实验一 平面点集的三角剖分

1 实验概要

1) 实现了基于分治策略的平面点集的三角剖分；
2) 提供了图形化的输入输出界面，并支持批量数据的导入、导出以及大规模随机据的生成；
3) 使用 DCEL 存储三角剖分的结果，实现了两种直观的验证 DCEL 正确性的方法；
4) 详细该平面点集剖分算法的效率。

2 实验内容

2.1 基于分治策略的点集剖分算法

算法的具体流程如下：

1) 由鼠标双击，随机生成或者读入文件的形式得到一个点集。首先就是对这个点集排序，找到 y 坐标最小的点（ y 坐标相同找 x 坐标最小的点）作为极点，然后对其它点进行排序。通过极点到其它点的向量位置进行排序，在左边的则沉到点数组中后面的位置，最后把极点保存到数组的末尾。如图 2.1 所示：

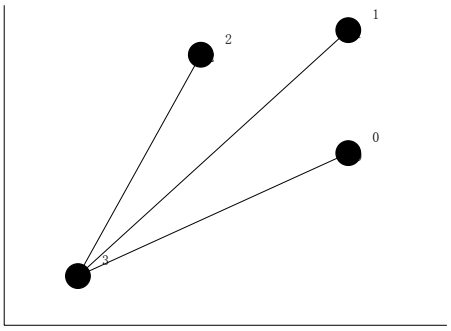

图 2.1

2) 用分治法（代码中用递归实现）把除开极点以外的所有点（已排好序）分成两部分，再分别在这两部分中分成两部分，直至其中一小部分中只剩下一个或者两个点时递归结束，并把这些边储存起来，只记录此边两点在点数组的下标值。如图 2.2 所示：

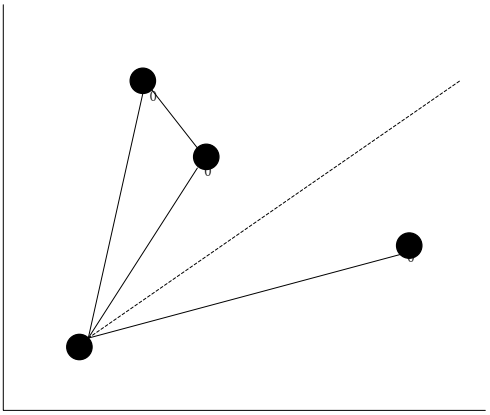

图 2.2

3) 最后的一步就是把已经三角剖分好的两部分结合起来，也就是说把两部分中间的区域三角剖分。

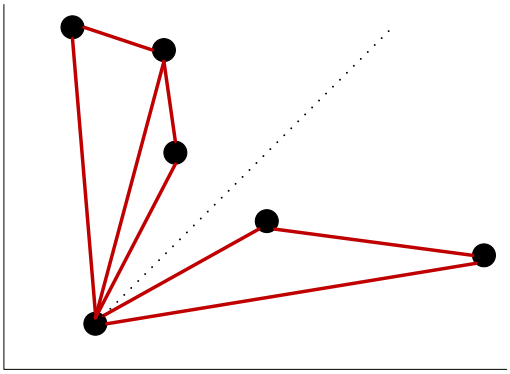

图 2.3

寻找两部分最靠近虚线（左半部分最右点和右半部分最左点）。从左半部分最右点开始向上和对面的点连线，直至右半部分所有点都在这条线段的右边为止。

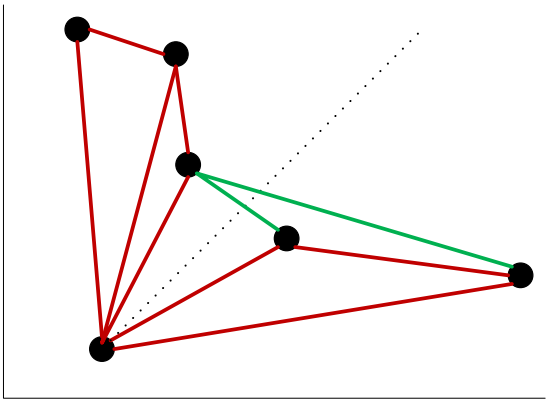

图 2.4

交替地从两边的点出发连线，直至两部分的点都在最后一条连线的左边为止，这条边也是所有点的凸包上的边。

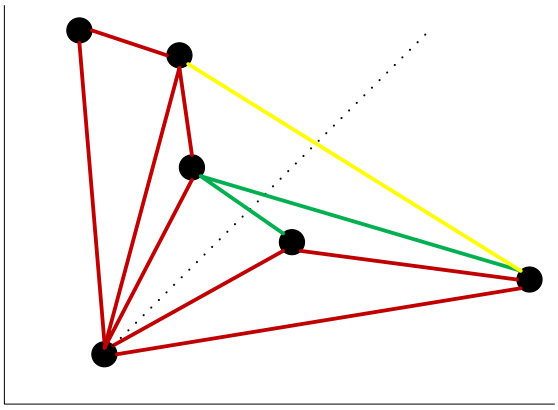

图 2.5

剖分算法中的点、线和面的数据结构如下：

1. 点：CPoint points[]

MFC 中自带的数据类型，里面的数据成员有 x 坐标和 y 坐标等，其中 x 和 y 均为整数，还有一些点运算的成员函数；

2. 线：CPoint lines[]

x， y 坐标表示线段的两个端点，都是点数组的下标值；

3. 面：Face faces[]

面使用我们自己定义的数据结构，由于是三角剖分，所以每个面有三个顶点，细节如下：

```
struct Face
    {
        CPoint l1;
        int l1_num;
        int l1_neighborFace;
        CPoint l2;
        int l2_num;
        int l2_neighborFace;
        CPoint l3;
        int l3_num;
        int l3_neighborFace;
    };
```

其中，l1、l2、l3 分别为三角形的三条边，x、y 坐标表示两个端点在点数组中的下标值；l1_num、l2_num、l3_num 分别为三条边在边数组中的下标值，用来匹配线段；l3_neighborFace 与该面相邻，且以 l3_num 为公共边，如果一个面的某一边没有对应的相邻三角形，说明这边是凸包上的边（最外层边界），则相邻面的值设为-1.

2.2 DCEL 结构

DCEL 结构（或者半边结构）可以进行点、边和面等各种相互关系的查询，非常高效，关于半边结构的定义各种参考资料中已经写得非常详细，在这里我们将实验中实现的半边结构进行说明。我们定义的主要数据结构和其属性方法如下：

1. class Vertex

表示半边结构中一个顶点，主要包含如下属性和方法：

```
bool inner;        //表示该点是内点还是边界点，在二维情况下边界点也就是凸包上的点
int startHe;       //顶点处的起始半边编号
int endHe;         //顶点处的终止半边编号（仅对边界点有效）
```

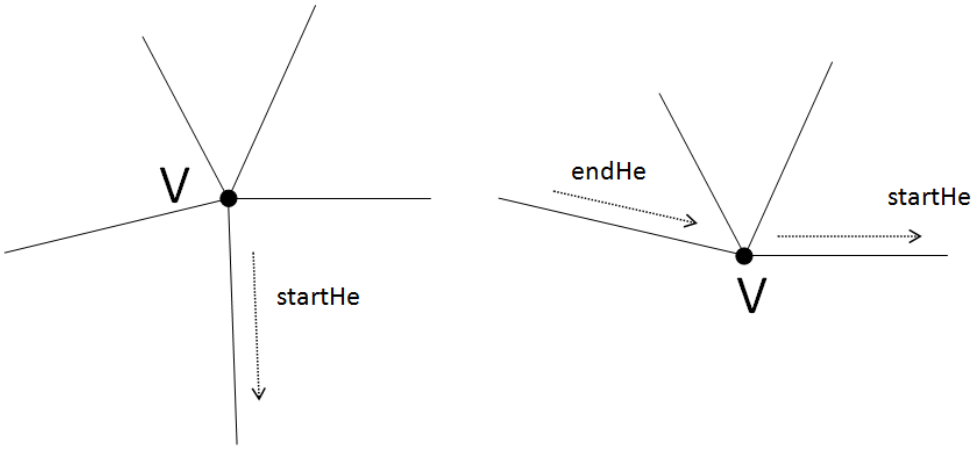

图 2.6 内部点和边界点的起始、终止半边

2. class HalfEdge

表示一条半边，即一条有向边，主要包含如下属性方法：

```
int fv;            //起始点
int tv;            //终止点
int fn;            //该半边所述的三角形编号
int prev;          //该半边的上一条半边
int next;          //该半边的上一条半边
int opp;           //和该半边方向相反的孪生对边
```

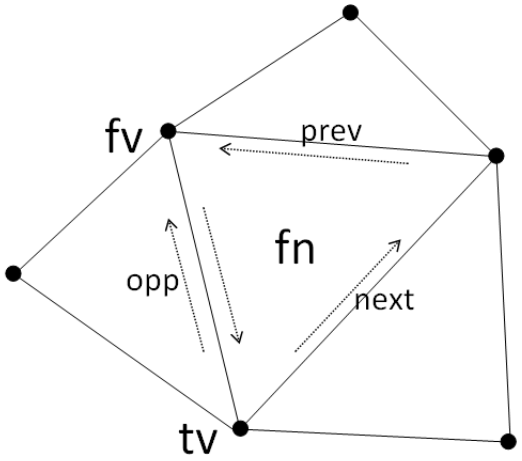

图 2.7 半边结构

3. `class HalfFace3`

表示和一个面相关的半边信息，包含如下信息：

```
int he[3];          //该面里的三条半边，注意应按照逆时针顺序排列
```

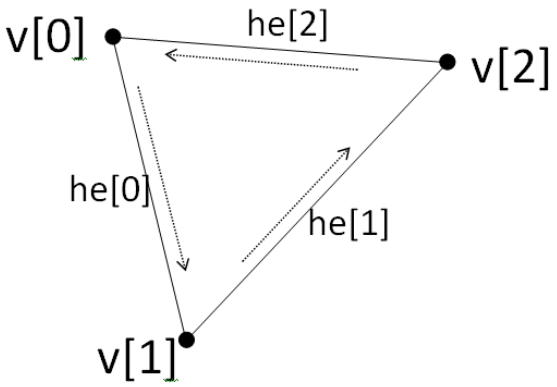

图 2.8 三角形面中点和半边的顺序

在我们的算法中，一个 DCEL 结构主要包含上述三方面的信息，有了上面的结构，我们可以很容易的进行各种操作，例如我们希望获得一个顶点周围的邻居，就可以从该点的 `startHe` 开始，顺序访问所有以该店作为出发点的半边，访问每一个邻居的时间复杂度都是 $O(1)$ 的。

3 程序运行说明

3.1 获得初始点集

我们的程序提供三种获得初始点集数据的方法：

1) 通过双击鼠标左键在相应位置增加点

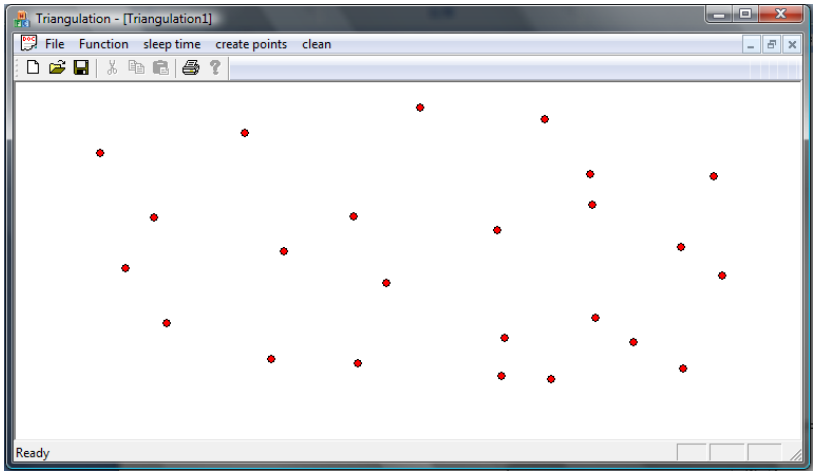

图 3.1 通过双击鼠标左键增加数据点

2) 设置点数，随机生成点集。可以通过程序菜单中的 Create Points->Random Points 菜单随机产生数据点，在弹出的窗口设想要的点数即可，为了方便查看，我们在距离边界 50 像素内不产生点。

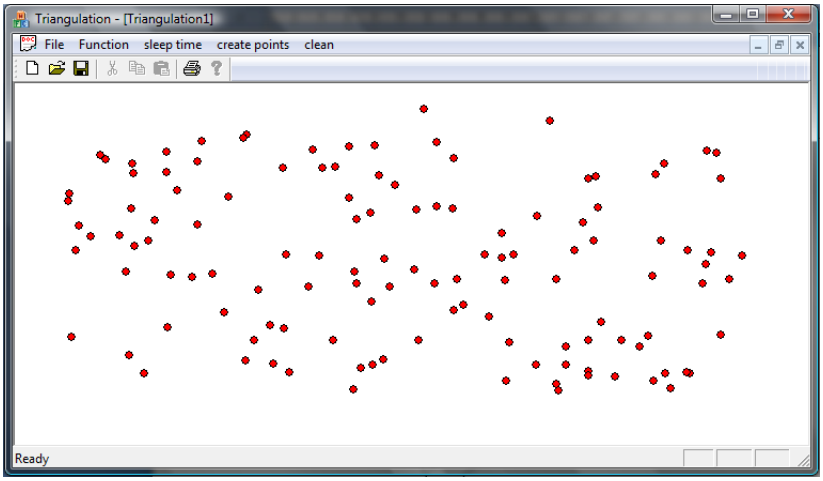

图 3.2 随机产生的 100 个数据点

3) 通过输入已有的 TXT 文档来读入点集，TXT 文档第一个数字必须是所含点数 n，下面 n 行则是每个点的 x， y 坐标值。

3.2 三角剖分

为了更方便地看清程序的每个步骤，可以通过程序菜单 Sleep Time->Set Sleep Time 设置每一步的延时时间。图 3.3 显示了有 100 个点的点集剖分结果。

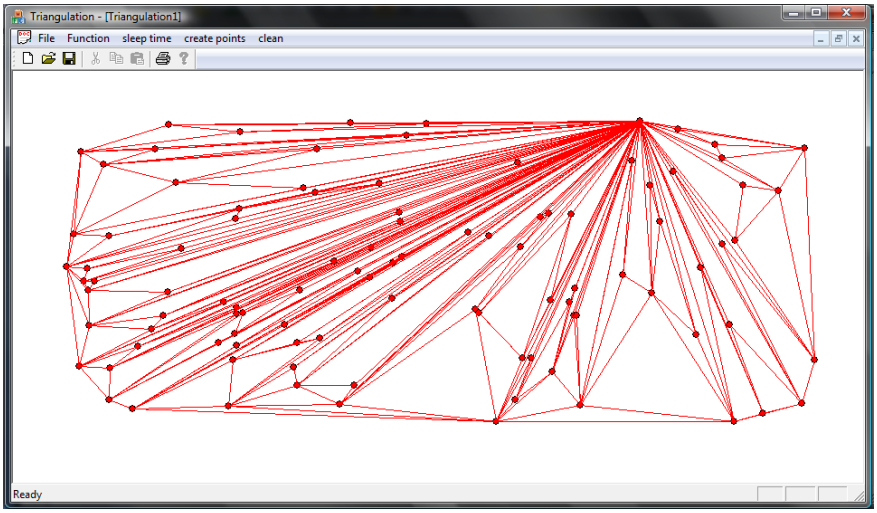

图 3.3 三角剖分结果

3.3 DCEL 数据结构的检验

为了检验程序中所建立的 DCEL 数据结构的正确性，我们使用了两种方法进行检验。

1) 随意画一条线，找出这条线经过的所有三角形。

具体操作方法为：用户通过鼠标左键按下，拖动鼠标，放开画一条线（绿色）。查看经过的三角形（都填充为绿色），经过的线段会自动闪烁并且加粗。算法的步骤为：

第一步找出用户线段于所有剖分线的交点的 x 坐标，找出在所画线段的 x 坐标范围内同时在剖分线的 x 坐标范围内但是 x 最小的那条剖分线为第一条相交线，找出以条剖分线为边的三角形；

第二步判断与三角形另外两条边中的哪条相交，并找出另外一个以这条边为边的三角形为下一个三角形；

第三步，重复以上步骤直至没有剖分线的另一个相邻面为-1（直线穿过凸包），或者交点比用户所画线最大 x 坐标还要大（用户线末端停留在凸包内部）。

检验的结果见图 3.4。

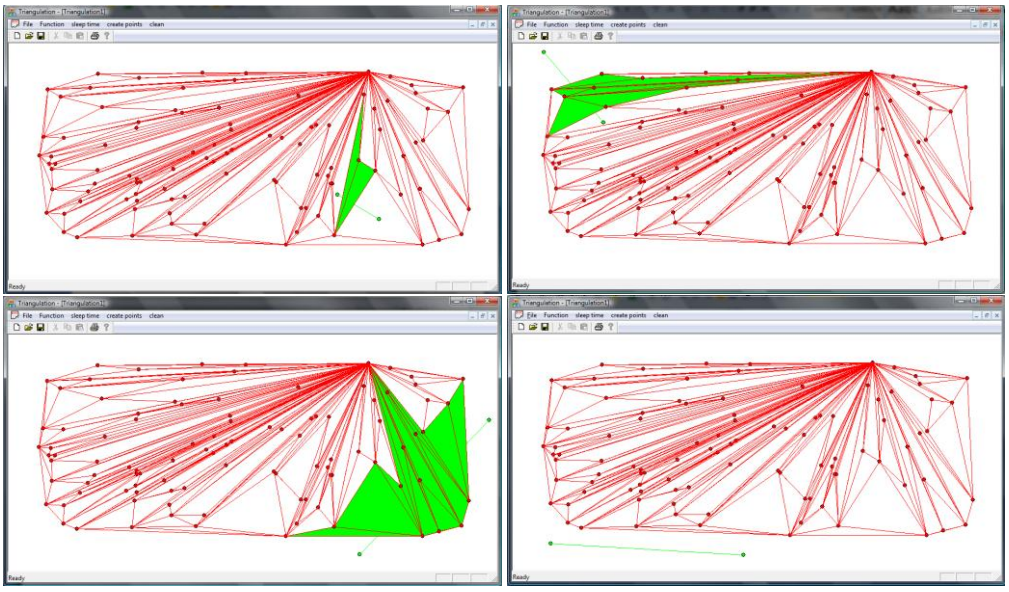

图 3.4 DCEL 正确性检测

2) 使用 DCEL 获取某个点的周围邻居。

具体操作方法为：使用右键双击一个点，程序能够顺序找出他的邻居并依次显示出来。效果如图 3.4 所示。

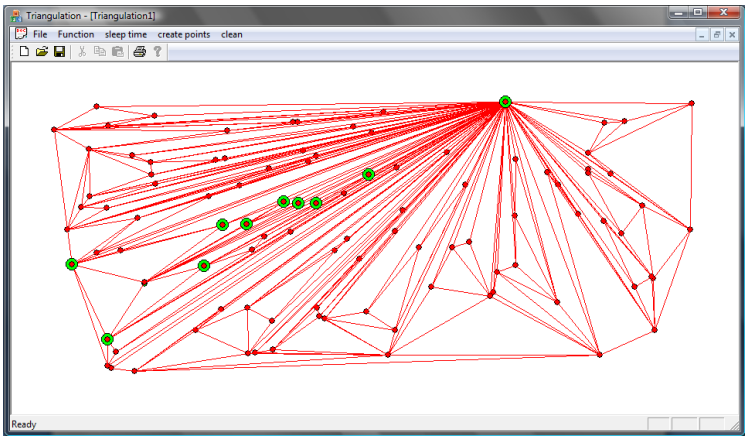

图 3.4 DCEL 正确性检测

4 结果与性能分析

具体数目的点集三角剖分所需时间如下图所示：

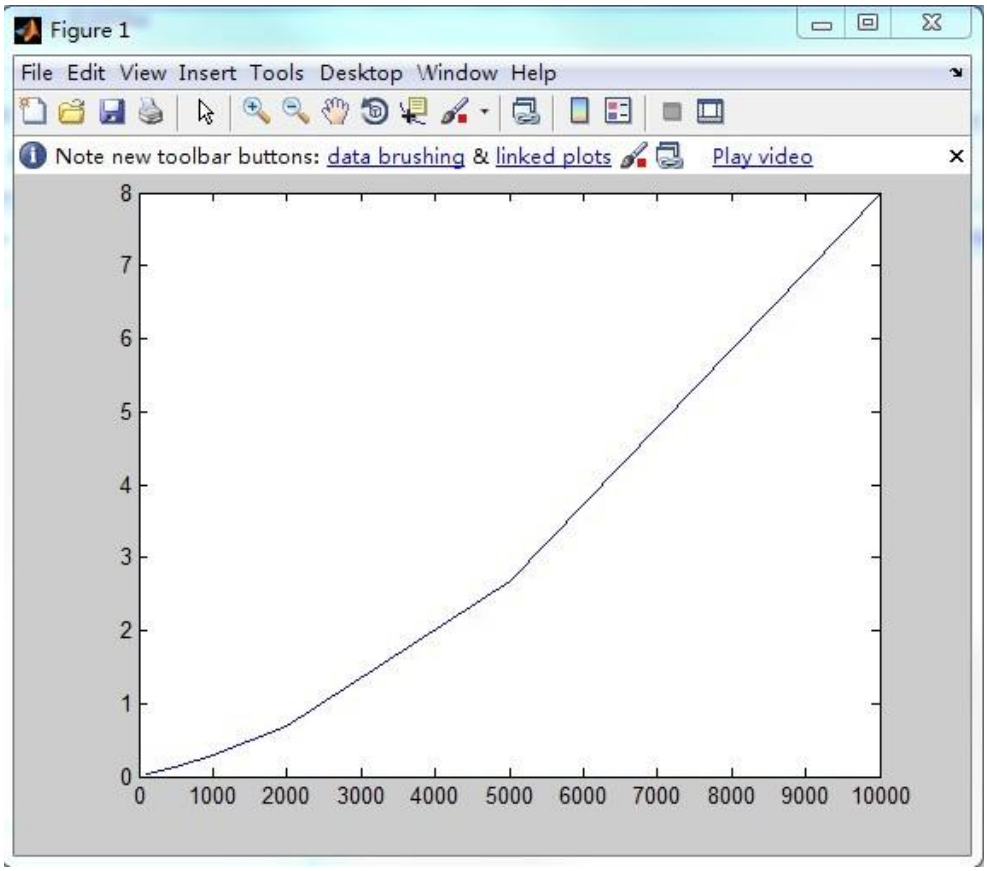

图 4.1　三角剖分所需时间表（X 坐标为点数-y 坐标为三角剖分所花时间）

三角剖分大部分时间都用于点集的排序。另外因为每个点的出度、入度差异很大（比如极点和其他点），以至创建的 DCEL 结构花费的时间过多。

5 不足与展望

这样剖分的三角形大部分形状细长，剖分质量不是很好，每个点的出入度差距可能很大，导致 DCEL 在点超过 10000 时的鲁棒性不够好。如果采用 Delaunay 三角剖分算法则可以得到剖分质量很好的三角形，每个三角形的外接圆里面不会出现第四个点。我们将进一步学习 Delaunay 三角剖分算法并把它加入到程序中。