文章编号: 1001 - 9081(2005)12Z - 0421 - 03

用VC++实现的任意多边形裁剪算法

李海姣,张维锦
(华东交通大学 土木建筑学院,江西 南昌 330013)
(haijiao_li@163.com)

摘 要:提出了一个用VC++语言实现的凸多边形、凹多边形,也可以是带内环的多边形的裁剪算法,可以求上述多边形的"交"、"并"以及"差"。首先,该算法使用VC++支持的CObList类和CArray类的对象存储数据,具有占用内存空间少及处理速度快的特点;再通过算法和数据结构的设计不仅使得多边形顶点可按顺时针方向或逆时针方向输入,而且减少了求解过程中对多边形顶点数据的遍历次数;基于判断和计算交点是裁剪算法的主要工作,文中引入了求交前的预处理,避免了大量不必要的求交,降低了算法的时间复杂度。最为重要的是该算法不需要对两多边形的边重合或两多边形在顶点处相交的情况作特殊处理。

关键词:多边形裁剪;VC++数据结构;凸多边形;凹多边形;交点计算预处理
中图分类号: TP391.41 **文献标识码**: A

0 引言

在图形系统中,二维裁剪是最为基础、最为常用的操作之一,其典型的应用是在图形的消隐处理等各种三维图形的处理以及各种排料算法等求交操作之中。如图形消陷、缩放、模式识别、导线和元件布局、线性规划[2]以及土木工程计算机辅助设计中梁与柱、墙与板、墙与柱特别是各类型基础工程量的计算等领域中都要广泛应用到多边形的裁剪。对裁剪算法的研究主要集中在裁剪直线和裁剪多边形两方面。在实用中,多边形裁剪与线剪裁相比具有更高的使用频率,因此它是目前裁剪研究的主要课题[1]。多边形愈复杂其裁剪算法就愈难以实现。文献[1]中综述了现有的解决方案,但这些方案或者局限于某一类多边形,或者时间复杂性和空间复杂性都较高。

本文对两个任意多边形相交时可能出现的位置关系做了严格的定义,用VC++语言实现了两个任意多边形的交、并和差算法。其中的裁剪多边形和被裁剪多边形都可以是一般多边形,既可以是凸多边形、凹多边形,也可以是带内环的多边形。本文针对多边形裁剪中相交的特点,做了多边形求交前的预处理,从而避免了不必要的求交,不仅减少了不必要的求交次数,而且将复杂的乘除法运算用简单的加减法代替,进一步提高了算法的运行速度。最为重要的是该算法不需要对两多边形的边重合或两多边形在顶点处相交的情况作特殊处理,这一点远远优于文献[1]中所述的算法,而且本文算法同样适用于有内环的多边形。算法最终通过简单的遍历多边形交线指针数组,就可以得到每一个结果多边形。

1 算法的数据结构

⑴多边形裁剪

多边形裁剪是指用于裁剪掉被裁剪多边形(又称为实体多边形)位于窗口(又称为裁剪多边形)之外的部分。

⑵相交的定义

如图1(a)、(b)所示,视为线段AB与线段CD不相交;如图1(c)所示,视为线段AB与线段CD相交,且交线为CD。

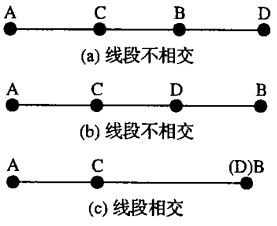

图1 线段相交示意

多边形裁剪算法需要一个适当的数据结构来存储多边形顶点及交线数组,并能够在其上进行正确的操作。在Weiler的算法中输入多边形组成一个树形结构。Greiner-Hormann算法采用双向链表结构,每个多边形由一个双向链表来表示。每找到一个交点就将其分别插入到实体多边形和裁剪多边形的两个双向链表中。Greiner-Hormann算法使用了线性链表与Weiler算法的树形结构相比降低了数据结构的复杂性。

本文算法用VC++语言编程实现。利用MFC类的CobList类以及CArray类的对象(动态数组),可以方便地实现多边形顶点及交线的数据存储,同时MFC类封装了对上述动态数组中数据元素的遍历、获取、插入、删除。

本文算法的每个多边形由一个动态数组来表示,数组按多边形顶点的输入顺序存储其顶点数据,数组的每一个元素对应于多边形的一个顶点。数组的最后一个元素与第1个元素相同以构成一个封闭的多边形,而每两个相邻的元素的一个组合对应于多边形的一条边,而有一个共同元素的一对组合对应于多边形的两条相邻边,公共元素是两条相邻边的相交顶点。

多边形顶点数组中每一个元素的数据结构定义如下:

```
typedef struct
{
  float x;                          //结点的X坐标分量
  float y;                          //结点的Y坐标分量
}
Vertex;
```

如图2所示多边形 *S*和 *C*的顶点数组分别为:

收稿日期: 2005 - 03 - 25;**修订日期**: 2005 - 05 - 26
作者简介:李海姣(1977 -),女,河南郑州人,助理工程师,硕士研究生,主要研究方向:三维CAD、图形算量.

```
CArray < Vertex, Vertex > S, C;
                    //声明多边形 S、多边形 C的顶点数组
S = { s1, s2, s3, s4, s1 };
C = { c1, c2, c3, c4, c5, c6, c1};
```

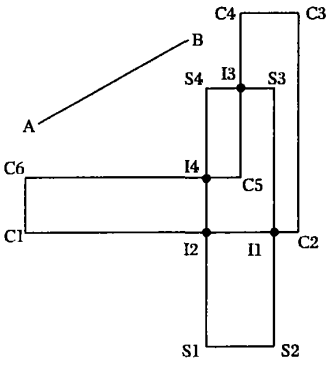

图 2 包围盒示例

如上所述,多边形 S和 C的数组中每两个相邻元素的一个组合表示相应多边形的一条边 $s1s2$,表示多边形 S的边 $s1s2$。而有一个共同元素的两个组合表示两条相邻边如 $s1s2$、$s2s3$表示多边形 S中相交于顶点 $s2$的两条邻边。

交点的数据结构如下:

```
typedef struct
{
  float x;
  float y;
  float z;
  MyPoint Star;          //该交点所在的另一多边形的边的起点
  MyPoint End;           //该交点所在的另一多边形的边的终点
}
MyJD;
```

定义交点数组为:

```
CArray <MyJD,MyJD > m_JD;
```

定义临时数组 m_tempP:

```
CArray <MyPoint,MyPoint > m_tempP;
```

交线的指针数组定义如下:

```
CTypedPtrList < CObList, CDrawBase * > CoLineList;
//存储指向每一条交线对象的指针, CDrawBase为绘图类的基类
```

3 算法描述

算法分为 4个步骤,为表述方便设多边形分别为 S和 C。

第 1步:求包围盒,避免不必要的求交运算。

1) 如果两多边形的包围盒不相交,则这两个多边形一定不相交,即它们的交集为空,并集为这两个多边形。

2) 若这两个多边形的包围盒相交,则求多边形 S的包围盒并剔除多边形 C中不在该包围盒内部的边,剔除后记为 C';再用多边形 C的包围盒(记为包围盒 2)剔除多边形 S不在包围盒 2中的边,剔除后记为 S'。

第 2步:对 S的各边循环,将各边与 C的各边求交,对 S'的任意一边 S_iS_{i+1}其具体操作如下:

1) 清空临时数组 m_tempP;

2) 调用库函数 $PointRgn()$测试边 S_iS_{i+1}的起点 S_i是否位于多边形 C内:若位于 C内,则将点 S_i加入临时数组 $m_tempP(CArray < MyPoint,MyPoint > m_tempP)$中;

3) 依次取出 C的两个相邻顶点,即 C的边 C_iC_{i+1},进行求交前的预处理。

4) 在预处理后,若判定两边相交(线段与线段交点不是虚交点),则调用子函数求交点,并分别将求得的交点加入到临时数组 m_tempP和交点数组 m_JD中。直至遍历了 C中的所有边;

5) 调用子函数对临时数组 m_tempP中的各点按边 S_iS_{i+1}的方向排序,并依次判断相邻两个交点的中点是否位于多边形 C内:若是,则将以这两个交点为起点和终点的边加入交线指针数组 $CoLineList$中;

至此,两多边形的所有交点和多边形 S在多边形 C内的部分都已经求出。

第 3步:组织交点,构造多边形 C在多边形 S内的边。具体操作是对 C内的边循环,对每一边 C_iC_{i+1}:

1) 清空临时数组 m_tempP;

2) 若 C_i位于多边形 S内,则将其加入临时数组 m_tempP;

3) 判断 m_JD中的所有各点的 $Start$和 End是否与 C_iC_{i+1}的起点和终点相同,若相同,则将其加入临时数组 m_tempP中;

4) 若 C_{i+1}位于多边形 S内,则将其加入临时数组 m_tempP;

5) 调用子函数对临时数组 m_tempP中的各点按边 C_iC_{i+1}的方向排序,并依次判断相邻两个交点的中点是否位于多边形 C内:若是,则将以这两个交点为起点和终点的边加入交线指针数组 $CoLineList$中;

第 4步:输出结果多边形。对交线指针数组循环,直至所有的线段均被输出。

当多边形是有内环的多边形时,仅需将带内环的多边的内环与外环的顶点分别存储在两个 CArray动态数组中,并使得内外环的顶点的输入顺序相反,即可采用上述算法将内环、外环多边形分别与被另一多边形求交。

在上述算法中将有多次用到的操作都做成了子函数以提高代码的复用率。将以上算法稍做修改即可用于多边形求并以及求差。

4 交点的判断与计算前的预处理

由上述分析可知,多边形裁剪的交点判断和计算就是用一多边形的每一条边与另一多边形求交点,可见求交计算是裁剪算法最核心的一部分,它的准确性与效率直接影响裁剪的可靠性与效率。为提高求交的效率本文在求交前采用如下方法做预处理:

⑴运用包围盒进行预处理

首先求出多边形 S和多边形 C的包围盒。在此需要做两步判断:

1) 若两多边形的包围盒不相交,则两多边形一定不相交,从而可得其交集为空,其并集为两个多边形的所有边。

2) 若两多边形的包围盒相交,则进行求解计算。用多边形 S的包围盒过滤多边形 C位于包围盒外的边以减少不必要的求交和判断。如图 2所示用多边形 S的包围盒可剔除多边形 C的边 $C6C1$、$C2C3$和 $C3C4$,接着用多边形 C剩余边的包围盒剔除多边形 S不在该包围盒内的边。从而在计算过程中仅需要判断或者求解多边形 S与多边形 C剩余边的交点,从而可以大大减少计算量。又因为做一次乘法运算远比加减法(比较运算)耗时,所以采用这一预处理大大减少了算法的时间复杂度。

设多边形的包围盒的四个角点的坐标为:$S1\ (X_{left}, Y_{bottom})$ $S2\ (X_{right}, Y_{bottom})$ $S3\ (X_{right}Y_{top})$ 和 $S4\ (X_{left}Y_{top})$ 被裁剪线

段 $Q1Q2$的坐标为:$Q1(x1, y1)$ 和 $Q2(x2, y2)$。只有当满足下列条件时:

$$MAX(x1, x2) \leq X_{left}$$

或 $$MIN(x1, x2) \geq X_{right}$$

或 $$MAX(y1, y2) \leq Y_{bottom}$$

或 $$MIN(y1, y2) \geq Y_{top}$$

才能判断线段 $Q1Q2$在包围盒外面可以直接舍弃否则不能判断还需另外处理。如图 2中的线段 AB还需要进一步判断。

⑵用两个判定条件做预处理

实际上,大多数情况下虽然线段 $Q1Q2$与多边形有交点,但线段 $Q1Q2$与多边形的多数边肯定是不相交的。如果能用比线段求交更简单的方法排除这类不相交的线段,则可以大大减少计算量。为此提出如下两个判定条件。

设被裁剪线段 $Q1Q2$的直线标准方程为:

$$F(x, y) = Ax + By + C = 0 \quad (1)$$

其中:$A = y2 - y1, B = x1 - x2, C = -(A \times x1 + B \times y1)$。

对于直线段 $Q1Q2$上的点 $F(x, y) = 0$;对于直线段上方的点 $F(x, y) > O$;而对于直线段下方的点 $F(x, y) < O$。

根据式 (1) 设 $F(x, y) = Ax + By + C$。由图 3可看出当多边形某条边 $P_iP_{i+1}(i = 0, l, \cdots, n)$ 的两个端点在线段 $Q1Q2$的同一侧时,线段 $Q1Q2$与多边形的边 P_iP_{i+1} 不相交。把符合这种情况的多边形的边 P_iP_{i+1} 的两个端点坐标分别代入 $F(x, y) = Ax + By + C$则有

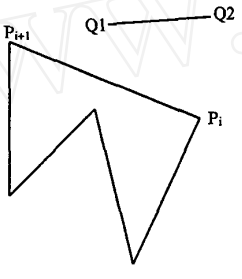

图 3　$Q1$、$Q2$位于边 P_iP_{i+1}的同侧

$$F(P_{i\,x}, P_{i\,y}) \cdot F(P_{(i+1),\,x}, P_{(i+1),\,y}) > 0 \quad (2)$$

为了避免进行乘法运算,上述不等式的判断可转化为:

$$SIGN(F_i) - SIGN(F_{i+1}) = 0,$$

$$SIGN(F_i) \neq 0 \text{且} SIGN(F_{i+1} \neq 0) \quad (3)$$

其中函数 $SIGN(x)$ 表示对 x符号的测试:

$$SIGN(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases} \quad (4)$$

特殊情况:当 $SIGN(F_i) = 0$或 $SIGN(F_{i+1}) = 0$时,即 $F(P_{i\,x}, P_{i\,y}) = 0$或 $F(P_{(i+1),\,x}, P_{(i+1),\,y}) = 0$时,则点 P_i或 P_{i+1} 在直线段 $Q1Q2$或其延长线上如图 4所示。这时需要根据具体情况分别处理。

①若 $F(Q_{1,\,x}Q_{1,\,y}) = 0$且点 $Q2$位于多边形 S的内部则将点 $Q1$和 $Q2$同时加入交点数组如图 4中 (c) 所示。

②若 $F(Q_{1,\,x}Q_{1,\,y}) = 0$且点 $Q2$位于多边形 S的外部则点 $Q1$不作为交点如图 4(d) 所示。

③对图 4中 (a)、(b) 所示情况按前述定义处理。在 (a) 中,若 $Q2$为有效交点必然会被包含 $Q1Q2$的多边形中与 $Q1Q2$相邻的边与边 P_iP_{i+1} 在求交时引入,为避免交点重复故此处不将其作为交点。这样处理使得多边形有重合边时也能作为一般情况来处理,从而避免了文献 1中的繁琐处理。

判断条件 1:

若线段 $Q1Q2$ 按顺时针逐边裁剪窗口多边形 $P_0P_1P_2\cdots P_{n-1}P_n$。对于多边形的边 P_iP_{i+1},若满足式 (4) 时,则可判断直线段 $Q1Q2$与多边形边 P_iP_{i+1}无有效交点,不用再求交,如图 3所示。从而直接判断多边形的下一条边界。

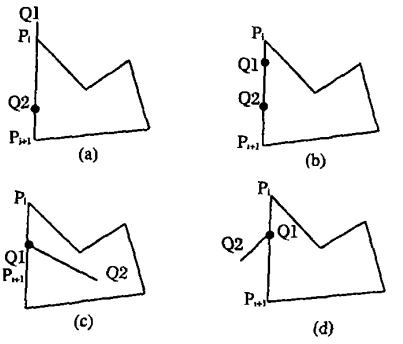

图 4　特殊情况

又设多边形边 $P_iP_{i+1}(i = 0, 1, \cdots, n)$的直线标准方程为:

$$G_i(x, y) = A_ix + B_iy + C = 0 \quad (5)$$

其中 $A_i = P_{i\,y} - P_{(i+1),\,y}, B_i = P_{(i+1),\,x} - P_{i\,x}, C_i = -(A_iP_{i\,x} + B_iP_{i\,y})$。

同理,对于多边形边 P_iP_{i+1}上的点有 $G_i(x, y) = 0$;对于 P_iP_{i+1}上方的点则有 $G_i(x, y) > 0$;而对于 P_iP_{i+1}下方的点则有 $G_i(x, y) < 0$。当 $G_i(x1, y1) \cdot G_{(i+1)}(x2, y2) > 0$时,即

$$SIGN(G_i) - SIGN(G_{(i+1)}) = 0 \quad (6)$$

表示线段 $Q1Q2$的两端点在多边形边 P_iP_{i+1}的同一侧,说明线段 $Q1Q2$与多边形边 P_iP_{i+1}没有有效交点,如图 3所示。

判断条件 2:

若线段 $Q1Q2$按顺时针逐边裁剪窗口多边形 $P_0P_1P_2\cdots P_{n-1}, P_n$ 对于多边形的边 P_iP_{i+1} 首先用判断条件 1来判断,若不满足式 (3),则继续用式 (6) 来判断。若满足式 (6) 则也可判断直线段 $Q1Q2$与多边形边 P_iP_{i+1}没有有效交点,从而直接跳出,进入多边形下一条边的测试。若两个判断条件都不满足则要进行求交运算。

5　结语

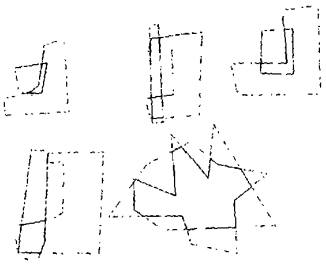
图 5　任意多边形求交举例

本文用VC++及其所支持的 MFC类的特性用动态数组实现了对多边形顶点、交点以及交线的数据存储,大大降低了算法的空间复杂性。算法的改进减少了裁剪过程中的对多边形顶点数据的遍历次数;同时通过求交前的预处理用简单的判断减少了复杂求交的次数,从而进一步降低了算法的时间复杂性。应用本算法成功实现了三维 CAD及图形算量软件中的各类扣减,多边形求交实例见图 5。其中虚线边界为两多边形的并,实线边界为两多边形的交。

参考文献:

[1] 刘勇奎, 高云, 黄有群. 一个有效的多边形裁剪算法 [J]. 软件学报. 2003, 14(4): 845 - 856.

[2] 周培德. 计算几何 [M]. 北京:清华大学出版社, 2000. 133 - 153.