Delaunay 三角网数字地形模型构建

小组成员：范文山、刘晓明、文雅玫

一、 问题背景

数字地形模型（Digital Terrain Model, DTM），特别是不规则三角网（TIN）的建立和优化是地理信息系统和虚拟现实中的一个基础研究问题。TIN模型具有数据冗余小，存储效率高，且能较好地顾及地形特征和适合多层次表达等突出优点。如何快速、高效地构建Delaunay三角网，一直是众多学者研究和关注的焦点。迄今为止出现了不少成熟的算法，基本算法为分割_合并算法、逐点插入法及三角网生长法等，其中三角网生长算法由于算法效率较低，目前较少采用；逐点插入法虽然实现较简单，占用内存较小，但它的时间复杂度差，效率较低；分割_合并算法最为高效，但相对复杂，由于其深度递归，对内存要求较高。近年来也有一些新的算法，结合以上分割_合并算法和逐点插入法算法, 以兼顾了空间和时间性能。我们采用分治法来求Delaunay三角剖分，在分割点集时，我们采用自适应划分方法双向交替均匀剖分。

二、 算法

1) 点集的分割

我们采用类似 kd—tree 的自适应划分方法对点集进行分割，直至每个点集只包含 2 或 3 个点。首先对所有点分别对按照 x 方向、 y 方向进行索引排序，将排序的结果分别放入两个数组，数组中存储的是这些点的索引

号。每一次分割先对 x 方向的索引数组进行二分，生成 x 方向上的左右两个子数组，然后根据在 x 方向上的划分情况生成对应的 y 方向上的左右两个子数组，即依次判断 y 方向的索引数组中的每一个点，若该点在 x 方向划分属于左子数组，则它在 y 方向的划分也应该属于左子数组，否则它在 y 方向的划分属于右子数组。循环这个过程，直到每个子数组中只有 2 或者 3 个点时结束，整个分割的流程如图 2.1 所示。

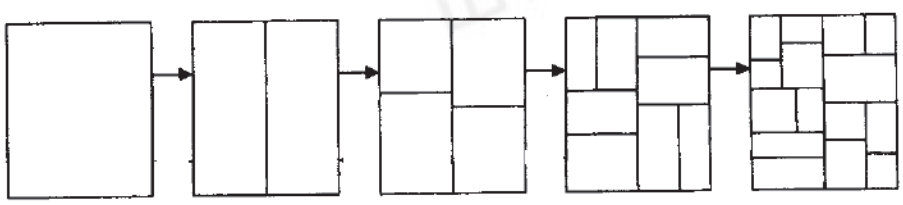


图 2.1 点集分割流程示意图

算法 1：点集分割

输入：n 个二维点、 分割方向。

输出：一个二叉树，每个叶子结点只包含 2 或者 3 个点。

1. 如果点数不大于 3，算法结束。
2. 对于所有点，根据它们的 x 值进行索引排序，将结果放入索引数组 Cx；
3. 对于所有点，根据它们的 y 值进行索引排序，将结果放入索引数组 Cy；
4. If（水平分割）
5. {
6. 按 x 方向进行二分，得到左右两个子数组；
7. For（Cy 中的每一个点）

```
8.                  If（该点被分割到 x 的左子数组中）
9.                      将该点分到 y 的下子数组；
10.                 Else
11.                     将该点分配到 y 的上子数组；
12.             递归调用本算法，对左右两个子数组进行垂直分割。
13.         }
14.         else
15.         {
16.             按 y 方向进行二分，得到上下两个子数组；
17.             For（Cx 中的每一个点）
18.                 If（该点被分割到 y 的下子数组）
19.                     将该点分到 x 的上子数组；
20.                 Else
21.                     将该点分配到 x 的下子数组；
22.             递归调用本算法，对上下两个子数组进行水平分割。
23.         }
```

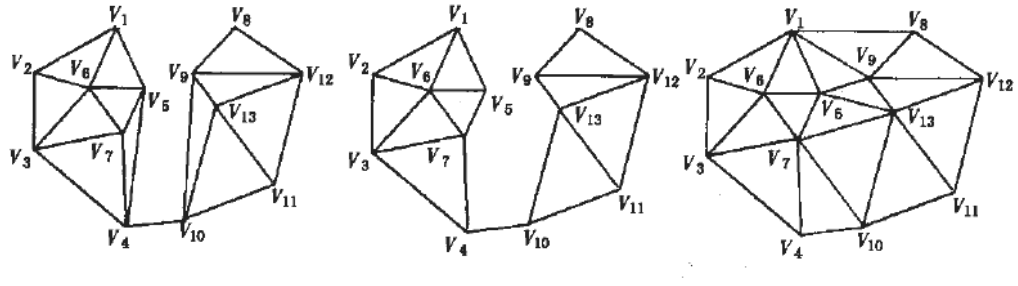
---

2) 构造初始凸包

分割后，每个点集里只包含 2 或者 3 个点，连接这些点，如果点集中有两个点，则构成一条线段；如果点集包含三个点，则可能构成一个三角形（三点不共线）或者两条线段（三点共线）。因为线段也属于凸包，因此我们将这三种情况统一考虑，即对每个点集中的凸包进行合并。

3) 合并算法

合并时，我们采用对凸包的合并方法。由于左右合并和上下合并原理完全相同，下面我们就以左右合并为例介绍合并算法。



（a）寻找起始线　　（b）删除非 Delaunay 三角形　　（c）重构三角形

图 2.2 合并示意图

a) 寻找两个凸包合并的起始线

首先找到左凸包的最右点和右凸包的最左点，将它们连接，然后用 zig—zag 的方法找到两个凸包的下支持边，如图 2.2（a）中的线段 V4V10，即为左右两个凸包的下支持线。

b) 删除非 Delaunay 三角形

在合并过程中，在边界处的某些三角形由于受相邻子三角网边界点的影响而不符合 Delaunay 法则，需要删除，如图 2.2（b）所示，当然，删除三角形伴随着对其相邻三角形拓扑关系的调整。假设当前边为 V4V10，对以左当前点（V4）为顶点按逆时针方向的一系列三角形（V4V5V7、V4V7V3）检查右当前点（V10）是否在其外接圆内，如果是，首先对该三角形的邻接三角形（V5V6V7、 V4V7V3）的拓扑关系进行调整，然后删除该三角形，并且对子三角网的边界点进行改动，在左当前点后插入三角形的另一点（V7，左子三角网边界点改动为 V1、V2、V3、V4、V7、V5）。

类似方法删除右边三角形 V9V10V13，右边界点改动为 V8、V9、V13、V10、V11、V12。

c) 生成三角形

以当前边的两点（V4、V10）和左当前点的后一点（V7）为测试三角形，如果右当前点的前一点（V13）在测试三角形内，则当前边与右当前点的前一点生成新三角形（V4V10V13），否则与左当前点的后一点生成新三角形（V4V10V7）。如图 2.2（c）所示则生成新三角形 V4V7V10，当前边调整为 V7V10，重复步骤 b,c，直至 V1V0,此时，V1V0 左侧不再有点，V0V1 右侧不再有点，V1V0 是点集的凸边，合并终止。

三、 显示

在显示结果时，我们考虑了教学演示的用途和三维地形的效果，实现了二维 Delaunay 三角剖分结果显示以及三维地形效果的显示，并且实现了二者的切换。

1) 二维显示

因为我们采用了 DCEL 结构，这种结构中每一个三角形都已存放在面表中，只需要遍历面表依次绘出即可。

2) 三维地形显示

显示地形时，我们利用 delaunay 三角剖分的结果，加上地形图中的高程数据信息，形成三维的三角网格，用 OpenGL 显示出来。同时，为了使显示的效果更加好，我们对剖分好的三角形进行了分层显示，即对于不同高度层的三角面片赋予不同的颜色，按照一般地形图的分层设色规则，由

批注 [J1]:
与检查时不同，不再是贴 RGB 格式的纹理

低到高分别以深绿色、浅绿色、黄色、浅棕色和深棕色表示，以体现高程的变化。

四、 数据结构

在这次作业中，我们采用了 DCEL 的数据结构存放所有的点、边、面。

定义了一个 DCEL 类：

```
class CDCEL
{
public:
    CDCEL(CGrid& grid);
    CDCEL(int* ptIndexs,int nCount);
    ~CDCEL();
    void Clear();
public:
    CObList m_Vertices;
    CObList m_Edges;
    CObList m_Faces;
    Face* OutFace;
```

DCEL 类中用三个链表来存储点表、面表和半边表。

```
class Vertex :public CObject
{
public:
    double x,y,z;
    HalfEdge *IncEdge;
    Vertex();
    Vertex(Point3D& point);
};

class Face :public CObject
{
public:
    bool GetPoints(Point3D& p1,Point3D& p2,Point3D  &p3);
    HalfEdge *OutEdge,*InnerEdge;
    void Draw(CDC *pDC,int index);
};

class HalfEdge: public CObject
{
public:

    Vertex *Origin;
    HalfEdge *Twin,*Next,*Prev;
    Face* IncFace;
    double LeftRight(Vertex* v); //判断点和边的左右关系,>0:左 0:线上 <0右
   double LeftRight (Point3D* v);//判断点和边的左右关系,>0:左 0:线上 <0右
rivate:

};
```

五、 结果

我们在 VC 6.0 环境下实现了这个系统，没有共享任何开源代码，所有代码都是由我们自己编写的。

图 4.1 给出了 10000 个点的 Delaunay 三角剖分的二维显示效果，图 4.2 给出了四种不同地形数据的三维显示效果。

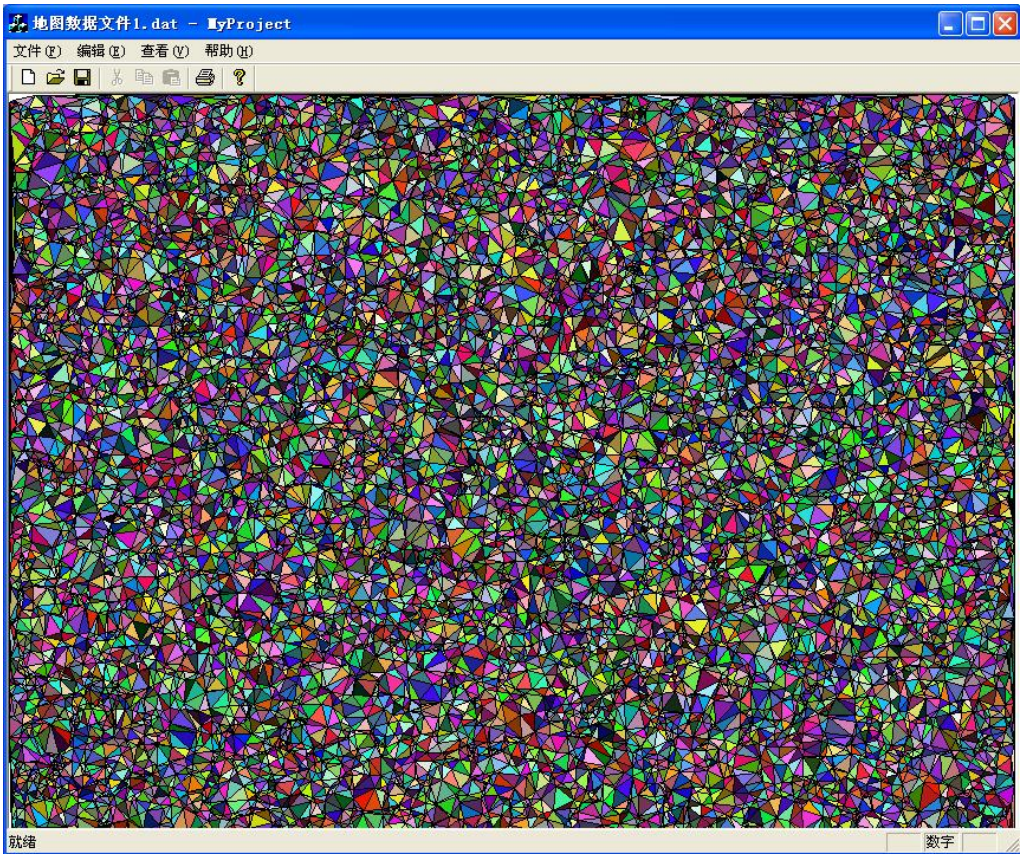
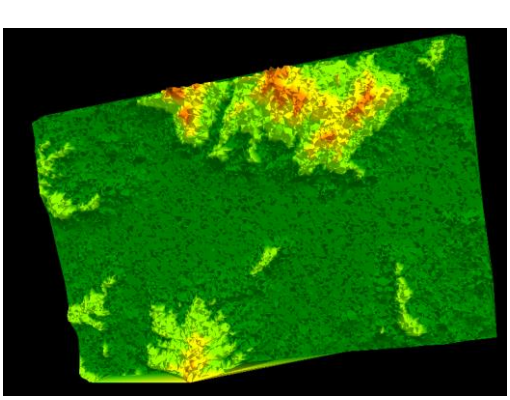
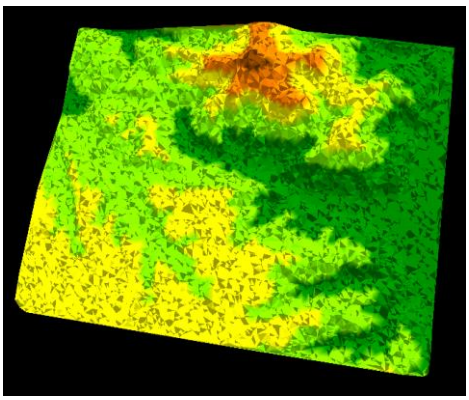


图 4.1 Delaunay 三角剖分的二维显示效果





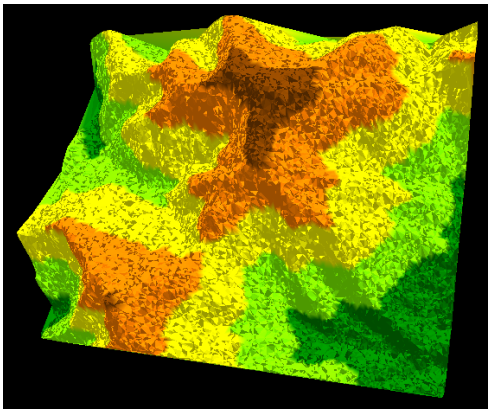
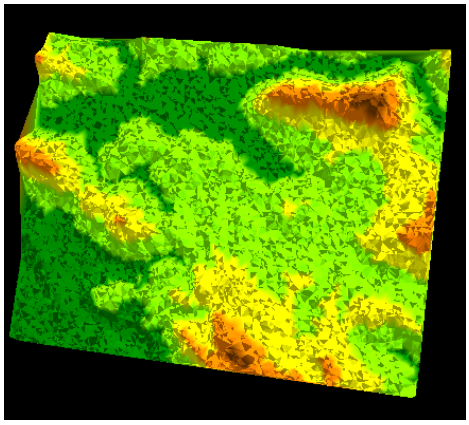



图 4.2 地形效果

六、 结论及讨论

由于我们采用的分割算法只需要对 x、y 方向进行一次排序即可，大大降低了算法的时间复杂度；同时，这种算法分割得比较均匀，避免了狭长三角形的出现，从而降低了在合并时删除无效三角形的开销。并且，在合并时，我们的算法也只需要找一条支持边。因此，我们的算法的时间开销较小，复杂度为 O（nlogn）。

同时，我们进行了大量的测试，以确保算法的鲁棒性，从稀疏点到非常密集的点都可以很好的处理，我们的测试中最多采用了 15 万个点。

批注 [J2]: 大部分情况下鲁棒，但未对输入数据合理性作检查。
比如，直接调入非*.dat 格式的文件，程序可能崩溃

我们在 Pentium（R）D CPU 2.80GHz ，1G 内存的机器上运行程序，各个数量的点的处理时间如表 5.1 所示，可以看出，处理时间确实是按照点数（n）的增加以 nlogn 的复杂度增加的。

表 5.1 不同数量采样点的运行时间

点 数（个）	500	1000	5000	10000	20000	50000	150000
时 间（s）	0.04	0.11	0.75	1.73	4.28	15.07	80.62

附：程序环境说明

本程序需要 OpenGL 实用库 glut 支持，我们已经将 glut 库置于 3DSLOADER\glut-3.7.6-bin 目录中，如果在 VC6.0 环境中运行，需要通过菜单 Tools→Options，选择 Directories 页面，将 3DSLOADER\glut-3.7.6-bin 目录添加到 include files 和 library files 目录列表中。

程序的数据文件放在“地图数据”子目录中，文件后缀名为.dat。运行时，通过“打开”文件菜单，选取要读取的数据文件若干秒后即可显示出二维剖分效果，按“t”键实现二维和三维的切换，通过按住鼠标右键前移或者后退缩放三维地形，按住鼠标左键拖动可以旋转三维地形。其中的数据文件都是随机采样的 150000 个点位置和高程信息，可以把文件第二行的 150000 改为其它的数（小于 150000），以获得其它点数的效果。