梯形图算法的实现

- 实验内容

对于平面上一组互不相交的线段集，构造其梯形图以及对应的一个查找结构。梯形图组织成类似 DCEL 形式，查找结构为有向无环图形式。

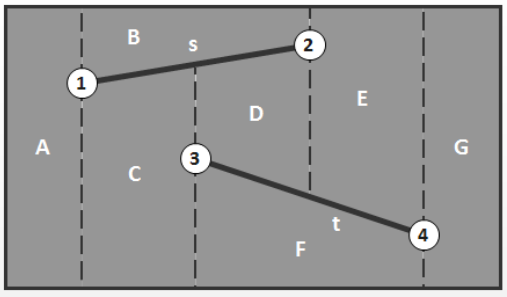
利用查找结构，在梯形图中进行点定位查找。这里点定位查找定义为：在输入的线段中，找出与发自查询点、垂直向上的射线相交的第一条线段，并返回其编号。

在这里做约定：

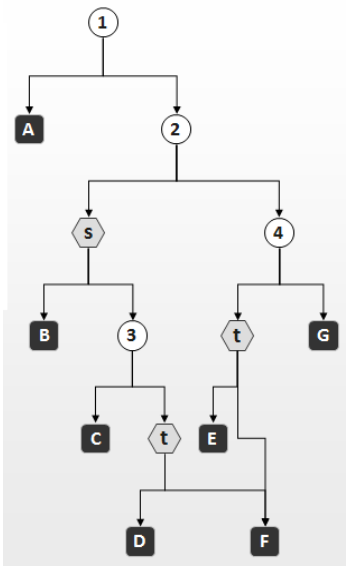
1. 射线为闭，即包含起点本身；
2. 输入线段均为左闭右开，即包含左端点但不包含右端点；
3. 假想为通过逆时针地轻微转动坐标轴，消除垂直线段，即垂直线段下闭上开；
4. 共左端点的线段，以右端点的纵坐标为序；
5. 若射线不与任何线段相交，则返回“inf”标识。

结合实验要求，本实验实现了以下几组功能：

* 通过鼠标输入、移动和删除线段；
* 线段集的随机生成；
* 梯形图构造过程的可视化演示；
* 在查找结构中搜索过程的可视化；
* 线段数据的批量导入和导出；
* 批量查询以及执行时间统计；
* 针对查找结构高度与查找效率的实验分析



(a) 梯形图的示意图



(b) 左图对应的查找结构

图 1　平面上互不相交的线段集的梯形图[i]

如图 1 所示，平面上的两条不相交的线段 s（线段端点为 1、2）和 t（线段端点为 3、4），

进行梯形化后，形成的梯形有 A~G，见图 1(a)，其构造的查找结构如图 1(b)所示，圆形为线段端点，六边形为线段，方形为梯形区域。

• 实验目的

本次实验是对计算几何中一种经典的梯形图问题进行了算法实现。通过本次实验，进一步了解区域点定位的含义，提高对算法理解和实现，以及算法各种退化情况处理的能力。

• 算法介绍

一. 数据结构设计

此次实验中需要维护两组数据，一个是 TM(S)(Trapezoidal Map)，一个是 SS(S) (Searching Structure)，S 是输入的线段集。

TM 数据结构

TM 主要采用了类 DCEL 的数据结构，针对随机增量法的需求和特性，我们采用了如下 4 种基本的元素来表示一种类 DCEL 的数据结构，分别是 PointTopology, Trapezoid, Segment, NeighborTraps。

顶点信息：

```
class PointTopology
{
    bool isLeftPoint_;              // 该端点是否为线段上的左端点
    PointTopology *brotherPoint_; // 所属线段的另一个端点
    Segment *seg_;                  // 所属线段
    NeighborTraps *head_;  // 围绕该点的逆时针排列的Trapezoid链的头指针
    Point2d *coord_;               // 该端点的坐标指针
};
```

梯形图

```
class Trapezoid
{
    Segment *topSeg_;           // 梯形图上线段
    Segment *bottomSeg_;        // 梯形图下线段
    PointTopology *leftPoint_;  // 梯形图左顶点
    PointTopology *rightPoint_; // 梯形图右顶点
    Point2d corner[4];          // trapezoid 角落的四个顶点
};
```

线段

```
class Segment
{
    PointTopology *leftPoint_;     // 左端点
```

```
    PointTopology *rightPoint_;    // 右端点
    size_t serialNumber_;          // 编号
};
```

邻居

```
class NeighborTraps
{
    SquareNode *squareNode_;   // ss结构SquareNode指针
    NeighborTraps *next_;      // 循环链表中的下一个NeighborTraps
    NeighborTraps *prev_;      // 循环链表中的上一个NeighborTraps
};
```

PointTopology类主要用来存储一个端点拓扑信息以及基本的数据信息，中的isLeftPoint用来判定一个顶点是否是一条线段的左端点，在退化情况的处理中会用到这个信息，brotherPoint_用来找到一条线段的另一个端点，在垂直共线ToLeft判定的时候会用上，seg_指向的是该端点所位于的线段，一个端点有且只能位于一条线段之上，head_用来指向与该顶点相关梯形图，如图2所示，通过该变量可以实现相邻梯形的O(1)跳转。Corner用来存储梯形图的四个顶点的左边，在绘制的时候能用到。

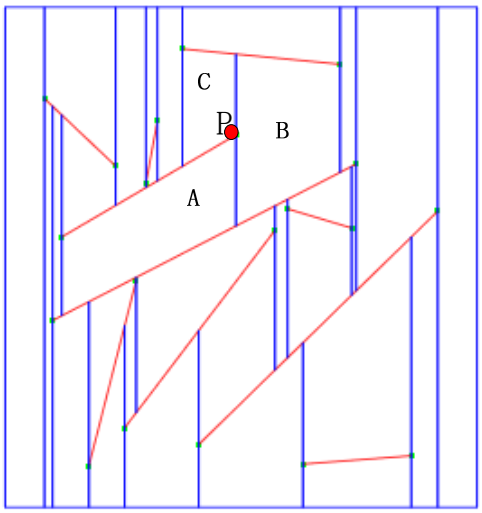


图2　P的相关梯形图A,B和C

Trapezoid用来存储单个的梯形，topSeg_存储梯形的上边界线段，bottomSeg_存储梯形的下边界线段，leftPoint_存储的是梯形的左边界点，rightPoint_存储的是梯形的右边界点，因为梯形图中的每一个梯形的左右边界都是垂直线，因此通过以上四个量可以计算出梯形的边界上的顶点。梯形的结构如图3所示。

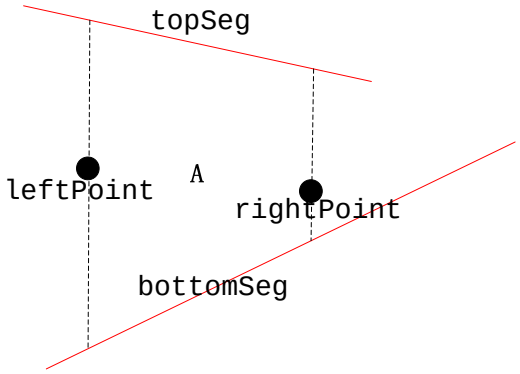


图3　梯形A的上下线段和左右端点

Segment用来存储一条直线段相关的信息，leftPoint_，rightPoint_分别记录线段的左右端点，serialNumber_是线段的编号，梯形图中的每条线段都有唯一的标识，因为点定位要求返回某点的向上发出垂直线碰到的第一条线段。

NeighborTraps用来存储某一线段的端点相关的梯形双向循环链表节点，用于实现在TM结构中快速的查找到相邻的梯形。一个顶点相关的所有的梯形在形成了一个环，如图X所示的点P相关的梯形A,B,C形成了如图4所示的环状结构。

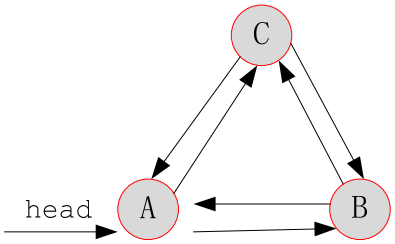


图4　图2中与顶点P相关的梯形形成了一个环，head为头指针

SS 数据结构

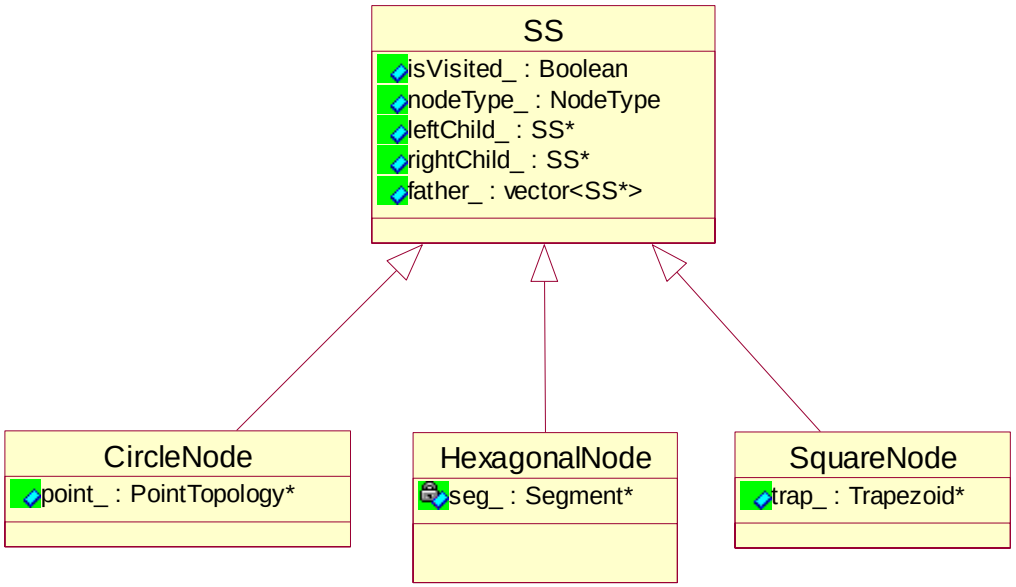


图 5 SS 结构的类之间的继承关系

SS 是一种类二叉树的用于梯形图定位的空间划分搜索结构。梯形图按两种原则来进行划分的，一种是按照线段的端点的 X 坐标来进行划分，在 SS 结构上形成了 X-NODE 节点（我们习惯叫它 CircleNode），另一种原则是按照直线段来划分的，在 SS 结构上形成了 Y-NODE（我们习惯叫它 HexagonalNode），最后所有的叶子节点都是单个的梯形（我们习惯叫它 SquareNode）。

因此梯形图的 SS 搜索结构总共有三种节点，这些不同的节点有一些共同的属性，因此设计成了如 X 所示的类的继承结构，SS 是三种节点的基类，isVisited_标记该节点是否被访问了，因为同一个 SquareNode 可能同时被多个 HexagonalNode 指向，因此为了访问遍历时不要进行重复的节点遍历。nodeType_用来标识该节点的类型，因为不用的节点，在遍历时的路径判定也是不同的，leftChild_左孩子的指针，rightChild_右孩子指针，father_为该节点的父亲，一个 SquareNode 可能有多个父亲，在随机增量进行构造的时候，当一个节点被替换成另一些节点的时候，通过 father_来重新来维护整个树的结构。

随机增量法构造

我们采用了邓俊辉老师课件上的随机增量法来进行梯形图的构建，该算法的基本伪代码如下：

```
Algorithm        ConstructTrapezoidalMap(S)
    //  initialization
        create a bounding box R that contains all segments of S;
        let TM(S) = the trapezoidal map with the single trapezoid R;
        let SS(S) = the search structure with a single node for R;
    //  randomized incremental construction
        for each segment s = pq ∈ S        //chosen randomly
        find the trapezoid Trap containing p;        //this is itself a PL query
        repeat    refine Trap into no more than 3 trapezoids;//perhaps 4 for once
            update TM(S) and SS(S);
            Trap = the trapezoid next to Trap along s;
    until (NULL == Trap);
```

ConstructTrapezoidalMap 大概的过程是，先对 TM 和 SS 进行初始化，然后每次随机的选择一条线段，插入到原来的已经构造好的梯形图当中，当插入一条新的线段时，先要找到该线段穿过了梯形图中那些梯形，然后动态的从左端点到右端点顺序的进行更新原有的梯形图和空间搜索结构，一直重复进行下去，直到插入完所有的线段为止。

算法细节

当一条新的线段要插入原有的梯形图时，它会给破坏原有的梯形图和空间搜索结构，但并非所有的梯形和 SS 上的节点都被破坏，新插入的线段只会破坏与它相交的梯形，因此算法的其中的重要一步是通过原有的 TM 和 SS 实现快速的找到该线段所穿过的梯形。

有的新插入的线段只落入了原有梯形图中某一个梯形中，而另一些可能和多个梯形相

交，找到线段所相交的梯形，关键的一点是确定该线段的起始端点位于哪个梯形，然后通过 TM 结构实现快速的查找下一个与其相交的梯形，一直查找下去，直到新插入的线段的右端点落入了某个梯形，该查找步骤结束。在查找的过程中，我们同时也对梯形图的 TM 和 SS 搜索结构进行必要的更新，当查找过程结束时，TM 和 SS 的结构更新同时结束。

在新插入一个线段时，我们可能会遇到四种情况(邓老师的课件提到了 3 中情况，我们把第 2 中情况扩展成分别针对于线段始点和终点两种情况)，分别如下：

情况 1:线段的两端点同时落入了一个梯形中，如图 6 所示：

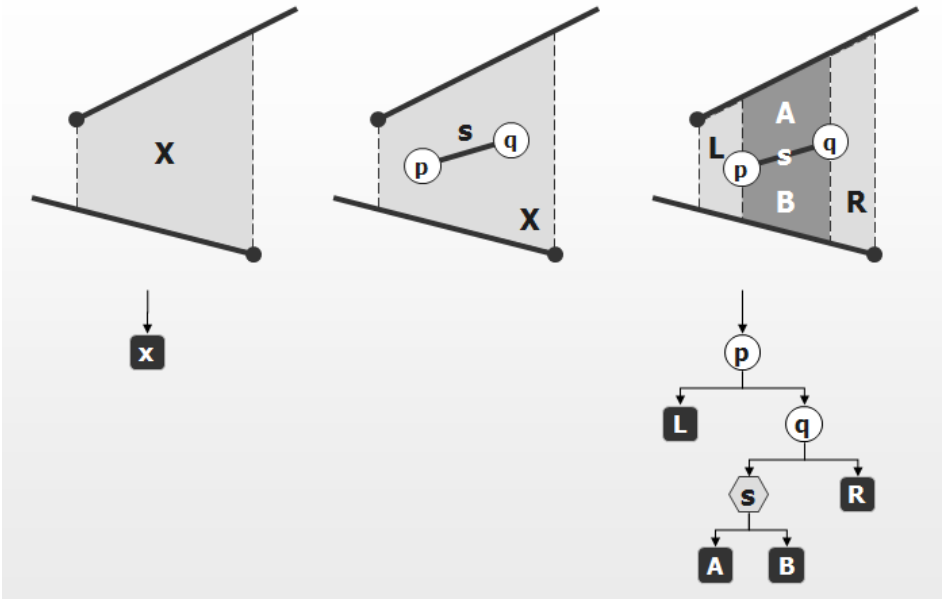


图 6　线段 s 的两端点 p, q 同时落入了 X 梯形中

两个端点同时落入了同一个梯形中，这种情况是最好处理的。

1. 更新 TM：原来的梯形 X 被拆分成 4 个梯形 L, A, B, R，因此只要删除 X，然后建立 L, A, B, R。
2. 更新 SS：原来在 SS 中的 X 节点被替换成 p, L, q, s, A, B, R, 7 个 SS 节点，并且建立如图 X 所示的节点拓扑关系，而且原来 X 节点的父亲，全变成了 p 节点的父亲。

情况 2：线段的端点落入了不同的梯形中，始点落入某一梯形情况，如图 7 所示：

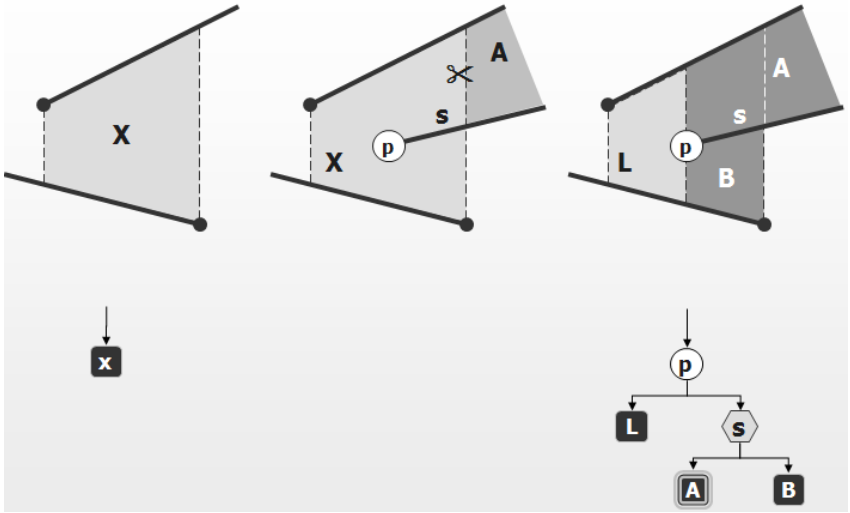


图 7　线段 s 的起点 p 落入了梯形 X 中，但是其终点没有落入其中

1. 更新 TM：原来的梯形 X 被拆成了 3 个梯形 L, A, B，因此只要删除 X，然后建立 L, A, B，并且标记 A 为一个临时的梯形，因为在梯形 A 在接下来的线段穿过的梯形中的一部分合并成一个较大的梯形，具体的细节接下来会详细讲解。
2. 更新 SS：原来在 SS 中的 X 节点被替换成 p, L, s, A, B，5 个节点，并且建立如图 X 所示的节点拓扑关系，而且原来 X 的父亲，全变成了 p 节点的父亲。

情况 3：没有一个端点落入某个梯形，但是该梯形被线段所穿过。如图 8 所示。

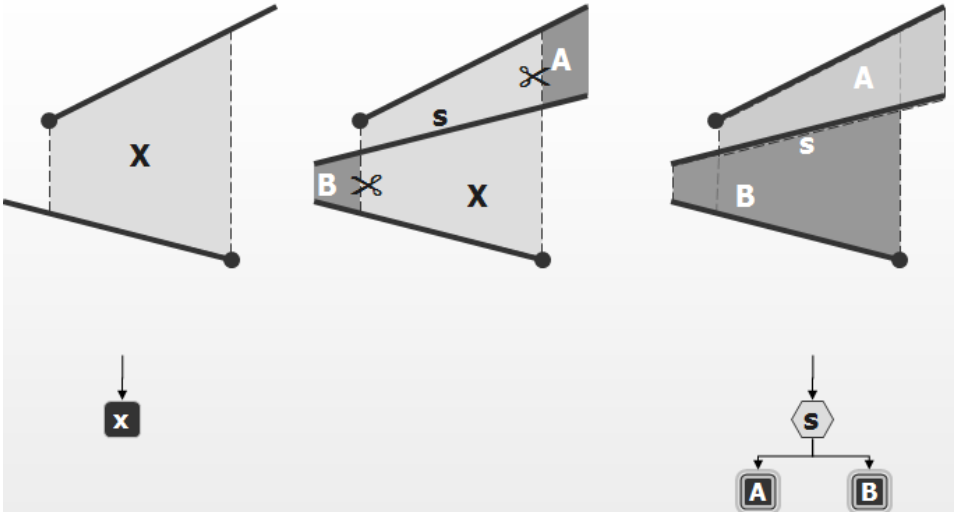


图 8　线段 s 穿过 X 梯形，但是其端点没有落入 X

1. 更新 TM：原来的梯形 X 被拆成了 2 个梯形 A, B，　因此只要删除 X，然后建立 A, B，　并且 A, B 梯形都为临时的，因为其中某一个可能和情况 2 中所形成的临时的梯形合并成一个新的梯形，而且可能某一个可能将来和情况 3 或是情况 4 的某个梯形进行合并，具体的合并细节在接下来会详细讲解。
2. 更新 SS：原来在 SS 中的 X 节点被替换成了 s, A, B，3 个节点，并且建立如果 X 所示的节点拓扑关系，原来 X 的父亲，全变成 s 节点的父亲。

情况 4：线段的端点落入了不同的梯形中，终点落入某一梯形情况，如图 9 所示。

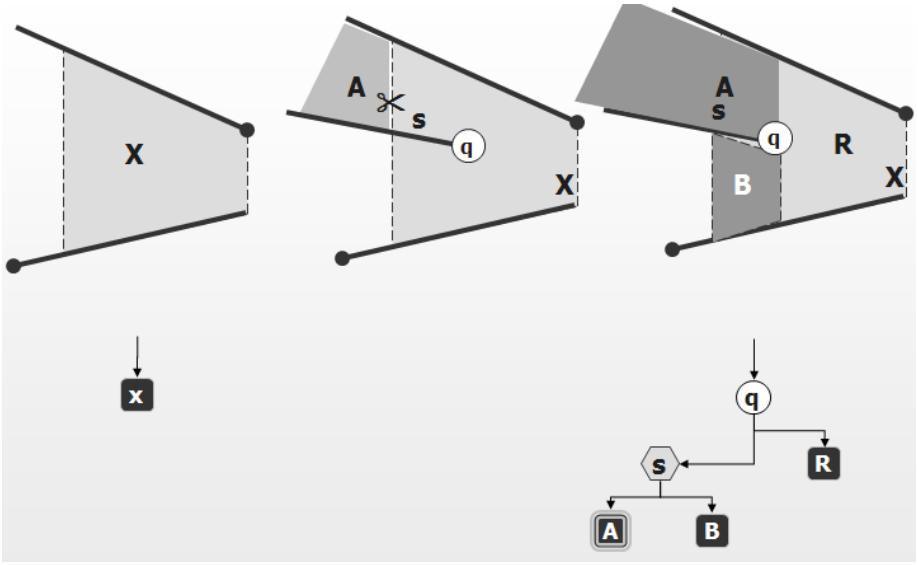


图 9　线段 s 的终点 q 落入 X 中，但是其始点没有落入其中

1. 更新 TM：原来的梯形 X 被拆成 3 个梯形 A, B, R，因此只要删除 X，然后建立 A, B, R。并且标识 A 为临时的，因为其可能和情况 2 和情况 3 中的临时梯形合并成一个新的梯形。
2. 更新 SS：原来在 SS 中的 X 节点被替换成 q, s, A, B, R，5 个节点，建立这 5 个节点的如图 X 所示的拓扑关系，然后把原来 X 的父亲，全变成 q 节点的父亲。

临时梯形的产生与合并分析

临时梯形的产生只可能在情况 2, 3, 4 中产生，合并只可能发生在情况 3，4，下面详细介绍临时梯形的产生与合并。（open 标识为产生临时梯形，close 标识为合并梯形，把位于线段上部分的临时梯形用 topTmp_存储，下部分的临时梯形用 bottomTmp_存储）

1. 情况 2 的临时梯形产生分为 2 中情况。如图 10 所示，(a)图标识梯形 X 的右端点位于线段 pq 的右边，因此 A 为临时梯形，topTmp_ → A，(b)图标识梯形 X 的右端点位于线段 pq 左边，标识 B 为临时变量，bottomTmp_ → B。

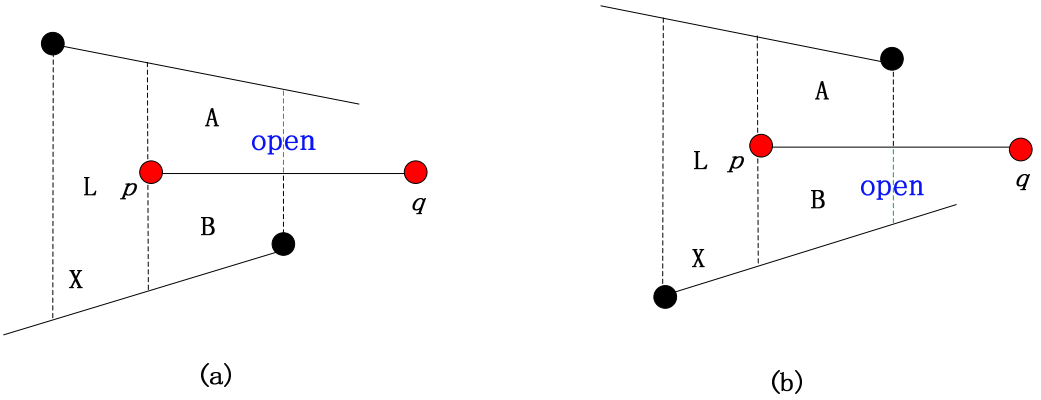


图 10　情况 2 产生临时梯形

2. 情况 3 的临时梯形产生的与合并分为如图 11 所示的 4 中情况。详细情况如表 1 所示。

表 1 情况 3 中的临时变量的产生于合并

	X 左边界点	X 右边界点	合并临时梯形	产生临时梯形
(a)	+	-	B∪bottomTmp_≫B	A →topTmp_
(b)	+	+	B∪bottomTmp_≫B	B→bottomTmp_
(c)	-	-	A∪topTmp_≫A	A→topTmp_
(d)	-	+	A∪topTmp_≫A	B→bottomTmp_

注意：+ 表示位于线段 pq 的左边，- 表示位于线段 pq 的右边，∪表示合并临时梯形操作，≫表示合并梯形后形成的新梯形，→表示产生新的临时梯形

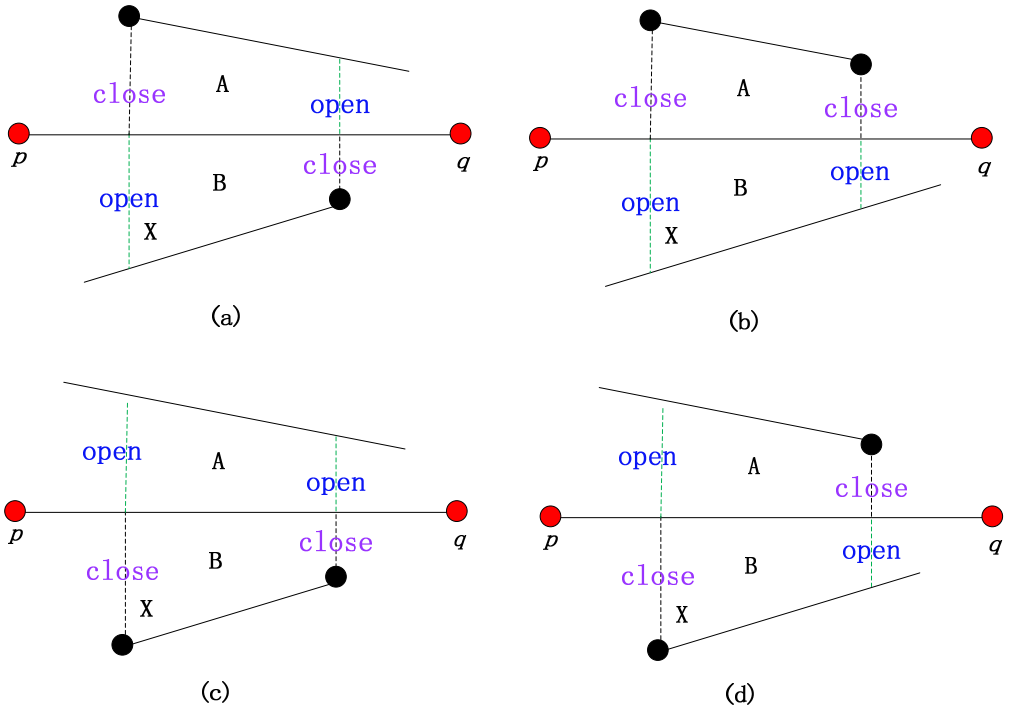


图 11　情况 3 的临时梯形产生与合并

3. 情况 4 产生临时梯形与合并梯形的情况如图 12 所示的 2 中情况。(a)情况梯形 X 的左端点位于线段 pq 的左边，这时 B 为临时梯形，要进行 B∪bottomTmp_≫B 合并操作，(b)情况梯形 X 的左端点位于线段 pq 的右边，要进行 A∪topTmp_≫A 的合并。

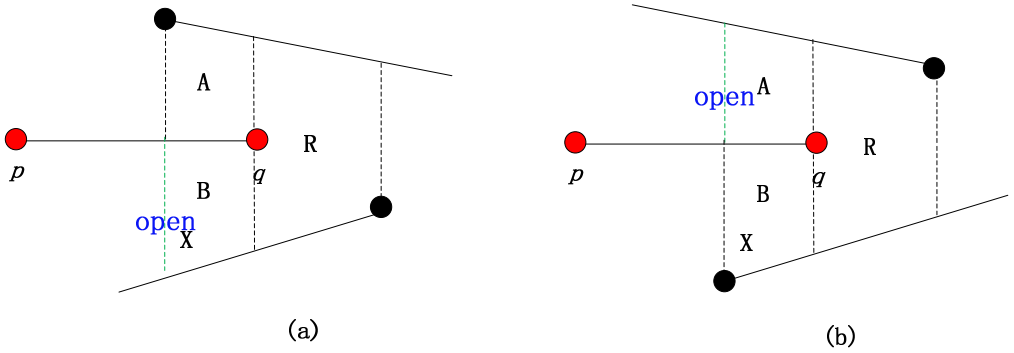


图 12　情况 4 产生临时梯形与合并梯形的情况

退化情况处理

退化情况的处理，我们全部是按照实验要求解决的。这里就不重复了。

随机线段集生成方法

随机生成线段需要判断新生成的线段是否和先前已经存在的线段相交。这个算法的时间复杂度大于 O(n^2)。本系统中提供的算法是这样的，每一条线段都预设一个固定大小的包围盒，也就是说生成的线段只能位于包围盒之内，对于不同的包围盒，其位置可以按照扫描线的顺序排列起来。这样使用的一个好处，就是后面随机生成的线段不必去跟所有的线段进行相交判断，只需要用那些包围盒跟当前包围盒相交的进行相交测试，这样大大减少了相交测试的次数，而生成的线段在视觉上又非常的随机。为了进一步增加其随机性，在完成线段的随机生成后，可以对线段次序进行打扰。

亮点：内存池的使用

由于随机增量法的特点，在每次插入一条新的线段时要产生一些新的 TM 和 SS 信息，同时也要删除一些 TM 和 SS 信息，频繁的进行内存的申请和释放会严重影响到程序的性能，特别是大规模数据处理的时候，这种内存的申请和释放给程序带来极大的延迟，如果用户想建立一个新的梯形图，但是要释放原来的梯形图的资源（包含 TM 和 SS 信息），如果原来梯形图的资源很多，而且内存不连续，这样执行释放内存操作会很慢。

鉴于该梯形图生成算法的特点，我们自己为程序内存的分配写了内存池。内存池的主要特点是每次都申请一大块内存(具体多少要看个人的机器环境)，当每次使用分配一块小内存时，不使用 new 或 malloc 进行内存的分配，而是在内存池中申请一块就可以，当程序每次释放一小块内存时，不适用 delete 或 free 进行释放，而是在内存池中进行垃圾内存回收机制，把可用的内存串联起来。使用内存池不仅减少了内存的申请与释放操作，而且在一定程度上减少了内存碎片，会给程序带来性能的提升。具体见图 13。

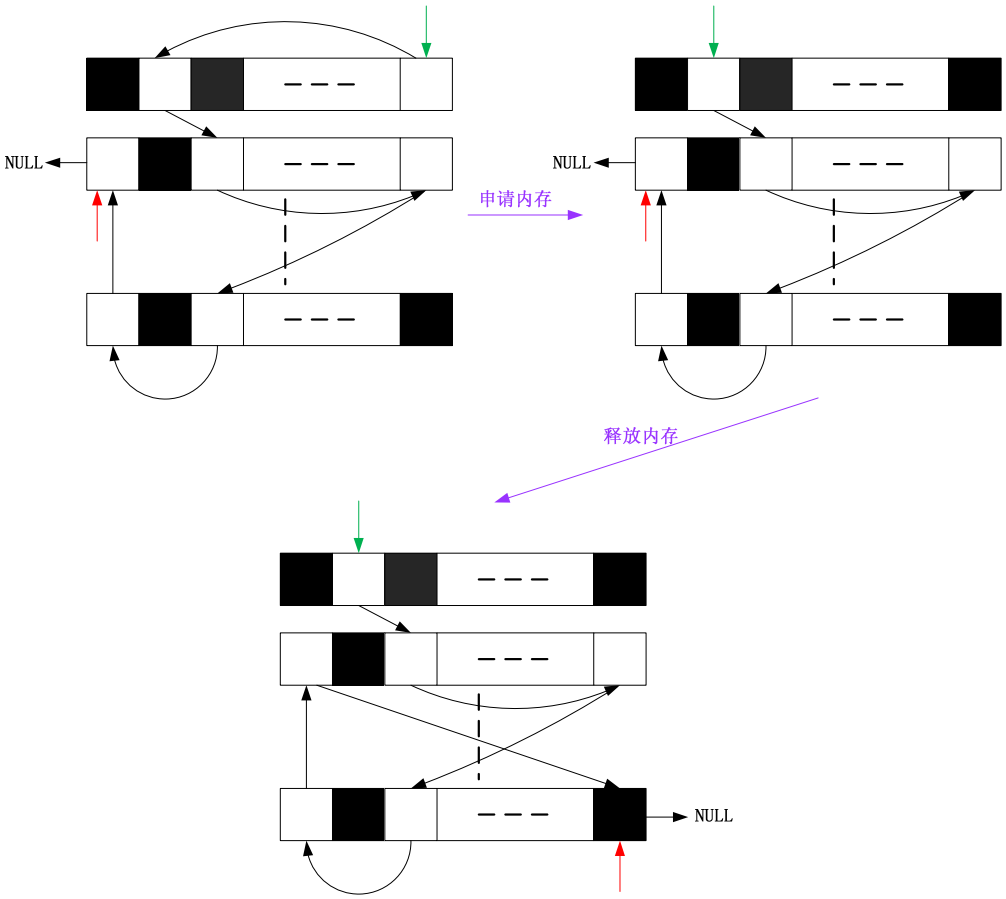


图 13 黑色方块表示内存池中已经被使用的内存，白色方块表示可以使用的内存，可用的内存在内存池中都是串联起来的，绿色箭头表示下一个可用的内存块，红色箭头表示内存池中最后一个可用的内存块。

• 结果分析

一. 线段集的输入

本系统提供三种线段集输入的方法。

1）文件输入。数据文件格式见其他说明部分。系统将界面已存在的数据进行删除，然后将文件保存的线段数据导入，不做相交测试处理。因此在进行文件输入时，一定要确保各线段之间不能相交。

2）随机生成线段集。系统提供随机生成线段集的功能，因为需要进行线段相交测试，所以效率比较低，系统同样把原先存在的线段集数据删除。生成 10k 的线段耗时为 1s，而且生成随机线段算法的时间复杂度大约是 O(n)。因此，生成线段集数目上限一般是 100k 的数量级，时间大约是 10s 左右。

3）鼠标输入。这时是进行添加线段，原先存在的数据不删除。程序同样进行相交测试，以防止有违法线段输入。

下图是不同数量级的线段输入，左图是 10 个，右图是 10k 个。线段的端点标为绿色，线段部分标为红色。

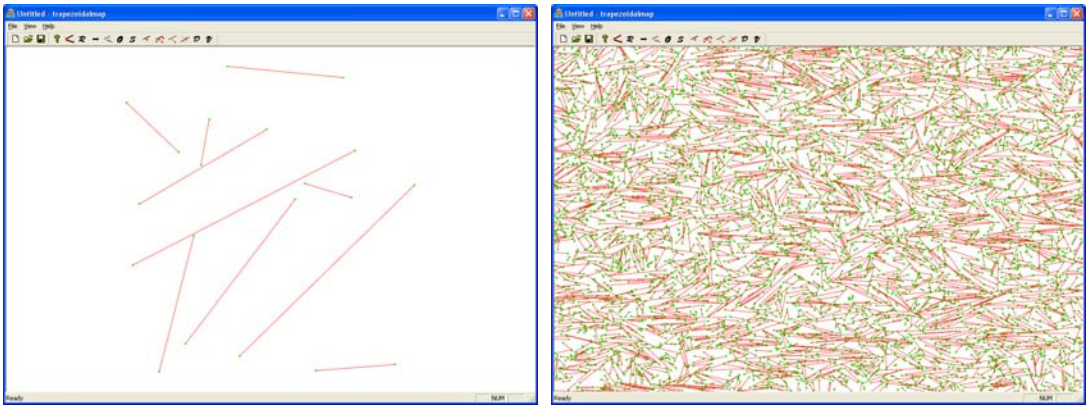


图 14　不同数量级的线段输入

二．线段集的梯形化

选择“梯形化”的图案，界面上将直接显示梯形化的结果，如下图所示。

同样地，红色线为输入线段集，蓝色线为求解的梯形化结果。为了显示需要，最外一圈蓝色线为线段集的包围盒。

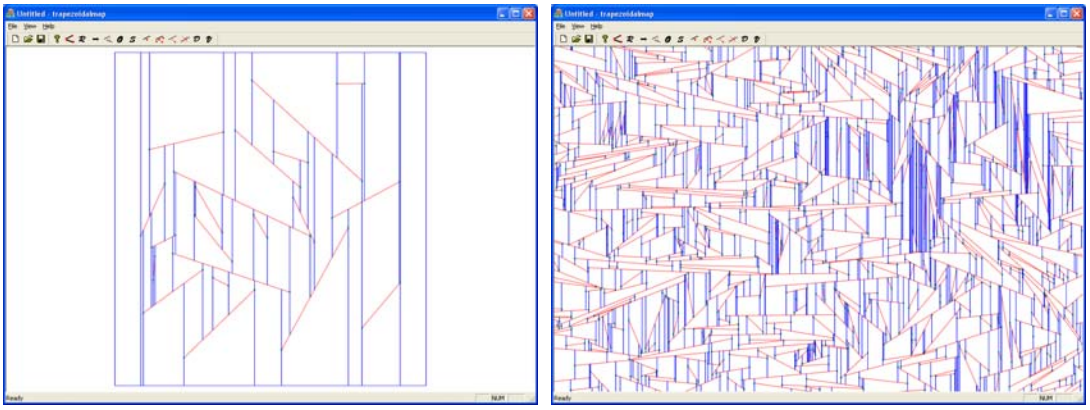


图 15　线段集的梯形化结果

三．单点查询

在进行梯形化之后，可以进行点定位查询。选择“鼠标点定位”图标，通过鼠标输入查询点，就可以显示单点查询的功能，如下图所示。黑色点为查询点，黑色线为点定位查询求得的线段。

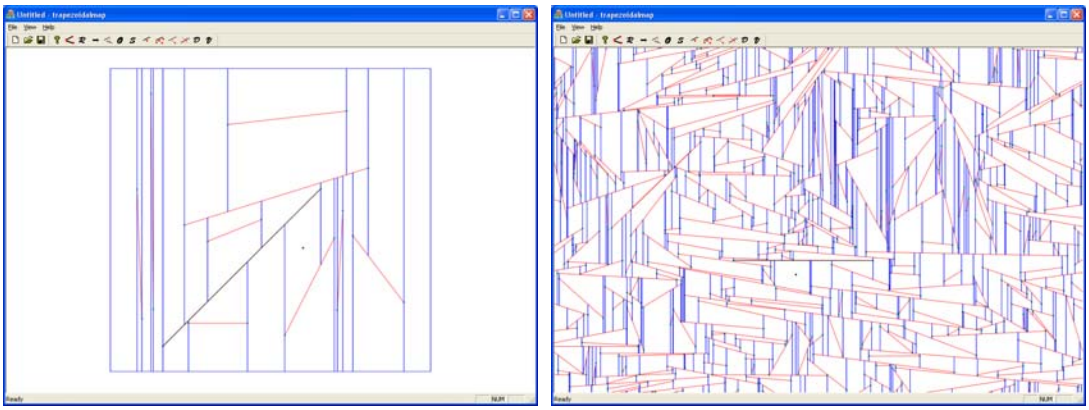


图 16　单点查询结果

四．随机多点查询

系统提供随机的多点查询功能。选择“随机生成点定位”图标，然后在对话框中输入定位点数目，系统将在包围盒之内随机生成要求数目的点，并将查询结果显示在界面上，包括所有查询点和查找到的线段。为了进一步观察各点对应的线段，可以通过“保存查询文件”图标，将查询结果保存到文件中再浏览。

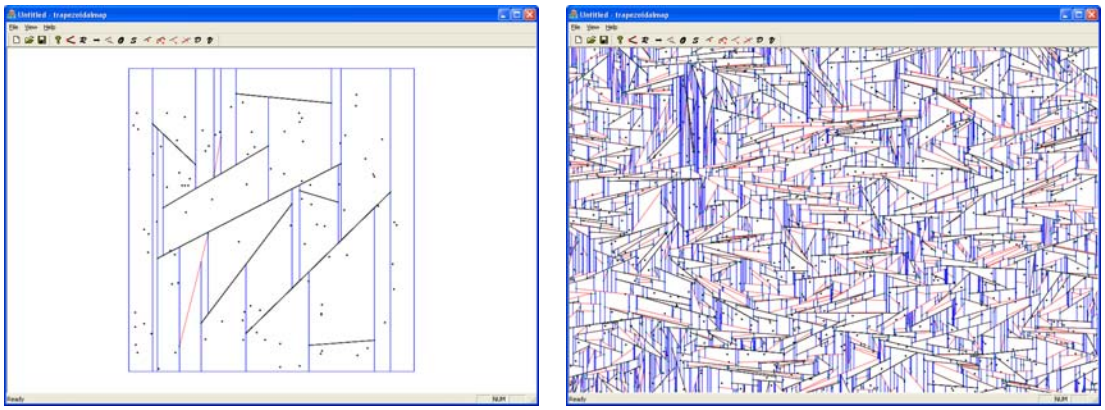


图 17　随机多点查询

另外，系统提供查询点数据输入的入口“打开查询文件”。系统同样把查询点集数据和查询结果显示在界面上。为了测试性能的要求，系统在批处理完查询任务后，将查询结构的高度和查询用时通过消息框进行报告。

五．线段移动

为了显示点定位的功能，系统提供了线段拾取移动的功能。在完成梯形图后，选择“移动线段”图标，然后按下左键选择线段（点定位的方式），移动鼠标，将线段移动到另外位置松开左键，如果该线段不与其他线段相交，则该线段将移至此位置，同时更新梯形图。如下图箭头所指的线段发生了位置的移动。

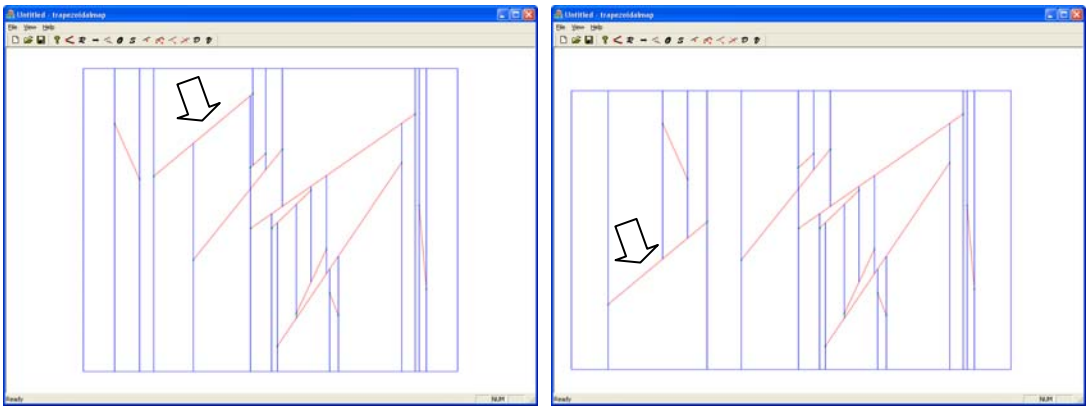


图 18　线段移动操作

六．线段删除

同样也可以实现线段的删除操作。在完成梯形图后，选择“删除线段”图标，然后通过单击鼠标把选择的线段删除了，然后梯形图即时进行更新。如下图，箭头所指的线段被删除。

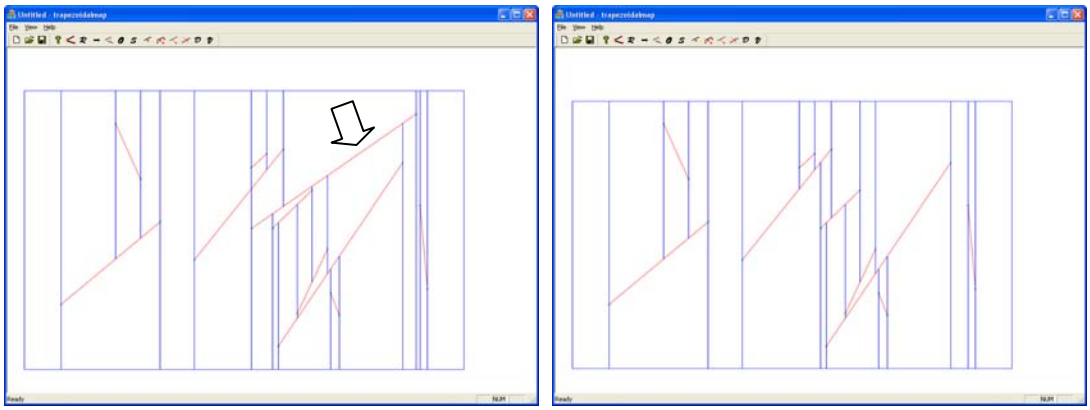


图 19　线段删除操作

七．梯形图构造过程演示

系统提供梯形图构造过程的演示，形象表现梯形图构造的整个过程。在输入线段集之后，选择“构造过程可视化”图标之后，设置好时间间隔，然后界面动态显示整个构造过程。下图是构造过程的两个不同时间的截图。红色框代表已经构造好的梯形，绿色线段为当前加入线段，蓝色部分梯形是当前加入线段涉及的梯形。在进行展示时，可以支持放缩和平移操作，其他操作将忽略。

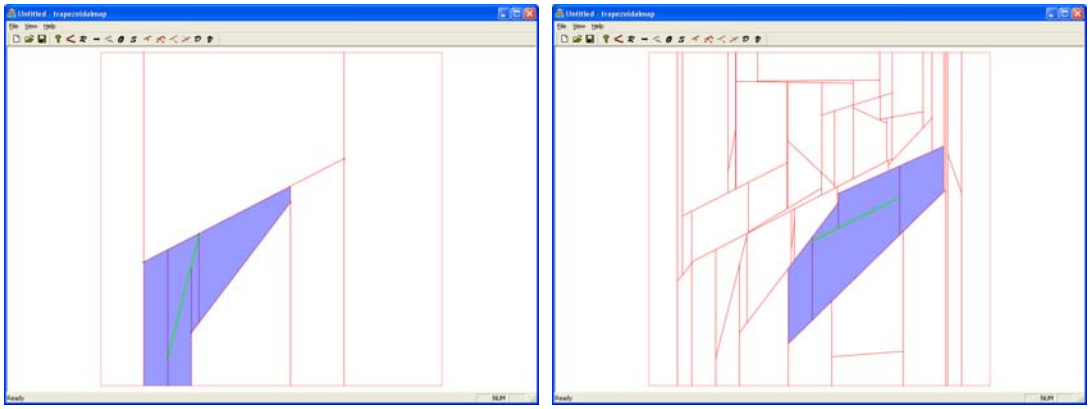


图 20　梯形图构造过程的演示

八．点定位搜索过程演示

为了展示点定位搜索的过程，系统提供点定位搜索过程的可视化。在完成梯形图之后，

选择“搜索可视化”图标，设置好时间间隔后，界面将动态显示整个搜索过程。若搜索到当前节点为线段的端点时，该端点用黑色加粗点标注，同时用箭头显示其方向；若当前节点为线段时，该线段用黑色加粗标注，箭头显示其方向；最后查找到定位点所在的梯形区域，该梯形区域用蓝色标注。在进行展示时，可以支持放缩和平移操作，其他操作将忽略。

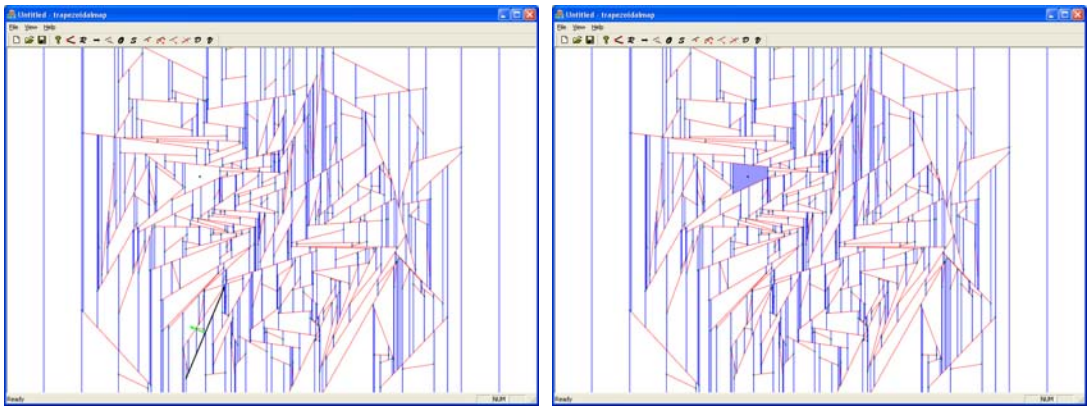


图 21　点定位搜索过程演示

九．梯形图构造过程的时间复杂度分析

经分析得出梯形图构造过程的时间复杂度为 O(nlogn)，下面是不同规模的线段集数目和构造时间的测试，整理为表格 2，画成曲线为图 22，大约验证其符合 O(nlogn)。

表格 2　梯形图构造过程的时间复杂度测试

数目	10	100	1k	5k	10k	30k	50k	70k	100k	200k
时间(ms)	0.327	1.185	10.4	60.5	134.1	479.4	843.1	1230.1	1814.4	3979.8

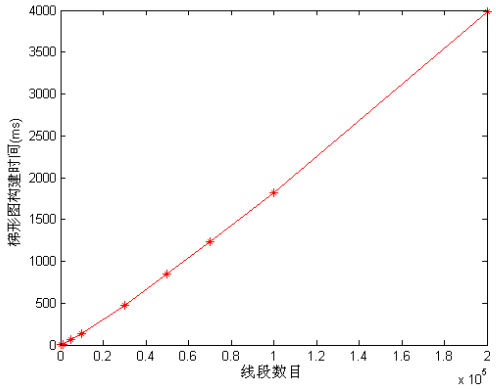


图 22　梯形图构造过程的时间复杂度曲线

十．查找结构高度与查找效率的实验分析

查找点数目统一设置成 10k 个，不同高度的查找结构对应的查找总时间关系如表格 3，拟合的曲线如图 23 所示。一般认为查找效率跟查找时间成反比，可见随着高度的增加，查找效率降低。

表格 3　查找结构高度与查找时间的关系

高度	8	11	22	34	37	54	56	60	82	90
时间(ms)	1.247	1.394	2.227	3.893	4.734	9.430	14.474	25.780	31.187	48.910

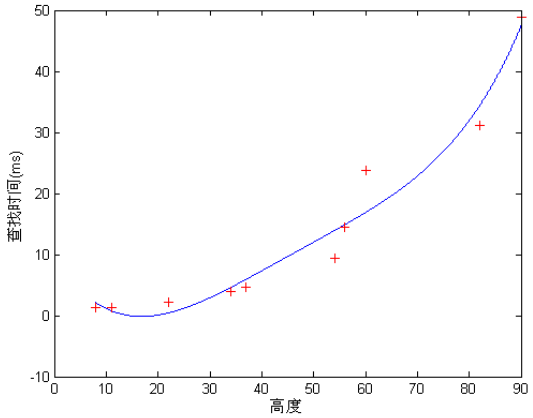


图 23　查找结构高度和查找时间的关系

• 用户手册

界面整体印象

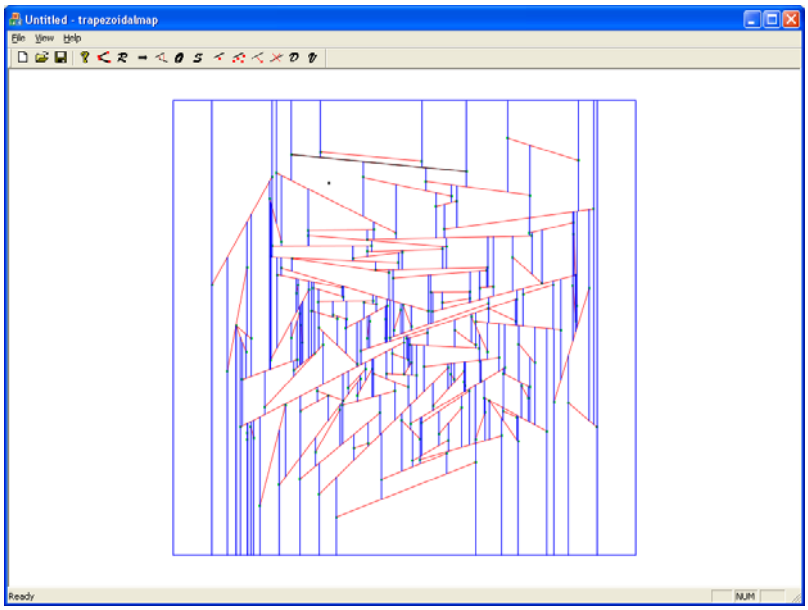


图 24　界面整体印象

界面操作说明

-  （或者“文件”->“新建”）：将已存在的数据和界面图案进行清除，重新初始化。
-  （或者“文件”->“打开”）：打开和读入线段集数据，在这里不做线段是否相交的测试，因此需要保证输入数据的有效性。
-  （或者“文件”->“保存”）：将界面已存在的线段集数据保存到文件中。
-  ：鼠标继续在界面上输入线段信息，系统提供相交测试功能。

-  ：随机生成互不相交的线段集数据，生成线段数目由用户在对话框中输入。
-  ：由鼠标控制进行界面平移操作。
-  ：梯形化操作，前提是界面已经存在线段集数据。
-  ：打开点查询数据，前提是界面已经存在线段集数据，并且已经进行图形化。将点查询数据读入后，批处理完点查询操作，将查询结果显示于界面。
-  ：保存点查询结果。将当前存在的点查询结果进行保存。
-  ：鼠标输入点查询。前提是对界面线段集进行梯形化。单击鼠标输入查询点，界面将查询到的线段用黑色来表示出来。
-  ：随机生成批量点查询。前提是对界面线段集进行梯形化。在对话框中输入点查询的数目，系统将查询结构的高度、查询用时都报告出来，然后界面显示点查询结果。
-  ：线段的移动操作。前提是对界面线段集进行梯形化。按下鼠标左键，如果通过点查询找到对应的线段，移动鼠标，该线段随着鼠标进行移动，放开鼠标左键，如果该线段和其他线段有相交，则该线段返回原位置，否则，该线段将移至此位置，梯形化进行了更新。
-  ：线段的删除操作。前提是对界面线段集进行梯形化。单击鼠标，如果通过点查询找到对应的线段，则将此线段删除，并同时更新梯形图。
-  ：动态演示梯形图的构造过程。前提是界面存在线段集数据。在对话框中输入演示时间间隔，确定后界面将动态显示梯形图的构造过程。构造过程中系统支持界面的放缩和平移操作，不支持其他操作。并且等演示过程结束后才能进行其他操作。
-  ：点定位查询过程的可视化。前提是对界面线段集进行了梯形化操作。在对话框中输入演示时间间隔，然后单击鼠标输入定位点，界面将动态显示点查询的过程。最后将定位点位于的梯形区域用蓝色标识。这里对位于包围盒外部的查询点不做处理。
- 鼠标滚轮放缩：由鼠标滚轮对界面进行放缩操作，方便用户从不用的范围观察数据。

• 其他说明

1） 系统开发平台为 vs2005，使用了基本的 OpenGL 库，对于编译和连接无特殊要求。
2） 提交文件夹 data 中包含我们设计的几组退化的测试数据。
3） 测试实验的配置环境为：Intel Core 2 Duo CPU 2.2GHz, 2.00GB RAM。
4） 对输入线段的文本文件格式要求：

*** 输入线段 ***

批量输入的线段集以如下格式存为文本文件:

```
*********************************************************************
_example_segments.txt
*********************************************************************
3       '#segments
-10.000     10.000      20.000      40.000      '(a1_x, a1_y) (b1_x, b1_y)
+10.000     -10.000     30.000      -15.000     '(a2_x, a2_y) (b2_x, b2_y)
0.000 0.000 40.000      05.000      '(a3_x, a3_y) (b3_x, b3_y)
*********************************************************************
```

线段之间保证无交。但某些线段的端点可能重合。允许水平或垂直线段，不同线段的横、纵坐标可能相等。

5）对批量查询点的文本文件格式要求：

*** 批量查询 ***

通过格式如下的文本文件给出一组查询：

```
*********************************************************************
_example_batch_query.input
*********************************************************************
5       '#queries
0.000 -20.000       '(q1_x, q1_y)
20.000      -20.000     '(q2_x, q2_y)
10.000      10.000      '(q3_x, q3_y)
10.000      30.000      '(q4_x, q4_y)
0.000 30.000        '(q5_x, q5_y)
*********************************************************************
```

6）对查询结果的文本文件格式要求：

查询结果按如下格式保存为文本文件：

```
*********************************************************************
_example_batch_query.output
*********************************************************************
5       '#queries
3       's3
2       's2
1       's1
1       's1
inf     'the vertical ray intersects no segments
*********************************************************************
```

[i] 邓俊辉，06.pl.e.TrapMap.(29).ppt. pp.5.