多边形布尔运算平台

开发技术文档

2014213501 贾晨

2014 年 12 月 18 日

一、 程序框架结构

（一） 模块划分

程序整体分为如下三个主要的模块：

- CP_Polygon(.cpp/.h)：定义多边形数据结构并初始化，实现对多边形的一些基本操作，例如移动元素、添加元素、显示方式等；
- CP_PolygonPlatformDoc(.cpp/.h)：主要用来完成保存、打开 txt 文档等功能；
- CP_PolygonPlatformView(.cpp/.h)：实现程序界面的显示、程序功能的内部代码实现等。

（二） 程序设计整体思路

1. 数据结构

   程序中定义了如下几种数据结构

```
/*点类*/
class CP_Point { ... }
typedef vector<CP_Point> VT_PointArray;//点集合

class CP_Polygon;//先声明，其他类里直接用

typedef vector<int> VT_IntArray;//整数向量（点序号集合）
typedef vector<VT_IntArray> VT_IntArray2; //点集的向量（点序号集合的集合）

/*定义一个线段的结构*/
struct Segment { ... };

/*环类*/
class CP_Loop { ... }
typedef vector<CP_Loop> VT_LoopArray;//多边形向量

/*区域类*/
class CP_Region { ... }
typedef vector<CP_Region> VT_RegionArray;

/*多边形类*/
class CP_Polygon { ... }
```

2. 函数设计

   1) 平台绘制函数

      最初搭建的平台中就带有完整的绘制函数，故此处便不再赘述。

2) 功能实现函数

实现布尔运算平台涉及到的所有函数如下所示：

```
/*计算线段长度的平方*/
double seglength(CP_Point p1, CP_Point p2){ ... }

/*计算交叉乘积(P1-P0)x(P2-P0)*/
double xmult(CP_Point p1, CP_Point p2, CP_Point p0){ ... }

/*判点是否在线段上,包括端点*/
int dot_online_in(CP_Point p, CP_Point l1, CP_Point l2){ ... }

/*判两点在线段同侧,点在线段上返回0*/
int same_side(CP_Point p1, CP_Point p2, CP_Point l1, CP_Point l2){ ... }

/*判两直线平行*/
int parallel(CP_Point u1, CP_Point u2, CP_Point v1, CP_Point v2){ ... }

/*判三点共线*/
int dots_inline(CP_Point p1, CP_Point p2, CP_Point p3){ ... }

/*判两线段相交,包括端点和部分重合*/
int intersect_in(CP_Point u1, CP_Point u2, CP_Point v1, CP_Point v2){ ... }

/*计算两线段交点,请判线段是否相交(同时还是要判断是否平行!)*/
CP_Point LineSegInter(CP_Point u1, CP_Point u2, CP_Point v1, CP_Point v2){ ... }

bool between(double v, double u1, double u2){ ... }
/*计算、添加交点，得到tempA和tempB*/
void AddNodes(CCP_PolygonPlatformDoc* pDoc){ ... }

/*判断点p是否在多边形poly内*/
bool PointInPoligon(CP_Point p, CP_Polygon poly){ ... }

/*判断线段u1u2是否在多边形内*/
bool SegInPolygon(CP_Point u1, CP_Point u2, CP_Polygon poly){ ... }

/*升序排序函数*/
bool ascending(const Segment& e1, const Segment& e2){ ... }

/*检查多边形中有没有面积过小的环，若有，为了避免其看起来像是一条边而不是环，将其删除*/
void PolygonExam(){ ... }

/*将边集合构建成多边形*/
void ConstructPolygon(CCP_PolygonPlatformDoc* pDoc){ ... }
```

```
/*显示布尔运算结果*/
void CCP_PolygonPlatformView::OnViewResult(){ ... }

void CCP_PolygonPlatformView::OnUpdateViewResult(CCmdUI *pCmdUI){ ... }

/*并运算*/
void CCP_PolygonPlatformView::OnPolygonUnion(){ ... }

/*交运算*/
void CCP_PolygonPlatformView::OnPolygonIntersection(){ ... }

/*差运算（A-B）*/
void CCP_PolygonPlatformView::OnPolygonAB(){ ... }

/*差运算（B-A）*/
void CCP_PolygonPlatformView::OnPolygonBA(){ ... }

/*合法性检查
  如下几种检查：
  1.外环应为逆时针，内环应为顺时针（手动添加内外环时可能会出错）；
  2.环 必须是简单多边形，即环中任意两条边不能有除端点外的其它顶点；
  3.多边形的一个区域中，内环必须在该区域的外环的内部；
  4.多边形的一个区域中，内环之间不能相互包含；
  5.多边形的一个区域中，内环间交点只能是端点，不能是边或其它；
  6.同一个多边形的不同区域不能相交，若相交，交点只能是端点，不能是边或其它。
    （注意当多边形只有两个外环，且这两个外环相互嵌套的情况！）
*/

/*检查各个环的方向(1)*/
bool LoopDirection(CCP_PolygonPlatformDoc* pDoc){ ... }

/*检查一个多边形中的所有边两两之间有没有除了端点外的其它交点，或者是重合边(2,5)*/
bool EdgeIntersection(CCP_PolygonPlatformDoc* pDoc){ ... }

/*检查多边形同一区域中的内环是否在该区域的外环中
--将该区域的外环作为一个多边形，判断该区域所有内环的任意一条边的中点
  是否在这个多边形内来判断这个内环是不是在外环内；*/
bool InOutLoop(CCP_PolygonPlatformDoc* pDoc){ ... }

/*检查多边形同一个区域中的内环是否存在包含关系
--将内环作为一个单独的多边形（点顺序应该不用考虑，因为只需要判断交点个数），
  判断其它内环的任意一条边的中点是否在这个多边形内来判断他们是否存在包含关系*/
bool InInLoop(CCP_PolygonPlatformDoc* pDoc){ ... }

/*检查同一个多边形中一个区域被另一个区域完全包含的情况
--将一个区域作为一个单独的多边形，
  判断其它区域的任意一条边的中点是否在这个多边形内来判断他们是否存在包含关系*/
bool InInRegion(CCP_PolygonPlatformDoc* pDoc){ ... }

void CCP_PolygonPlatformView::OnCheck(){ ... }
```

二、 用到的一些技巧

（一） 编程技巧

1. 使用宏定义

对于常使用的常数，使用宏定义更加简洁、方便，且能避免出错：

```
#define DOUBLE_PI            6.28318530717958647692
#define PI                   3.14159265358979323846
#define HALF_PI              1.57079632679489661923
#define eps 1e-6//精度范围
#define INFINITY -10000//定义无穷远为-10000，用来产生“伪射线”
```

2. 处理容差

由于坐标是 double 型数据，存储和计算的过程中会出现偏差，所以“判零”或者“判等”时需要考虑容差，使用如下的方式可以方便简洁地处理容差问题：

```
#define zero(x) (((x)>0?(x):-(x))<eps)//判断是不是0,如果是0返回true，不是0返回false
```

3. 功能隔离

将一些常用的简单的，但是编码过程有些复杂的功能封装到一个独立的函数中，使用时直接调用函数，更加直观、简洁、方便。如下图所示函数，实现的功能为：判断 u1<=v<=u2 或 u2<=v<=u1 是否成立：

```
bool between(double v, double u1, double u2)
{
    double min = u1<u2 ? u1 : u2;
    double max = u1>=u2 ? u1 : u2;
    if(v>min && v<max)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

（二） 测试技巧

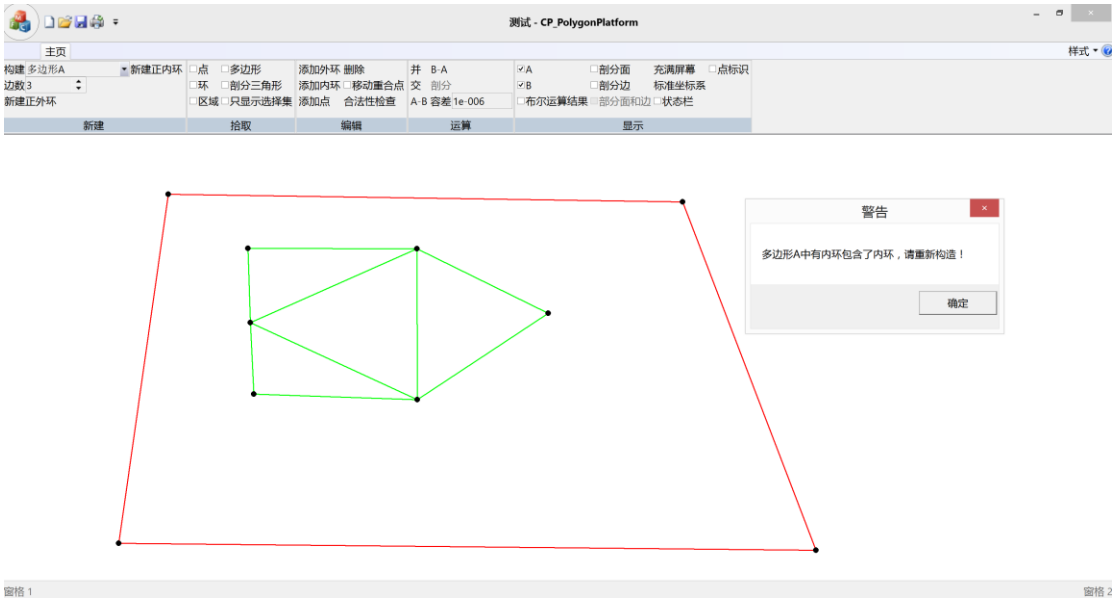
1. 代码重用测试

在代码开发的过程中有些模块功能几乎完全相同，此时应该先写好一个

模块，测试完成保证基本无误后再移至到另一个模块上。同时要注意，重用代码时可能会出现忘记在原有代码上进行修改或修改不全面的情况，而造成错误，所以代码重用时要注重测试这些方面 bug。

2. 边界测试

由于布尔运算过程中会出现一些边界情况，例如边重合、顶点重合等情况，此时可以通过手动调整 txt 文档中的参数来生成一些带有边界情况的用例，如下图所示，为两内环交点为顶点，但是内环相互重合的非法情况：



3. 认真做好测试记录

在做测试时及时记录测试数据，及测试结果。在修改代码后可以重新做一次测试，避免由于修改代码产生其它的不可预知的 bug。

三、 程序编译、配置及运行

此程序使用 MFC 和 C++结合编写，运行环境为 Windows 操作系统下的 VS2012，无需进行其它配置，简单易行。点击工具栏中的“本地 Windows 调试器”或菜单栏中的“调试”->“开始执行（不调试）”即可进行程序的编译运行，得到运行结果。