**Rectangles Combine To polygons**

要求：

1. 输入若干个矩形，输出若干个多边形；
2. 根据图形的位置关系，两个矩形的位置关系分别有相交，相容，重合，相离：

若相交且超过一个公共点，则合并产生一个新的多边形，保证外边界不变；

若相交且只有一个公共点，则设置一个define，设定它的值来选择相交还是相离；

若相容，则留大一点的多边形；

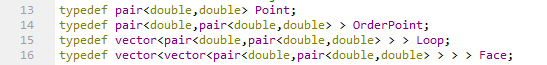
若重合，则看作相容；

若相离，则不处理。

总结：

1. **数据结构**

Polyon.cpp



点用pair表示，两个参数代表点的横坐标和纵坐标。记为point；

带序号的点用pair<double, point>表示，第一个参数表示当前点的序号，第二个参数表示点的坐标。记为orderpoint；

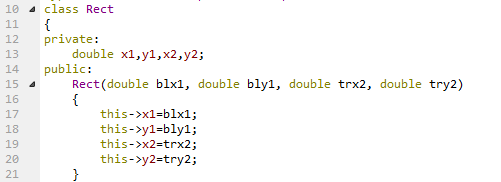
多边形和孔用vector<orderpoint>表示，表示带序号点的集合。记为loop；

所有多边形的集合用vector<loop>表示，记为face；

另：loop中两个相邻的orderpoint，就是一个向量。

1. **Rectangle类**有四个参数double。

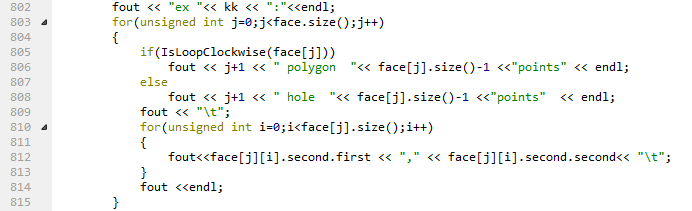
Rect.h



前面两个为矩形左下点横坐标和纵坐标，后面两个double为矩形右上角点的横坐标和纵坐标。

1. **Polygon输出形式**为多个有序点的坐标。

Polygon.cpp

其中顺时针顺序的点序列为polygon，逆时针顺序的点序列为hole；

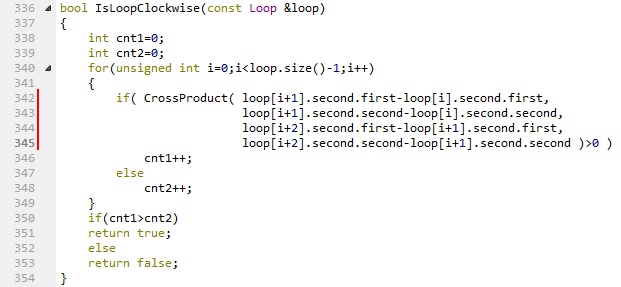
**区分polygon和hole**的定义是，沿着loop的顺序走，右手在外为polygon，右手朝内为hole。

我们根据单个loop的两个连续的向量的外积的正负的个数来确定某个loop是否为polygon或者hole：

正数的个数多于负数的个数，则为polygon；

负数的个数多余正数的个数，则为hole。

Rect.cpp



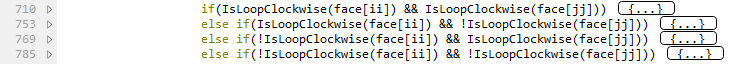
若多边形没有孔，则输出形式为“polygon”；

若多边形有若干个孔，则输出顺序为“…-hole-hole-polygon”；

若孔中还有多边形，则输出顺序为“polygon-hole-polygon”。

这个根据原来loop 的性质（是否为polygon或hole）和形成的新loop来讨论实现：

Polygon.cpp



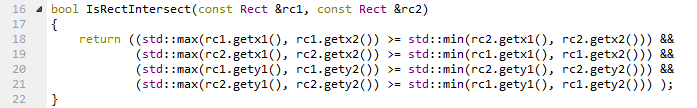
1. 首先我们需要判断两个多边形的位置关系，虽然输入都是矩形，但是一旦有两个相交的矩形合并之后，就是多边形了，再加上矩形本身就是多边形的一种特例，所以我们第一步将所有的矩形转化成Polygon；然后对多边形进行相交测试：

**快速相交测试：**

Polygon.cpp



Rect.cpp



对于两个多边形进行相交测试，原理和矩形的相交测试一样，将两个多边形进行外包，产生外包矩形，对两个外包矩形进行相交测试：

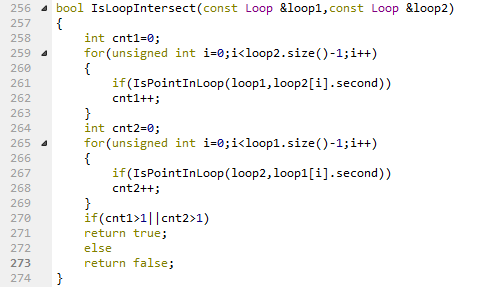
1、相交测试结果为相交的时候，两个多边形不一定为相交；

2、相交测试结果为不相交的话，两个多边形一定不相交。

这个测试能在输入矩形非常多的情况下在程序的的开始就为不相交的多边形省去很多步骤，能够提高算法效率。

1. **相交测试：**

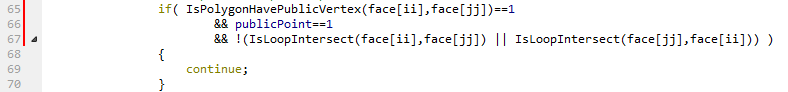
**Rect.cpp**



两个polygon相交的话，一定有一个顶点在另一个polygon内（含边）。条件如下：

1. Po1至少一个顶点在po2内（含边）；
2. Po2至少有一个顶点在po1内（含边）；
3. 排除po1与po2只有一个公共点的情况；（并对这种情况根据define进行分类讨论）；





当define的publicPoint为1的时候，两个只有一个公共点的多边形算作两个多边形；

当define的publicPoint为0的时候，两个只有一个公共点的多边形算作一个多边形。

1. **合并过程：**

Polygon.cpp

106-669

对通过相交测试的polygon进行合并

loop1=“<1,p1>，<2,p2>，<3,p3>，<4,p4>，<5,p1>”

loop2=“<1,q1>，<2,q2>，<3,q3>，<4,q4>，<5,q1>”

两个多边形loop1，loop2相交，所以必定有交点，也就是说loop1中必有一个向量和loop2中的某一向量相交。假设向量p2p3与向量q1q2相交。

**求交点：**

则可以求得交点A，根据A在p2p3和q1q2的几何位置可以求得序号，序号的值在0～1之间，记为m、n，并加上p2的序号和q1的序号，所以m在2～3之间，n在1～2之间；

**取正负：**

根据向量的外积，p2p3和q1q2的外积为正，则q1q2和p2p3的外积为负，反之亦然。为m和n加上对应的正负号，其中负号表示这个点是入点，正号表示出点。

**插入原loop：**

得到：

loop1=“1，2，3，4，5”

loop2=“1，2，3，4，5”

两个多边形相交至少有两个交点，所以我们补全之后得到：

loop1=“1，2，-m1，3，+m2，4，5”

loop2=“1，+n1，2，3，4，-n2，5”

**处理loop：**

现在取+m2作为新的多边形的起点，按顺序往后遍历，当遍历到序号为负数-m1的时候跳出，改为去po2的序列中寻找与-m1的点+n1，然后按着顺序继续往后遍历，当到达序列最后的时候跳转到开头继续遍历，知道遇到序号为负数的点-n2，然后跳出再去po1中寻找，反复循环直到遍历到起点为止。

我们先对po1，po2从首元素进行遍历，如果是正数且是交点，则跳出；若不是则将其放入loop的尾部。

loop1=“+m2，4，5，1，2，-m1，3”

loop2=“+n1，2，3，4，-n2，5，1”

对两个新loop进行提取，取出每一段完整的sub-loop，放到一个vector<Loop> face0中：

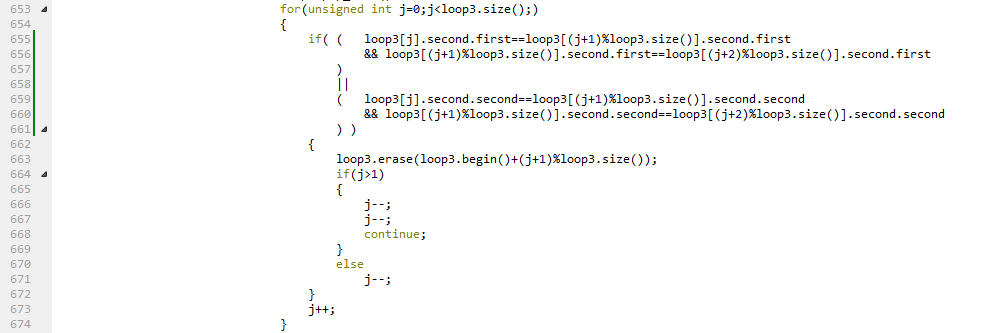
face0=“+m2，4，5，1，2，-m1”，“+n1，2，3，4，-n2”

然后对face0中的所有sub-loop进行拼接，直到遍历face0的所有元素，并将新生成的loop放到vector<Loop> face1中：

Face1=“+m2，4，5，1，2，-m1，+n1，2，3,，-n2”

考虑到共边的情况，新生成的loop的点可能出现三个连续的点出现在同一条线上，我门加一个整理过程：

Polygon.cpp



PS：起点一定要是正号，正号肯定是在新的loop上面。而随机找一个点则不一定。

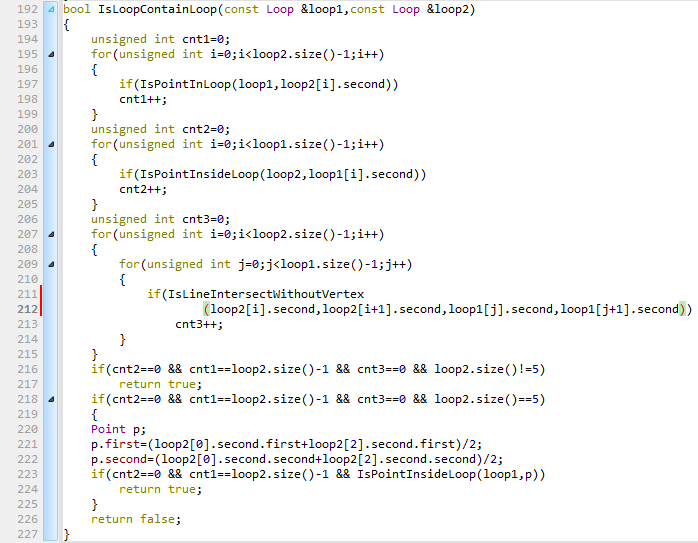
当两个polygon交点较多的时候，会产生多个新loop，这些loop里面只有一个是polygon，其余的都是hole。

当polygon和hole进行合并运算的时候，本质其实是两个polygon的减法运算。

当hole和hole进行合并运算的时候，本质其实是两个polygon的相交运算（求公共部分）。

1. **相容测试：**

Rect.cpp



当polygon1与polygon2相容（包括重合）的时候（po1包含po2），考虑到二者可能是凹多边形，相容测试的原理如下：

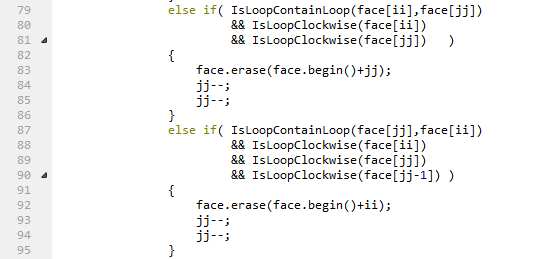
1. po2的所有顶点都在po1内（包括在边上）
2. po2的所有向量与po1的所有向量没有交点（顶点除外）
3. po1没有顶点在po2内部（不包括边上）

由于程序操作的结构，两个会被测试为相容的polygon中必有一个是矩形。

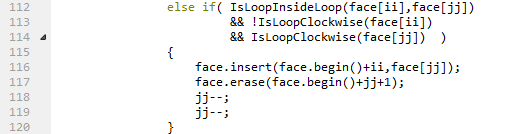
当po2是矩形时，po2内部（不包括边上）中任意一点都在po1内部，这里我们取矩形的中心

当po1是矩形的时候，满足上面的条件1-3即可.

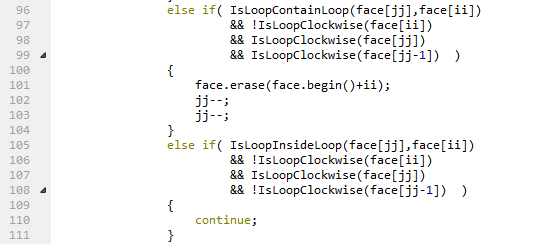
当**polygon包含polygon**的时候，删去被包含的polygon；



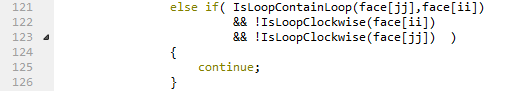
当**hole包含（不重合）polygon**的时候，将polygon放到hole前面去；



当**polygon包含（含重合）hole**的时候，删掉hole；



当**hole包含hole**的时候，不处理。



上述情况还包含几种带孔多边形的规则：

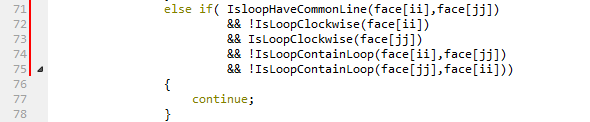
因为当前hole所在的polygon，一定是包含这个hole的，所以将前面一个loop是hole的polygon排除在外；

因为当前polygon后面一个loop是hole的时候，一定是孔内多边形，所以也将这样的polygon排除在外。

1. 当hole与polygon只有公共边、并且分别在公共边两侧的时候，本质上属于相交，但是我们得要的结果是两个保持不变。

所以单列出来，做一个测试，从而提高算法效率。

Polygon.cpp



Hole和polygon只有公共边的测试，要求如下：

1. Hole没有点在polygon内部（不含边）
2. Polygon没有点在polygon内部（不含边）
3. Hole和polygon的交点，都在二者的边上。
4. Hole和polygon不互相包含。

