梯形图及点定位

目录

实验内容

对于平面上一组互不相交的线段，构造梯形图以及对应的一个查找结构。梯形图组织为梯形结构集合形式，查找结构为有向无环图形式。

这里的点定位查找定义为：在输入的线段集中，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。

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，即包含左端点但不含右端点；

3. 假象为通过逆时针地轻微转动坐标轴，消除垂直线段——亦即，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

数据结构

二维点

```
class TMPoint2D{
public:
    double x, y;          // 表示点对应的x坐标和y坐标
}
```

线段

```
class TMSegment{
public:
    TMPoint2D start;      // 线段的起始点
    TMPoint2D end;        // 线段的终止点
}
```

辅助矩形

```
class TMRectangle
{
public:
    double left;          // 矩形左侧边x坐标
    double right;         // 矩形右侧边x坐标
    double bottom;        // 矩形底边y坐标
    double top;           // 矩形顶边y坐标
}
```

梯形

```
class TMTrapezoidal{
public:
    /*梯形自身的拓扑信息*/
    size_t up;            //作为梯形上边的线段
    size_t bottom;        //作为梯形下边的线段
    TMPoint2D leftp;      //生成左边射线的某端点
    TMPoint2D rightp;     //生成右边射线的某端点
    double leftUpY;       //梯形左边上端点的Y坐标
    double leftBottomY;   //梯形左边下端点的Y坐标
    double rightUpY;      //梯形右边上端点的Y坐标
    double rightBottomY; //提醒右边下端点的Y坐标
```

```
    /*梯形与邻居的邻接关系*/
    TMTrapezoidal* rightUpNeigh;        //梯形的右上方邻居，与当前梯形共享同一个up线段
    TMTrapezoidal* rightBottomNeigh;    //梯形的右下方邻居，与当前梯形共享同一个bottom线段
    TMTrapezoidal* leftUpNeigh;         //梯形的左上方邻居，与当前梯形共享同一个up线段
    TMTrapezoidal* leftBottomNeigh;     //梯形的左下方邻居，与当前梯形共享同一个bottom线段

    size_t nodeIndex;     //当前梯形在查找结构中所对应的节点下标
};
```

有向无环图节点

```
#define TypePoint 1000  // 节点类型，表示x类型节点
#define TypeEdge  1001  // 节点类型，表示y类型节点
#define TypeTrap  1002  // 节点类型，表示叶子节点，即对于梯形

class TMDAGNode
{
public:
    int type;                // 节点类型

    double x;                // x节点时，节点是一个点，记录点x坐标
    size_t segIndex;         // y节点时，节点是一条边，记录线段在集合中的下标
    size_t trapIndex;        // 叶子节点时，记录所对应梯形在梯形集合中的位置
    size_t leftChildIndex;   // 左孩子在节点集合中的位置下标
    size_t rightChildIndex;                 // 右孩子在节点集合中的位置下标

    TMDAGNode *pLeftChild, *pRightChild;  // 指向节点左孩子和右孩子，叶子节点时为空
};
```

搜索结构

```
class TMSearchDAG{
public:
    vector<TMDAGNode> nodes;            // 搜索结构的节点的集合
}
```

梯形图及点定位算法类

```
class TrapMap{
```

```
public:
    TMSegment *segmentSet;                  // 输入线段集合
    size_t segmentNumber;                   // 输入线段个数
    size_t segmentReserve;                  // 输入线段集合的预留
空间

    TMSearchDAG searchDAG;                  // 搜索结构
    vector<TMTrapezoidal*>  traps;          //当前所有的梯形集合
    TMRectangle rect;                     // 梯形图边框
}
```

算法说明

随机线段生成算法

这里提供一种伪随机方法生成不相交的线段集合。首先，由用户输入要求生成的线段个数 N，系统会根据线段数目生成一个矩形包围框，表示生成线段的数据范围，之后按如下步骤生成不相交的线段：

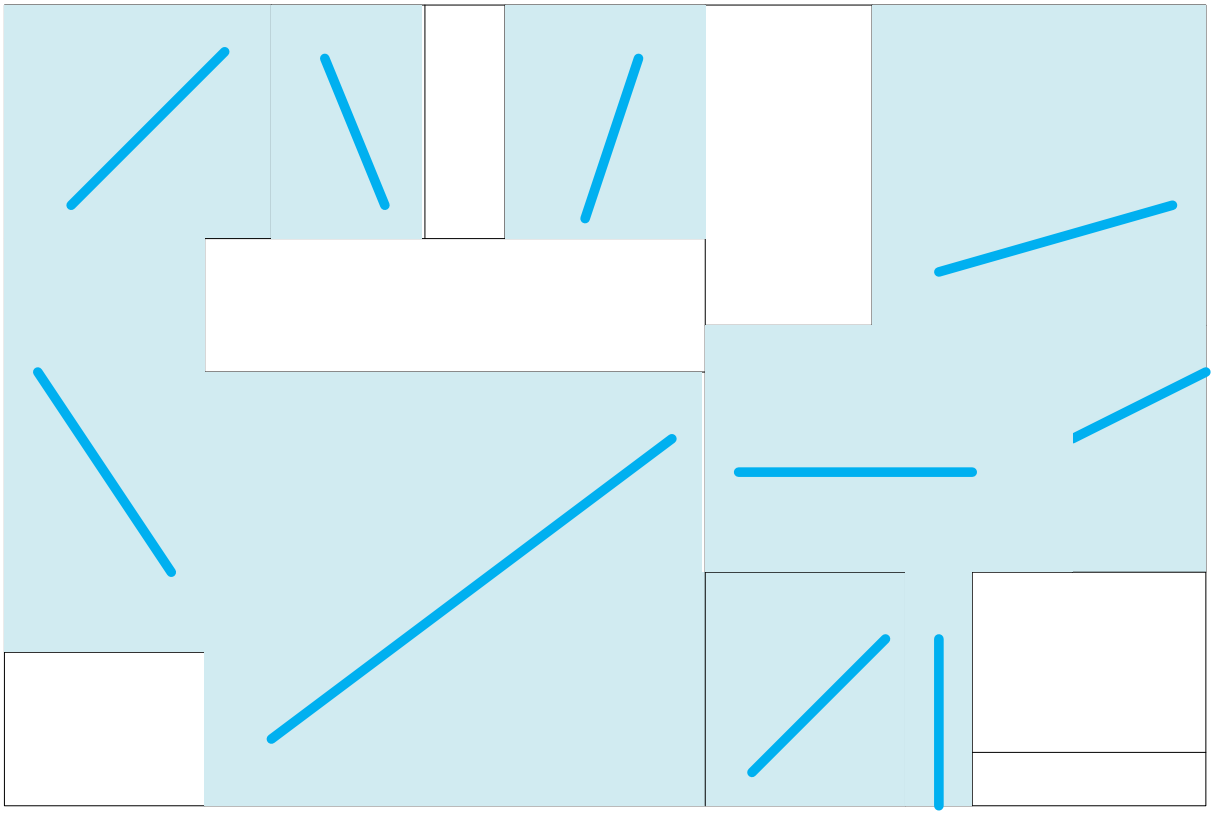
1）对矩形包围框进行随机切割：如果包围框的宽度(width)大于高度(height)，则分成左右两个包围框（按 x 坐标），如果包围框的宽度(width)小于高度(height)，则分成上下两个包围框（按 y 坐标）。切割的比例是按照矩形边的三等分点随机划分的，即要么选在 1/3 处切割，要么选在 2/3 处切割。

2）判断矩形框的数目是否已经不小于 N，如果已经不小于 N，对当前的所有矩形框重复“步骤 1）”中的操作。否则进入“步骤 3）”。

3）从生成的矩形框中（大于等于 N 个）随机选取 N 个矩形框出来，作为备选矩形框。

4）在每个备选矩形框中随机产生两个点，按照“左”在“右”前，“下”在“上”前的原则构成线段的起点和终点。

算法如下图所示，蓝色线段表示生成的随机线段：



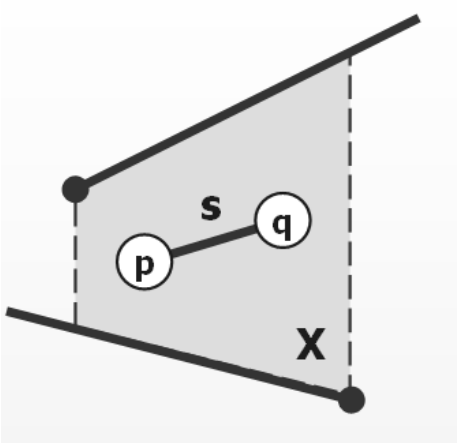
梯形图和查找结构构造算法

算法思路

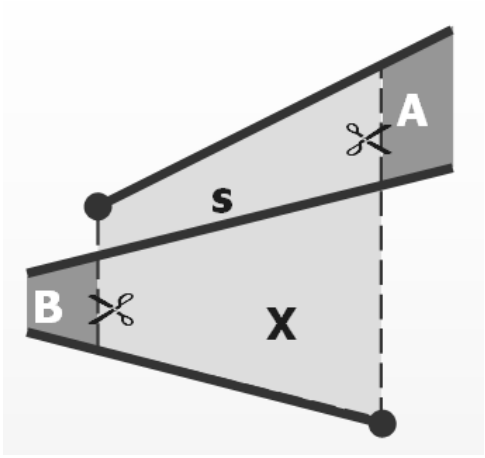
首先计算线段集合的包围盒，将该包围盒作为初始梯形图中的梯形，接下来对于输入线段集中的每一条线段，都调用一个插入算法，将其插入到现有的一个完整的梯形图中，形成一个新的梯形图。

插入线段的过程即为在已有梯形图中找到相交梯形，然后对其进行删除并生成新梯形的过程。线段与梯形相交的情形分为以下三种。（图片选自老师课件）

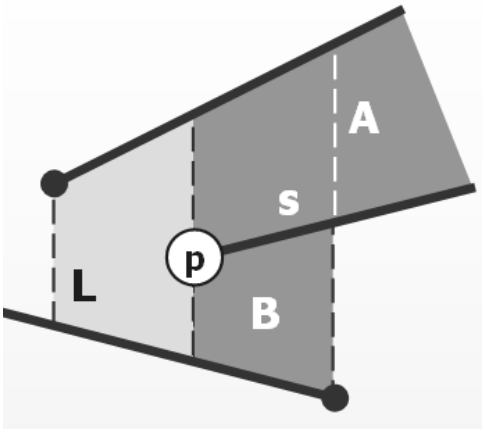
第一种类型，线段端点全部位于梯形内。



第二种类型，线段完全穿越梯形



第三种类型，线段的某一个端点落在梯形内



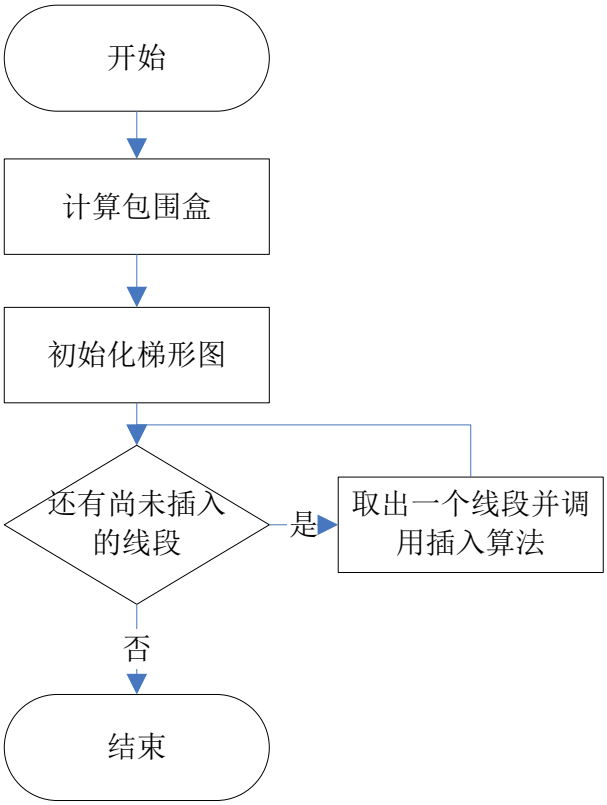
我们的程序处理线段插入过程如下：首先找到左端点所在的梯形，先将该梯形进行修改，将其 rightp 向左移动，变成线段的左端点，然后对该梯形所对应的搜索结构节点也进行更改，将其类型由叶子节点修改成 X 类型的节点。接下来，生成两个左边封闭右边未知的梯形，其中一个位于输入线段的上方，另一个位于下方，并生成对应的搜索结构节点以及线段所对应的 Y 节点，将其作为 X 类型节点的右孩子。

接下来，从当前梯形开始，只要当前梯形的右边界位于线段右端点的左侧，就做如下处理：根据当前梯形的 rightp 与输入线段的上下位置关系，决定对上下两个半封闭梯形中的哪一个进行闭合处理，闭合处理后再次生成一个对应的半封闭梯形。并更新对应的搜索结构。

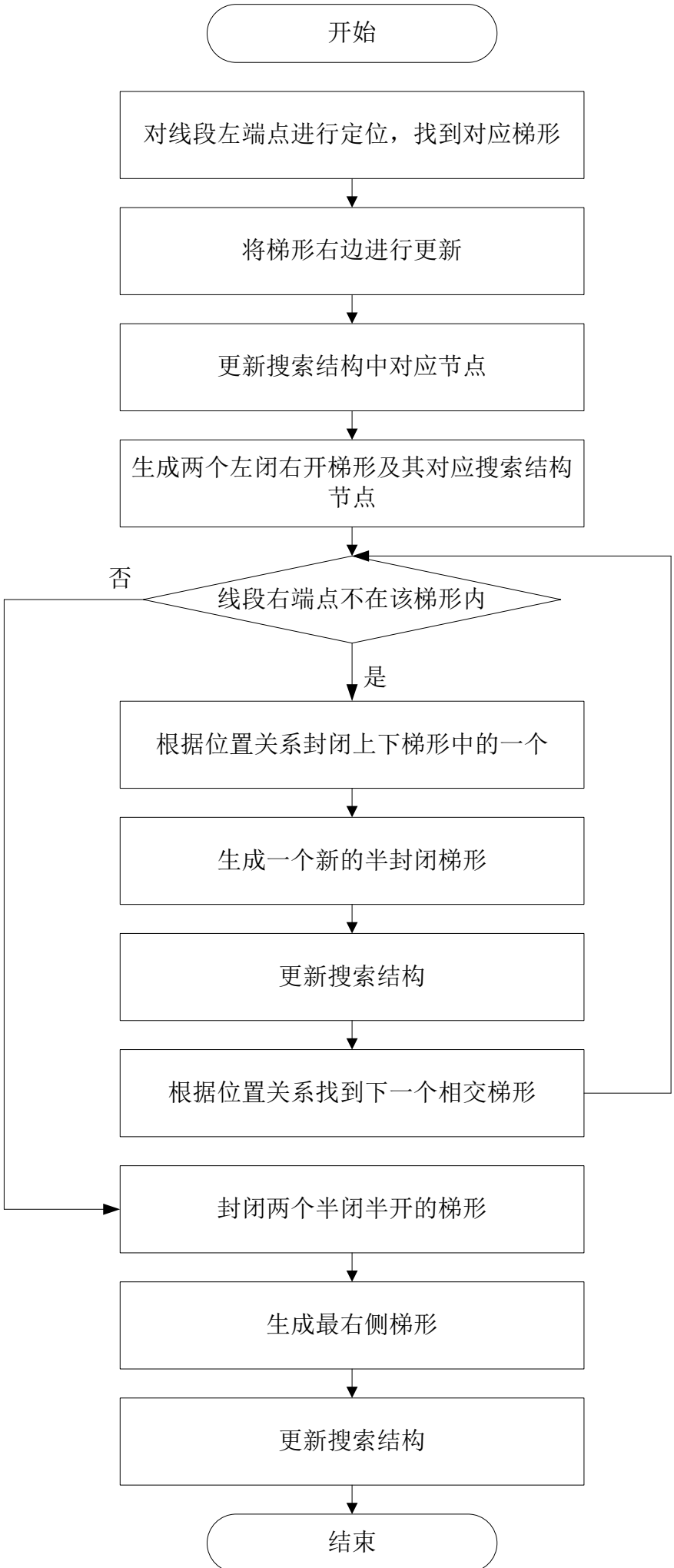
在处理到线段右端点所在的梯形时，根据右端点对上下两个半封闭梯形进行封闭处理，并进行最后一个梯形（最右侧）的生成，并更新搜索结构。

算法流程图

算法整体流程图：



插入线段流程图


开始
对线段左端点进行定位，找到对应梯形
将梯形右边进行更新
更新搜索结构中对应节点
生成两个左闭右开梯形及其对应搜索结构节点
否
线段右端点不在该梯形内
是
根据位置关系封闭上下梯形中的一个
生成一个新的半封闭梯形
更新搜索结构
根据位置关系找到下一个相交梯形
封闭两个半闭半开的梯形
生成最右侧梯形
更新搜索结构
结束

相交梯形的查找与退化情况的处理

从当前相交梯形找到下一相交梯形是需要判断当前梯形的 rightp 与插入线段 s 的位置关系的。如果 rightp 位于 s 上方，则下一个相交梯形为当前相交梯形的右下方邻居，如果 rightp 位于 s 下方，则下一个相交梯形为当前相交梯形的右上方邻居。

而当 rightp 刚好被线段 s 穿过的时候，就产生了退化情况，这种情况应该予以进一步判断。此时分两种情况处理：

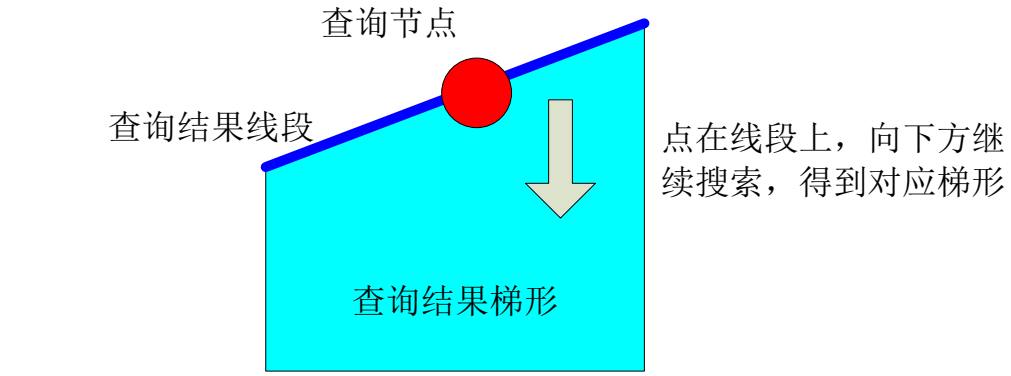
第一种情况，当前梯形在右侧只存在一个邻居。则下一个相交梯形就为唯一的一个右侧邻居（有可能是右上方邻居，也可能是右下方邻居）。

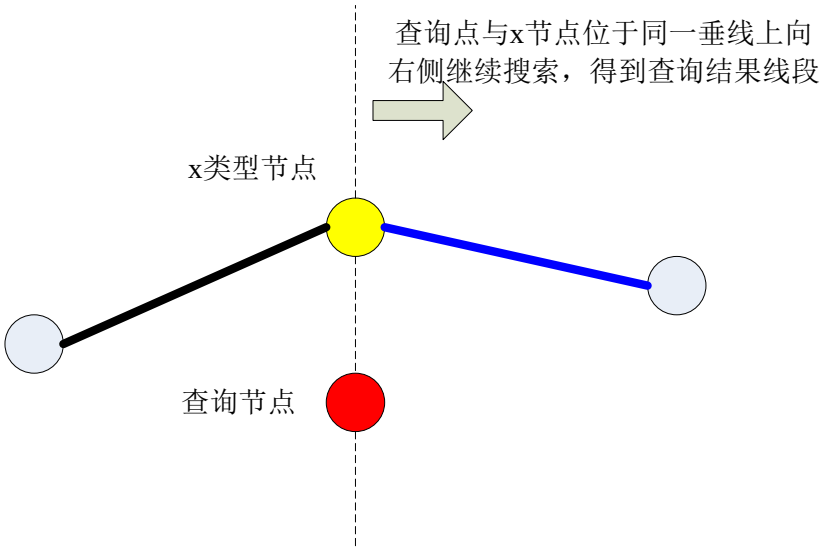
第二种情况，当前梯形存在两个右侧邻居。这说明决定当前梯形的 rightp 来自与其他线段，而不是当前梯形 up 或 bottom 中任何一条线段的端点。假设 rightp 来自与线段 s1 的做端点 left(s1)，那么我们判断 right(s1)与插入线段 s 的位置关系，如果 right(s1)位于 s 的上方，则我们选择当前梯形右下方邻居作为下一个相交梯形，否则选择当前梯形的右上方邻居作为下一个相交梯形。

点定位算法

在已经构造好的搜索结构中（即对应的有向无环图 DAG），可以快捷方便的进行点定位查询，但考虑到一些退化情况，仍然需要做一定的处理。首先，一些位于包围矩形之外的点应该被事先排除，这里需要注意那些位于矩形外但在下方的点，它们不应被排除，因为它们仍可能会与线段相交，在处理这类点时，由于它们不在任何构造出梯形内，所以要对它们加一定的偏移，使其位于包围矩形下边稍微往上一点的位置，这样即使得其有包含它的矩形，也使得搜索的结果（所交的射线线段）不变，保证结果正确性。

在搜索过程上，如果遇到的节点为 x 类型节点，此时该节点对应为一个点，比较该 x 节点与查询点的 x 坐标，若查询点在该点右侧或位于同一条垂线上，则继续访问右孩子，即将查询方向导向右侧，否则则访问左孩子，将查询方向导向左侧；如果遇到的节点为 y 类型节点，此时该节点对应为一条线段，比较该线段与查询点的位置，若查询点在线段下方或位于线段所在直线上，则继续访问该节点的右孩子，将查询方向导向下方，否则则继续访问该节点的左孩子，将查询方向导向上方；如果节点为梯形类型，则表示已经搜索到对应的梯形，该梯形中记录的上方线段即为所需要的第一个与该查询点向上发出射线的第一个相交线段。在搜索过程中，对于点在线段上的情况，我们采取往下方方向继续搜索，这样使得得到的结果线段为点所在的那条线段。而对于在查询过程中遇到与查询点在同一垂线上的 x 类型节点时，采取往右侧继续搜索，这是由于线段均是左闭右开的，此时点所交的线段应该取右侧区域中的线段。





程序结构

文件说明

文件名	功能
myglshow2d.h myglshow2d.cpp	用于在 MFC 中应用 openGL 库，提供相关的初始化以及设置环境操作
TMDataStructure.h	数据结构定义，以及一些基本操作函数的定义
TMBasisFunction.cpp	一些基本操作函数的实现
TMShow.h TMShow.cpp	控制各种元素，包括点、线段、梯形等的显示
TrapMap.h TrapMap.cpp	算法的具体实现部分，包括随机生成线段、构造梯形图和查找结构、点定位查询等

类和结构体说明

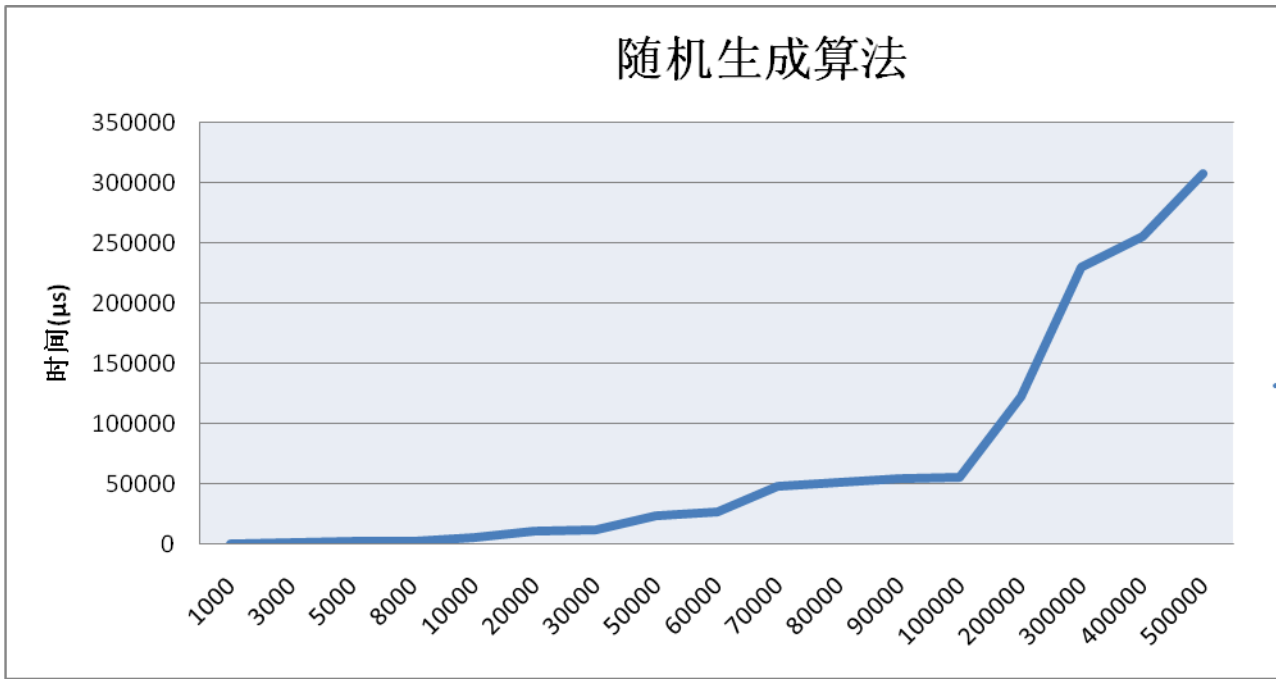
类名	功能
MyGLShow2D	用于在 MFC 中应用 openGL 库，提供相关的初始化以及设置环境操作

CRandomSetDLG	随机生成线段的对话框类
TMPoint2D	二维点类
TMSegment	二维线段类
TMRectangle	用于随机生成线段的辅助矩形类
TMTrapezoidal	梯形类，用于生成梯形图
TMDAGNode	搜索结构的节点类
TMTimer	测试时间类，用于性能测试
TMDraw	控制各种元素，包括点、线段、梯形等的显示
TMSearchDAG	搜索结构类，包括对应的节点数组，根节点，以及相关的维护操作
TrapMap	梯形图和点定位算法类，包括随机生成线段、构造梯形图和查询结构、点定位等操作
TMSleepTimeDLG	设置动画演示间隔时间的对话框类

实验结果

随机生成线段

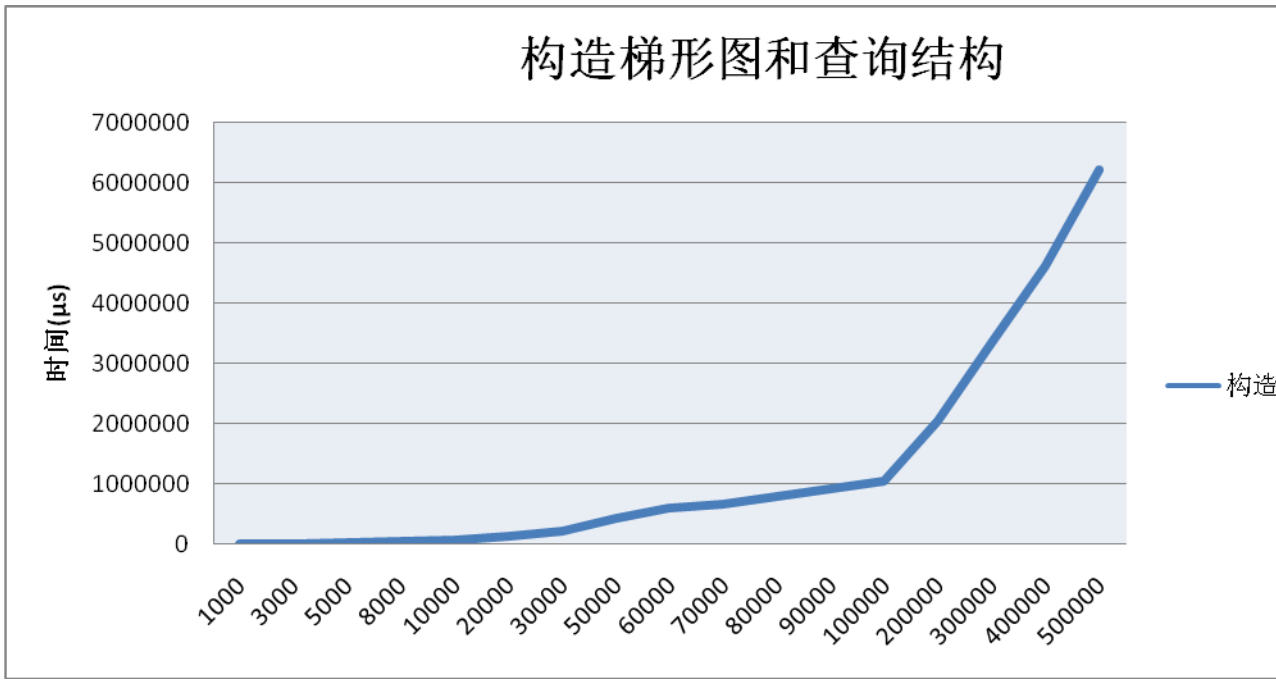
	1000	**3000**	**5000**	**8000**	**10000**	**20000**	**30000**	**50000**	**60000**
时间(μs)	403.083	1293.37	2440.32	3016.37	5810.36	11299.3	12400.7	23940	26756.2
	70000	**80000**	**90000**	**100000**	**200000**	**300000**	**400000**	**500000**	
时间(μs)	48205.9	51483.6	54330.9	55240.4	122209	230298	255295	307704	



构建梯形图和查找结构

下表位构造梯形图和查询结构的时间

	1000	3000	5000	8000	10000	20000	30000	50000	60000
时间(μs)	3343.74	10323.9	25269.8	40272.9	59161.1	127196	225873	429375	591677
	70000	**80000**	**90000**	**100000**	**200000**	**300000**	**400000**	**500000**	
时间(μs)	667907	794644	923441	1.05255e+006	2.04058e+006	3.33467e+006	4.62635e+006	6.21787e+006	



下表为所构造的搜索结构的高度，这里的数据是对于 N 个随机线段，分别构造了 5 次搜索结构，平均而得的搜索结构高度。

	1000	3000	5000	8000	10000	20000	30000	50000	60000	70000	80000	90000	100000	200000	300000	400000	500000
搜索结构高度	6	49	49	50	53	57	64	67	67	71	71	68	71	76	80	83	84

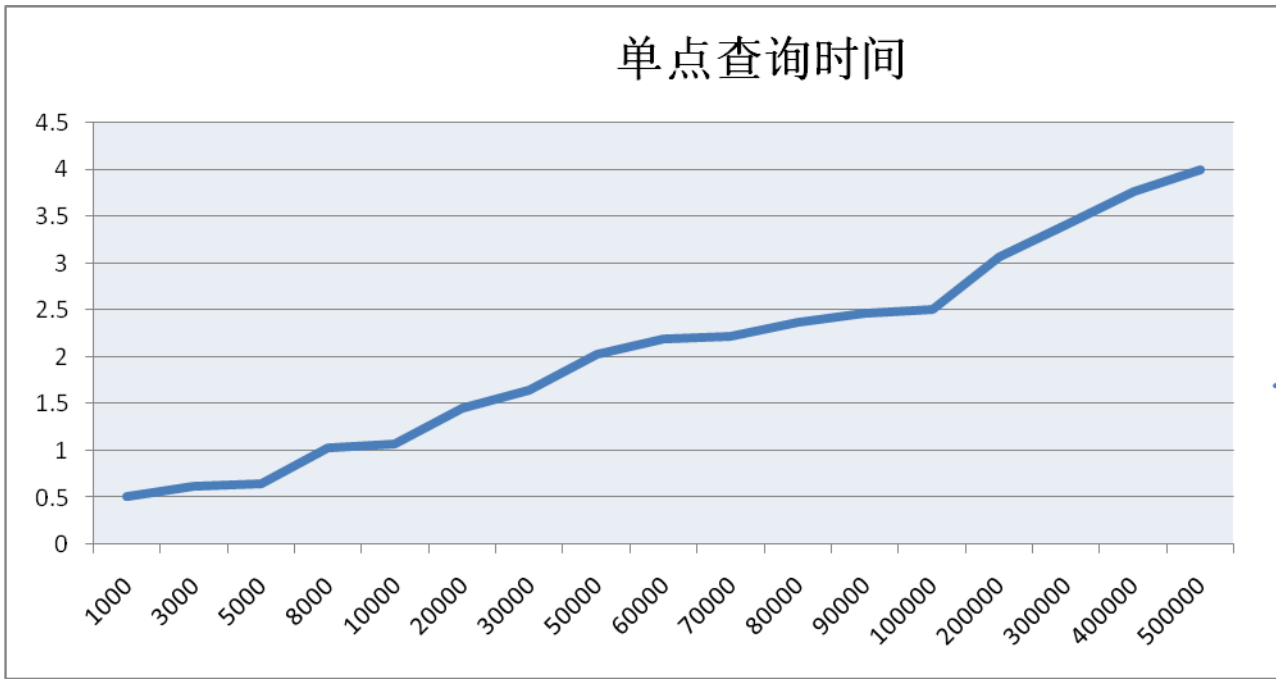
点定位

为避免因为特殊情况造成误差，我们的点定位查询采用下面的方法进行测量。首先根据给定的线段数 N，分别生成 0.2N、0.4N、0.6N、0.8N 和 N 个随机点进行点定位查询，记录查询时间，最后将所得到的查询时间平均起来作为单个点的查询时间。

下表为点定位查询算法消耗时间表，时间单位为微秒（μs）：

	1000	3000	5000	8000	10000	20000	30000	50000	60000	70000	80000	90000	100000	200000	300000	400000	500000
0.2N	47.688	202.573	510.331	1163.22	1955.94	3901.01	6473.54	13635.6	17144.9	20797.3	24434.9	29928.4	34384.9	89484.3	150759	220665	302192
0.4N	108.781	389.32	796.475	1685.91	2392.07	6869.68	13858.4	29802.8	39984.7	46421	58297.1	66412	76473.2	196016	357675	480472	668011
0	15	95	13	2	3	1	2	4	6	7	9	1	1	3	5	761	1.0

.6N	8.364	1.011	08.37	566.82	529.61	172.87	323.95	864.88	279.31	539.95	218.29	095.90	216.01	235.07	281.72	101	3287e+006
.8N	218.6	863.756	1597.22	3425.3	4940.33	1686.19	32412	663.153	886.048	1062.40	1297.47	1152.37	1172.70	443.73	718.545	1.0793 2e+006	1.4151 8e+006
N	279.791	1096.13	2000.83	486.92	747.664	212.934	437.225	872.753	1179.76	1385.76	1657.81	1952.57	2269.71	2560.32	932.471	1.3538 1e+006	1.8086 5e+006
单点查询时间	0.05053 45	0.06154 09	0.06407 55	1.02523	1.06865	1.44515	1.64156	2.02739	2.19006	2.21833	2.36866	2.45575	2.50038	3.06838	3.4033	3.76553	3.99242



实验分析

由实验数据可得，在随机线段生成算法部分，由于采用的是通过辅助矩形的算法生成线段，从而算法的时间复杂度为 O(nlgn)。而对于构造梯形图和搜索结构的算法，由数据可得算法的复杂度为 O(nlgn)，另外，从得到的搜索结构的高度变化趋势可以看出，该高度是以 O(lgn)的复杂度趋势变化的，从而我们可以推论出查询时间的复杂度应为 O(lgn)，而之后的单点查询结果也印证了这一推论，从而证实了查询算法的复杂度为 O(lgn)。