第三次实验报告

实验目的

本次实验要求实现二维平面的点定位问题的求解，使用的方法是用梯形图对二维平面进行划分然后建立对应的搜索结构以用于点定位时进行查找.

实验环境

System: Windows XP / Vista　　IDE: VS 2005.net　　界面: MFC

数据结构

为了支持梯形图的构造，我们使用了课本中介绍的有向无环图来构造查找结构。同时，为了支持局部的梯形查找，我们定义了三个结构，分别表示梯形图中的 X-node， Y-node 和梯形的信息。下面具体介绍每个结构如下：

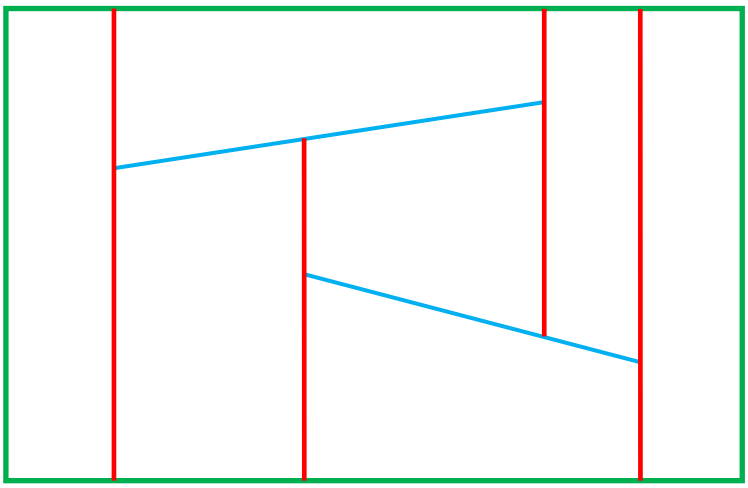

图 1：梯形图示意图

梯形图结构

LocationPoint（图中红线段）

梯形图中的一条竖线，对应查找结构中的 X-node 节点

-- Point2d point;
-- double upper;
-- double lower;
-- LocationTrapezoid* upperTrapezoid;
-- LocationTrapezoid* lowerTrapezoid;
-- int index;

LocationSegment（图中蓝线段）

梯形图中的一条横线段，对应查找结构中的 Y-node 节点

-- Point2d point1;
-- Point2d point2;
-- int index;

LocationTrapezoid

梯形图中的一个梯形

-- LocationPoint* left;
-- LocationPoint* right;
-- LocationSegment* top;
-- LocationSegment* bottom;
-- LocationNode* node;
-- int index;

查找结构

LocationNode

梯形图查找结构的节点

-- LocationNode* left;

```
-- LocationNode* right;
-- LocationNodeType type;
-- union LocationNodeUnion
   {
        LocationPoint* nodeX;
        LocationSegment* nodeY;
        LocationTrapezoid* nodeTrapezoid;
   } pointer;
```

LocationGraph

梯形图查找结构对应的有向无环图

```
-- LocationNode* root;
-- double minx, maxx, miny, maxy;
-- std::vector<LocationNode*> nodes;
-- std::vector<LocationPoint*> xnodes;
-- std::vector<LocationSegment*> ynodes;
-- std::vector<LocationTrapezoid*> trapezoids;
-- LocationPoint boundaryNodeXLeft;
-- LocationPoint boundaryNodeXRight;
-- LocationSegment boudaryNodeYTop;
-- LocationSegment boudaryNodeYBottom;
```

算法描述

不相交随机线段生成算法：

简单生成算法：

简单的思想是每条线段都完全随机生成，生成同时判断是否与已有线段相交，如果相交则抛弃该线段，否则，将该线段加入到线段数据集中。这样的算法每生成一条线段，都要对所有的线段进行遍历，同时当生成线段比较多时，很容易出现相交的情况。鉴于这两点原因，算法会运行的很慢。生成效果方面，因为后生成的线段会受到前面生成线段的影响，同时生成的线段集形状也会比较差，长线段会决定大部分线段分布的方向。

基于区域划分的算法：

根据简单算法的特点，我们提出了一种基于区域划分思想的算法。首先将区域划分为多个矩形的小区域，我们在每个区域“附近”生成长度具有一定限制的线段。这样，生成的线段跨越的只是该矩形周围一定范围内的矩形。根据不同的限制条件，我们在一个矩形内生成线段时，只需要在其一定范围内判断相交即可。这样的算法一方面降低了线段的长度，从而避免了长线段对大面积区域内线段形状的影响，另一方面，算法每生成一条线段，只需要判断周围有限范围内的线段是否相交，不在和所有线段进行相交判断，大大降低了算法的复杂度。用户可以自定义矩形区域的大小或者每个矩形区域内线段的数量，可以改变生成线段的效果。

基于多分辨率的算法：

基于区域划分的算法有一个问题，就是不会出现长线段，在某些情况下，可能我们需要长线段来使我们的算法看起来更“随机”。对于这样的要求，我们可以基于多分辨率的思想来生成随机线段。首先，使用四叉树将整个区域进行划分。树的每一个节点（包括子节点和父节点）都代表一个矩形区域。对于这颗四叉树，由上倒下的在每个区域内生成一定数目的线段。在一个节点内生成线段时，只需要在对其所有的祖先节点内的线段进行相交判断就可以了。

在我们的程序中，实现了简单生成算法，和两种基于区域划分的算法，根据生成线段的数目不同，我们使用不同的算法来产生随机不相交线段，为了保证效率，生成数量越多，效果越不随机。

梯形图构造算法：

梯形图的构造基本采用了课本中的算法。算法中的几个重点部分介绍如下：

新插入一条线段时处理梯形的查找过程

为了支持新插入一条线段查找该线段穿过的梯形，在每个 X 节点对应的结构 LocationPoint 中，保存了这条竖直线右上方和右下方（可能相同）指向的梯形。这样，每次判断穿过的 X 节点和新插入线段的上下关系，就可以找到线段穿过的下一个梯形

梯形图查找结构的析构

由于梯形图查找结构是一个有向无环图，两个不同的父节点可能指向同一个子节点。如果用递归的方法析构图的节点，可能会造成同一个节点被析构两次，使得程序崩溃，为了防止这种情况的发生，可以将所有的图的节点唯一的保存到一个 vector 中，最后统一的析构所有的节点。

点定位算法：

点定位算法比较简单，从图的根节点出发，递归的判断定点的位置，如果是 X 节点，判断测试点在 X 节点左侧还是右侧，如果是 Y 节点，判断在上方还是下方。最终定位到一个梯形节点中。

用户手册

提供功能：

-- 从文件中导入线段，随机生成不相交线段，手工生成线段，线段的编辑

-- 生成梯形图，生成梯形图的动画过程，交互式添加线段同时动态生成梯形图

-- 点定位，点定位的动画过程，批量点定位

程序特点和可能存在的问题

-- 程序提供了大量的动画和演示功能，提供了友好的界面操作和提示

-- 但是由于程序使用 mfc 进行界面的演示，在数据量比较大时，绘制效率比较慢，因此，在数据量比较大是，程序自从关闭了现实功能，而且数据量较大时，也关闭了动画功能

-- 由于时间比较急迫 ,没有对动画进程进行处理 ,在动画演示时 ,不能进行其他操作，也不能暂停和结束动画进程

-- 程序算法具有较好的稳定性，但是当数据量很大是，仍然可能会产生一些未知的问题

菜单

-- **Input** **产生线段数据**

-- Load From File 从文件中导入线段数据

-- Store To File 将线段数据到出到文件

-- Input Random 随机产生不想交线段数据

-- Input By Hand 手工交互产生线段数据

-- **Edit** **数据编辑**

-- Select Line Segment 选择单个线段，选中后，可以对线段进行移动和删除操作

-- Delete 删除选中线段

-- Clear Line Segments 清除所有线段数据

-- Clear Points 清除测试点数据

-- Clear All 清除所有数据

-- **View** **显示控制**

-- Enlarge 局部放大，单击后在屏幕上拖动鼠标，将放大至选择区域

-- Enable/Disable Draw 屏幕显示开关，打开或关闭屏幕显示，当数据量大时，为了提高预算效率，可以关闭显示

-- Show Point 控制是否显示测试点数据

-- Show Input Line Segment 控制是否显示输入线段

-- Show Trapezoid 控制是否显示生成梯形图

-- **Trapezoid** **生成梯形图**

-- Construct Trapezoid 生成输入线段的梯形图

-- Incremental Construct Trapezoid 递增式的生成梯形图，单击该按钮，用户在界面上输入线段，动态的插入相应的梯形图

-- Animation Construct Trapezoid 输入线段梯形图的动画生成过程

-- **Location** **点定位操作**

-- One Point Location 定位鼠标单击点所在的梯形，鼠标单击显示区内一点，显示该点所在的梯形和他的上边

-- Animation Point Location 点定位的动画演示过程

-- Animation Point Location2 另一种动画演示过程

-- Random Point 随机产生点

-- Load Point File 从文件中导入测试点数据

-- Store Point File 将测试点数据导入文件

-- Point Location 定位测试点的位置，并在屏幕上显示最后一个点所在的梯形和上边

-- Store Result 将测试点位置数据导出到文件

试验结果：

鲁棒性测试：

对于算法，我们进行了相应的鲁棒性测试，主要是一些存在共点和点在直线上等特殊情况的测试。测试用例保存在 data 文件夹中，部分测试结果截图如下：

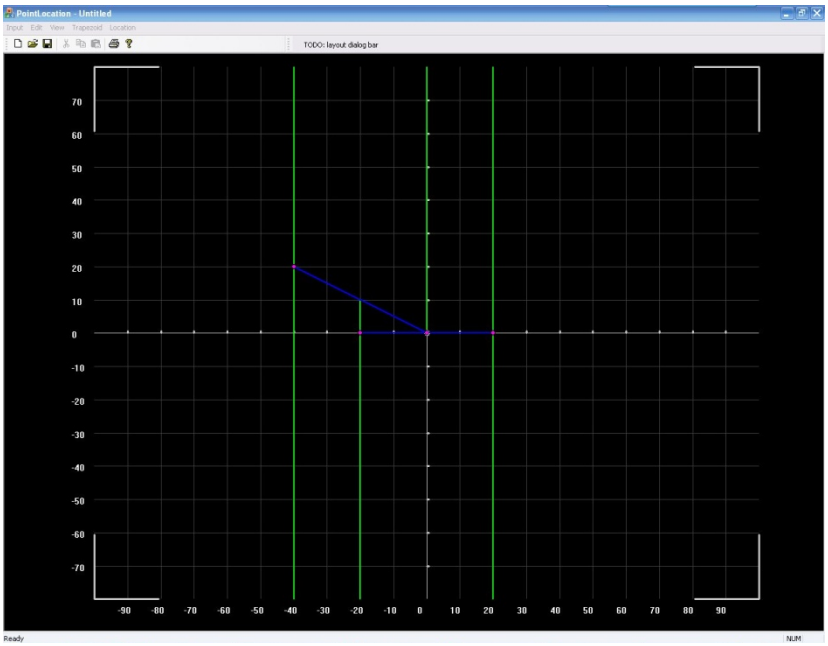

图 2：点在直线上的情况

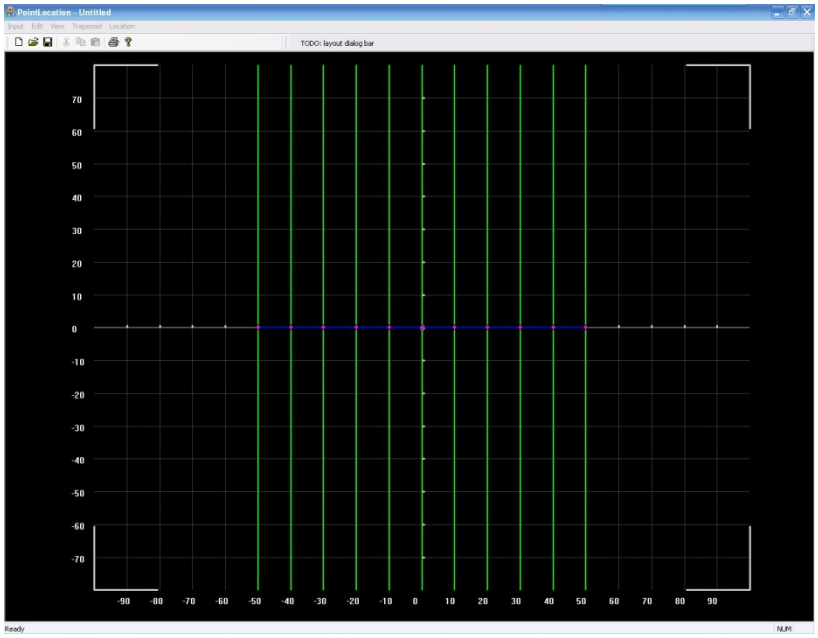

图 3：多点共线的情况

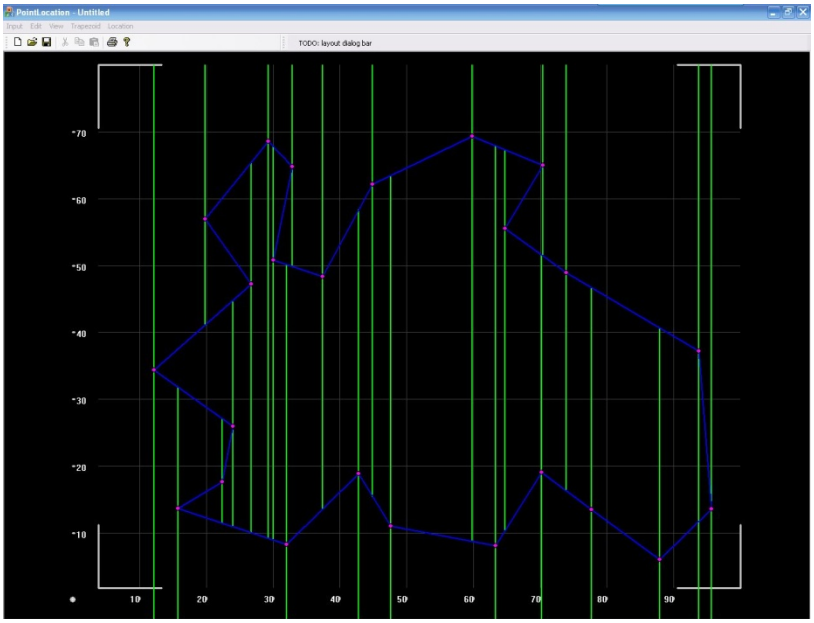

图 4：多边形的梯形图

效率测试：

对于作业中的三个主要算法，我们进行了大数据量的测试。选取的测试集如下：

随机生成线段数：5000，10000，20000，50000，100000

测试点数：10000，100000，1000000

因为随机生成线段算法性能，我们的平台最多生成了 100,000 条不相交线段，为测试性能，还是用了其他小组的 200,000 和 500,000 的测试数据，但是因为生成随机线段的性质不同，在生成梯形图和点定位时的复杂度也不完全匹配，最终的测试时间只用作参考。

主要的测试性能如下表格所示：

随机线段生成：

生成线段数	5,000	10,000	20,000	50,000	10,0000
消耗时间（微妙）	500	953	500	1750	2656

梯形图构造：

线段数	10,000	20,000	50,000	100,000	200,000	500,000
消耗时间（微妙）	63	141	515	1313	1859	5203

点定位：

线段数	10,000	20,000	50,000	100,000	200,000	500,000
10,000 测试点	16	31	110	188	31	32
100,000 测试点	250	406	1171	1875	282	421
1000,000 测试点	2656	4549	13031	19000	2766	3531

动画演示：

对于梯形图生成和点定位，都提供了两种不同的动画展示方案。

对于梯形图生成：

数据量比较少时（小于等于 50 条线段），可以动态显示每插入一条线段后，梯形图每条线段逐渐的变化生成过程。

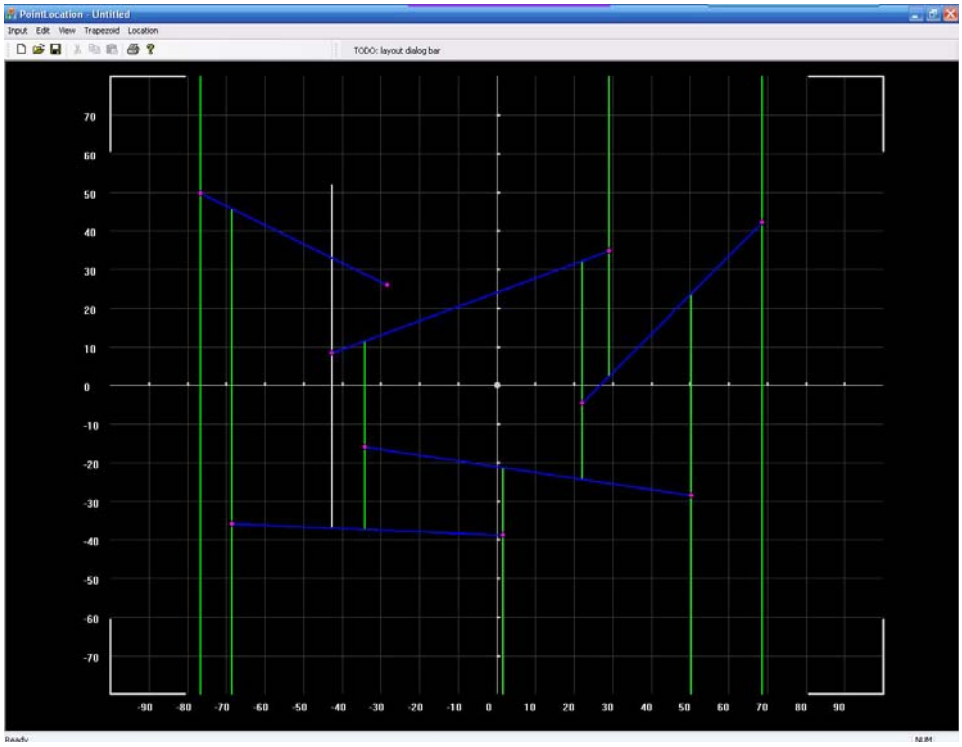

图 5：生成梯形图动画 1 截图

数据量较多时，为了加快显示过程，每插入一条线段，对梯形图进行更新

图 6：生成梯形图动画 2 截图

对于点定位：

第一种显示方案通过缩小屏幕范围来显示动态查找过程，最终找到点所在的区域

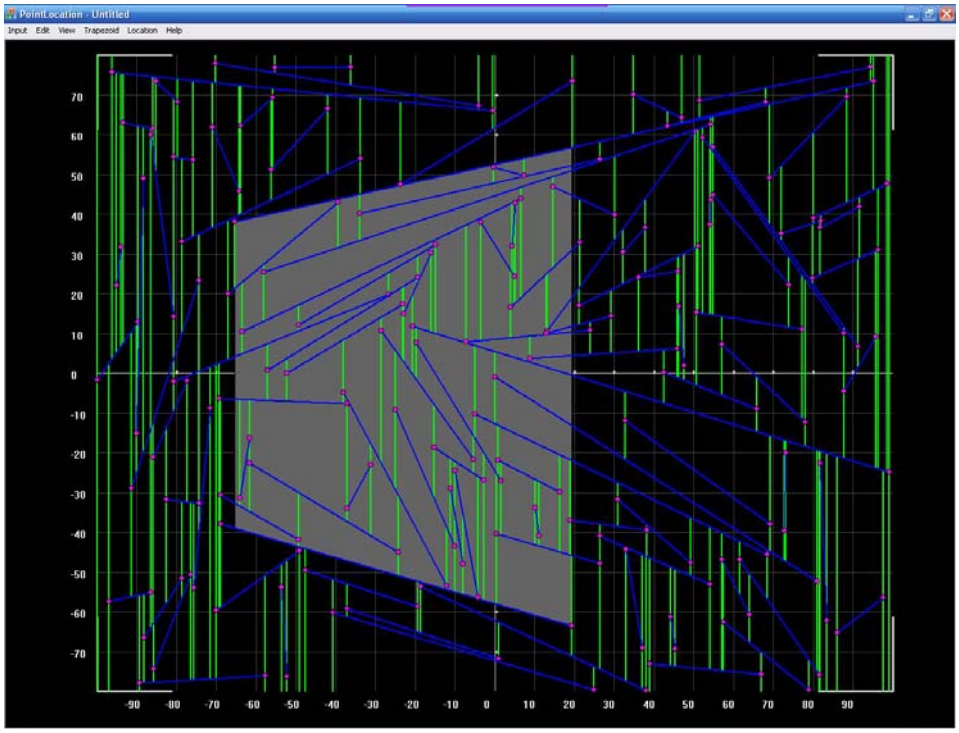

图 7：点定位动画 1 截图

第二种显示方案更接近于查找的过程 ,逐步定位该点可能落到的梯形 ,缩小梯形的范围，最终找多所在的梯形.

图 8：点定位动画 2 截图