平面点集的三角剖分

1　问题背景

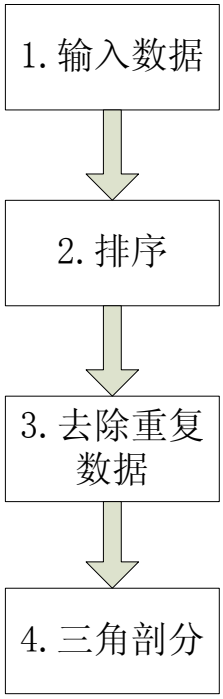
三角剖分是计算几何中的一类经典问题，我们研究的是这一问题的简化版本－－平面点集的三角剖分。所谓平面点集的三角剖分，是一种特殊的子区域划分－－在这种划分中，如果在其中任意两个（没有直接相联的）顶点之间引入一条新的边，都将破坏其平面性。换言之，对于平面上的一个点集 P={p_1, …, p_n}，令 S 为 P 的一个三角剖分，则任何不属于 S 的边，都必然与 S 中已经存在的某条边相交[1]。

按照这一定义，三角剖分的存在性是显而易见的。我们的问题就是求出这样的一个三角剖分，并将其在图形界面上加以显示。

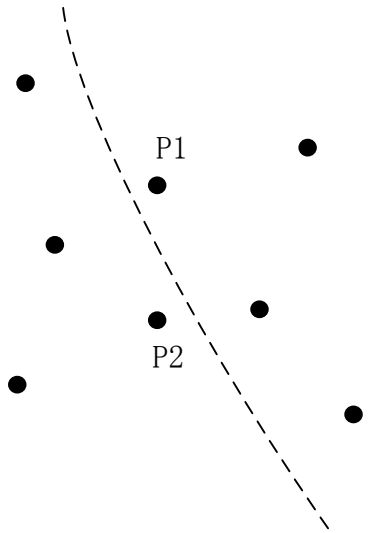
2　算法及原理

2.1　三角剖分

三角剖分的算法流程图如下：

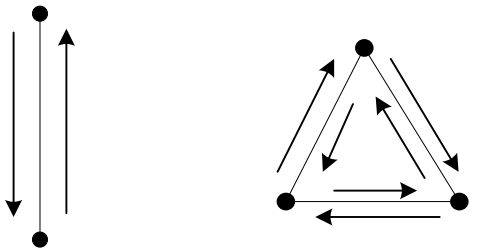


其中第四部分为算法的核心部分，采用分而治之的思想，将一个点集按照其 X 轴坐标，不断地分成左右两部分，如果 X 轴坐标相等，则按 Y 轴坐标分为上下两部分，如下图所示：



注意：P1，P2 点 X 轴坐标相等，是按照 Y 轴坐标进行划分。

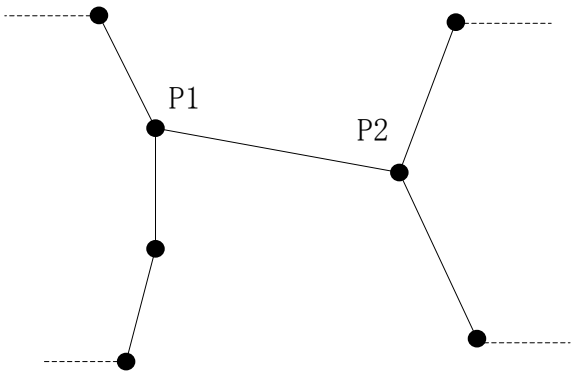
直至只有两个点或三个点的基本类型的情况(任何大于一个点的点集都可以表示为两个点和三个点的组合)，基本类型的表示图如下：



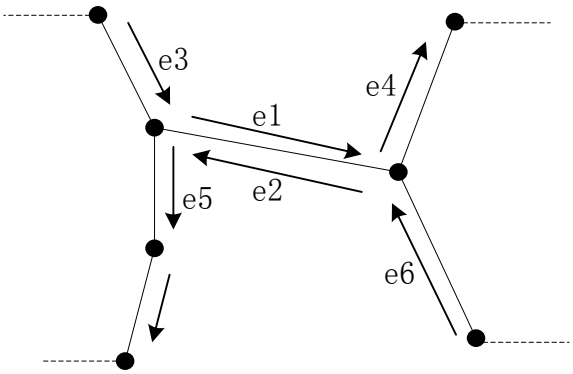
基本类型一　　基本类型二

其中，每一条单边都可以用一条 DCEL 记录表示，故在基本类型一中，有两条 DCEL 记录，有类型二中，有六条 DCEL 记录。

在构造好了基本类型后，要对它们进行合并，即 Merge 算法，每次找到左半部分的最右方（当存在多个最右方点时，取最上）的点，右半部分的最左方（当存在多个最左方点时，取最下）的点，首先连接这两个点，如下图所示：

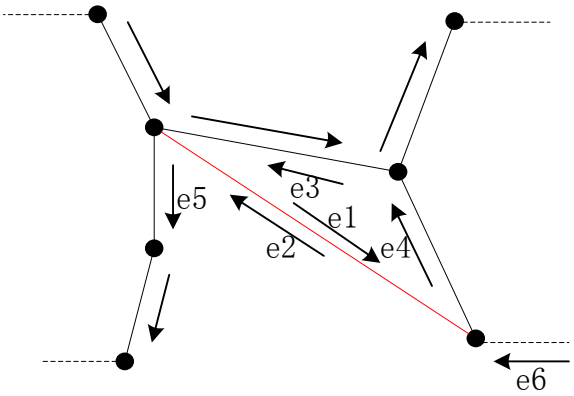


并处理其中的 DCEL 记录，即：



e1->twin=e2，e2->twin=e1，e3->next=e1，e5->pre=e2，e4->pre=e1，e6->next=e2.

接着按照构造凸包时的合并算法一样，进行三角剖分，并处理 DCEL 结构，我们介绍其中的某个状态：



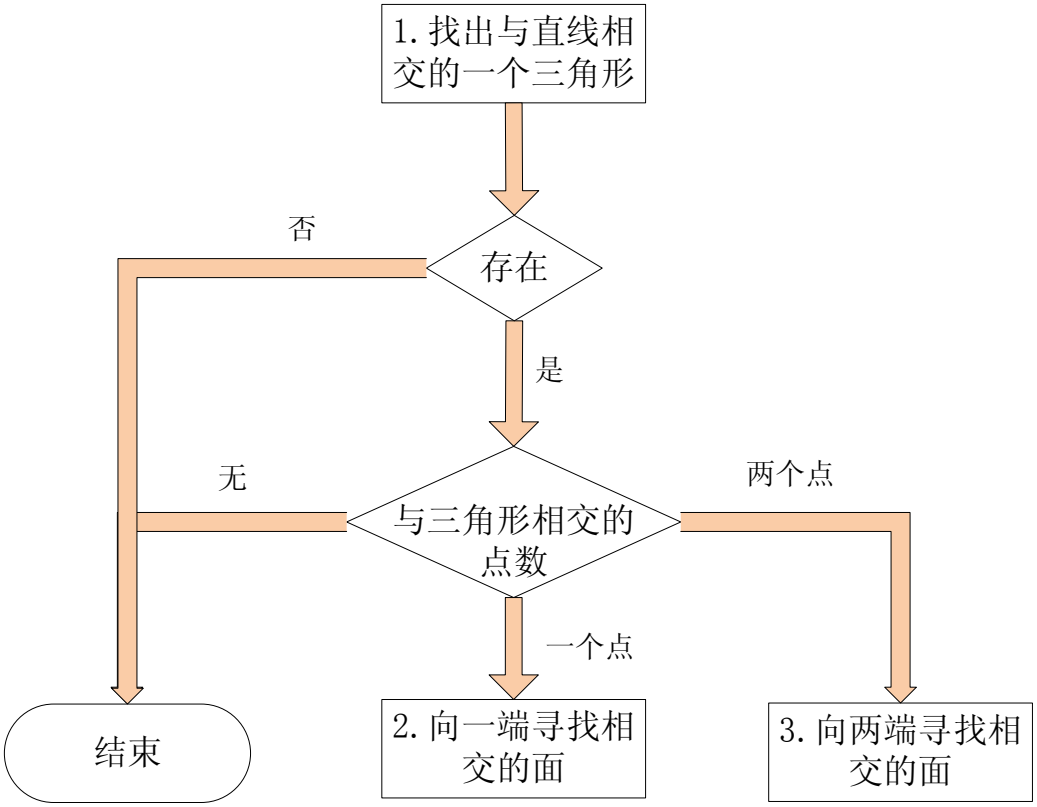
红线为当前所需处理的直线，同样，加入这条线后，处理 DCEL 记录：

e1->twin=e2，e2->twin=e1，e3->next=e1，e5->pre=e2，e4->pre=e1，e6->next=e2.

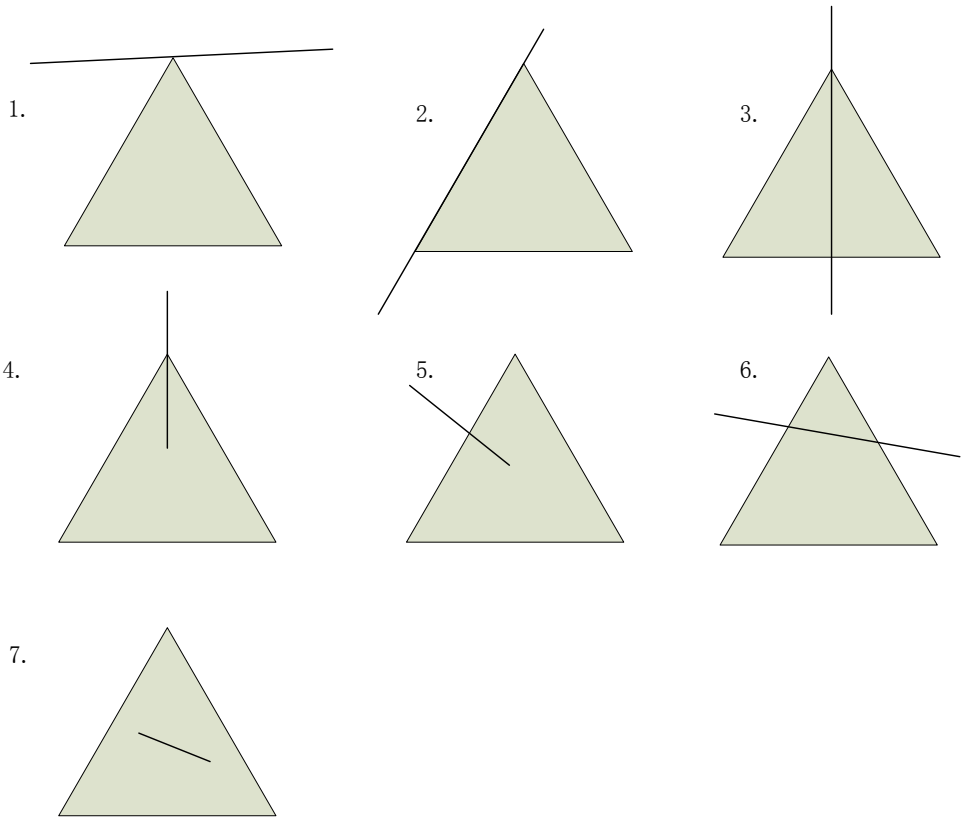
逐条边地加入，直至所有的直线都加入，此时即完成了合并的程序，即 Merge 算法。

2.2 直线穿越

与介绍的直线穿越算法不同，本算法通过先找到任何一个与直线相交的三角形，再分别向两边穿越，最终找出所有经过的三角形，流程图如下：

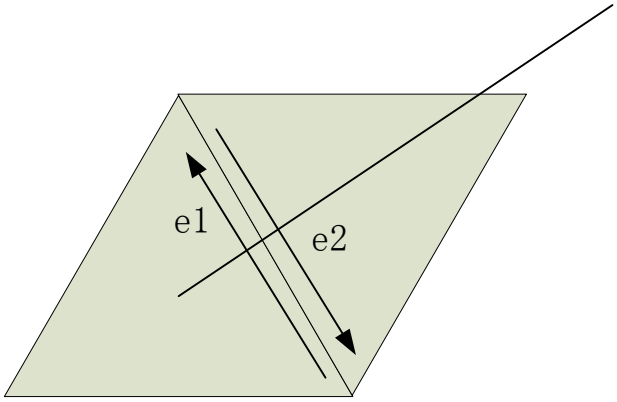


找出与直线相交的第一个三角形面的做法是通过遍历所有的面(除了最外的不封闭的面)进行的寻找的，当不存在时，即直线不穿越任何面。对于所有的三角形面，与直线有交集的情况有如下几种：



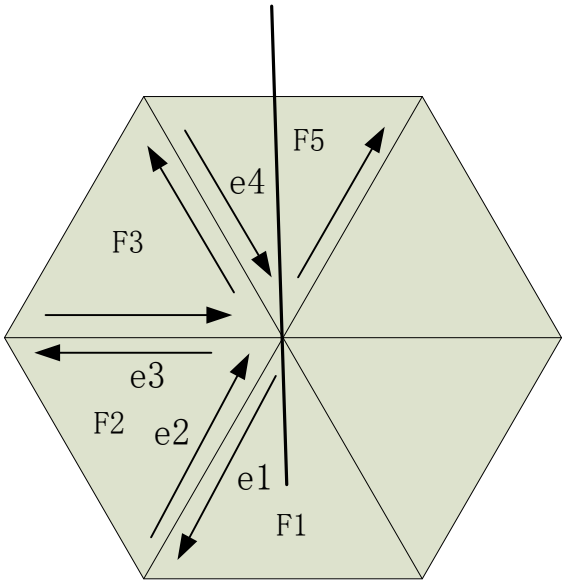
其中，第 1，2 种情况不能算在穿越三角形面的情况之内，3，4，5，6 种情况可分为三种，一种是有两个点在三角形里（包括位于三角形的边或点上），第二种是有一个点在三角形里，第三种是没有点在三角形里。对于第一种（第 7 种情况），不需要进行穿越处理；对于第二种（第 4，5 种情况），需要向一端进行穿越；对于第三种（第 3，6）种情况，需要向两端进行穿越。

穿越时，也分两种情况，一种是通过边进行穿越，另一种是通过点进行穿越，第一种情况如下图所示：



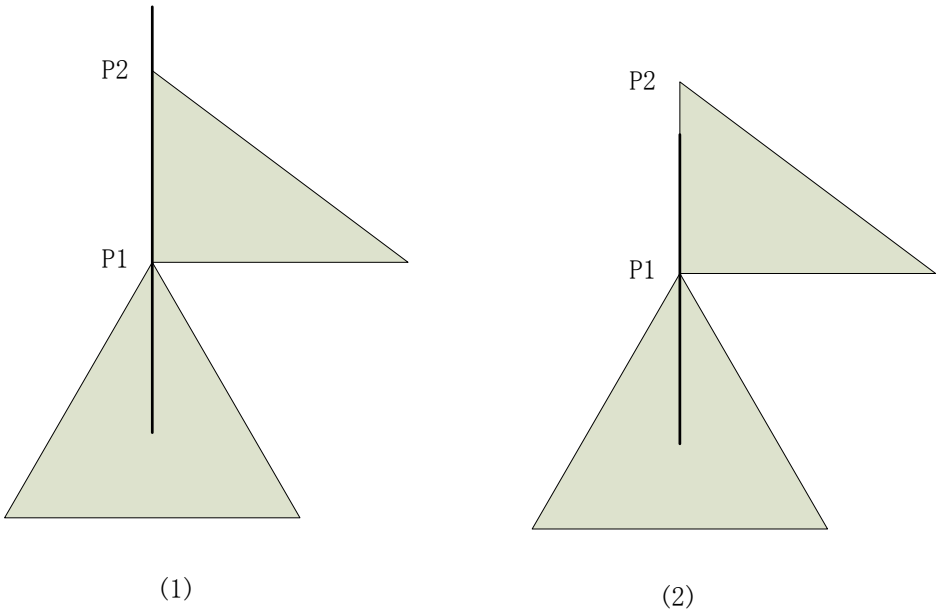
通过求出 e1 的对偶边 e2, 就可以找到下一个面。

对于第二种情况：



此时可以通过 e1 的对偶边得到下一个面 F2，通过对偶边的下一条边的对偶边（e1->twin->next->twin）可以得到 F3，不断寻找下一条边择偶边，可以把围绕此点的所有的三角形遍历，在遍历的过程中寻找与直线相交的三角形，如果找到，则退出遍历，如上图，当找到 F5 时，则找到了需要找的面。

寻找的过程中，会遇到退化的情况，比如点穿越时，会遇到直线与边重合的情况，此时只需要对边另一端的点继续做点穿越即可（也可以存在线不够长，无法到达下一个点的情况），如下图所示：



(1)　　(2)

对于第 1 种情况，只需继续对 P2 进行点穿越即可，第 2 种情况下即可停止寻找下一个面。

通过穿越，最终无法找到与直线相交的三角形或已经到达最外面不封闭的面的时候，即认为已经找到此方向所有与直线相交的面。

3　系统设计

3.1　开发环境

Windows XP + Visual studio 2008。

3.2　编译与链接

工程中使用了 MFC 自带的链接库，不需要特殊操作。

3.3　总体框架

系统主要分为三大模块：

① 系统框架模块

这部分是整个程序框架，负责数据的输入及输出，与其余两大模块进行连接发，完成所有控件操作。

② 核心算法模块

这部分主要实现核心算法，对输入点集进行三角剖分，并将结果返回给系统。

③ 图形绘制模块

这部分主要完成图形界面的绘制，实现对算法结果的图形演示。

4　数据结构

4.1 点的结构

```
typedef struct point
{
    int x;
    int y;
}Point;
```

4.2 边的数据结构

```
typedef struct edge
{
    Point start;
    Point end;
}Edge;
```

4.3 边的双链接表结构

```
typedef struct Dcel_Edge
{
    Edge e;
    Point origin;
    Dcel_Edge *twin;
    int face;
    Dcel_Edge *next;
    Dcel_Edge *pre;
}D_Edge;
```

4.4 点的双链接表结构

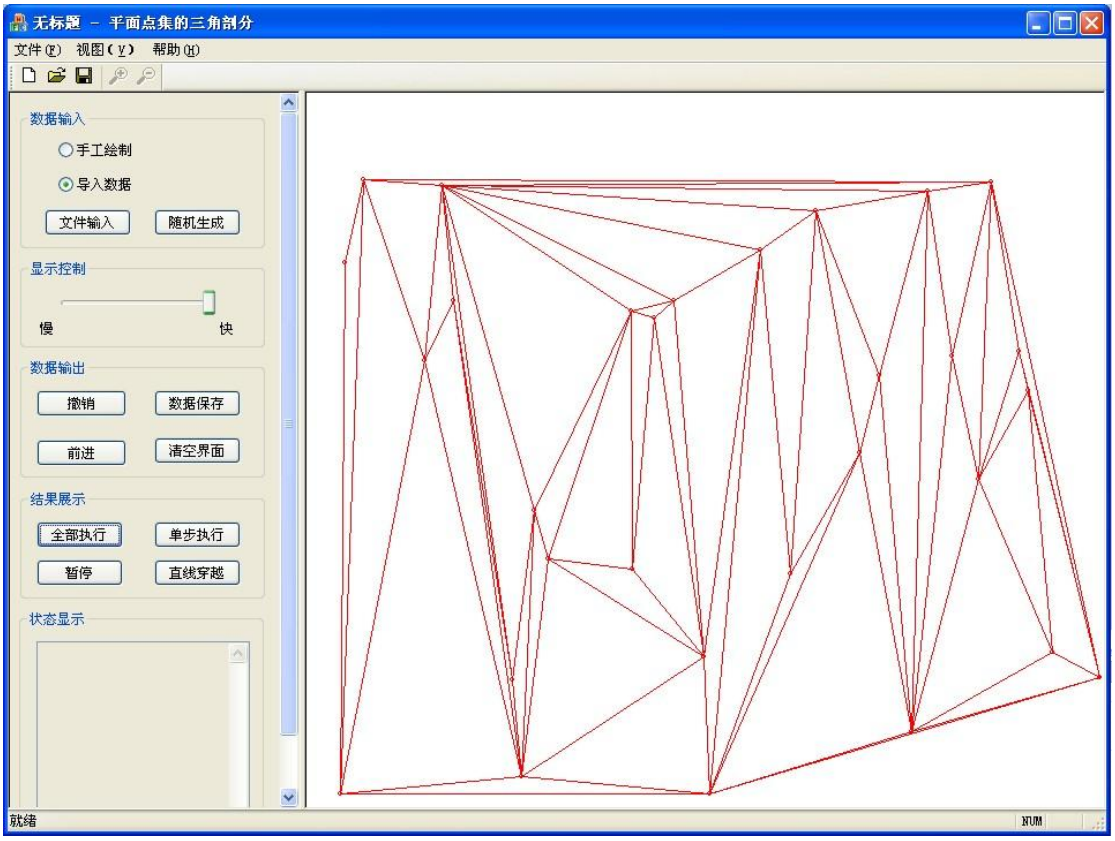
```
typedef struct Dcel_Point
{
    Point v;
    D_Edge *e;
}D_Point;
```

4.5 面的结构

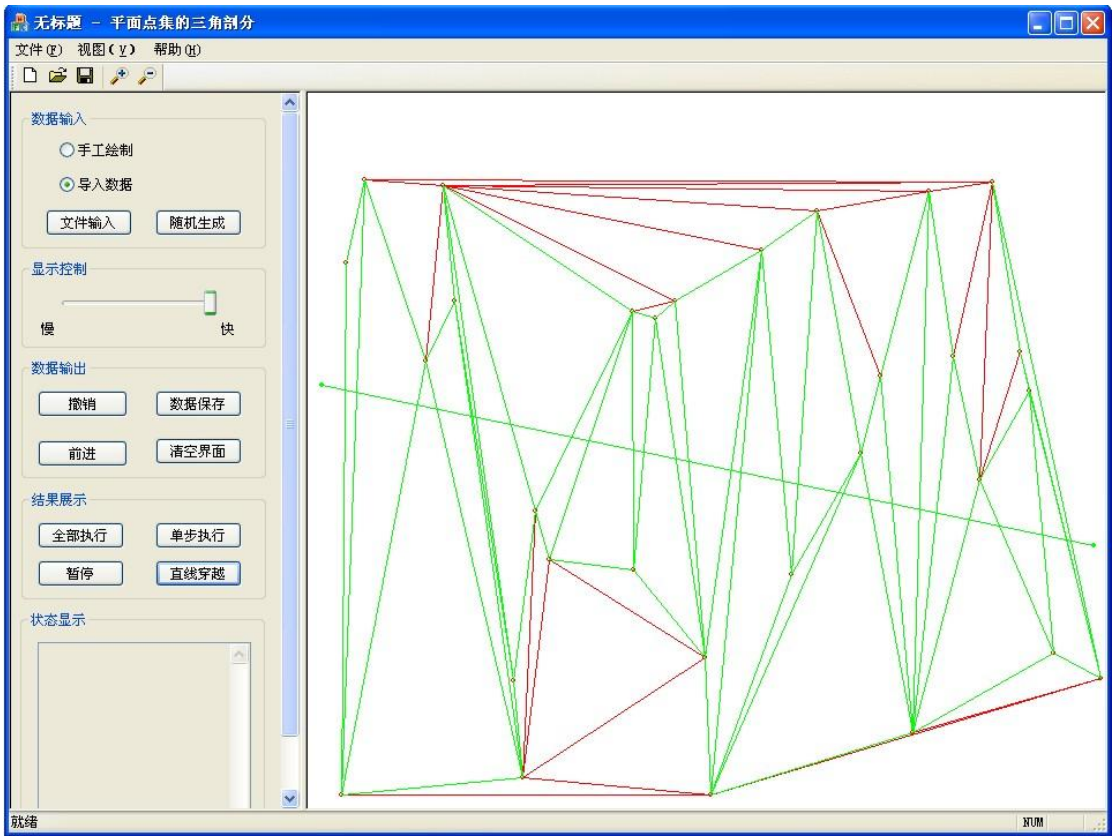
```
typedef struct face
{
    D_Edge *Edge_ptr;
}Face;
```

5 运行结果与分析

5.1 一般情况下的结果展示



30 个点的三角剖分

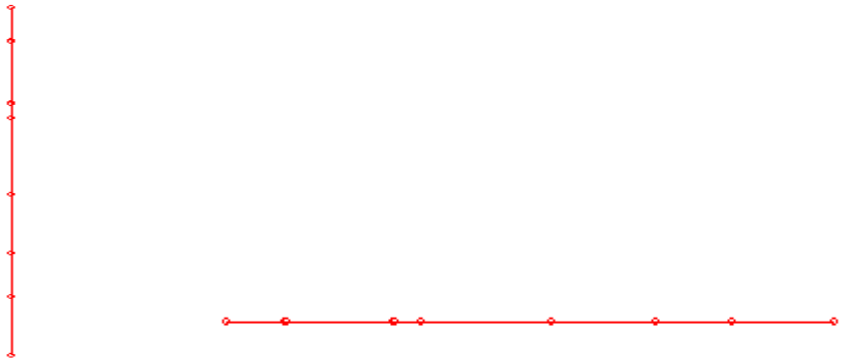


剖分后的直线穿越

5.2 退化与临界情况的测试

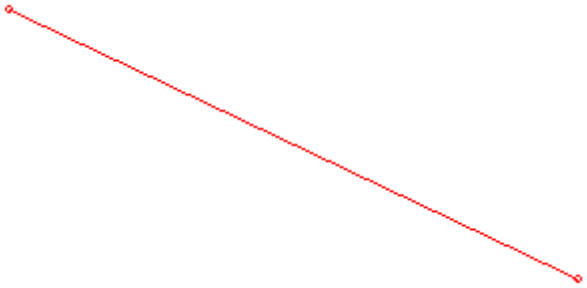
5.2.1 输入点全共线

在这种情况下，将不做剖分，直接输出直线。



5.2.2 点数不足 3 点

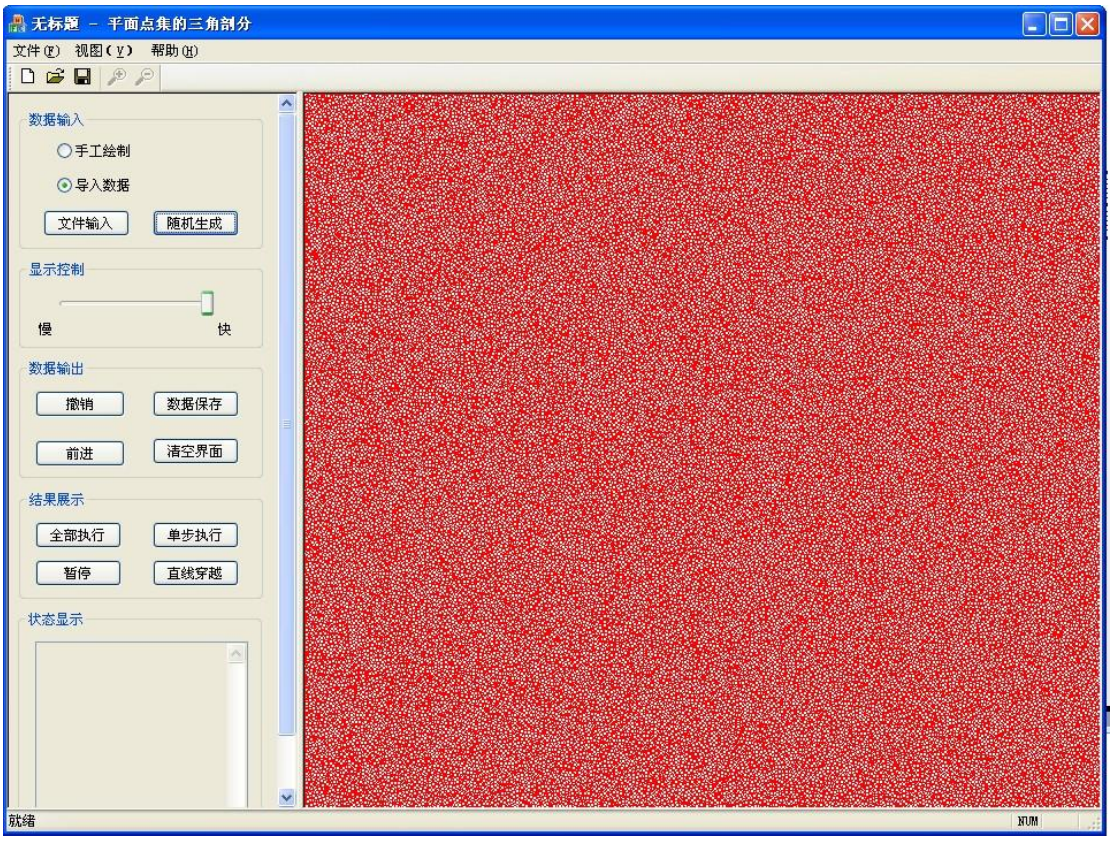
2 个点的时候，不做剖分，直接输出直线。



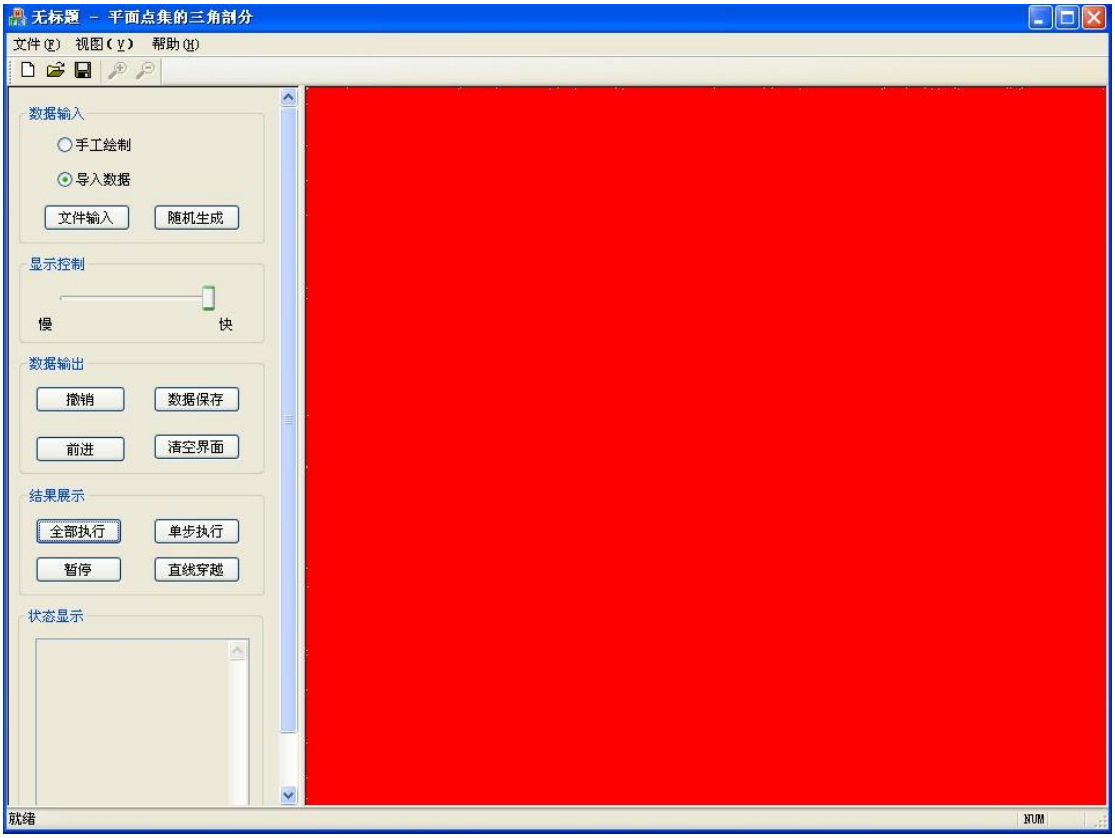
1 个点的时候，不做剖分，直接输出点。

5.2.3 压力测试

随机生成 100 万个点，能够运行通过。由于没有做图像的放大功能，在这种情况显示的结果基本看不清楚。这是我们的一个不足之处，今后要重点改进。



生成 100 万个点后的效果



剖分后的结果

6 不足与展望

6.1 知识方面的不足

1) 由于本组三人均不是图形学或图像处理方向，因此在这方面的基础知识比较薄弱。
2) 小组成员中的 2 人之前均未进行过 VC++方面的编程开发，致使项目是在边学语言边写程序的情况下完成的，影响了项目的实现效果。

6.2 算法方面的不足

1) 本算法只实现了基于分治策略的三角剖分，算法主要参考了邓老师关于凸包算法的课件中的“Divide-and-conquer (2)”一节。这个算法没有对三角形状进行限定，因而得到的三角形状不够美观，也不够实用。
2) 在图形显示的过程中未采用双缓冲技术，致使显示大规模数据的时候产生了一定的延迟。

6.3 界面方面的不足

1) 在显示界面中为加入图像缩放功能，致使在显示大规模数据时不够清晰。
2) 图像显示得不够美观。

6.4 未来工作

经过本项目的开发，小组成员积累了编程方面的相关经验，对计算几何有了更深刻的理解。在未来的工作中，我们将深入学习计算几何的相关知识，熟练自己的图形编程技巧，解决本项目中所未能解决的问题，更好地完成邓老师所布置的第二个任务。

7 参考文献

[1] 计算几何：算法与应用(第 3 版)。(德）Mark de Berg 等著；邓俊辉译。北京：清华大学出版社，2009.8。