平面点集三角剖分

triangulation of planar point sets

1. 问题背景

计算机在石油、矿山和城市地质等地学领域的广泛使用,使得对三维地学信息系统的需要更加迫切。雨量等值线在水文、防汛领域应用广泛，Delaunay三角剖分具有空外接圆和最走的最小角度两个良好性质，对于非规则分布的离散点数据进行三角剖分内插是生成等值线的最常用的算法,但实际应用中往往都不是凸壳进行三角化，而是有限定边(或限定点)对三角剖分进行约束。

三角剖分是计算机辅助几何设计、几何造型及计算机图形学中研究的重要内容之一。对于设计一个三角剖分算法来说,最重要的就是其复杂度低和高质量网格的形成。三角剖分算法在计算几何、曲面重构及有限元网格生成中有着重大的应用价值。

对于无限定条件的三角剖分，其剖分方案无确定性，无论对点集做怎样的剖分，效果都似乎差不多，我们所需要考虑的更多的是算法的复杂度和运行的成功率。然而，就其外观的自然性和布局的合理性而言，某些三角剖分的确更好。特别是对于那些给予了约束条件的情况，三角剖分的方案比较有限，比如最小夹角和条件或者最短路径条件等等。这时，我们就需要采取某些特定的算法来实现，比如 Delaunay 剖分或是贪心算法。

本次实验我们基于分治策略，利用DCEL结构对平面的散乱点集实现了三角剖分。主要考虑分治的合理性以及算法的时间复杂度问题。

2. 算法及原理

Divide：按点的x坐标的中值（或先按点的二坐标分类，然后从中间分割）分割点集S为S1和S2，使S1与S2包含的点数近似相等。如果，S1与S2点的数目大于k（k为设定的常数），则继续分割点集，直至点集规模小于或等于k。对每个点集按照增量算法求其三角剖分。

Merger：对于已经三角剖分过的子点集，由于其外围边界构成凸壳，故而相邻的子点集间的合并可采用与凸壳合并相似的处理方法。具体操作如下：遍历两凸壳上顶点连线u1u2与下端点连线v1v2间的所有点;找出其间D1的最右点Pi和D2的最左点Pk，连接PiPk；从Pi开始与Pk上方的各点连线，使得与凸壳的边界无交点方止；从停止点仿照上述步骤向P1各点连线，如此往复直至将PiPk与U1U2间的区域完全三角剖分；自PiPk向下的三角剖分与之类似。示意图如下：

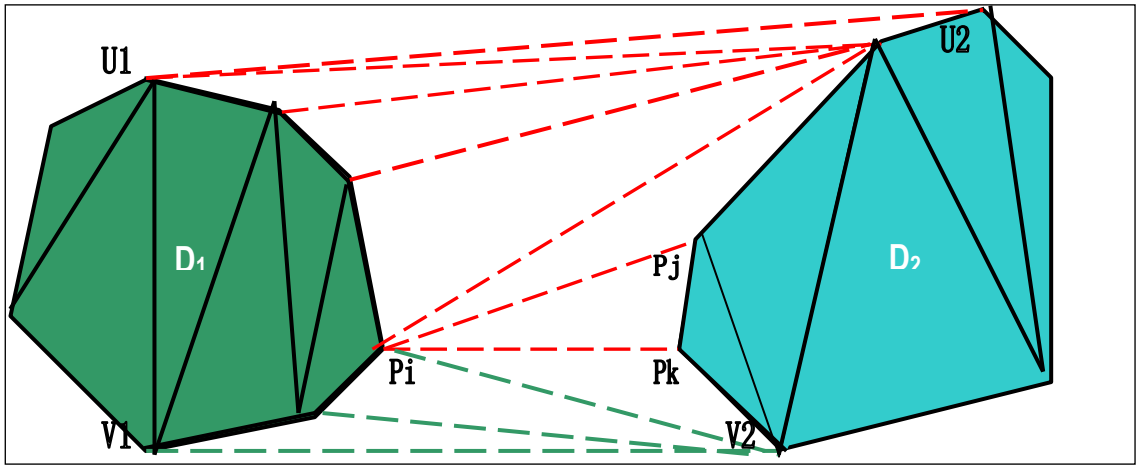

图1

3. 系统设计

本次实验，我们基于Visual studio 2008 c# 工作平台编制。在输入端，设计了三种输入方式，手动输入，文件输入和随机生成。手动输入便于验证算法的普遍性，文件输入则对于某些特殊的情形给予了演示，随机生成方便演示于用户。

4. 实现过程中遇到的问题及解决对策

实现过程中有很多简单的细节问题，这里就不赘述了。这里说明一下算法程序经常需要注意的问题。

(1)DCEL数据结构如何具体实现。DCEL只是一种逻辑上的数据组织形式，在使用时的具体实现我们按照它的基本结构，分别设计了VerTex、Face和HalfEdg三个类，作为点、面和半边的类型。三个类的如下示：

```
class VerTex //点类
    {
        private Point startPoint;
        private HalfEdge edge;
        public Point StartPoint// 点坐标
        {
            get { return startPoint; }
            set { startPoint = value; }
        }
        public HalfEdge Edge // 以该点为起点的有向边
        {
            get { return edge; }
            set { edge = value; }
        }
        public VerTex(Point StartPoint, HalfEdge Edge)
       {
```

```
            startPoint = StartPoint;
            edge = Edge;
        }
    }
class Face //面类
    {
       private HalfEdge outComponent;
       private HalfEdge innerComponent;
       public HalfEdge OuterConponent // 围成这个面的有向边中的一条，
如果为无界面，则此值为null
        {
            get { return outComponent; }
            set { outComponent = value; }
        }
       public HalfEdge InnerComponent //面中洞的有向边中的一条，如果该
面无洞，则此值为null
        {
            get { return innerComponent; }
            set { innerComponent = value; }
    }
class HalfEdge
    {
        private VerTex originPoint;
        private HalfEdge twinEdge;
        private Face incdentFace;
        private HalfEdge nextEdge;
        private HalfEdge preEdge;
        public VerTex OriginPoint// 该有向边的起始点
        {
            get { return originPoint; }
            set { originPoint = value; }
        }
        public HalfEdge TwinEdge // 该有向边对应的半边
        {
            get { return twinEdge; }
            set { twinEdge = value; }
        }
        public Face IncdentFace // 所在的面
        {
            get { return incdentFace; }
            set { incdentFace = value; }
        }
        public HalfEdge NextEdge// 下一条半边
        {
```

```
            get { return nextEdge; }
            set { nextEdge = value; }
        }
        public HalfEdge PreEdge  // 上一条半边
        {
            get { return preEdge; }
            set { preEdge = value; }
        }
    }
```

最后又设计了一个DCEL类， 记录剖分的结果：

```
    class DCEL
     {
        private List<VerTex> verTexCollection;
        private List<HalfEdge> halfEdgeCollection;
        private List<Face> faceCollection;
        public List<VerTex> VerTexCollection// 点的集合
        {
            get { return verTexCollection; }
            set { verTexCollection = value; }
        }
        public List<HalfEdge> HalfEdgeCollection// 有向边的集合
        {
            get { return halfEdgeCollection; }
            set { halfEdgeCollection = value; }
        }
        public List<Face> FaceCollection // 面的集合
        {
            get { return faceCollection; }
            set { faceCollection = value; }
        }
        public DCEL()//构造方法
        {
            verTexCollection = new List<VerTex>();
            halfEdgeCollection = new List<HalfEdge>();
            faceCollection = new List<Face>();
        }
    }
```

根据这样的构造方法，我们实现了对剖分结果的 DCEL 存储。

(2)记录凸壳边。剖分成最小的点集中我们采用增量算法进行三角剖分，各个三角剖分后的点集进行 Merge。在这些过程中都需要对先前生成的点集的凸壳边上进行遍历。针对这个问题我们记录了剖分时生成的唯一无界面 OuertFace。所有凸壳边都是 OuterFace 的 InnerComponent，从其中的一条 HalfEdge 都可以遍历整个凸壳。

（3）Merge 过程时左右两部分 D1 的最右点和 D2 最左点定位问题。在进行 Merge 时，

首先要找到D1的最右点和D2的最左点。我们设计了 private HalfEdge GetMinEdge(HalfEdge outerEdge)和 private HalfEdge GetMaxEdge(HalfEdge outerEdge)两个函数，从遍历 OuterFace 就可以找到 D1 的最右点和 D2 的最左点。

5. 时间复杂度分析

对n个点进行快速排序需要nlogn的时间，对划分的子点集进行三角剖分可在线性的时间内完成，而对生成的子凸壳进行merge操作亦可在线性时间内完成，所以算法时间复杂度满足nlogn的要求。

6. 测试结果

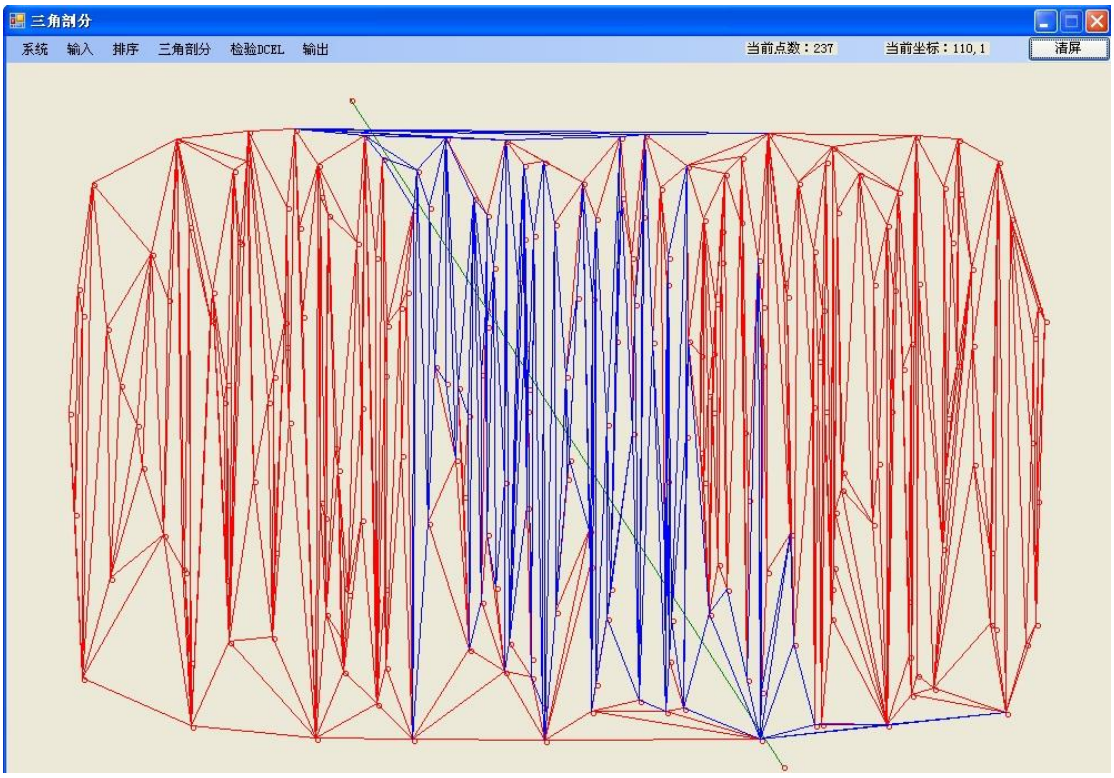

图2

7.测试对比结果

根据测试，这个程序能处理很多点的情况，以及特殊情况。而且这个程序提供了方便的输入点的坐标，以及存取盘的功能。这样可以精确的演示算法的各种特殊情况的处理。

8. 没有解决的问题

本算法仅仅针对坐标是整数类型的情况，对于坐标是小数的情况未作处理。同时，对于点数比较密集时，未设计放大处理环节。

9. 有关文献

参考文献主要有：

1)计算几何算法与应用
2)计算几何----算法设计与分析
3)地理信息系统算法基础
4)地理信息系统基础