{L-CODE1}(V_1,V_2)$$

(2) 根据整条线不可见性的第2个条件, 线段 V_1V_2 是完全不可见的, 如图5(b)所示, 此时线段必然穿过2条延长线, 其编码为

$$\text{LEN-CODE}(1)=\text{IN-CODE}(V_1)*\ 16+\ \text{IN-CODE}(V_2)$$

$$\text{LEN-CODE}(2)=\text{IN-CODE}(V_2)*\ 16+\ \text{IN-CODE}(V_1)$$

如果线段端点 V_e 位于角区域, 即 P-CODE(V_e) ≠IN-CODE(V_e), V_e 定义了一个与延长线的交点, 此延长线的编码为

$$\text{LEN-CODE}=(\text{P-CODE}(V_e)-\text{IN-CODE}(V_e))*\ 16+\ \text{IN-CODE}(V_e)\cdots\cdots(*)$$

(3) 线段 V_1V_2 是部分可见的, 即 L-CODE1(V_1, V_2) = 0 AND L-CODE2(V_1, V_2) ≠0, 如图5(c)所示, 位于角区域中的线段每一端点 V_e 定义了线段与延长线(其编码由(*)式指定)之间的交点.

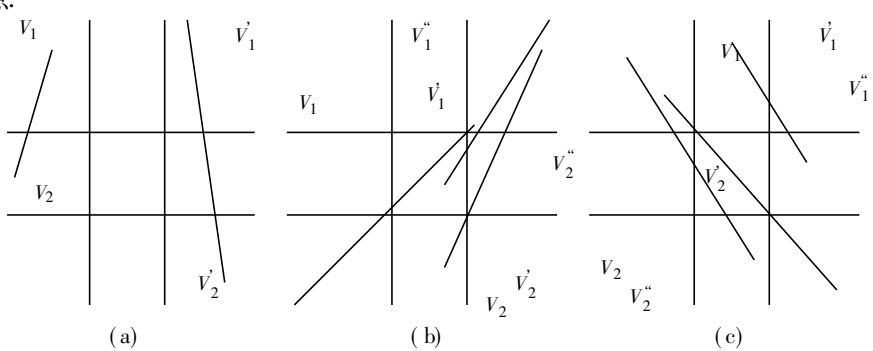

图5　线段在窗口中的位置

本文先用判交点个数法[2]来确定窗口角点是否位于输入多边形内. 算法中存贮了输入多边形与引自这一角点的每一延长线的交点数, 当两条延长线的交点数均为奇数, 则角点作为两个(相重的)裁剪点; 当仅一条延长线的交点数为奇数, 而另一条的交点数是偶数或零, 该角点仅作为延长线交点数为偶数或零的窗口边线的裁剪点; 当两条延长线的交点数均为偶数或零, 则该角点位于多边形之外.

2.5 线裁剪算法的类-PASCAL描述

由以上2.1～2.4小节的分析,现把作为任意多边形裁剪算法基础的线裁剪算法用类-PASCAL描述如下

```
procedure Line - Clip(x1, y1, x2, y2: real);
VAR  P - CODE, IN - CODE: array [1…2] of integer;
     L - CODE1, L - CODE2: integer;
begin
  确定端点V1(x1, y1), V2(x2, y2)的P - CODE[1]和P - CODE[2];
  L - CODE1= P - CODE[1] .AND. P - CODE[2];
  if L - CODE1≠0 then
    确定与线相交的延长线编码且其计数器增1
  else
    [L - CODE2= P - CODE[1]. OR. P - CODE[2];
    if L - CODE2= 0 then
        存贮整条线
    else
        if 在L - CODE2中仅一位被置为'1' then
            计算线与用P - CODE≠0确定的窗口边界的裁剪点坐标
        else
          [计算端点V1, V2的IN - CODE[1]和IN - CODE[2];
          for i: = 1 to 2 do
            if IN - CODE[i] ≠ P - CODE[i] then
    端点在角区域,确定与线段相交的延长线编码且其计数器增1;
    if P - CODE[1]≠0, AND. P - CODE[2]≠0 then
    [计算线段V1, V2与用IN - CODE[1]确定的窗口边界的交点坐标;
    if 整条线不可见性的第2条件为true then
      计算完全位于窗口外的线段必定穿过的2条延长线的编码;
    else 计算线段与用IN - CODE[2]确定的窗口边界的交点坐标并存贮线段的可见部分
  ]
    else 计算线段与用IN - CODE[i]≠P - CODE[i]确定的窗口边界的交点坐标并存贮线段可见部分
    ]
  ]
endp; {Line - Clip}
```


有了以上线裁剪算法,多边折线裁剪算法的编写便迎刃而解,限于篇幅,不再赘述.

3 确定输出多边形的顶点表

每一输出多边形用其轮廓线的顶点表 OPVL 表示. 在过程中, 如果 OPVL 的第一和最后顶点重合, 即多边形已封闭, 则从 OPVL 中删去最后顶点, 所得到的 OPVL 即为该输出多边形的顶点表. 基本操作步骤为:

(1) 过程始于可见多边折线表 VPL 的第一条多边折线. 如果 VPL 和可见线段表 VSL 为空, 则过程结束.

(2) 从多边折线的第一点到最后一点, 按顺序把各结点加入到 OPVL 中, 并从 VPL 中删去该多边折线.

(3) 如果多边形已封闭, 输出顶点表并转(1).

(4) 在 VSL 中确定一个点对, 其中一点的坐标与当前 OPVL 的最后顶点坐标重合, 将另一点加入OPVL 中, 并将该点对从 VSL 中删去. 如果点对的第2点与裁剪窗口的角点重合, 重复(4).

(5) 如果多边形已封闭, 输出顶点表并转(1).

(6) 从VPL 中选取一条多边折线, 其第一或最后结点坐标与当前OPVL 的最后顶点坐标重合. 如果 OPVL 的最后顶点与多边折线的第一结点重合, 则按第一结点到最后结点顺序把多边折成中各结点加入 OPVL 中, 否则按相反顺序把多边折线各结点加入 OPVL 中, 从 VPL 中删去该多边折线并转(3).

本算法在486微机上用 TURBO PASCAL 实现, 能够正确地实行任意多边形的裁剪. 测试实例因篇幅所限从略.

参 考 文 献

1 罗杰斯 D F. 计算机图形学的算法基础. 梁友栋等译. 北京: 科学出版社, 1987, 119～210

2 金延赞. 计算机图形学. 杭州: 浙江大学出版社, 1988, 215～256

3 刘乃琦, 顾书吉, 徐朝寅. 计算机图形技术基础. 成都: 电子科技大学出版社, 1994. 108～121

4 Mathew A J. Polygonal clipping of polylines. Computer Graphics Forum, 1987, 4(4): 407～414

5 Andreev R D. Algorithm for clipping arbi-trary polygons. Computer Graphics Forum. 1989, (8) 183～191

Clipping of Arbitrary Polygonal Region

Fan Huilin① Yang Xing②

(① Dept. of Computer Science; ② Dept. of Election. Eng., Huaqiao Univ., 362011, Quanzhou)

Abstract A discussion is devoted to the clipping technology of arbitrary polygonal region. An algorithm for arbitrary polygonal clipping is given and it is implemented by TURBO PASCAL.

Keywords polygonal region, polygonal line, clipping, window, algorithm