Decomposing Polygonal Regions into Convex Quadrilaterals

田延军	2004210956	tyj04@mails.tsinghua.edu.cn
戴少伟	2004210961	daisw04@mails.tsinghua.edu.cn

一. 问题背景

多边形区域是平面上的一个闭区域，它由一个边界多边形（bounding polygon）和 0 个或多个洞多边形（hole polygons）组成。多边形区域的顶点包括边界多边形的顶点和洞多边形的顶点。多边形区域的弦（chord）指的是连接多边形的两个顶点，且在多边形区域内部的一条线段。多边形区域的四边形剖分问题是：在多边形区域加入一些互不相交的弦，把多边形区域分为四边形的集合。这又分为简单四边形和凸四边形两种情况。将多边形区域分为相对简单的多边形区域是计算几何中一个非常重要的问题，它的应用包括计算机图形学，模式识别，VLSL 设计等。简单的多边形区域因为含有更少的顶点（四边形或者三角形等），或一些特殊的结构（凸性，单调性，星型等），从而更容易分析问题。

已经证明，判断一个多边形区域是否可以分为凸四边形的问题是 NP-完全的。事实上，有些多边形区域是不能被四边形剖分的，如图 1 所示就是一个例子。但是，有些特殊的多边形区域可以进行四边形剖分，如正交多边形区域。正交多边形是所有边都水平的或垂直的多边形。正交多边形区域是一个边界多边形和洞多边形都正交的多边形区域。[KKK83]证明了正交多边形区域可以被四边形剖分。

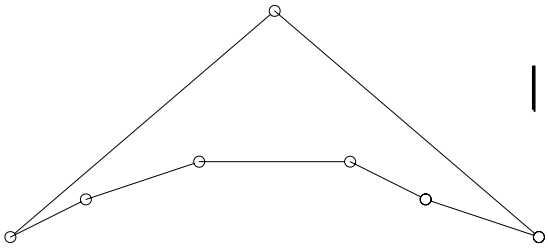


图 1 不能被剖分为凸四边形的多边形区域

首先定义一个多边形区域的一个性质，CQ hereditary。多边形区域满足 CQ hereditary 指的是：多边形区域可以不是凸四边形，但是它有一个可去除的凸四边形(removable quadrilateral)。这个凸四边形的边由原多边形区域的边和弦组成。

从多边形区域去掉这个四边形后得到的仍是一个包含可去除凸四边形的多边形区域。

一类满足 CQ hereditary 的多边形区域是伪正交多边形区域。我们实现的算法就是对伪正交多边形区域进行四边形剖分。所谓伪正交多边形区域是满足如下条件的多边形区域：

1）组成多边形区域的每条边交替为水平边(horizontal)(不妨假设与 X 轴平行)，其它边称之为斜边(tilted)。

2）所有的内角都≤270°。

3）每一条斜边的 shadow 不包含任何多边形的顶点。

斜边的 shadow 定义如下：

shadow：斜边 e 的 shadow 定义为多边形区域内部的一些点的集合，这些点可以通过多边形区域内的一条垂线与 e 连起来，也即从 e 的两个端点向多边形区域内发出两条射线，直到遇到多边形区域的边为止，由斜边 e，两条射线，以及射线与多边形区域相交的一条或者多条边围成的区域（其中不包括从 e 的左端点，以及左端点向下发出射线；e 的右端点，右端点向上发出的射线）。图 2 为 e 的 shadow 的一个例子。

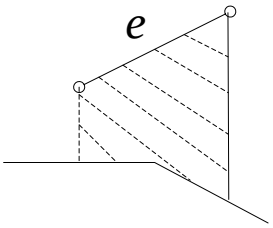


图 2 斜边 e 的 shadow(阴影区域)

二． 算法描述与分析

1. 定义

$<_x$：按 x 坐标从小到大对顶点排序，如果 x 坐标相同则按 y 坐标从小到大排序

$<_y$：按 y 坐标从小到大对顶点排序，如果 y 坐标相同则按 x 坐标从小到大排序

left edge：伪正交区域的一条*斜边*是一个 left edge：如果沿着 y 坐标小的顶点向另外一个顶点走，伪正交区域在右手边。

right edge：伪正交区域的一条*斜边*是一个 right edge：如果沿着 y 坐标小的顶点向另外一个顶点走，伪正交区域在左手边。

顶点 v 的 right neighbor：在 $r >_x v$，且对 v 可见的顶点集合中（如果存在这样的集合），按 $<_x$ 排序，其中最小的一个顶点定义为 v 的 right neighbor。

边 e 的 right most vertex：按 $<_x$ 排序，e 上最大的顶点，即 e 中 x 坐标大的顶点。

2. 伪正交多边形的性质

[定理]：伪正交多边形区域满足 CQ hereditary 条件

证明的过程实际上就是对多边形区域进行四边形剖分的过程。我们不对这个定理进行证明，在算法描述部分，我们将详细介绍剖分过程。

在伪正交多边形区域中找到一个可去除四边形：找到一条 left edge (u,v)，$u <_x v$，v 有一个 right neighbor r，r 在一条 right edge (r,s) 上，从而 (u,v)，(r,s) 组成了一个可去除的四边形 Q。

[引理 1]：伪正交多边形区域的任意一条 left edge (a,b) 的顶点 b 都有一个 right neighbor。

引理 1 说明，只要对 left edge 的右端点按照 $<_x$ 排序，那么序列最右端的 left edge 的右端点的 right neighbor 一定在一条 right edge 上。因此，(u,v)，(r,s) 是存在的。

[引理 2]：Left edge (u,v)，$u <_x v$，v 的 right neighbor r，r 在 right edge (r,s) 上，则 $u <_x v <_x r <_x s$。

[引理 3]：伪正交多边形区域 P，删除了由 (u,v)，(r,s) 组成的一个凸四边形（其中，$(u,v), u <_x v$ 是一条 left edge，v 的 right neighbor r，r 在 right edge (r,s) 上）

后，形成一个或多个互不相交的多边形区域集合P'。则P'中的一条 left edge 的 right most vertex v的 right neighbor 就是v在原多边形区域P中的 right neighbor。

引理 3 说明一个伪正交多边形区域的 right neighbor 在算法执行过程中不会改变，因而我们只要在开始找到每一个顶点的 right neighbor，在剖分的过程中就不需要再重新计算。这个 right neighbor 称为 initial right neighbor。具体地，顶点v的*初始右邻居(initial right neighbor)*定义为：在满足$r >_x v$的顶点集合里，按$<_x$排序最小，与v可见，且不与v组成一条斜边的顶点。顶点的初始右邻居可能不存在。如果存在，必唯一。

3. 算法描述

Step 1：验证多边形区域P满足伪正交性，，对P的每个顶点v，找到它的*初始右邻居*(initial right neighbor)，按照 left edge 最右顶点(right most vertices)的$<_x$顺序建立一个 left edge 的有序表L。

Step 1 是用扫描线实现的。首先对所有顶点以$<_x$排序，一条垂直的扫描线从$x=-\infty$到$x=\infty$顺序扫过，在每个顶点处停留进行处理。垂直的扫描线自底向上与一系列线段相交，算法按照与扫描线相交的顺序将这些线段保存为一个有序链表。设想一条扫描线自底向上穿过多边形区域，与扫描线相交的线段每两条组成一个 pair（称为 in-out pair），扫描线由 pair 的第一条线段进入多边形区域，由 pair 的第二条线段离开多边形区域。算法对每个顶点找到它在有序表中的相对位置，判断属于以下四种情况（如图 3）的哪种，然后做相应的处理。

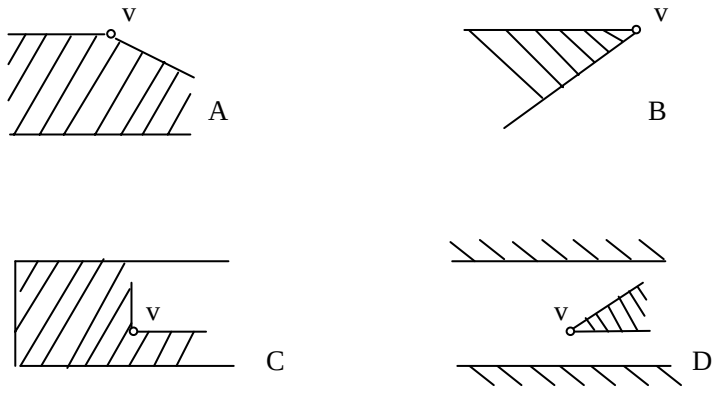


图 3 顶点 v 的四种情况

Step 1 实现算法流程如图 4：

```
Algorithm   ScanLine(P)
Input. 多边形区域P
Output. 每个顶点的initial right neighbor;所有left edge的一个
有序表L(对left edge的right most vertex按照<x排序)
{
 1. Initialize an empty event queue Q; insert all vertices
    to the event queue.
 2. Initialize an empty status structure S
 3. while(Q is not empty)
 4.      do find the next event point p, delete it
 5.         HandleEventVertex(p)
}
```

```
HandleEventVertex(p)
{
1.  Validate whether polygon region is pseudo rect.
2. Find its place relative to the status structure
S.
3. if(p is in case A){
       Replace one edge in S by another
       ProcessWaitingVertex(p);
   }
4.  if(p is in case B){
       Delete two edges in S
       ProcessWaitingVertex(p)
   }
5.  if(p is in case C){
       Add two new edges to S
       ProcessWaitingVertex(p)
   }
6. if(p is in case D){
       Add two new edges to S
   }
```

图 4 扫描线算法的流程图

Step 2:实现四边形的剖分，见图 5

Algorithm Quadrilateral(P,L)

Input:多边形区域 P（包含每个顶点的 intial right neighbor）,step 1 计算 left edge 的有序表 L

Ouput：分割为凸四边形几何的 DCEL 结构。

1.从 L 中删除 left edge (u,v)，$u <_x v$。

2.设 r 为 v 的初始右邻居，r 在一条斜边(tilted edge)上，s 为斜边上的另一个点。

3.按照下面的顺序输出 (u,v)，(r,s) 组成的四边形：设 u' 为 (u,v) 中 y 坐标值较大的那个顶点，v' 为 (u,v) 中另一个顶点，r' 为 (r,s) 中 y 坐标值较大的那个顶点，s' 为 (r,s) 中另一个顶点。输出：v',u',r',s'。

4．对新的斜边做如下处理：如果 (u',r') 不是水平边(horizontal edge)，而且，$u' >_x r'$ 则将 (u',r') 加入到 L 的表尾。如果 (v',s') 不是水平边

图 5 四边形剖分流程图

三． 系统设计&数据结构

我们编程实现了该四边形剖分算法，开发环境为 Microsoft Visual Studio.NET 2003，全部程序使用 C++语言实现。整个程序主要包括三个模块：I/O 界面模块、扫描线模块和四边形剖分模块。

I/O 界面模块用于实现多边形数据的导入、导出，用户手工绘制，以及扫描过程和剖分结果的动态显示。其中多边形数据使用 DCEL 结构存储，封装在 CDCEL 类中，描述了顶点（vertex），边（dEdge），面（face）和洞（hole）之间的拓扑结构。图形界面的绘制则是使用 MFC 来实现的，并且采用了双缓冲技术来实现动态显示效果和避免图形闪烁。

```
class vertex
{
public:
    int     id;                         //顶点的ID
```

```
    double x,y;                         //顶点的坐标值
    dEdge* inc_edge;                    //与顶点相关联的一条边
    vertex* m_pRightNeighbor;           //顶点的右邻居
    vertex* m_pCurInciTiltVertex;       //与顶点相关联的斜边上的另一个顶点
    …
};
class dEdge
{
public:
    int     id;   //边的ID
    dEdge*  twin; //边的twin
    vertex* org;  //边的起始顶点
    face*   left; //与边相关联的面
    dEdge*  prev; //前一条边
    dEdge*  next; //后一条边
    …
}
class face
{
public:
    int    id;                          //面的ID
    dEdge* inc_edge;                    //面相关联的一条边
    int Holes_Number;                   //面内部包含的洞的个数
    dEdge* Holes_Edge[MAX_HOLES];       //与每个洞相关联的一条边
    …
}
class CDCEL
{
public:
    vertexes vertexList;     //顶点链表
    edges    edgeList;       //边链表
    faces    faceList;       //面链表
    UINT     m_vertexid;     //顶点的个数
    UINT     m_edgeid;       //边的个数
    UINT     m_faceid;       //面的个数
    …
}
```

扫描线模块用于实现算法中的 Step 1 部分，判断多边形的伪正交性，为每个顶点找到初始右邻居，并且建立一个 left edge 的有序表。整个模块的实现封装在 CScanLine 类当中。

四边形剖分模块用于实现算法中的 Step 2 部分，根据 left edge 的有序表和每个顶点的初始右邻居来进行四边形剖分，并且相应的更新 DCEL 结构。整个模块的实现封装在 CQuadrilateral 类当中。

四． 实验中遇到的问题及解决方法

程序采用DCEL结构来存储多边形拓扑信息，我们从www.cgal.org网站上下载了一些资料，参考了CGAL里面的一些代码和介绍，但是由于程序所处理的多边形含有洞，而且在进行四边形剖分之后需要对DCEL结构进行复杂的更新，即需要插入新的edge和face，所以我们重新写了CDCEL类的大部分代码。

实验中遇到退化的四边形的情况，经测试，这两种退化的情况均能够正确的处理，如图 6 所示：

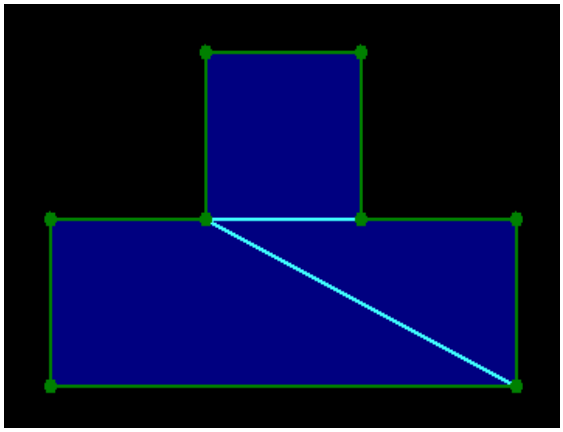
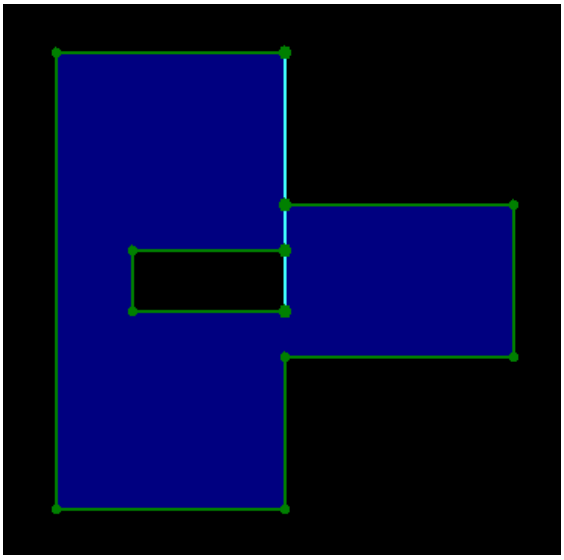
四边形的四个顶点中有三点共线	
四边形的四个顶点共线	

图 6 退化的几种情况

五. 总结和进一步的工作

我们实现了一种时间复杂度为 $O(\log n)$ 的四边形剖分算法，并且能够动态的显示扫描和剖分过程，非常直观的演示了整个算法的执行过程，便于进行程序的调试工作。而且根据实验检查所提出的建议做了几点改进：修改了多边形的绘制顺序，由面到边，再到顶点，使得显示出来的结果更美观；添加了【重新执行】功能，对于导入的同一个多边形数据，可以重复进行扫描线和剖分的演示。

程序在以下一些方面还需要进一步的改进：

1. 大规模数据的随机生成和测试比较：由于伪正交多边形的定义相对比较苛刻，顶点、边和面之间的相对关系需要满足的条件比较复杂，所以程序中没有实现随机生成多边形的功能，用户只能手工绘制，因此很难生成较大规模的多边形数据进行测试。
2. 在算法的 Step 1 中存储状态结构（即与当前扫描线相交的所有线段）时，我们采用了链表的结构，这样使得程序处理起来相对比较简单，但势必会影响算法的效率，可以改用平衡二叉树来处理。
3. 一些特殊的退化情况还没有正确处理，如下图所示，左边的多边形中有两对顶点分别重合，变成图中右边的多边形，程序对这种退化的情况不能进行正确的四边形剖分，见图 7。

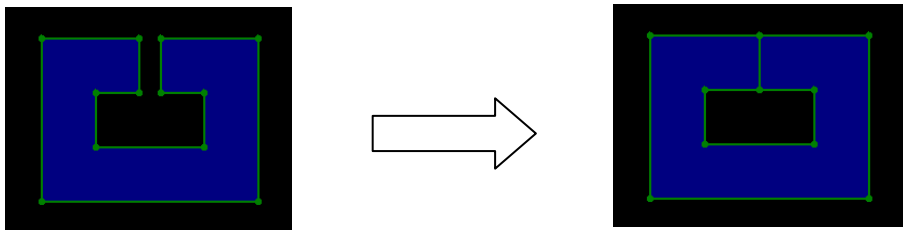


图 7 没有处理的一种退化情况

六. 操作说明

1. 多边形的绘制

【导入数据】　直接导入多边形数据文件

【导出数据】　将用户手工绘制的多边形保存到数据文件

【手工绘制】　用鼠标绘制多边形，先绘制外围的多边形，然后在其内部绘制洞，右键结束

【移动顶点】　　点击选中多边形的顶点进行平移

【绘制结束】　　完成多边形的绘制，程序自动检验所绘制的多边形是否为简单多边形，并初始化 DCEL 结构

2. 多边形的操作

【自动执行】　　自动执行扫描过程和剖分过程

【单步执行】　　每次处理一个事件，扫描一个顶点或者剖分一个四边形

【重新执行】　　对于当前的多边形数据，重新进行一次自动的执行

【暂停】　　暂停程序的执行

【清屏】　　清空屏幕和内存数据

3. 显示效果

选择【延时】的时间，来调节程序执行速度；在手工绘制时，可以选择是否【画正交多边形区域】;在程序执行过程中可以选择是否【显示四边形边界】，是否【显示四边形颜色】，是否【显示动态效果】。

七. 参考文献

[Anna84] Anna Lubiw . Decomposing Polygonal Regions into Convex Quadrilaterals. Proc. First ACM Computational Geometry Symposium, Baltimore, pp. 97-016,(1985).

[KKK83] J.Kahn, M.Klawe, and D.Kleitman. Traditional galleries requires fewer watchman. SIAM J.Algebraic and Discrete Methods 4,pp,194-206,(1983)

[Sack82] J.R.Sack. An $O(n\log n)$ algorithm for decomposing simple rectilinear polygons into convex quadrilaterals. Proc. 20th allerton conference. 64-74 (1982)