平面简单多边形的布尔运算

计研 12　董未名（014894）、张松海（014888）

【问题背景】

平面多边形的布尔运算是几何和实体造型中的几个基本问题之一。当前有许多方法可以计算多边形的交，但是大部分算法都没有一个正规的数学模型，以至于绝大多数只能对凸多边形进行操作，而很少有比较好的方法可以对凹多边形进行计算。然而，在实际的应用中，特别是在计算机图形学领域，任意多边形的交、并、差的运算是很重要的。例如，在进行平面消隐的操作时，它的基础就是二维平面多边形的布尔运算。因此，设计一个稳定而有效的多边形布尔运算的算法是必要的。在本次实验中，我们采用一种比较新颖的方法来对任意平面多边形（有洞和无洞）进行布尔运算的操作，算法以一种对几何模型进行数学形式化的算法为基础，可以避免去考虑布尔运算中各种各样繁琐的特殊情况，从而使系统有着较高的效率和较强的鲁棒性。

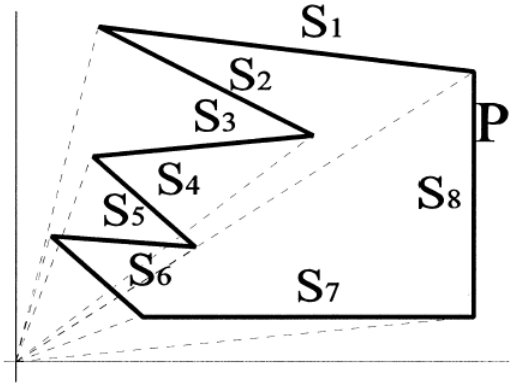
【算法及原理】

这里只对二维的情况进行说明，对于三维或者更高维的情况是一样的。

(1) 多边形表示

设 P 为一简单多边形，它的 n 条边 e1，e2…en 逆序排列，P 可以用如下的三角形链表示：$P = P_\lambda, where\ \lambda = \sum_{i=0}^{n} \alpha_i S_i$，其中 $\alpha_i \in \{1,-1\}$，S_i 为原点与边 ei 组成的三角形。

这里要说明一点：多边形所有定点必须落在第一象限。我们定义：$f_\lambda : R^d \to Z, f_\lambda(Q) = \sum_{Q \in S_i} \alpha_i$。对于 P 内部任意一点 Q，有 $f_\lambda(Q) = 1$，对于多边形外部一点 $Q^{'}$，有 $f_\lambda(Q^{'}) = 0$。



λ= S_1-S_2+S_3-S_4+S_5-S_6+S_7+S_8

Layer 1: S_1,S_8 **Layer 3:** S_3　**Layer 5:** S_5
Layer 2: S_2　**Layer 4:** S_4　**Layer 6:** S_6, S_7

Fig. 1. Representation of a polygon through its simplices.

(2) 多边形求交

设 P1，P2 为简单多边形，$\lambda_1=\sum_{i=0}^{n}\alpha_i S_i$，$\lambda_2=\sum_{j=0}^{m}b_j S_j$ 为对应的三角形链，则

$P_1\cap P_2$ 对应的三角形链为 $\lambda_{P_1\cap P_2}=\sum_{i=0}^{n}\sum_{j=0}^{m}\alpha_i b_j Tri(S_i\cap U_j)$，其中 $Tri(S_i\cap U_j)$ 为

$S_i\cap U_j$ 的结果再分解成三角片的集合。

(3) 求交化简

给定两个三角片，$S=OQ_1Q_2$ 和 $U=OQ_1'Q_2'$，如果 S=U 或者存在 $Q_1'Q_2'$ 上一点 Q（边界出外），Q 在 S 的内部，则 U 属于 S（$U\lhd S$）。

设 P1，P2 为简单多边形，$\lambda_1=\sum_{i=0}^{n}\alpha_i S_i$，$\lambda_2=\sum_{j=0}^{m}b_j S_j$ 为对应的三角形链，如果两多边形的边界重合或者仅相交于端点。则 $P_1\cap P_2$ 对应的三角形链为

$$\lambda_{P_1\cap P_2}=\sum_{i=0}^{n}\sum_{j=0}^{m}\alpha_i b_j Tri(S_i\cap U_j)=\sum_{i=0}^{n}\sum_{j=0}^{m}c_k R_k$$

where

$$c_kR_k=\begin{cases}\dfrac{a_ib_j}{\mathrm{card}(H_c(S_i;P_2))} & if\ S_i\lhd U_j,\ where\ \ c=\mathrm{level}(U_j),\\ \dfrac{a_ib_j}{\mathrm{card}(H_c(S_i;P_2))} & if\ U_j\lhd S_i,\ where\ \ c=\mathrm{level}(S_i)\\ 0 & if\ there\ does\ not\ exist\ a\ subordination\ relationship\ for\ S_i,U_j\end{cases}$$

(4) 多边形求并、求差

设 P1，P2 为简单多边形，$\lambda_1=\sum_{i=0}^{n}\alpha_i S_i$，$\lambda_2=\sum_{j=0}^{m}b_j S_j$ 为对应的三角形链，$\lambda_{P_1\cap P_2}$ 为

两多边形交 $P_1\cap P_2$ 所对应三角形链。$\lambda_{P_1\cap P_2}=\sum_{k=1}^{l}c_kR_k$。则 $P_1\cup P_2$ 对应的三角形链为

$\lambda_{P_1\cup P_2}=\sum_{i=1}^{n}a_iS_i+\sum_{j=1}^{m}b_jU_j-\sum_{k=1}^{l}c_kR_k$。

两多边形差 P_1-P_2 对应的三角形链为 $\lambda_{P_1-P_2}=\sum_{i=1}^{n}a_iS_i-\sum_{k=1}^{l}c_kR_k$

(5) 算法设计

```
procedure CalculationIntersection(pol1,pol2,pol3)
Begin
  (1).    CalEdgeIntersection (pol1,pol2)
  (2).    CalculationLevel (pol1)
          CalculationLevel (pol2)
  (3).    Intersection (pol1,pol2,coefficient)
  (4).    TraverseCoeff (coefficient,pol1,pol3)
          TraverseCoeff (coefficient,pol2,pol3)
    end{CalculationIntersection}

procedure Intersection (pol1,pol2,coefficient)
begin
  for each Si of pol1
    for each Uj of pol2
      if (subordinated (Uj,Si)==true) and level(Si)∉(LevelsDom(Uj)) then
        coefficient (Uj) ← coefficient (Uj)+ ai*bj
        InsertLevel (level(Si), LevelsDom(Uj))
      else if (subordinated (Si,Uj)==true) and level(Uj)∉(LevelsDom(Si)) then
        coefficient (Si) ← coefficient (Si) + ai*bj
        InsertLevel (level(Uj), LevelsDom(Si))
      end {if-else}
    end {for}
  end{for}
end {Intersection}

procedure TraverseCoeff (coefficient,polBegin,polEnd)
    begin
      for each Si of polBegin
        if (coefficient (Si)= +1) or (coefficient (Si)= -1) then
          insert (coefficient (Si),Si,polEnd)
        end-if
      end {for}
    end {TraverseCoeff}

procedure UnionCalculation (pol1,pol2,pol3)
Begin
  (1). CalPointsIntersection (pol1,pol2)
  (2). CalculationLevel (pol1)
       CalculationLevel (pol2)
  (3). Intersection (pol1,pol2,coefficient)
  (4). SumCoeffU (coefficient,pol1)
       SumCoeffU (coefficient,pol2)
       TraverseCoeff (coefficient,pol1,pol3)
       TraverseCoeff (coefficient,pol2,pol3)
end{UnionCalculation}
```

```
procedure SumCoeffU (coefficient, polBegin)
Begin
   for each Si of polBegin
      coefficient (Si) ← ai - coefficient (Si)
   end
end {SumCoeffU}

procedure DifferenceCalculation (pol1,pol2,pol3)
Begin
   (1). CalPointsIntersection (pol1,pol2)
   (2). CalculationLevel (pol1)
        CalculationLevel (pol2)
   (3). Intersection (pol1,pol2,coefficient)
   (4). SumCoeffD (coefficient,pol1,pol2)
        TraverseCoeff (coefficient,pol1,pol3)
        TraverseCoeff (coefficient,pol2,pol3)
end{DifferenceCalculation}

procedure SumCoeffD (coefficient,pol1,pol2)
Begin
   for each Si of pol1
      coefficient (Si) ← ai - coefficient (Si)
   end
   for each Uj of pol2
      coefficient (Uj) ← - coefficient (Uj)
   end
end {SumCoeffD}
```

【系统设计】

系统采用 Microsoft Visual C++ 6.0 进行设计和开发，全部程序在 Microsoft Windows XP Professional、AMD Athlon™ processor 1.39 GHz、256MB RAM 下调试通过。

程序采用单文档结构，使用黑色背景进行绘图，主界面如下图所示。坐标原点在视图的左下角，x 轴和 y 轴的方向分别为向右和向上。多边形（组）1 和多边形（组）2 分两次进

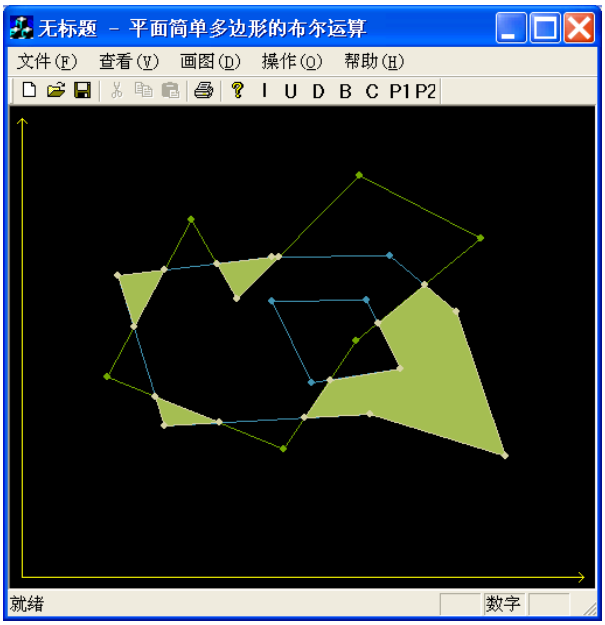
行输入。由于算法的需要，多边形的外环输入时要按照逆时针方向，内环则须按照顺时针方向，且不能自交。然后可利用菜单或工具栏中的快捷键进行交、并、差的操作，下图所示是对两个多边形进行求差操作的结果。

为了简单起见，程序使用 struct 对变量进行结构化的封装，数据结构如下所示：

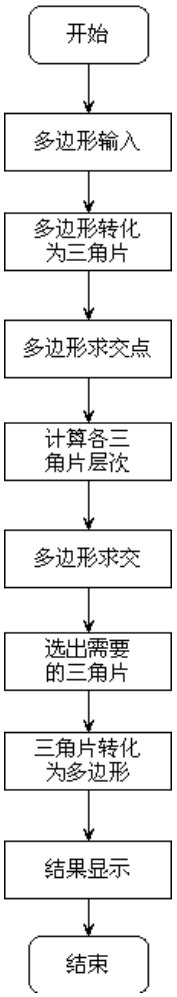
```
struct Point2D //多边形点的数据结构
{
    CPoint p;
    bool IsVertex;//标志该点是顶点还是交点
};

struct trian     //三角形片的数据结构
{
    Point2D p1,p2;//三角形对应多边形边的两个顶点(start,end)
    int coef;//三角形片的参数(1,0,-1)
    int level;//三角形片的 level
    int num1,num2;//三角形片的标号
};

struct loop     //多边形环的数据结构
{
    CArray<Point2D,Point2D&> points;//存储环上各顶点
    bool IsHole;//标志环是否为洞
};
```



如上所示，多边形用环进行表示，其中的每一条边与三角形片一一对应。结果仍采用多边形进行表示，由若干个环构成。程序执行过程与算法相对应，以求交为例，流程图如下所示。

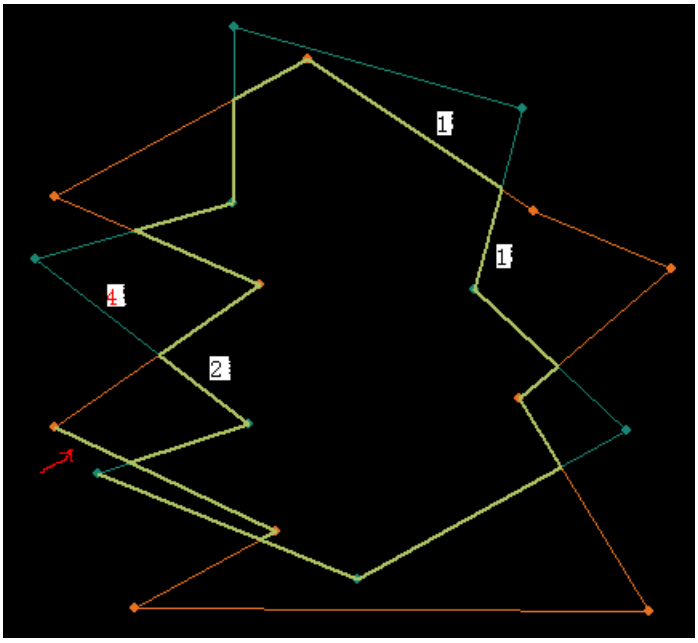


【测试、对比结果】

任意简单多边形求交的时间复杂度为$n^2 \log n$，这篇文章提供的算法的时间复杂度为n^2，最好为$n \log n$。

【遇到的问题】

在实验过程中，我们发现文章的一个漏洞，使用当前算法实现在某种情况下会出现错误。如图



查看箭头所示线段，该线段从属于图中标明层次的线段，按文章提出的算法，它的参数为－1。这是错误的。我们曾经考虑先求 level，再做多边形求交，但是结果也是错误的。经过老师提醒，现在已经有一个修改的思路，等到 20 号计算机视觉实验检查完毕就来考虑这个问题。

【没有解决的问题】

由于时间仓促，多边形输入时判断其简单性和多边形边的顺序调整还没有做。多边形简单性可以通过判断新输入的边和已经输入的边进行求交来判断。多边形边的顺序可以通过其包围盒的边的顺序判断。

【代码来源】

由于系统用到的算法涉及的数据结构和功能比较简单，所以其中用到的算法全是由我们自己编写。

【有关文献】

[1] M. Rivero, F.R. Feito. Boolean operations on general planar polygons. Computers & Graphics 24 (2000) 881-896

[2] F. R. Feito, M. Rivero, A. Rueda. Boolean representation of general planar polygons.

[3]周培德，《计算几何——算法分析与设计》，清华大学出版社，2000 年 3 月