

# 梯形化及点定位

## 1.问题背景

### 1.1 点定位概念

平面点定位就是对点位置的查询。在一个包含  $n$  条边的平面字区域划分  $\tau$  中，对处于其中的任意点  $q$ ，在  $\tau$  中找出  $q$  所处的那张子平面。本实验的要求是，在输入的线段集中，找出与从  $q$  出发的垂直向上的射线相交的第一条线段，并返回其编号，要是  $q$  恰好落在某条边上，或者恰好与某个顶点重合，查询算法也必须返回这些信息。

### 1.2 线段不相交

若平面上两条线段不相交，或者只相交于它们的一个共同端点，都称它们是互不相交的

### 1.3 一般性位置线段

由  $n$  条线段构成的集合  $S$ ，其中的线段互不相交，而且任何两个端点都不会处于同一条垂线上，这样的集合称为一般性位置线段。

### 1.4 梯形图 $\tau(S)$

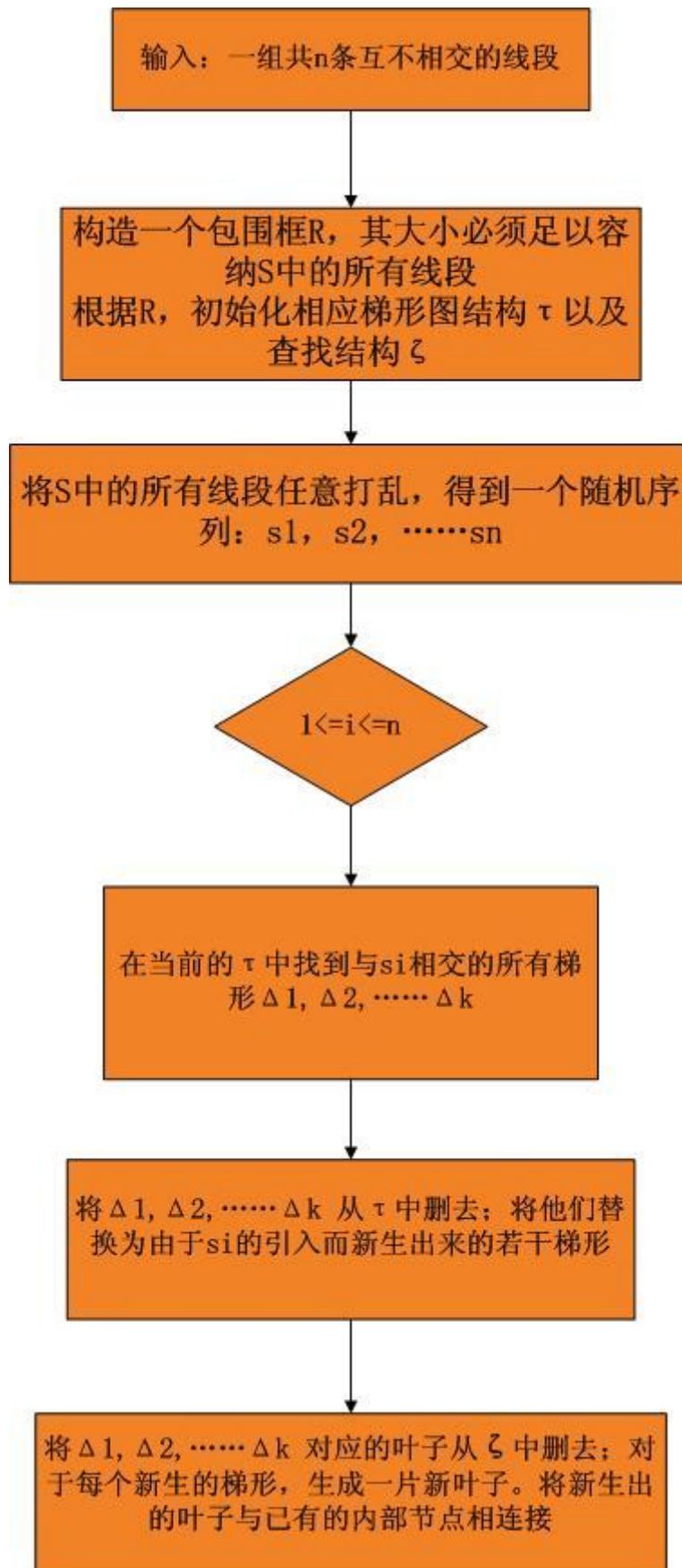
引入一个与坐标轴平行的矩形  $R$ ， $R$  必须大到能容纳下  $S$  中的所有线段。 $S$  的梯形图  $\tau(S)$  也称为  $S$  的垂直分解或者梯形分解。梯形图是这样得到的：经过  $S$  中每条线段的每个端点，向上的和向下个发出一条垂直射线；在碰到  $S$  中的另一条线段或者  $R$  的边界后射线终止，将它们分别称为上垂直延长线和下垂直延长线。 $S$  的梯形图就是由线

段集  $S$ , 矩形  $R$  以及所有的上垂直延长线和下垂直延长线导出的一个之区域划分。

## 1.5 所构造的数据结构

这种支持点定位的数据结构  $\zeta$  称作查找结构。它是一副有向无环图, 有唯一的根节点, 同时对应于  $S$  的梯形图中的每个梯形有且仅有一片叶子, 每个内部节点的出度都是 2。所有内部的节点分为两类:  $x$  节点和  $y$  节点。每个  $x$  节点都被标记为  $S$  中某条线段的一个端点; 每个  $y$  节点都被标记为某条线段。

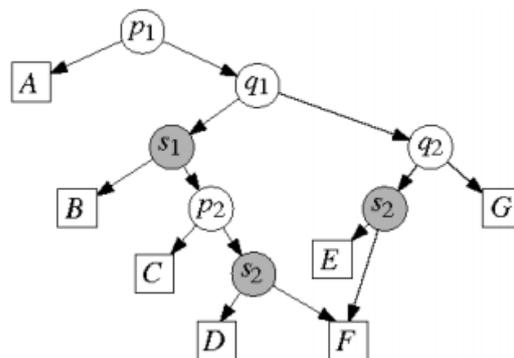
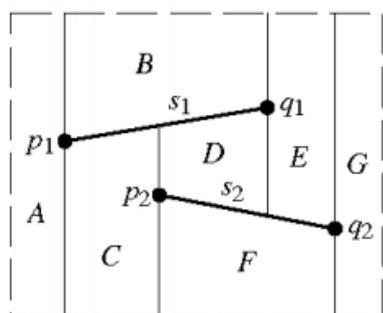
## 2. 算法描述



在引入线段  $s_i$ ，查找该线段穿过的梯形时，采用了 DCEL 结构，首先根据点定位 ( $S_i$  的左端点) 找到第一个梯形，然后根据 DCEL 半边找出下一个梯形  $S_{i+1}$ 。由于线段集是不相交的，从一个梯形区域翻到相邻的梯形区域时，线段只会穿过垂直边，因此 DCEL 结构中只需存储梯形的垂直边，在此实验中，顶点的信息也不重要，DCEL 只需存储边和面得关系即可。

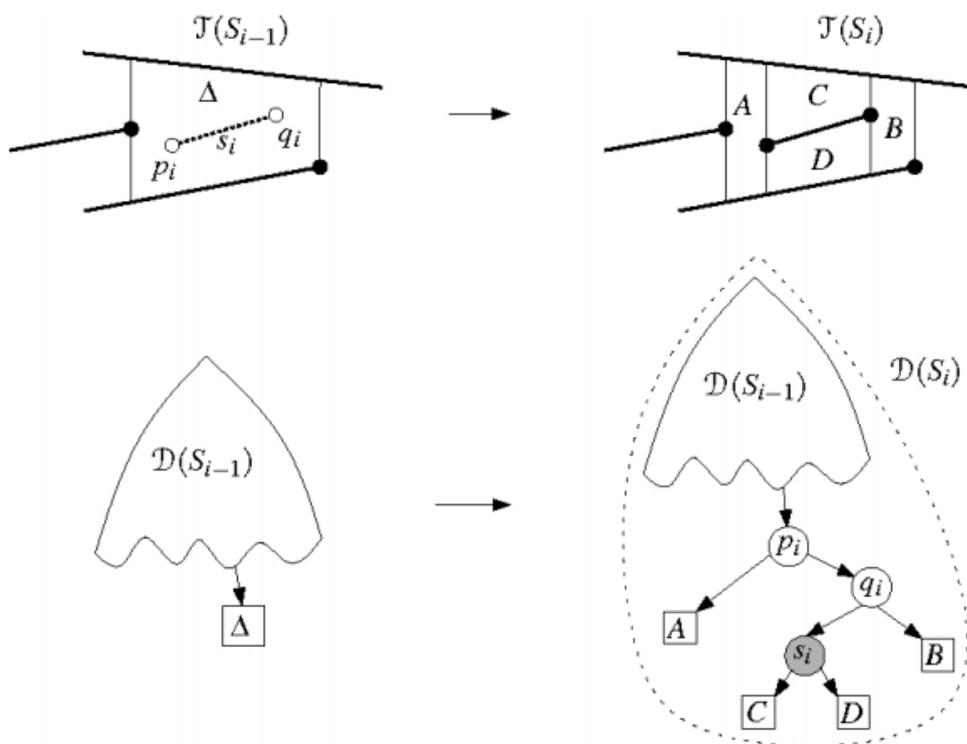
### 3.算法分析

在对点  $q$  进行查询时，要从根节点出发，沿着某条有向路径达到某片叶子，最终达到那片叶子，就对应于  $\tau(S)$  中包含  $q$  的那个梯形。再沿查找路径前进时，每遇到一个新的节点，都要将其与  $q$  进行对比，以确定应该继续前进到其两个子节点中的哪一个。若是  $x$  节点，则按如下形式进行比较：取出存储于该节点处的那个端点；相对应于经过该端点的那条垂线， $q$  是位于左侧还是右侧？若是  $y$  节点，则比较的方法如下：取出存储于该节点的那条线段  $s$ ；相对于这条线段， $q$  是位于上方还是下方？这种算法构造出来的查找结构和梯形图是相互关联的：梯形与叶子是一一对应的；每个梯形都可以通过指针，找到与自己对应的叶子；反过来，每片叶子都可以通过指针，找到与自己对应的梯形。



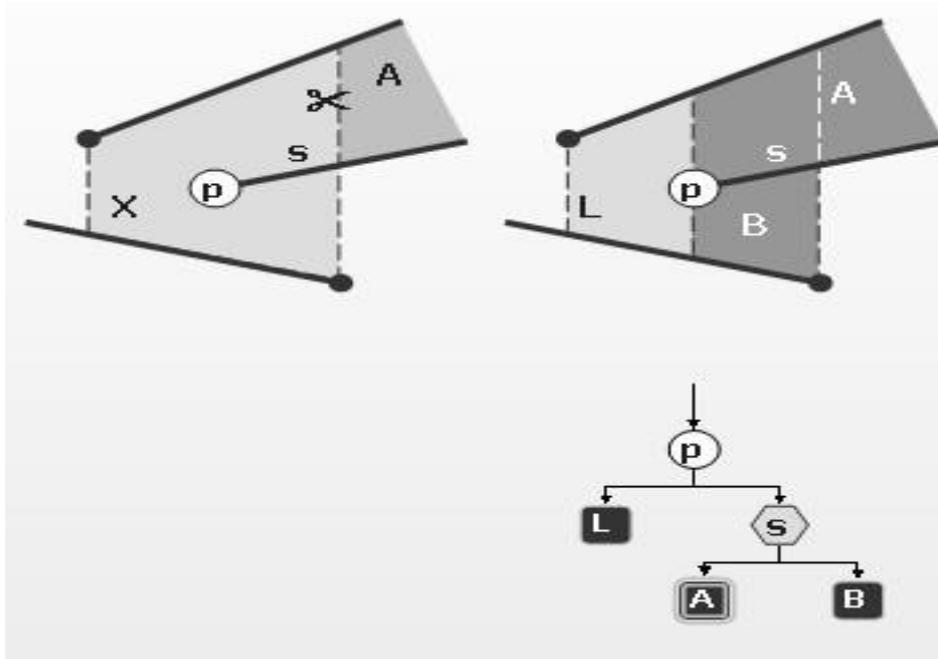
下面是引入线段时的两种情况。

(1) 线段完全落在某个梯形内



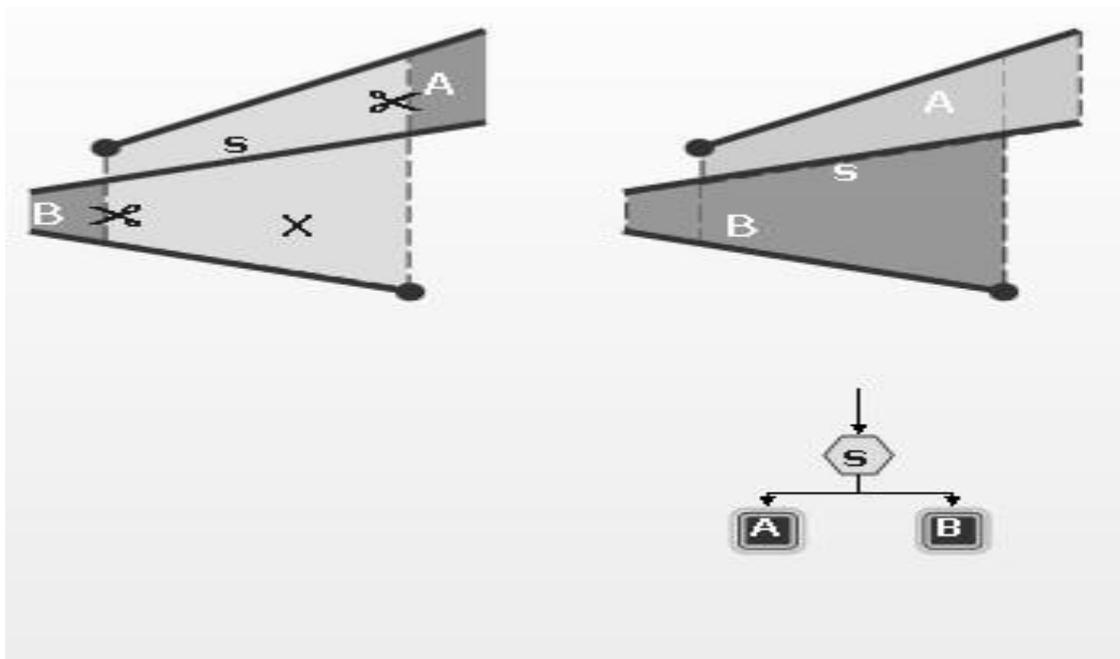
将原先对应于  $\Delta$  的那片叶子，替换为一颗包含四片叶子的小树。这棵树中含有两个  $x$  节点，分别用来对  $s_i$  的左右端点进行比较；还有一个  $y$  节点，用来对  $s_i$  本身进行比较。

(2) 线段一个端点落在梯形中



在上图所示这种情况中（线段的左端点在梯形中），L、B 为新生成的梯形区域，A 将和后面的梯形的一部分融合。如果线段的右端点在梯形中，其情况恰好相反。

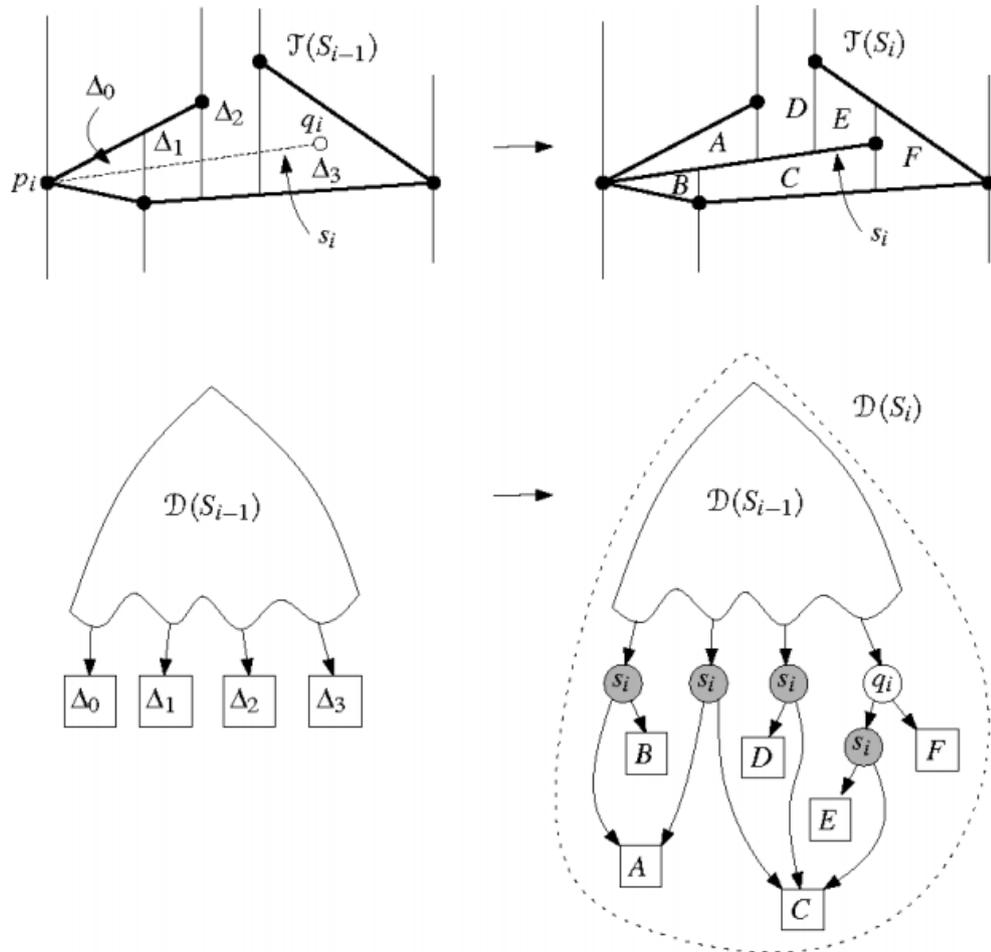
### (3) 线段穿过梯形



可分为两类情况，一类是梯形的左右端点都在线段同侧，一类在异侧，如在异侧，就如上图所示，其中 A 与后面的梯形融合，B 与前面的梯

形（如情况 2 中向后融合的梯形）融合。

按照上述的集中情况不断地分裂合并，最终会完成如下图所示的过程。



删除对应于  $\Delta_0, \Delta_1, \Delta_2, \dots, \Delta_k$  的叶子，然后为每个新的梯形分别生成一片叶子，最后还要引入若干新的内部节点。

对于构造梯形图和搜索结构的算法，时间复杂度为  $O(n \lg n)$ ，查找点时，时间复杂度为  $O(\lg n)$ 。

## 4. 实验所用数据结构

### 1. 查找结构

由三类节点构成，线段的顶点、线段和梯形区域。在梯形区域对应的数据结构中需存储它的父节点，其父节点数目不定，用链表存储。

顶点对应的数据结构：

```
class CVertexNode : public CNode
{
public:
    float x;
    float y;
};
```

线段对应的数据结构：

```
class CLineNode : public CNode
{
public:
    CVertexNode * s;    //起始节点
    CVertexNode * d;    //尾节点
    int no;             //编号
};
```

梯形区域的数据结构：

```
class CTrapezoidalNode : public CNode
{
public:
```

```

void AddParent(CNode *p);

CLineNode* top;          //梯形区域的上边界
CLineNode* bottom;      //下边界
CVertexNode *left;      //梯形区域的左端点
CVertexNode *right;     //梯形区域的右端点
typedef struct Parent{
    CNode *parent;
    struct Parent *next;
}Parent;                 //父节点
Parent *phead;
CHalfEdge *edge;        // 对应的第一条半边
};

```

## 2、梯形图的结构

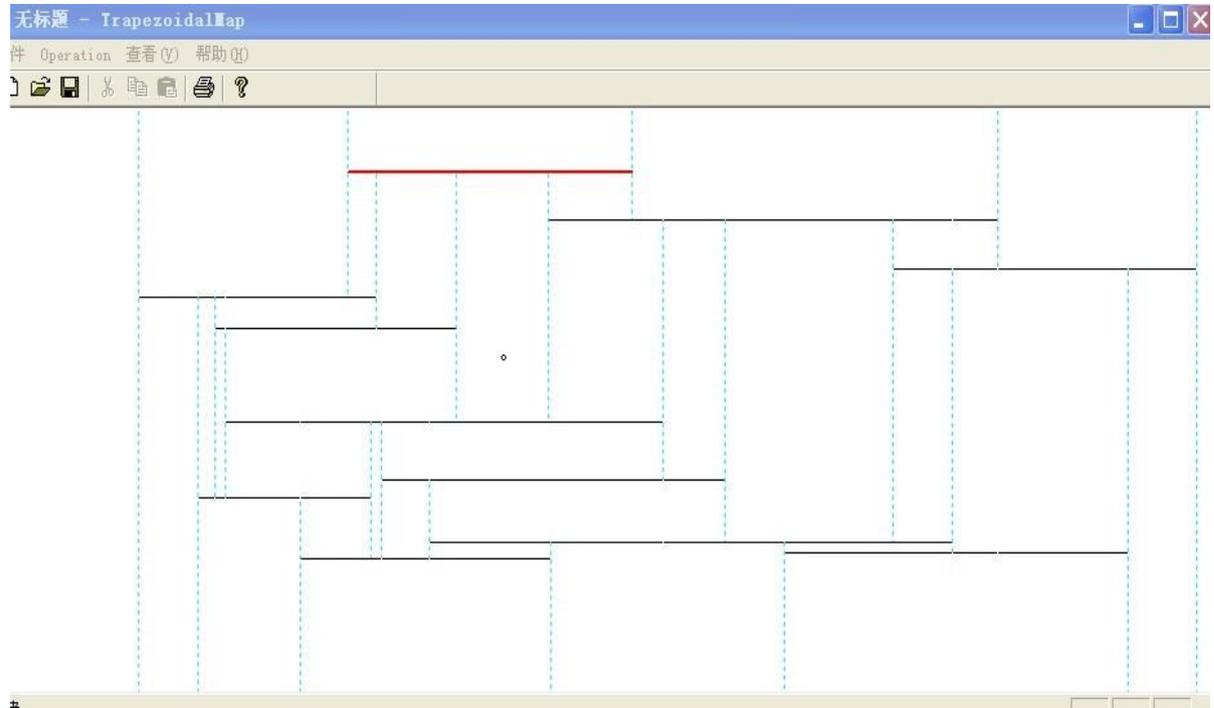
```

class CHalfEdge
{
public:
    float x;              // 半边所在的 x 坐标
    float ymin;          //半边 y 坐标方向的最小值
    float ymax;          //半边 y 坐标的最大值
    CNode *face;
    CHalfEdge *twin;
    CHalfEdge *next;
};

```

};

## 5.实验结果



测试结果