计算机图形学
--实区域填充算法

授课教师： 陈佳舟

邮箱：cjz@zjut.edu.cn

单位：计算机学院数字媒体所

可选课堂报告内容

- 颜色系统
- 颜色的冷暖
- 高动态图像HDR

- 图像去噪
- 图像去运动模糊
- 彩色到灰度的转换

- 图像分割
- 图像显著度检测

- 线条矢量化
- 图像矢量化

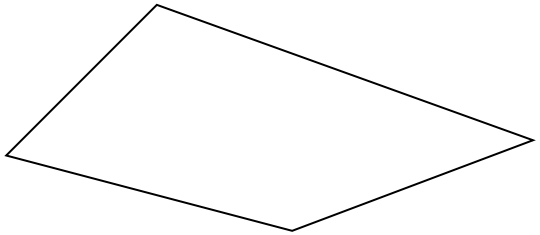
基本图形生成算法

- 图元扫描转换
  - 直线段扫描转换
  - 圆弧扫描转换
- 实区域填充
- 多边形扫描转换
- 图形反走样

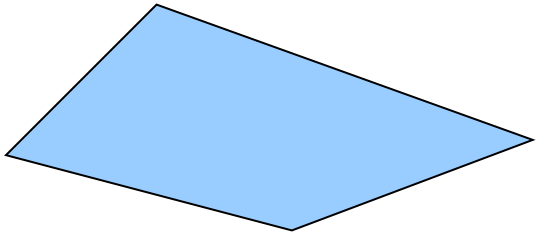
基本概念

光栅图形的基本概念

- 关于光栅图形
  - 本质：点阵表示
  - 特点：面着色，画面明暗自然、色彩丰富
  - 与线框图相比：更加生动、直观、真实感强

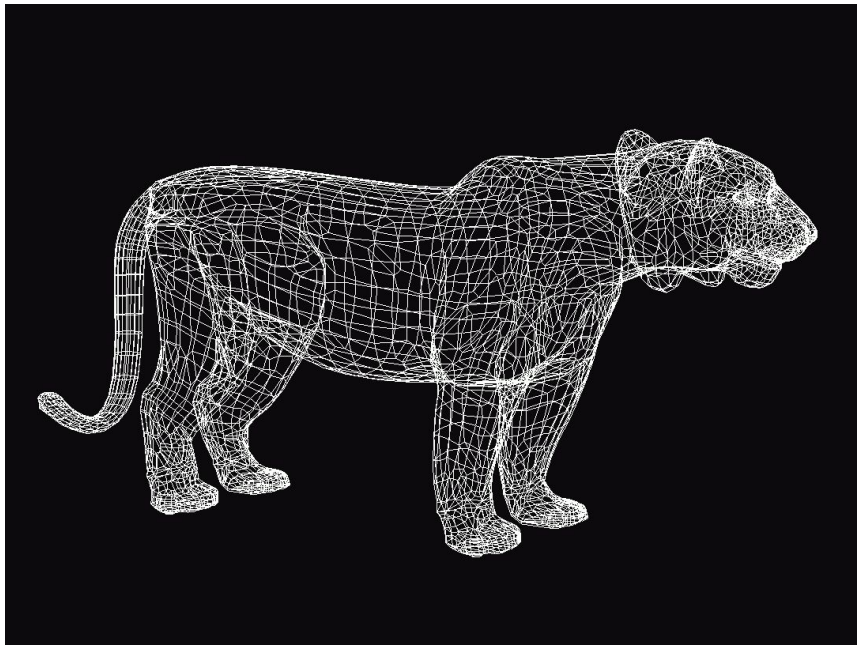


线框平面多边形

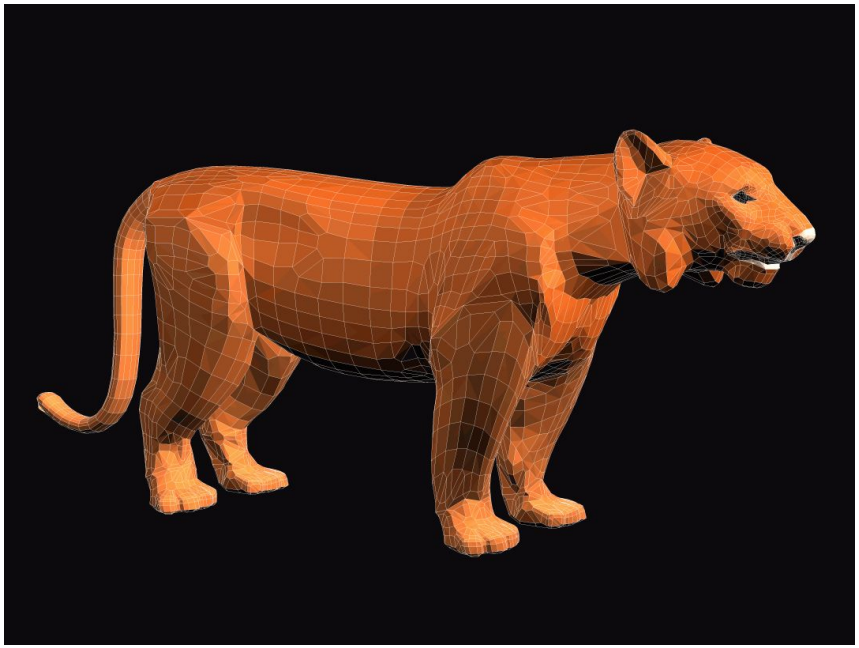


着色的平面多边形

光栅图形的基本概念



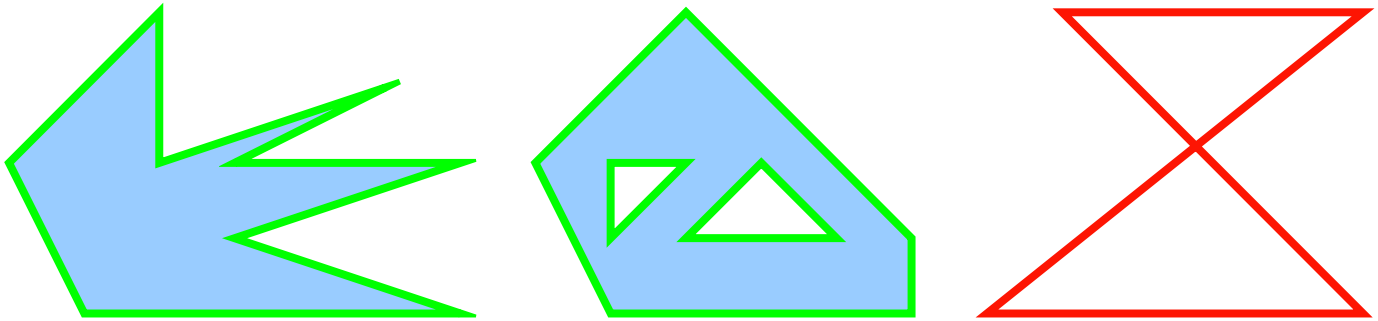
线框多边形物体



填充多边形物体

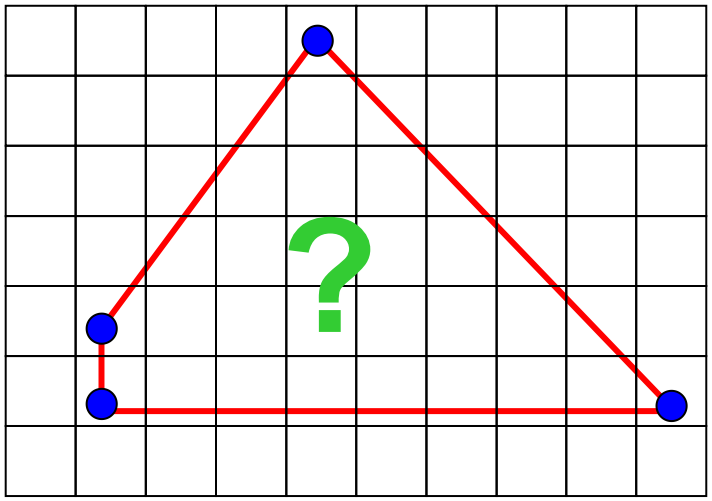
光栅图形的基本概念

- 关于平面多边形
  - 图形学中的多边形：无自相交的简单多边形



  - 图形学中多边形的两种表示方式
    - 顶点表示：用多边形的有序顶点序列表示多边形
    - 点阵表示：用位于多边形内部的像素集合来表示多边形

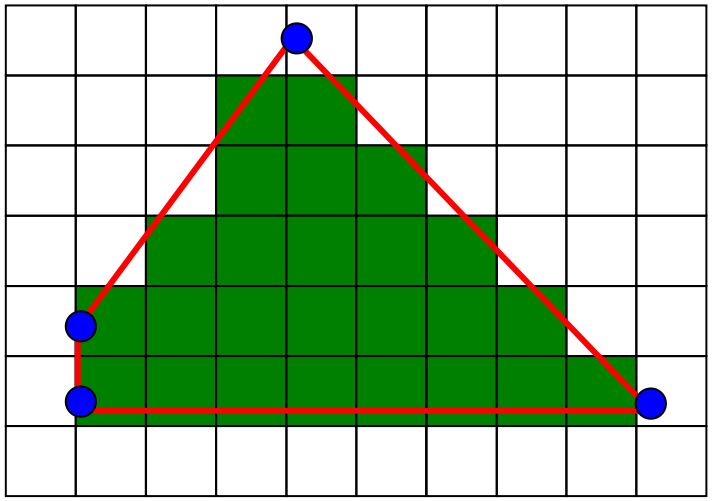
顶点表示



多边形的顶点表示

- 优点
  - 直观
  - 几何意义明显
  - 存贮量小
- 不足
  - 难以判断哪些像素位于多边形内部
  - 不能直接用于多边形着色

点阵表示

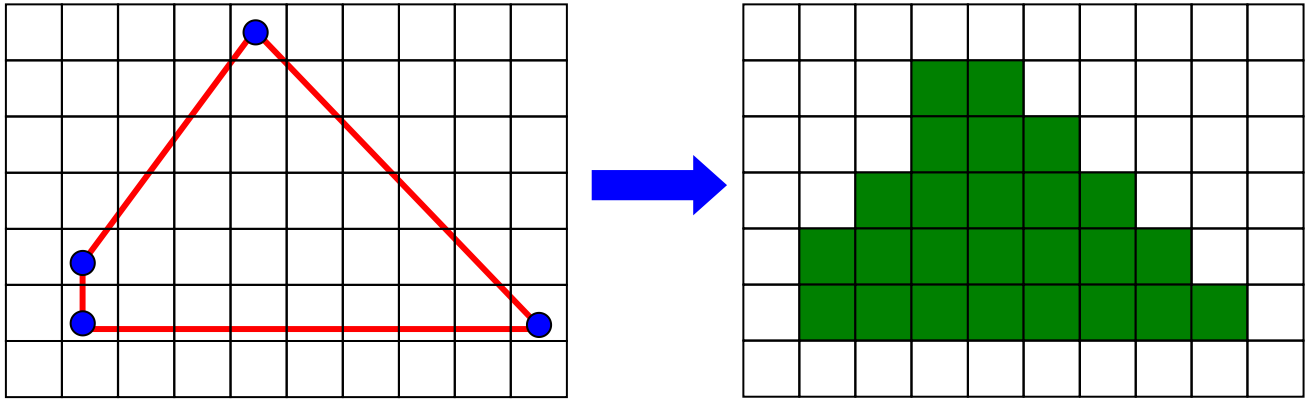


多边形的点阵表示

- 优点
  - 便于用帧缓冲器(frame buffer)表示图形
  - 面着色所需的图形表示
- 缺点
  - 丢失了几何信息
  - 占用存储空间多

多边形的扫描转换

- 多边形的扫描转换：把顶点表示转换为点阵表示
  - 从多边形的给定边界出发，求出其内部的各个像素
  - 并给帧缓冲器中各个对应元素设置相应灰度或颜色



多边形的顶点表示　　多边形的点阵表示

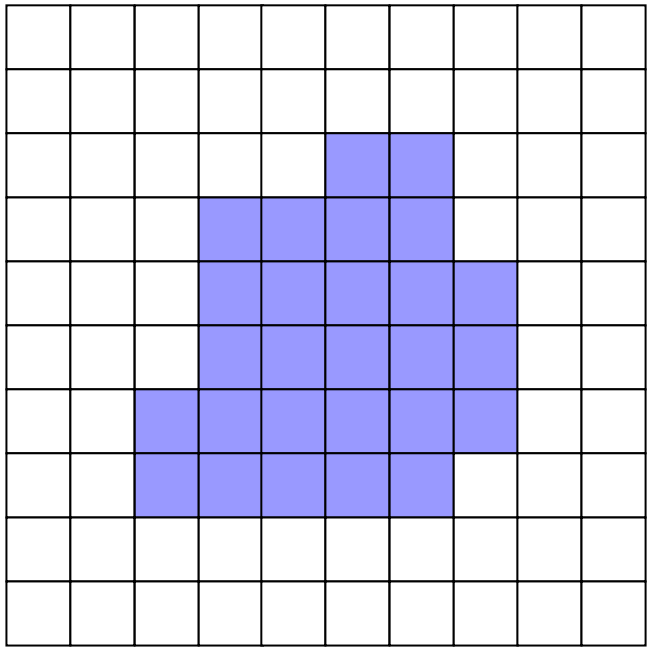
区域填充

点阵表示的区域填充

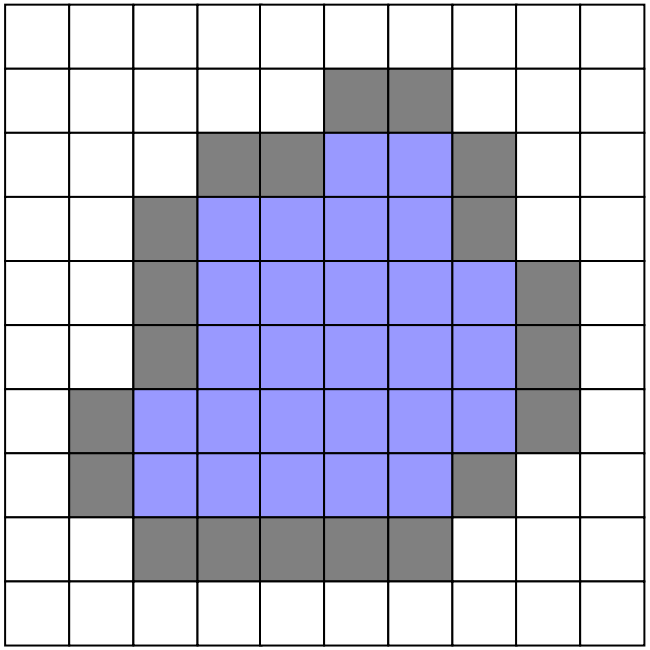
- 区域的定义：已经表示成点阵的像素集合
- 区域的表示：
  - 内部表示：把给定区域内的像素枚举出来
    - 区域内所有像素都着同一种颜色
    - 区域边界像素不能着上述颜色
  - 边界表示：把区域边界上的像素枚举出来
    - 边界上所有像素都着同一种颜色
    - 区域内部像素不能着上述颜色

区域表示

- 内部表示

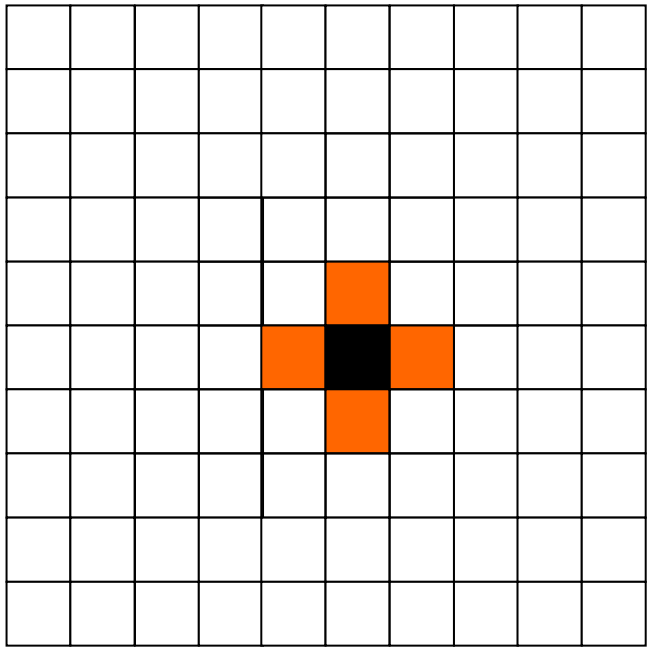


- 边界表示

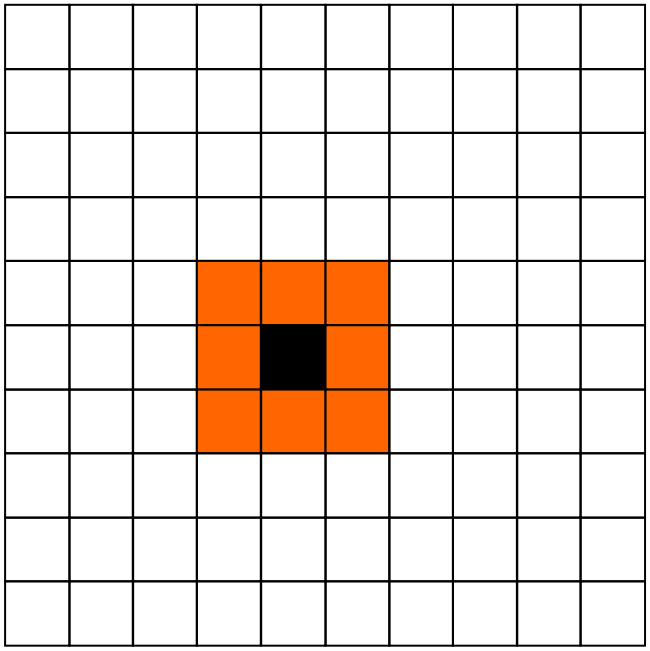


区域的类型

- 四连通邻域



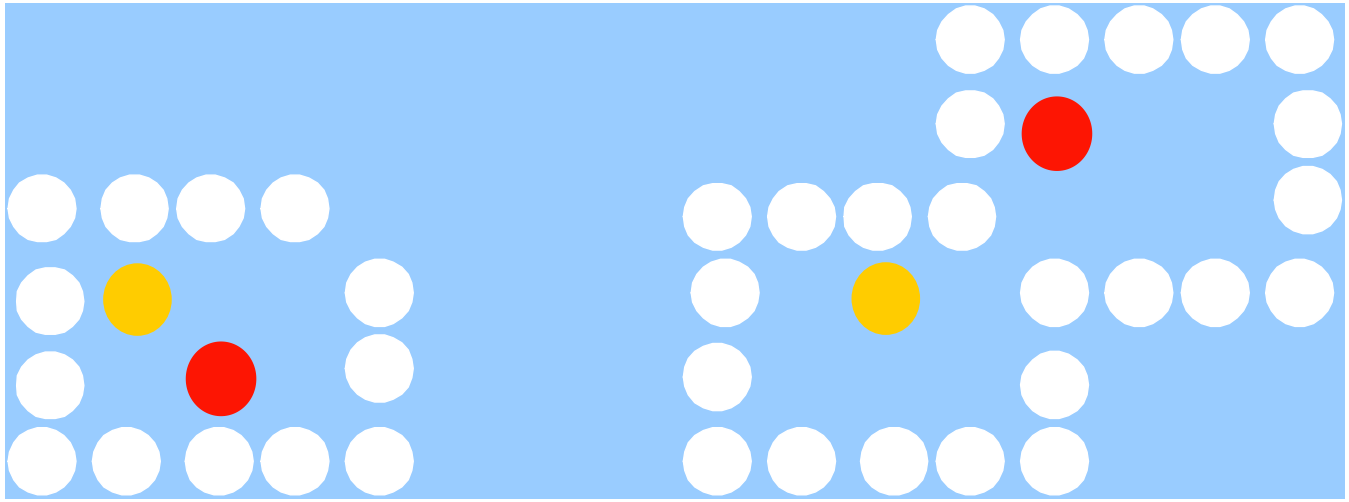
- 八连通邻域



区域的类型

- 四连通区域：区域内任意两个像素，从一个像素出发，可以通过上、下、左、右四种运动，到达另一个像素
- 八连通区域：区域内任意两个像素，从一个像素出发，可以通过水平、垂直、正对角线、反对角线八种运动，到达另一个像素

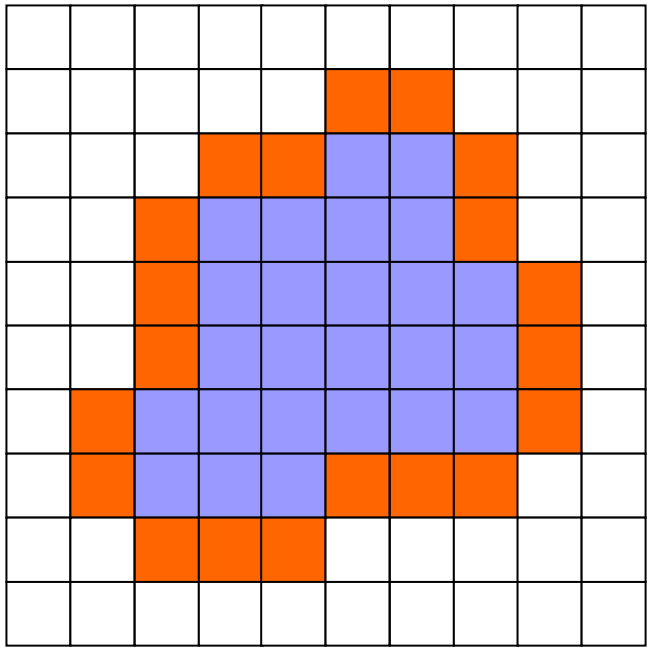
区域的类型



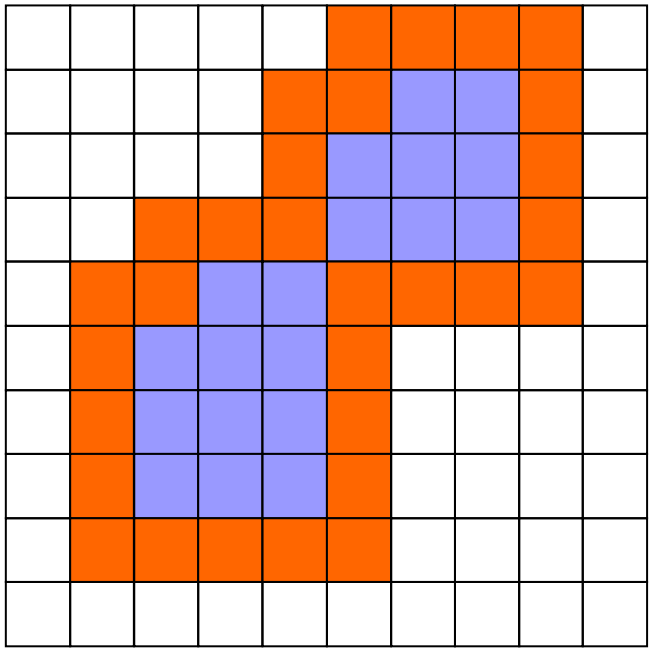
四连通区域　　八连通区域

区域的类型

- 四连通区域实例



- 八连通区域实例



区域的类型

- 四连通和八连通区域的关系
  - 四连通区域 $\subseteq$ 八连通区域 (反之不成立)
  - 四连通区域的边界是八连通区域
  - 八连通区域的边界是四连通区域

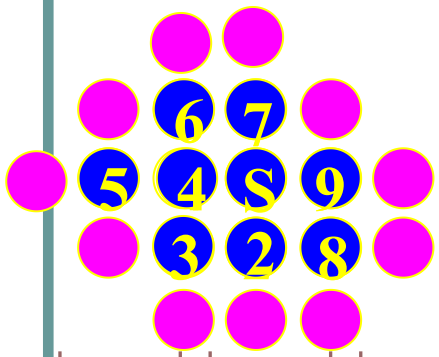
内部表示区域种子填充算法

- 假设内部表示区域为G，其中的像素原有颜色为G0，需要填充的颜色为G1。
- 算法需要提供一个种子点(x,y)，它的颜色为G0。
- 具体算法如下(四连通区域)

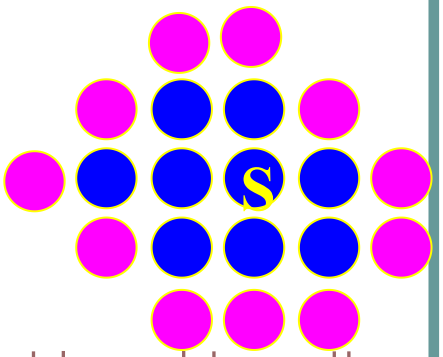
简单的种子填充算法（4连通边界）

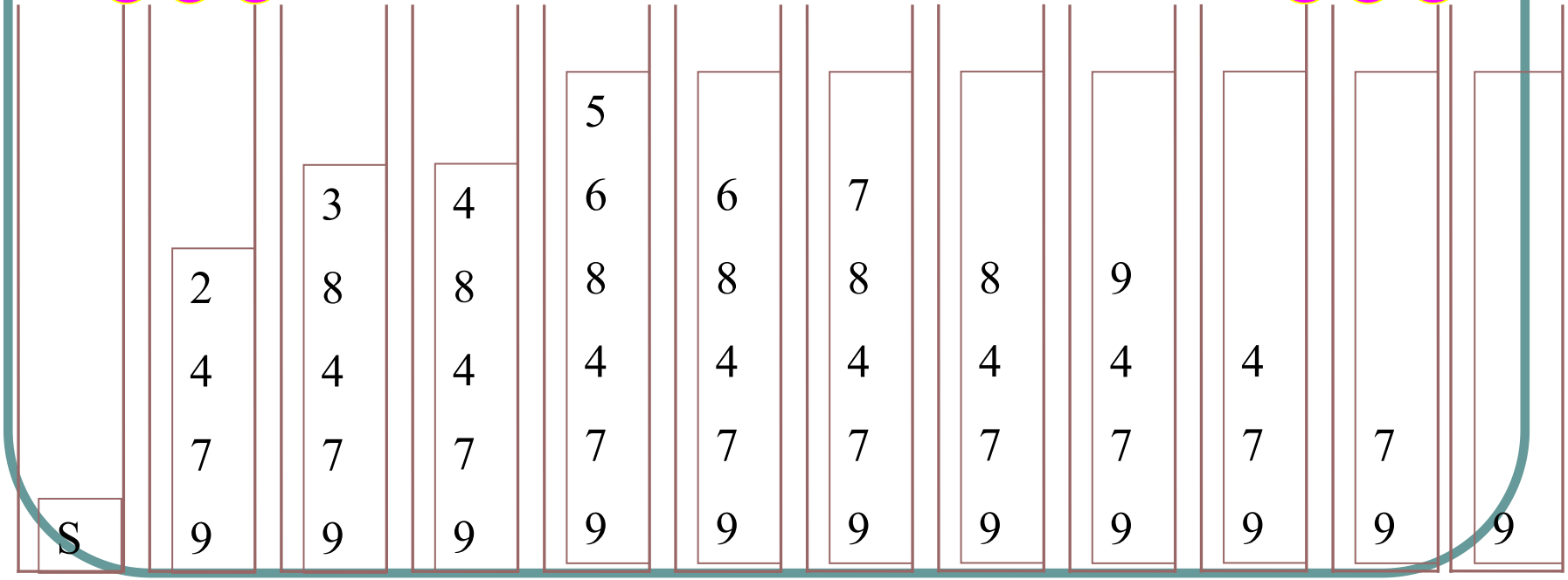
- 种子像素入栈
- 当栈非空时，重复以下步骤：
  - 栈顶像素出栈
  - 将出栈象素置成填充色
  - 按左、上、右、下顺序检查与出栈象素相邻的四象素，若其中某象素不在边界上且未被置成填充色，则将其入栈

填充算法演示


6 7
5 4 S 9
3 2 8

缺点？


S


S
2 4 7 9
3 8 4 7 9
4 8 4 7 9
5 6 8 4 7 9
6 8 4 7 9
7 8 4 7 9
8 4 7 9
9 4 7 9
4 7 9
7 9
9

4-connected boundary-fill

```
void BoundaryFill4(int x,int y,int fill,int
     boundary)
{
  int current;
  current = getpixel(x, y);
  if ((current != boundary) && (current != fill))
  {
     putpixel(x, y, fill);
     BoundaryFill4(x+1, y, fill, boundary);
     BoundaryFill4(x-1, y, fill, boundary);
     BoundaryFill4(x, y+1, fill, boundary);
     BoundaryFill4(x, y-1, fill, boundary);
  }
}
```

4-connected boundary-fill

```
void FloodFill4(int x,int y,int fillColor,int
     oldColor)
{
  int current;
  current = getpixel(x, y);
  if (current == oldColor)
  {
     putpixel(x, y, fillColor);
     BoundaryFill4(x+1, y, fillColor, oldColor);
     BoundaryFill4(x-1, y, fillColor, oldColor);
     BoundaryFill4(x, y+1, fillColor, oldColor);
     BoundaryFill4(x, y-1, fillColor, oldColor);
  }
}
```

种子填充的优缺点

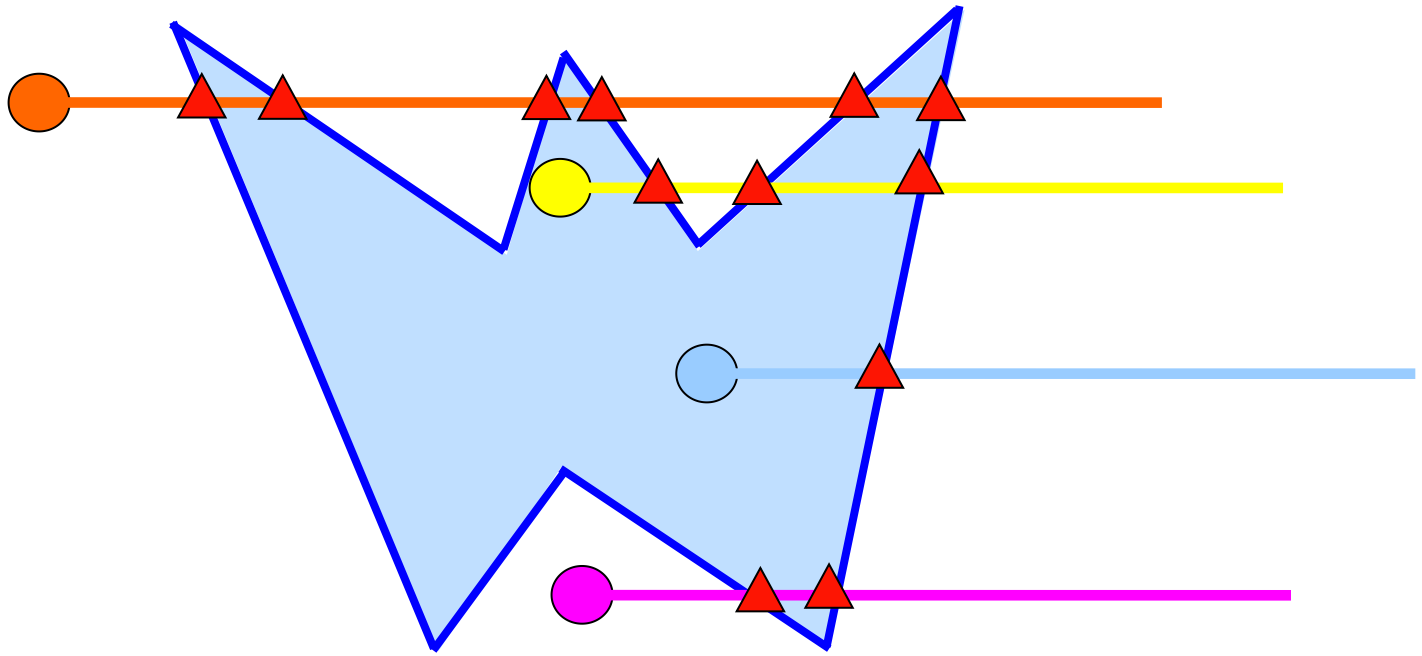
- 优点
  - 适用于任何封闭的形状
- 缺点
  - 需要制定种子点
  - 效率较低

多边形的扫描转换

逐点判断算法

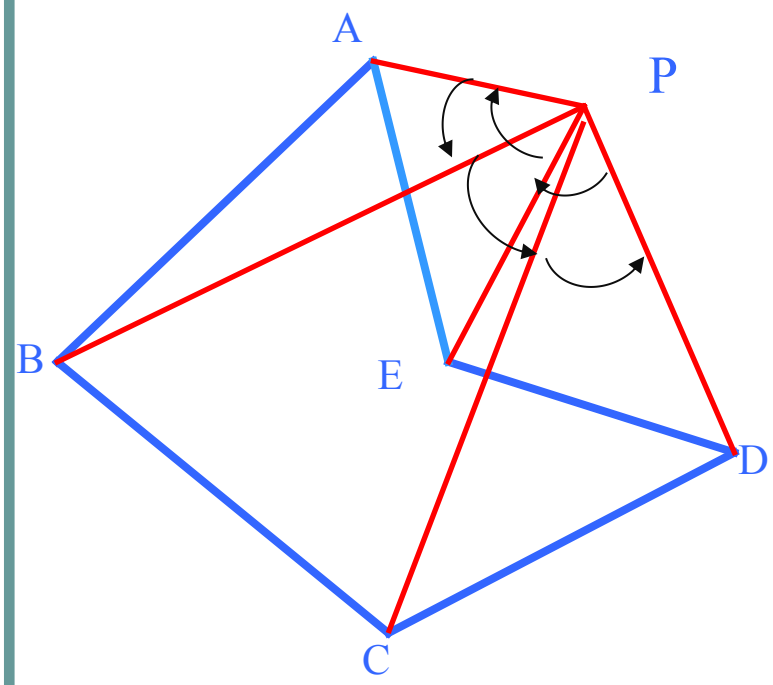
- 逐点判断算法：逐个像素判别其是否位于多边形内部
- 判断一个点是否位于多边形内部：射线法
  - 从当前像素发射一条射线，计算射线与多边形的交点个数
    - 内部：奇数个交点
    - 外部：偶数个交点

逐点判断算法 - 射线法

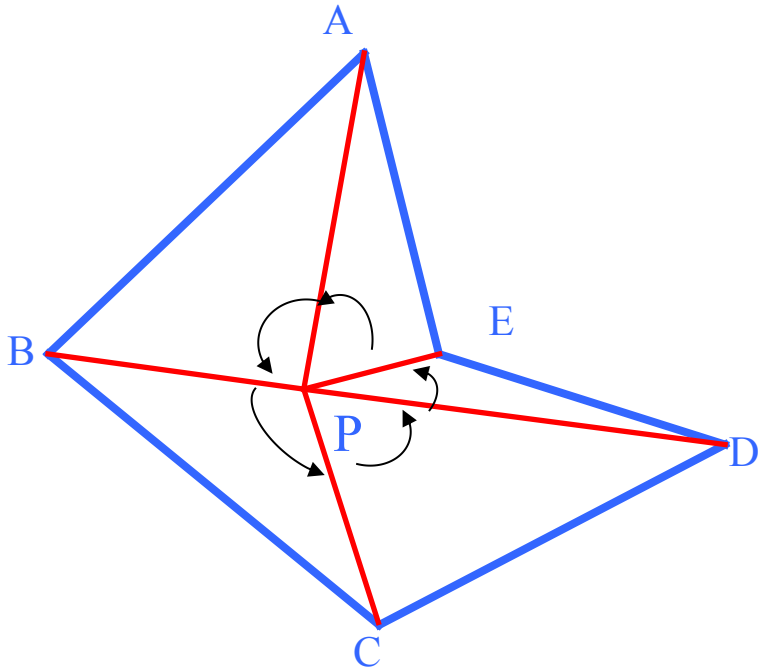


判断一点是否位于多边形内部？

逐点判断算法 - 夹角法



若夹角和为0，则点p在多边形外

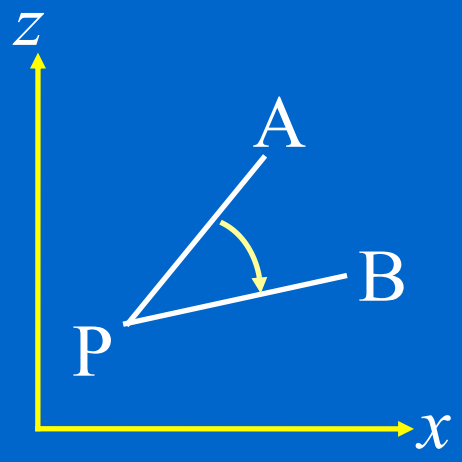


若夹角和为360°，则点p在多边形内

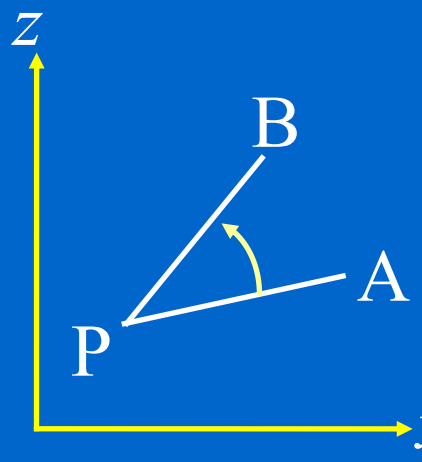
夹角如何计算？

- 大小：利用余弦定理 $c^2 = a^2 + b^2 - 2ab\cos(\gamma)$
- 方向：令

$$T = \begin{vmatrix} x_A - x_P & z_A - z_P \\ x_B - x_P & z_B - z_P \end{vmatrix} = (x_A - x_P)(z_B - z_P) - (x_B - x_P)(z_A - z_P)$$



当T<0时，AP斜率>BP斜率，为顺时针角



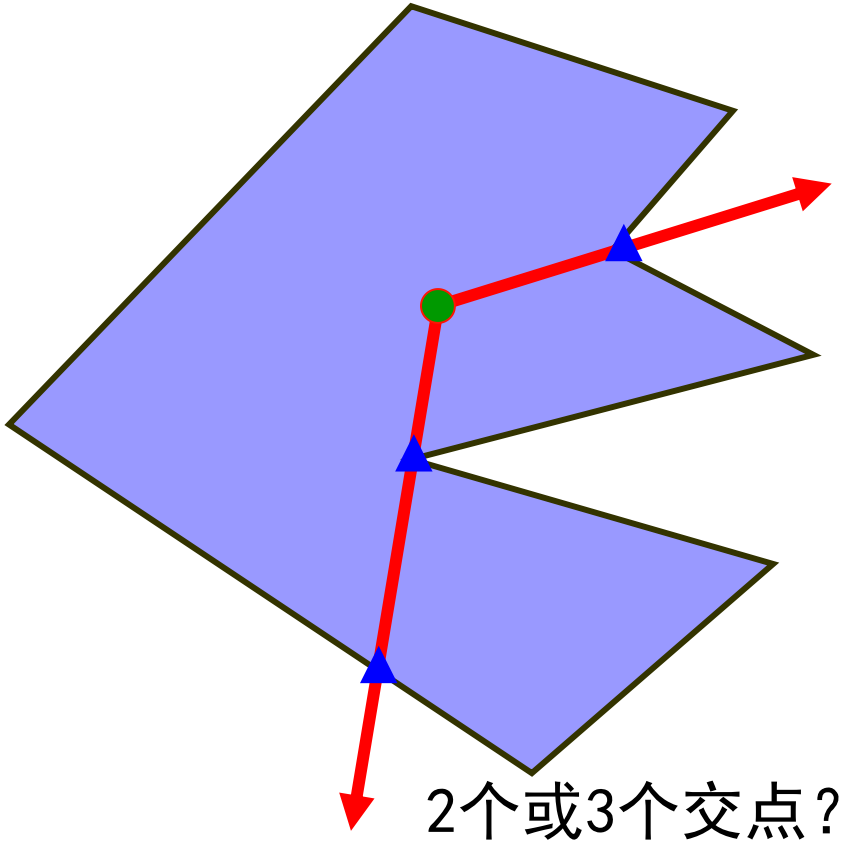
当T>0时，AP斜率<BP斜率，为逆时针角

逐点判断算法

- 算法描述

```
for(y=0; y<=y_resolution; y++)
for(x=0; x<=x_resolution; x++)
{
    if(inside(polygon, x+0.5, y+0.5))
      setpixel(framebuffer,x,y,polygon_color)
    else
      setpixel(framebuffer,x,y,background_color)
}
```

逐点判断算法中的奇异情况



1个或2个交点？

2个或3个交点？

逐点判断算法的不足

- 速度慢：几十万甚是几百万像素的多边形内外判断，大量的求交、乘除运算
- 没有考虑像素之间的联系

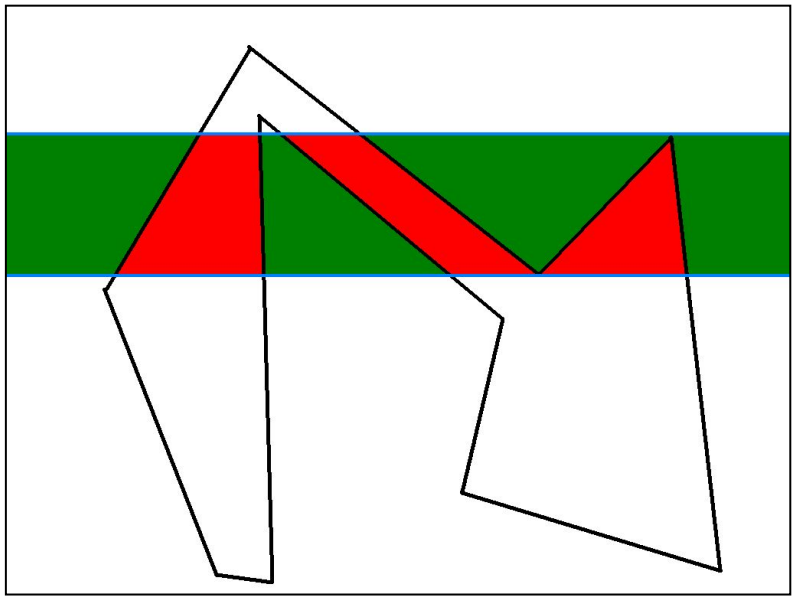
- 结论：逐点判断算法不可取！

相邻像素之间的连贯性

- 扫描线算法充分利用了相邻像素之间的连贯性，避免了对像素的逐点判断和求交运算，提高了算法效率
- 各种连贯性的定义
  - 区域连贯性
  - 扫描线连贯性
  - 边的连贯性

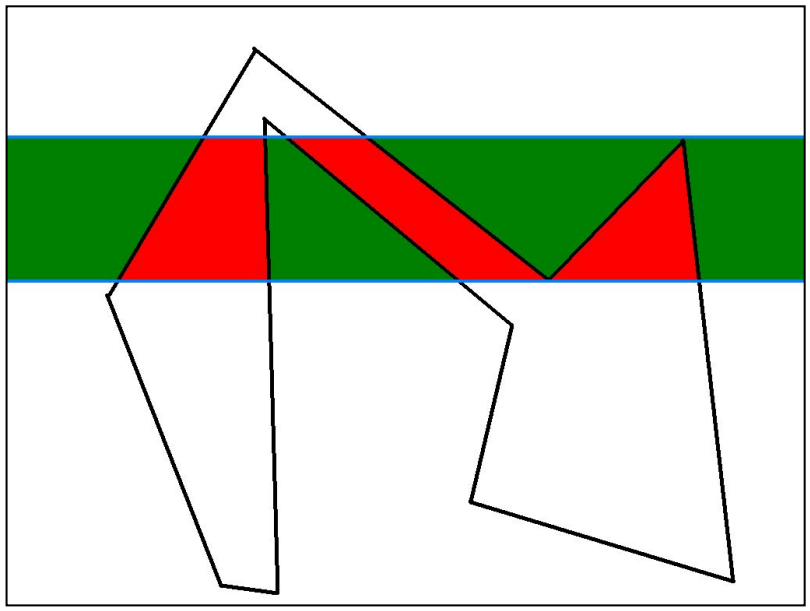
区域连贯性

- 区域的连贯性是指多边形定义的区域内部相邻的像素具有相同的性质。例如具有相同的颜色



区域的连贯性

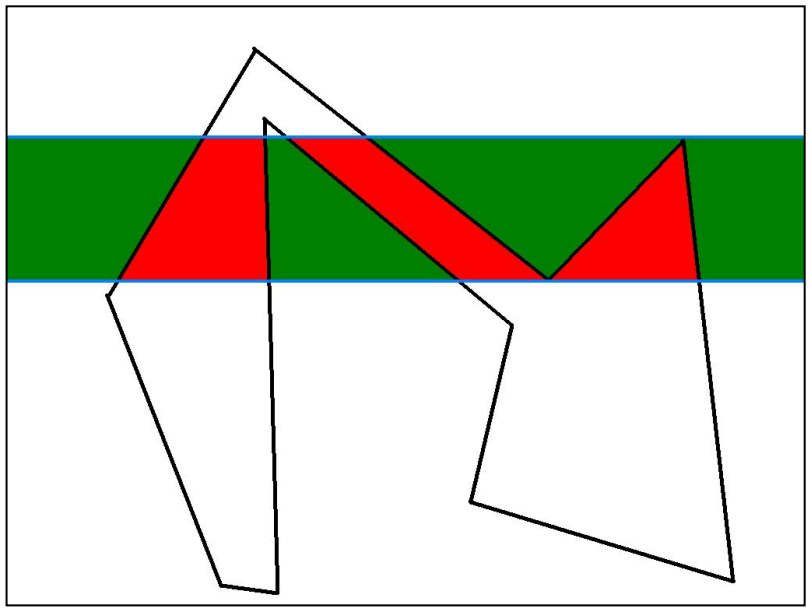
区域连贯性



区域的连贯性

- 两条扫描线之间的长方形区域被所处理的多边形分割成若干梯形(三角形可以看作退化梯形)
- 梯形的底边为扫描线，梯形的腰为多边形的边或窗口边缘

区域连贯性



区域的连贯性

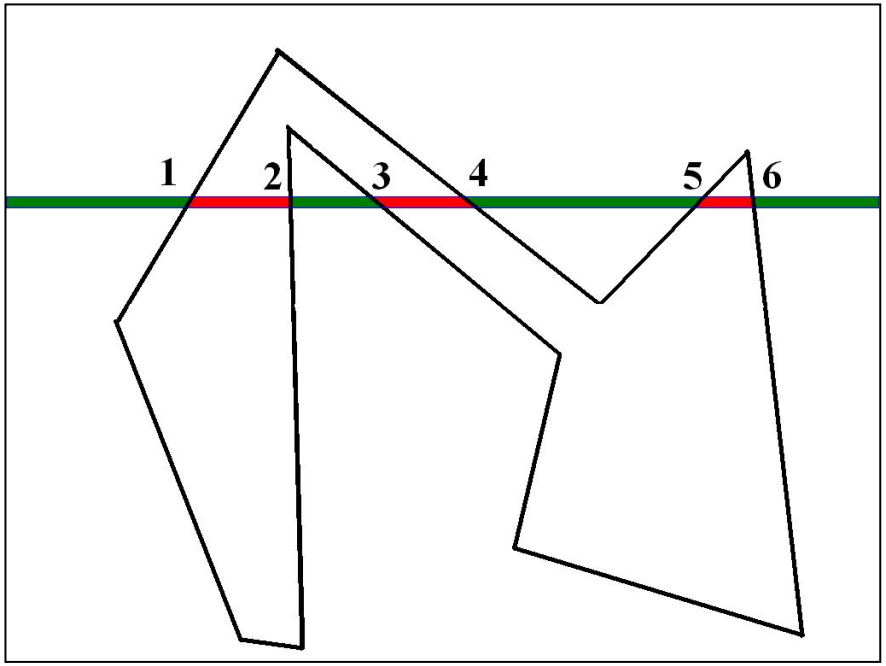
- 梯形分为两类：多边形内部和多边形外部
- 两类梯形在多边形内部相间排列(相邻的两个梯形必然有一个位于多边形内部，有一个在多边形外部)

区域连贯性

- 推论：如果上述梯形属于多边形内(外)，那么该梯形内所有点的均属于多边形内(外)。
- 效率提高的根源：逐点判断→区域判断

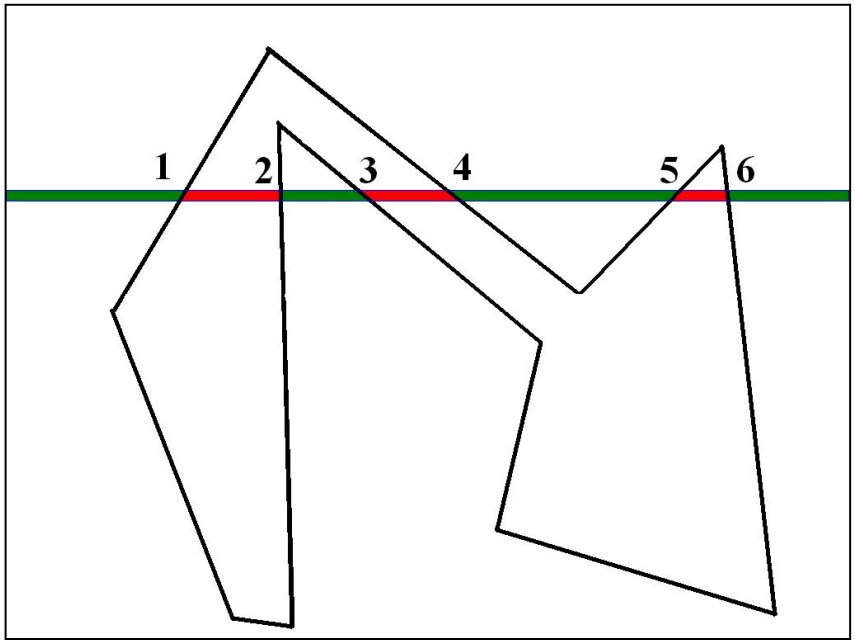
扫描线连贯性

- 区域连贯性在一条扫描线上的反映



扫描线的连贯性

扫描线连贯性



扫描线的连贯性

- 交点序列：扫描线与多边形的交点个数为偶数(1,2,3,4,5,6)
- 红色区间(1,2)、(3,4)、(5,6)位于多边形内部
- 其余绿色区间位于多边形外部
- 两类区间相间排列

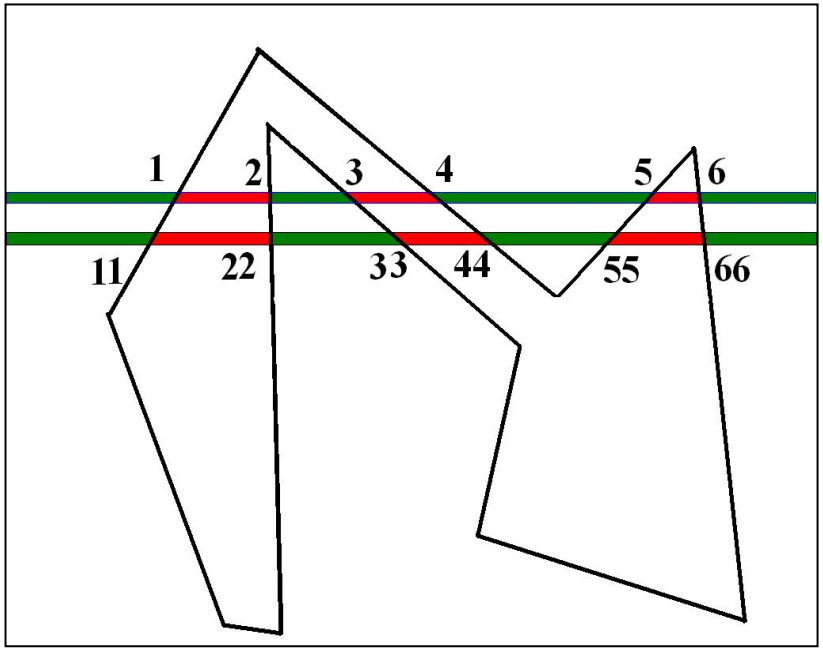
扫描线连贯性

- 推论：如果上述交点区间属于多边形内(外)，那么该区间内所有点均属于多边形内(外)。

- 效率提高的根源：逐点判断→区间判断

边的连贯性

- 边的连贯性：直线的线性性质在光栅上的表现



边的连贯性

扫描线与边的交点

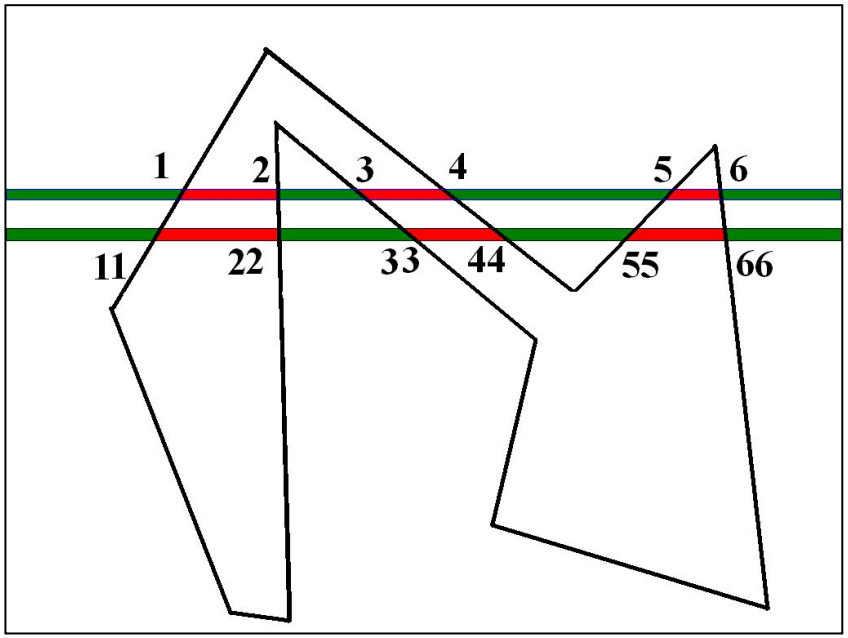
1 (x_1,y_1)

2 (x_2,y_2)

11 (x_{11},y_{11})

22 (x_{22},y_{22})

边的连贯性



边的连贯性

- 相邻扫描线($y_1=y_{11}+1$)与多边形的同一条边的交点存在如下关系：

$$\frac{y_1 - y_{11}}{x_1 - x_{11}} = k$$

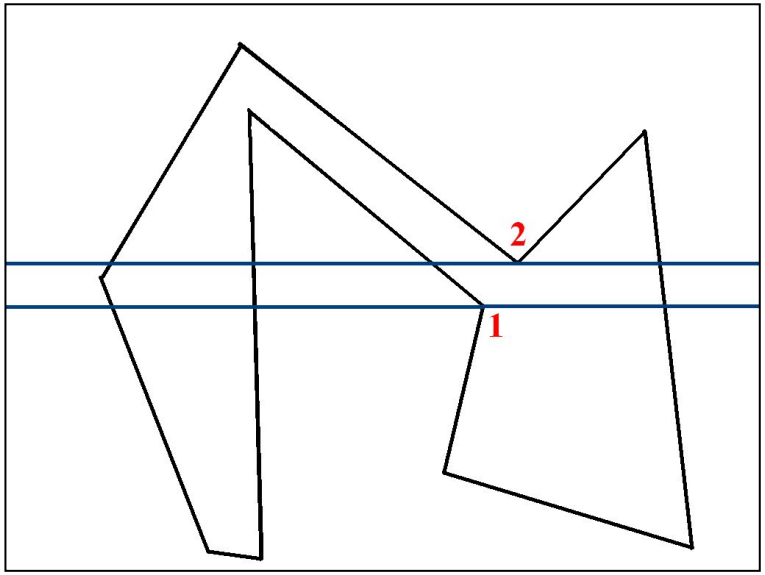
$$x_1 = x_{11} + \frac{1}{k}$$

- 当知道扫描线与一条边的一个交点之后，通过上述公式可以通过增量算法迅速求出其他交点

边的连贯性

- 推论：边的连贯性是连接区域连贯性和扫描线连贯性的纽带。
- 扫描线连贯性“＋”边连贯性“＝”区域连贯性

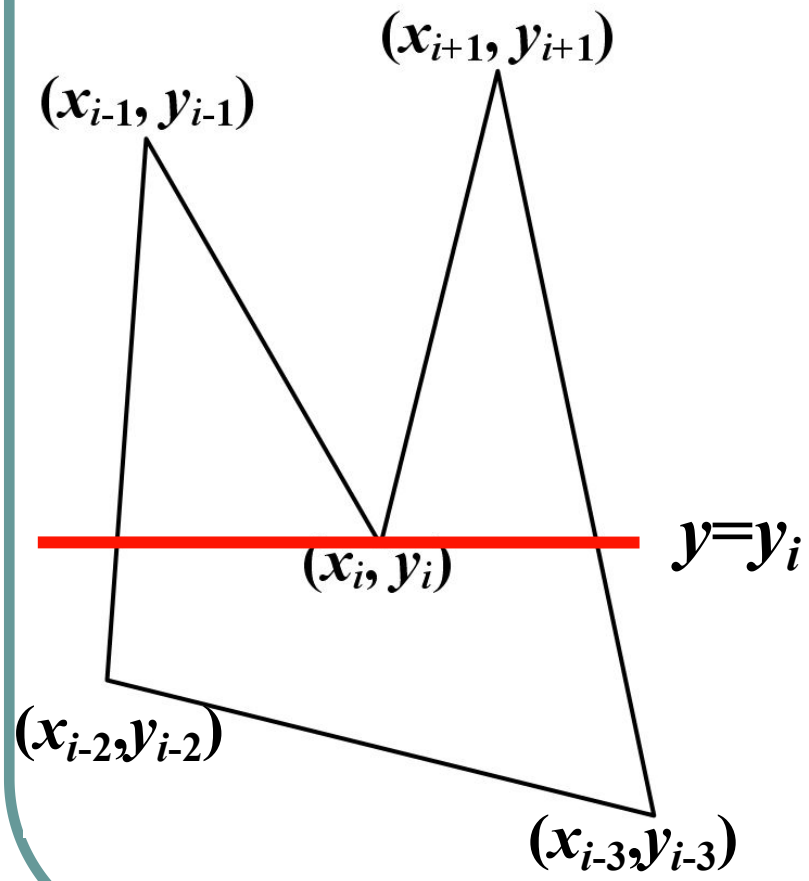
奇异点



奇异点示意图

- 奇异点：扫描线与多边形交交于多边形的顶点
- 奇异点计为几个交点？
  - 扫描线1：一个交点
  - 扫描线2：两个交点

奇异点的分类

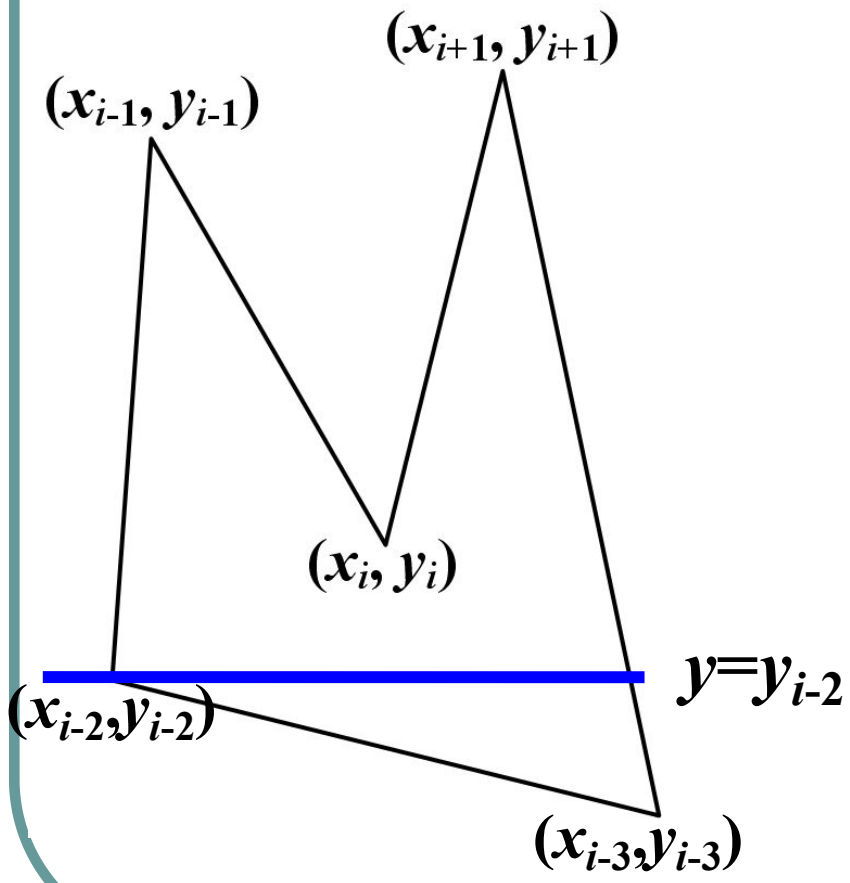


- 极值点：相邻三个顶点的y坐标满足如下条件：

$$(y_{i-1}-y_i)(y_{i+1}-y_i)\geq 0$$

即相邻三个顶点位于扫描线的同一侧

奇异点的分类



- 非极值点：相邻三个顶点的y坐标满足如下条件：

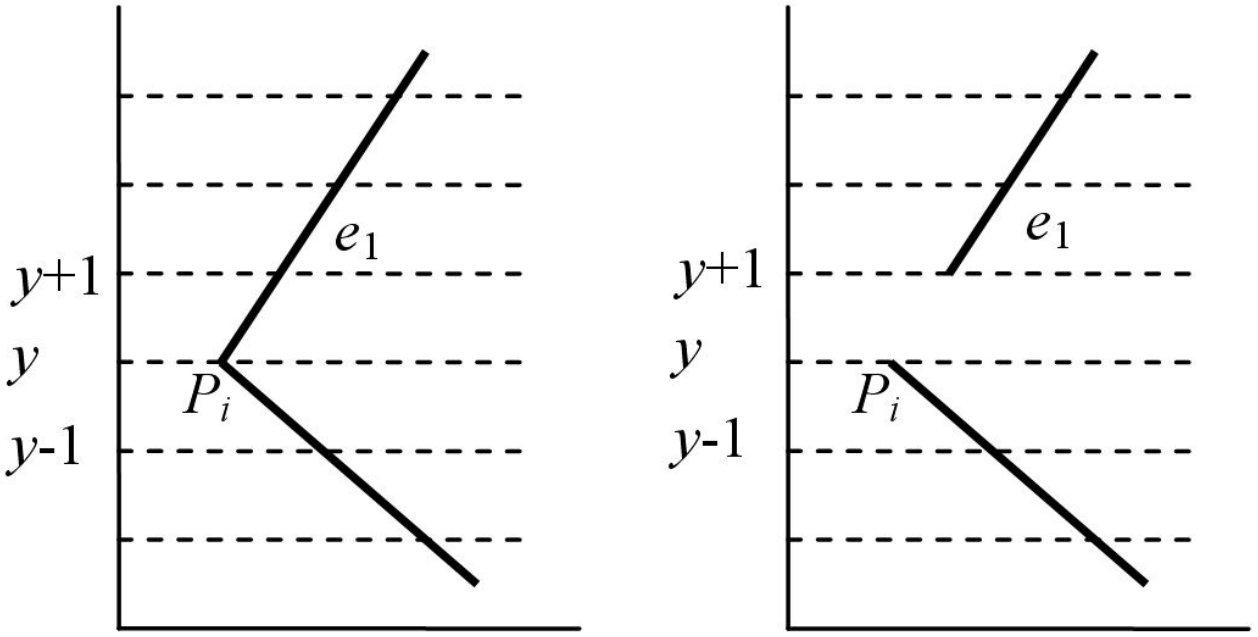
  $(y_{i-3}-y_{i-2})(y_{i-1}-y_{i-2})<0$

  即相邻三个顶点位于扫描线的两侧

奇异点的处理

- 奇异点的处理
  - 在**极值点**处，按**两个交点**计算
  - 在**非极值点**处，按**一个交点**计算
- 实际计算前，奇异点(非极值点)的预处理
  - 将扫描线上方线段截断一个单位，这样扫描线就只与多边形有一个交点。

奇异点的处理

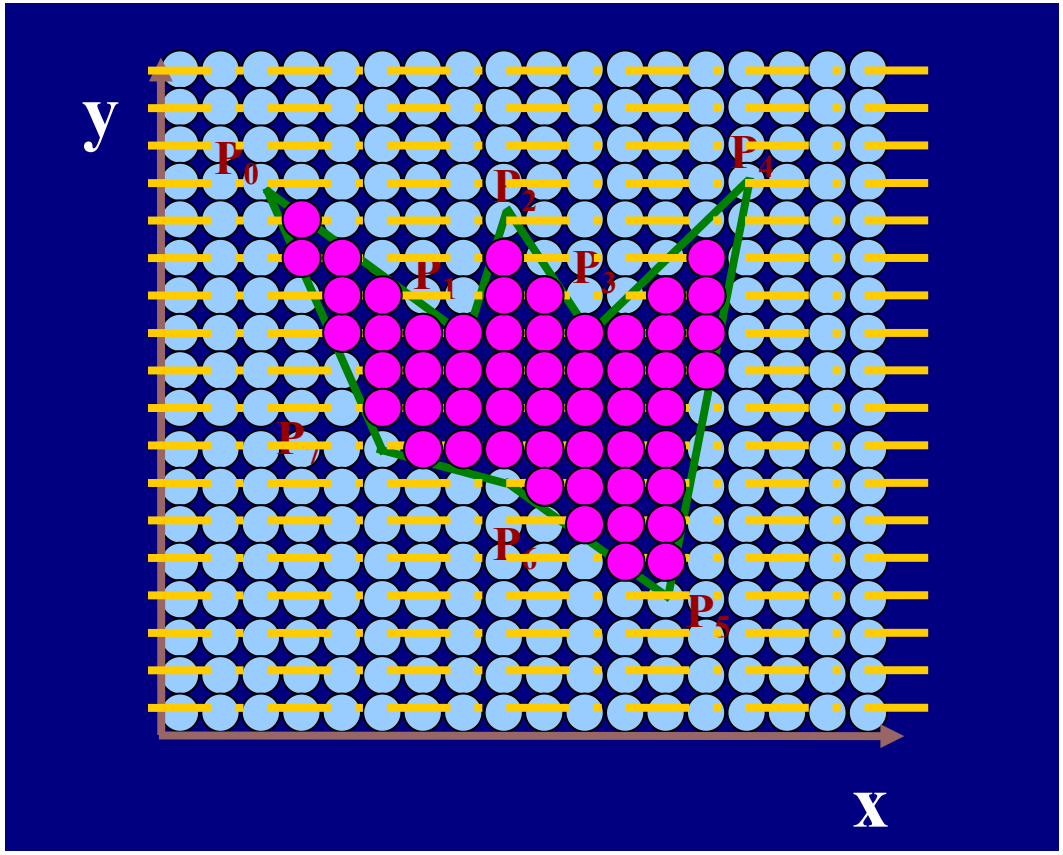


扫描线 y 通过非极值顶点 P_i，在进行扫描转换时，以 P_i 为下端点的边 e_1 在 P_i 处截掉一个像素的宽度。

多边形扫描转换算法

- 核心思想(从下到上扫描)
  - 计算扫描线 $y = y_{\min}$ 与多边形的交点，通常这些交点由多边形的顶点组成
  - 根据多边形边的连贯性，按从下到上的顺序求得各条扫描线的交点序列
  - 根据区域和扫描线的连贯性判断位于多边形内部的区段
  - 对位于多边形内的直线段进行着色

多边形扫描转换算法



扫描转换示意图

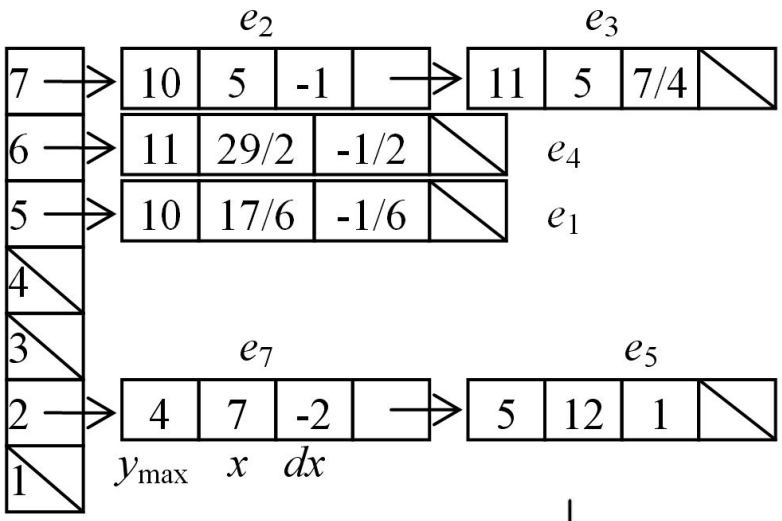
多边形扫描转换算法

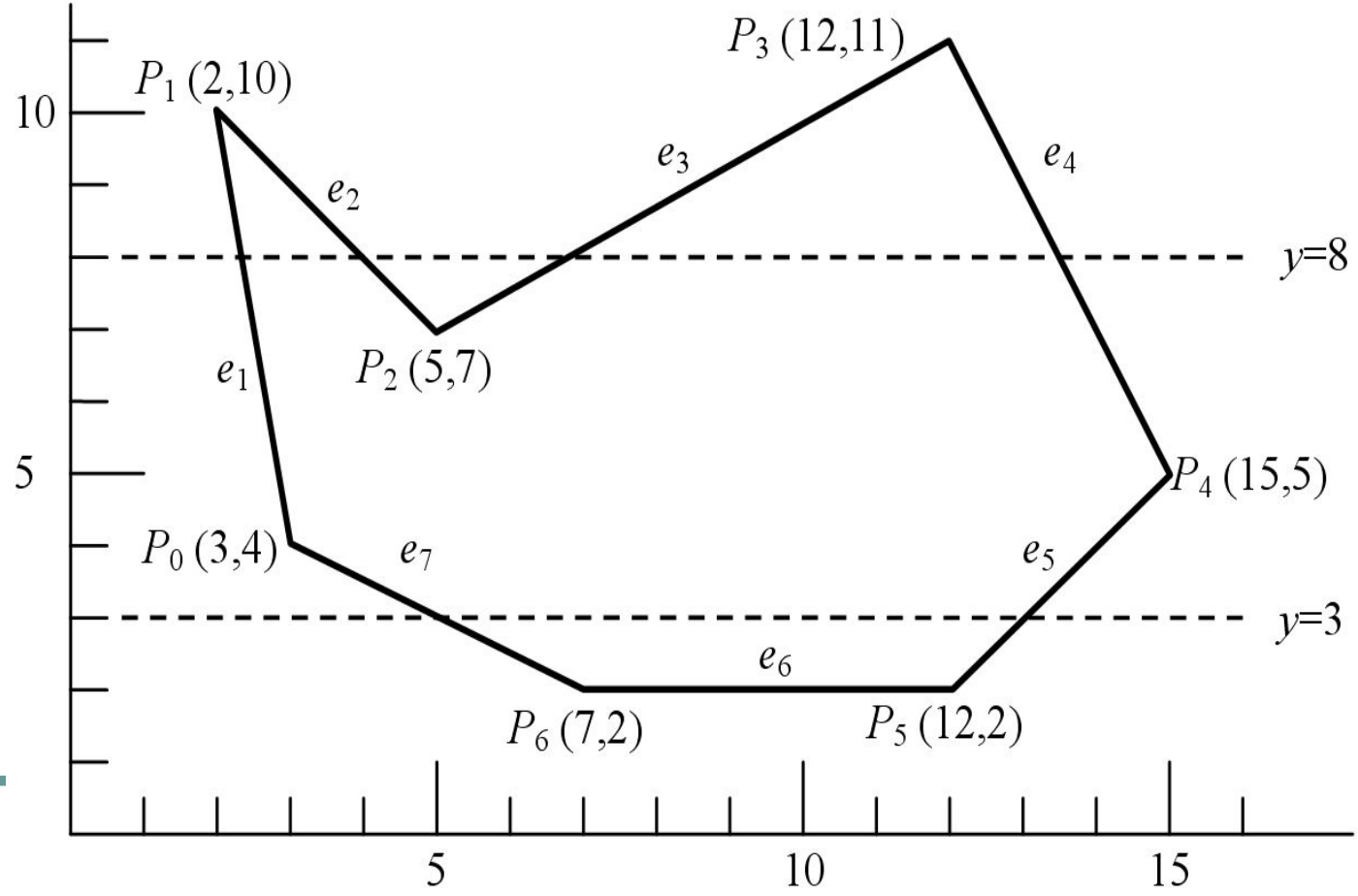
- 为了实现上述思想，算法中需要采取灵活的数据结构。
  - 分类的边表ET (Sorted Edge Table)：记录多边形信息
  - 活化边链表AEL (Active Edge List)：记录当前扫描线信息
- 它们共同基础是边的数据结构

边的数据结构

- 边的数据结构
  - y_{max}：边的上端点的y坐标
  - x：边的下端点x坐标，在活化边链表中，表示扫描线与边的交点的x坐标
  - dx：边的斜率的倒数
  - next：指向下一条边的指针

边的数据结构实例


e_2
e_3
7
10
5
-1
11
5
7/4
6
11
29/2
-1/2
e_4
5
10
17/6
-1/6
e_1
4
3
e_7
e_5
2
4
7
-2
5
12
1
1
y_{max}
x
dx

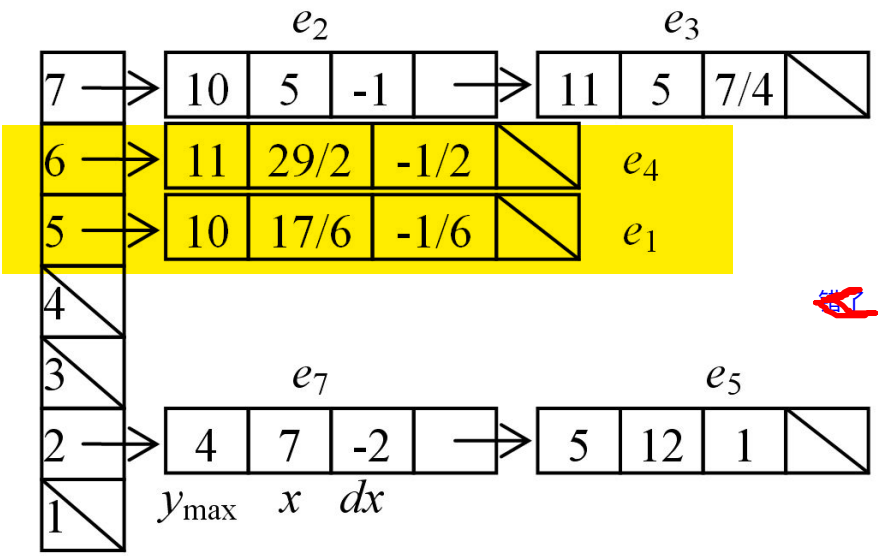

P_1 (2,10)
P_3 (12,11)
10
e_2
e_3
e_4
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15

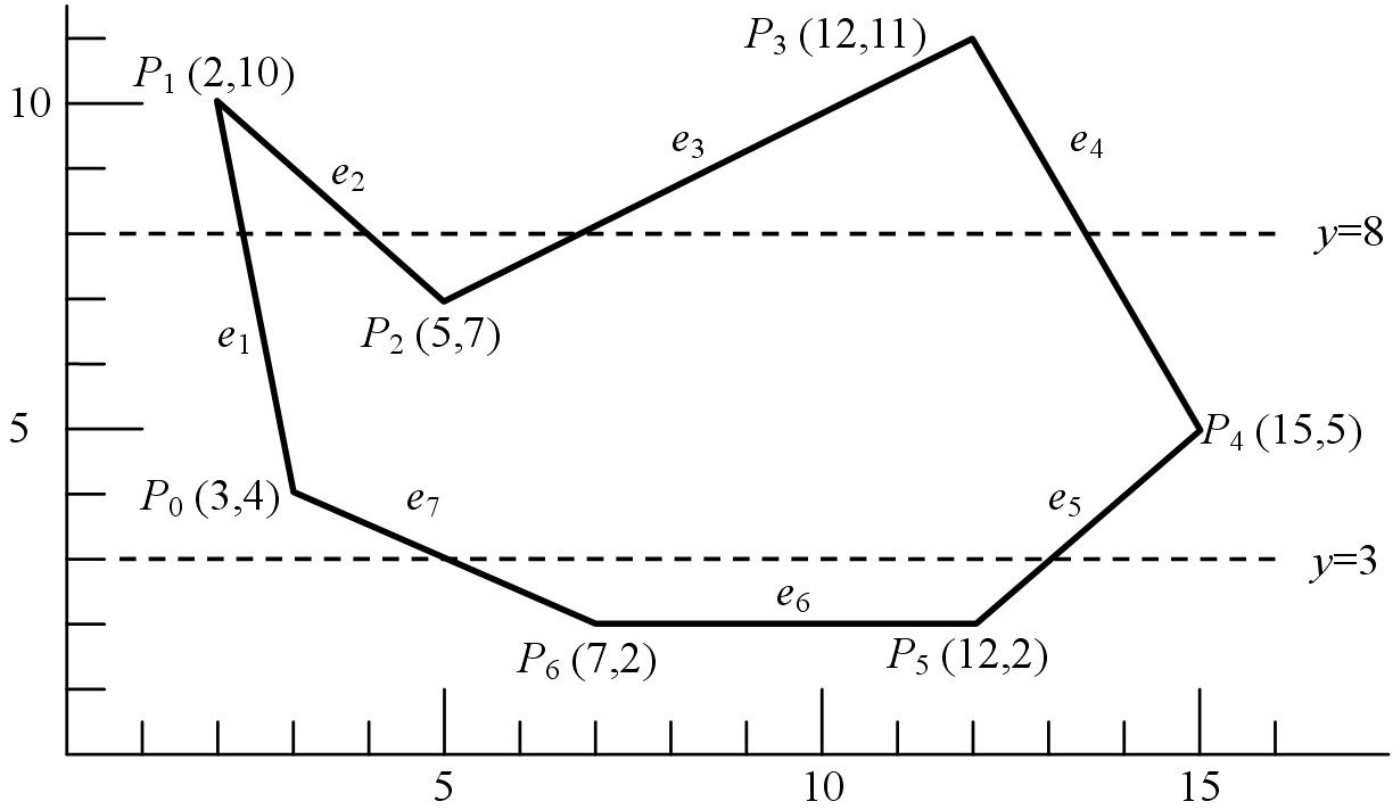
分类的边表 (ET)

NET

- 分类的边表是按边的下端点的纵坐标y对非水平边进行分类的指针数组
  - 下端点的纵坐标y值等于i的边，归入第i类
  - 同一类中，各边按x值(x值相等时，按dx的值)递增的顺序排成行
  - 水平边不加入分类边表中

分类的边表实例

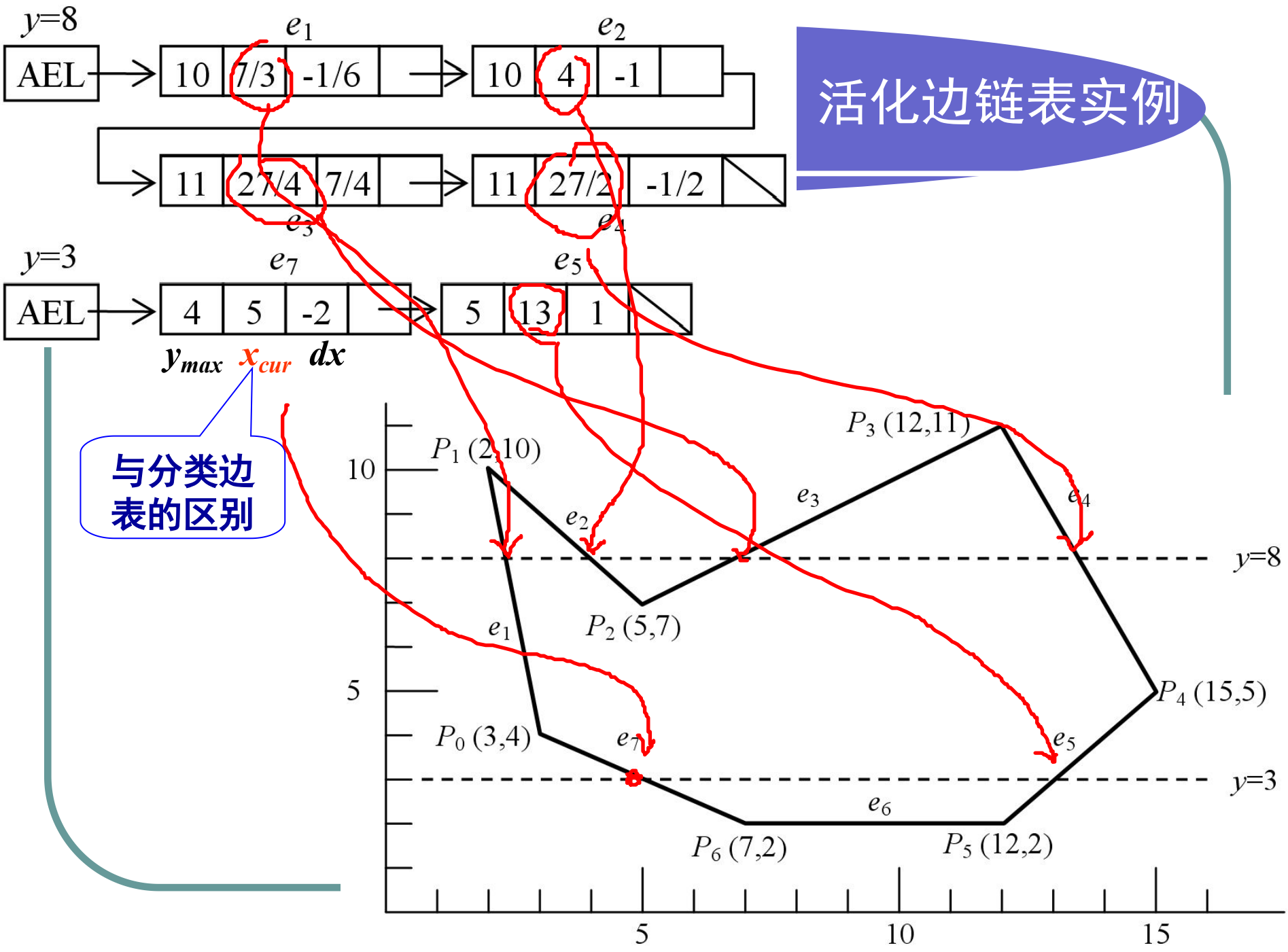

e_2
e_3
7
10
5
-1
11
5
7/4
6
11
29/2
-1/2
e_4
5
10
17/6
-1/6
e_1
4
3
e_7
e_5
2
4
7
-2
5
12
1
1
y_{max}
x
dx


P_1 (2,10)
P_3 (12,11)
10
e_2
e_3
e_4
y=8
P_2 (5,7)
e_1
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15

活化边链表(AEL)

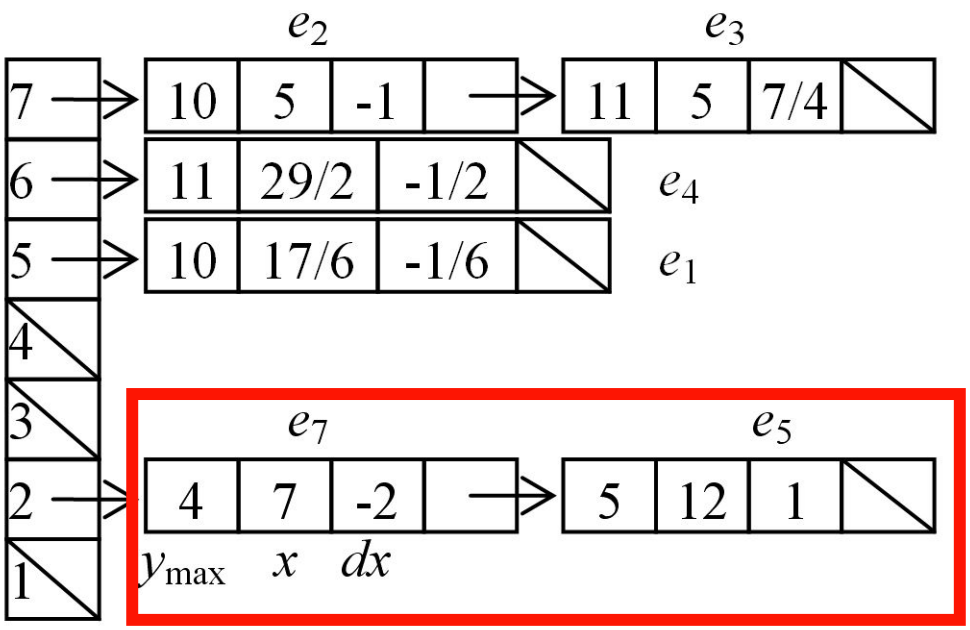
- 活化链表由与当前扫描线相交的边组成
  - 记录了多边形的边沿扫描线的交点序列
  - 根据边的连贯性不断刷新交点序列
- 基本单元是边(与扫描线相交的边)
- 与分类边表不同
  - 分类边表记录初始状态
  - 活化边表随扫描线的移动而动态更新

活化边链表实例


y=8
AEL
e1
e2
10
7/3
-1/6
10
4
-1
11
27/4
7/4
11
27/2
-1/2
e3
e4
y=3
AEL
e7
e5
4
5
-2
5
13
1
ymax
xcur
dx
与分类边表的区别
P1 (2,10)
P3 (12,11)
P2 (5,7)
P4 (15,5)
P0 (3,4)
P6 (7,2)
P5 (12,2)
e1
e2
e3
e4
e5
e6
e7
y=8
y=3
10
5
5
10
15

多边形扫描转换算法

1. (*y*初始化) 取扫描线纵坐标*y*的初始值为ET中非空元素的最小序号 (*y*=2)



多边形扫描转换算法

2. (AEL初始化) 将边的活化链表AEL设置为空
3. 按从下到上的顺序对纵坐标值为*y*的扫描线(当前扫描线)执行如下步骤，直到分类边表ET和边的活化链表AEL都变成空为止

多边形扫描转换算法

a) 如果分类边表ET中的第*y*类元素非空，则将属于该类的所有边从ET中取出并插入边的活化链表AEL中(同时将ET中相应的边表删除)，AEL中的各边按照*x*值(*x*值相等时，按*dx*值)递增方向排序；

b) 若对于当前扫描线，边的活化链表非空，则将AEL中的边交点两两依次配对。每一对边与当前扫描线的交点区间位于多边形内部，依次对这些区间上的像素按多边形属性着色；

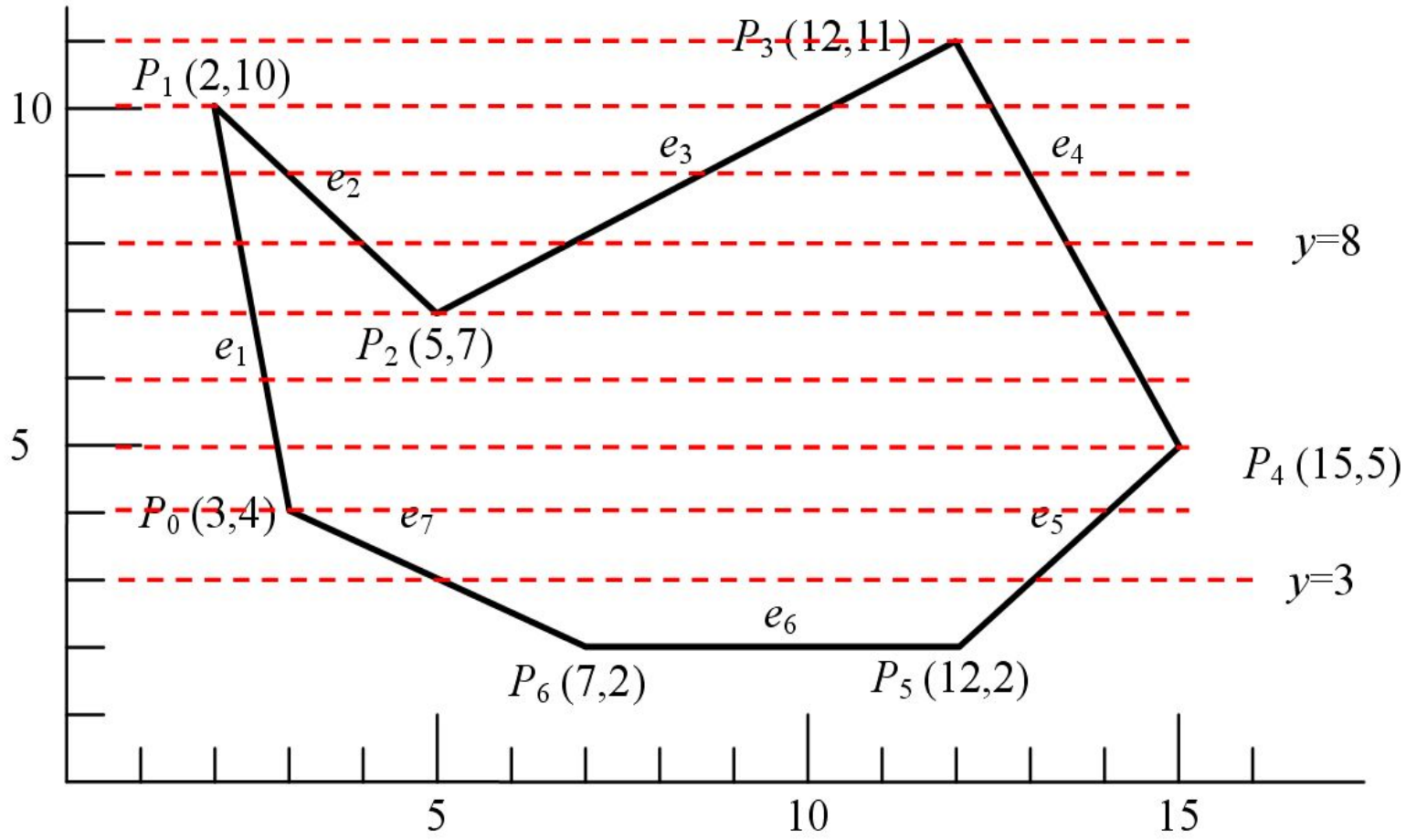
多边形扫描转换算法

c) 将边的活化链表AEL中满足$y=y_{max}$的边删除；

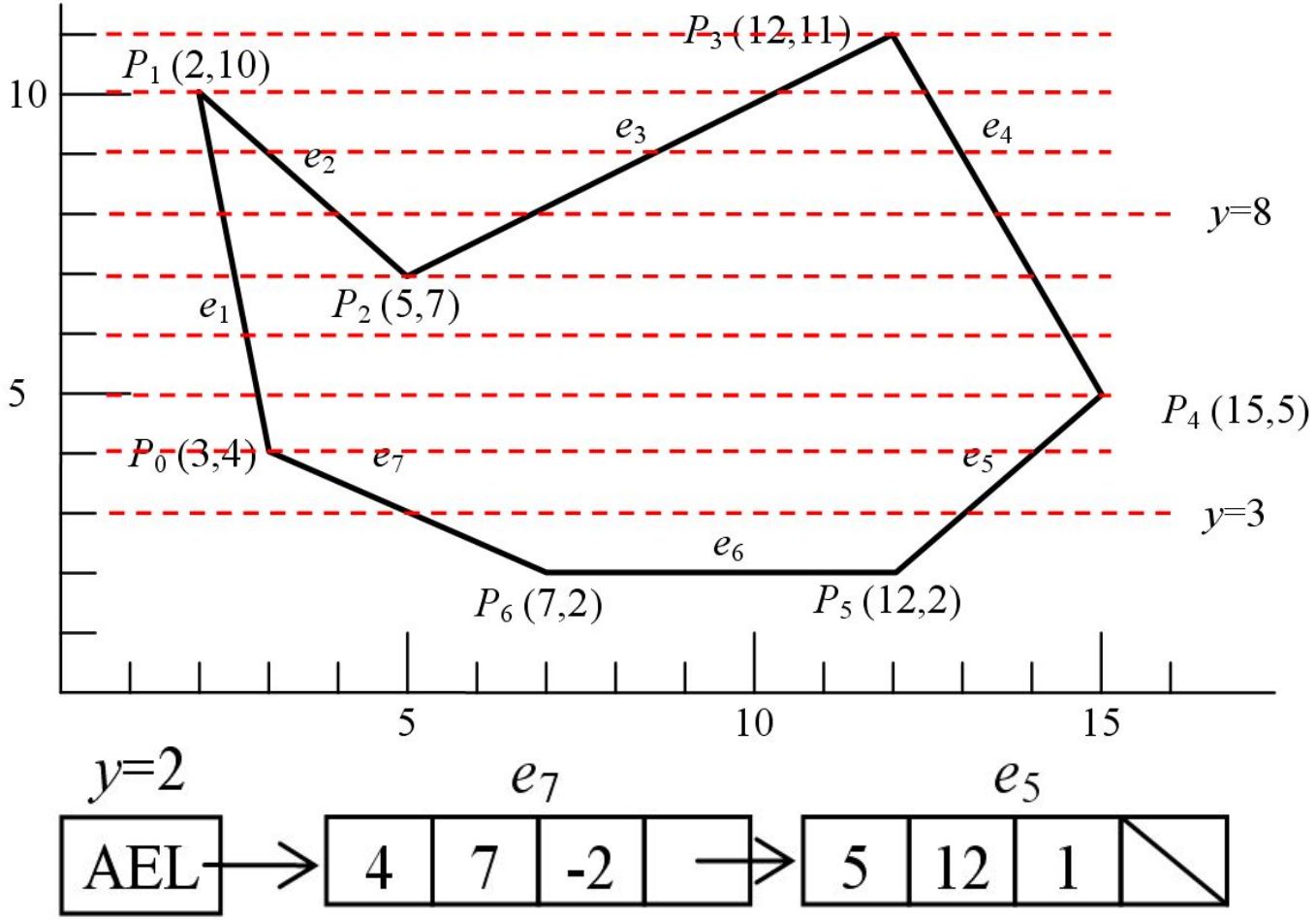
d) 将边的活化链表AEL中剩下的每一条边的x累加dx，即：$x=x+dx$;

e) 将当前扫描线的纵坐标值y累加，即$y=y+1$。

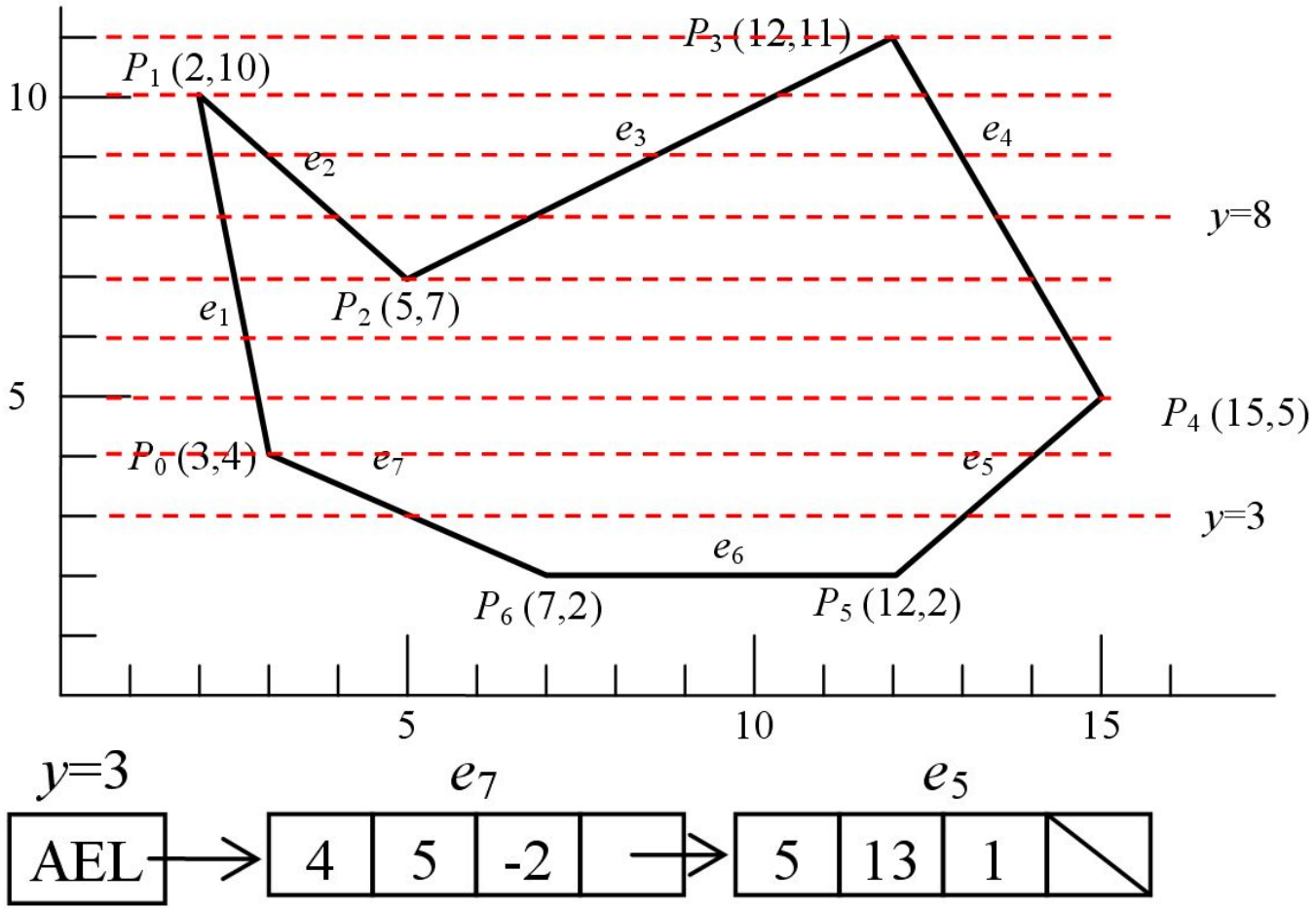
多边形扫描转换实例


P_3 (12,11)
P_1 (2,10)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15

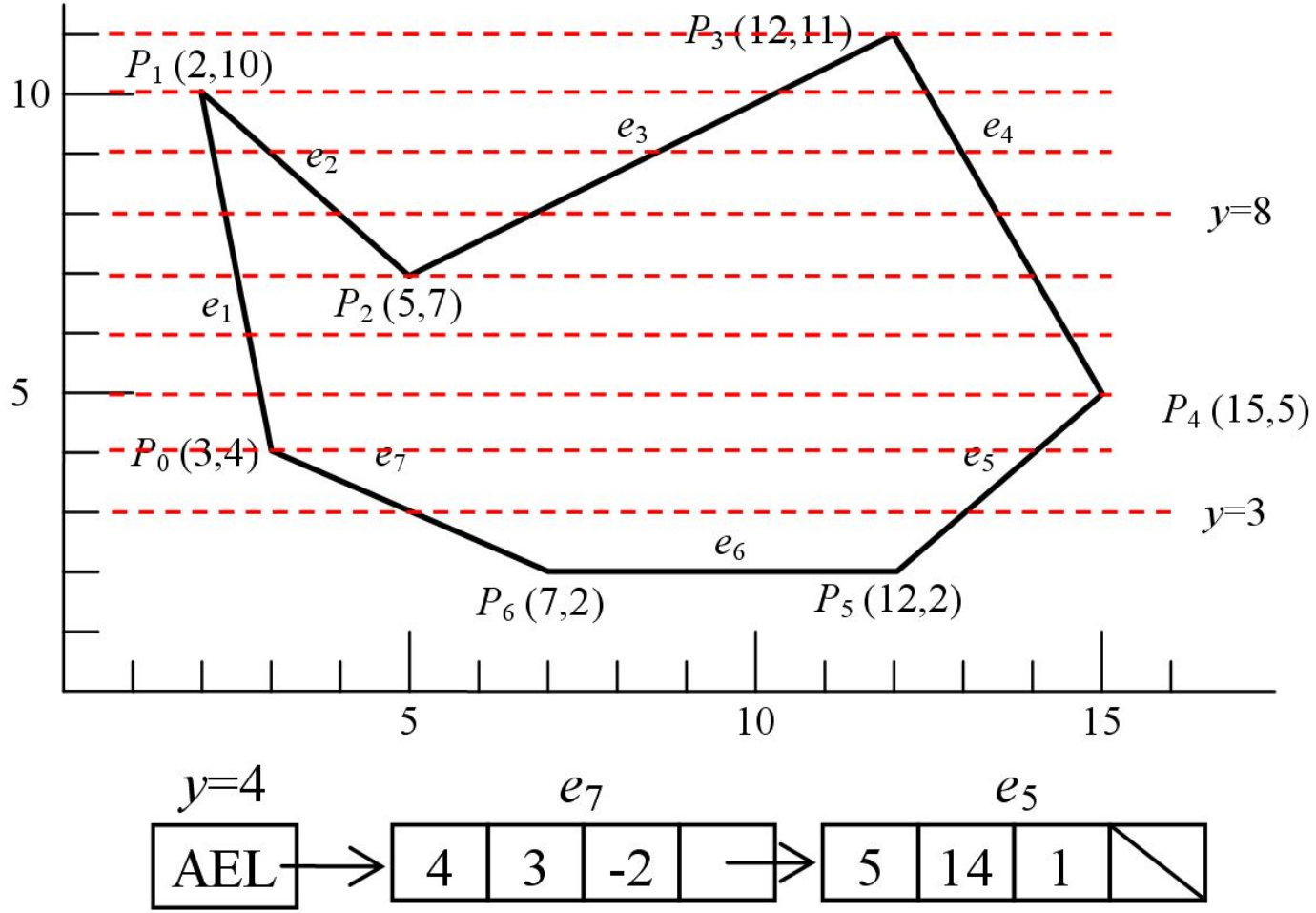
多边形扫描转换实例


P_3 (12,11)
P_1 (2,10)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15
y=2
e_7
e_5
AEL
4
7
-2
5
12
1

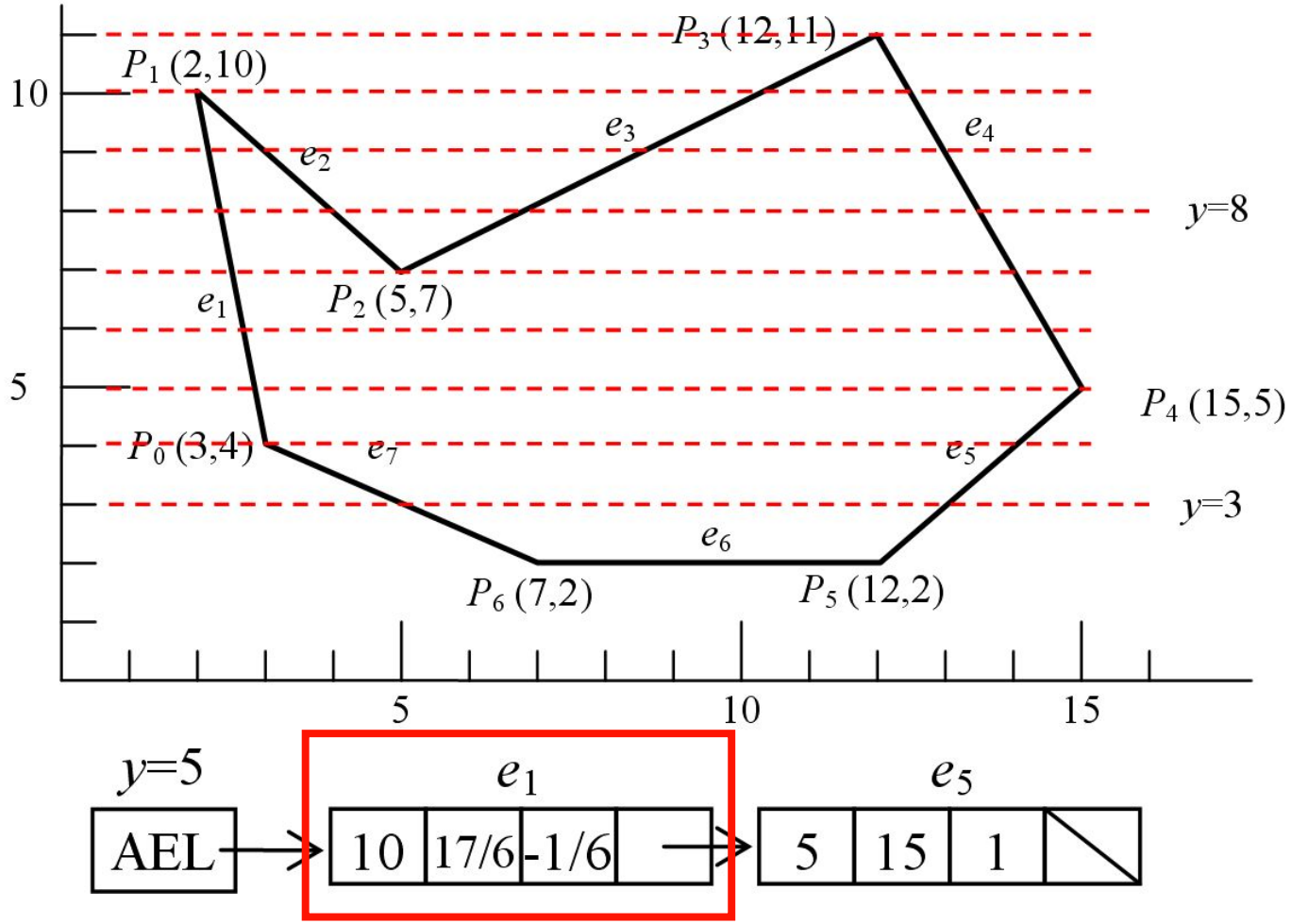
多边形扫描转换实例


P_3 (12,11)
P_1 (2,10)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15
y=3
e_7
e_5
AEL
4
5
-2
5
13
1

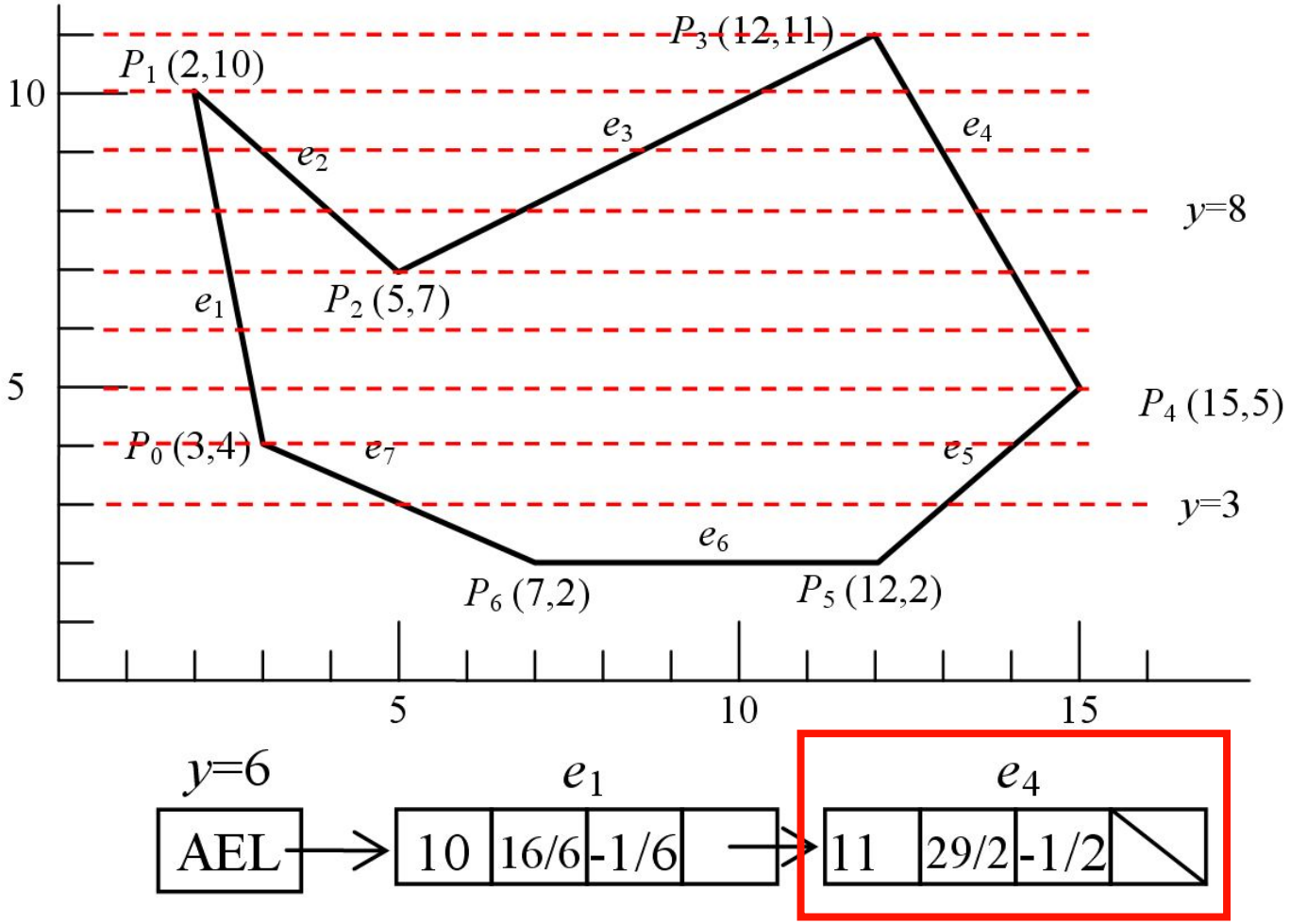
多边形扫描转换实例


P_1 (2,10)
P_3 (12,11)
10
e_2
e_3
e_4
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15
y=4
e_7
e_5
AEL
4
3
-2
5
14
1

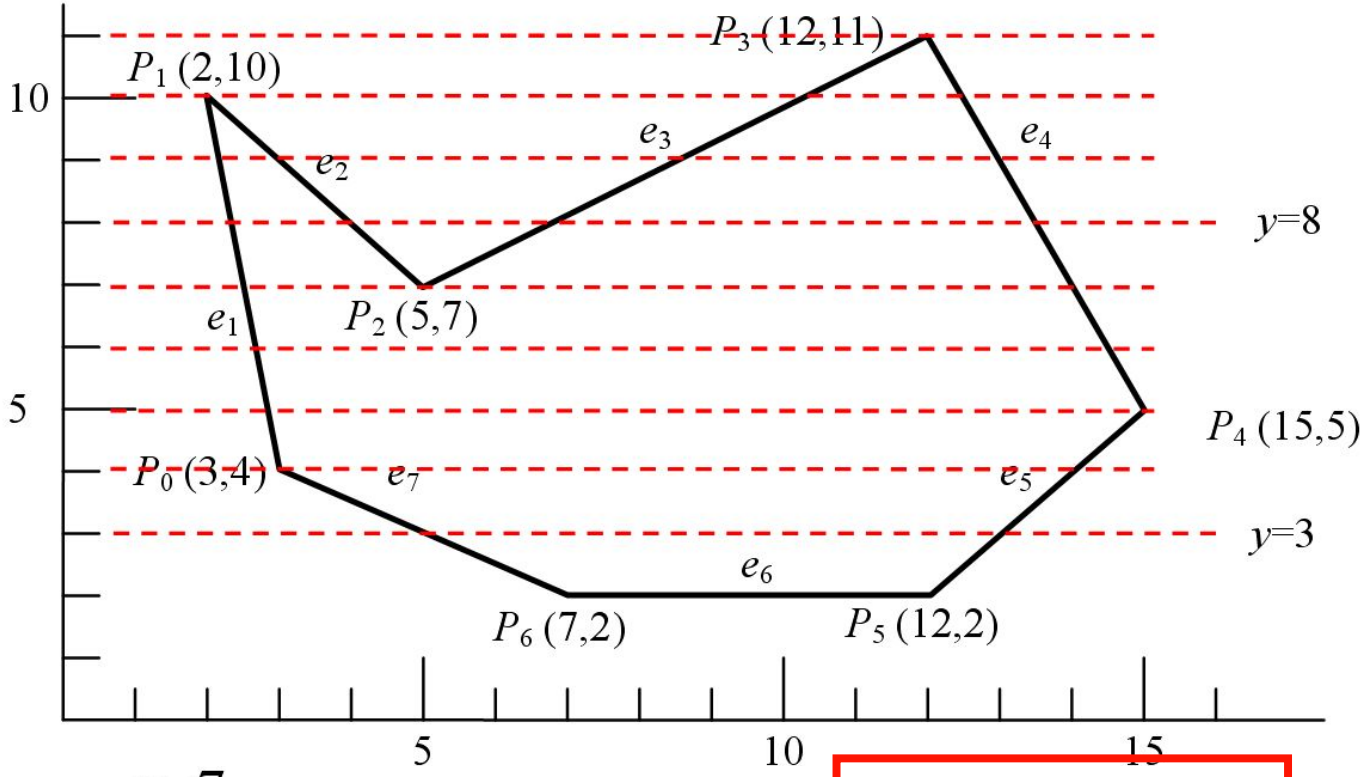
多边形扫描转换实例

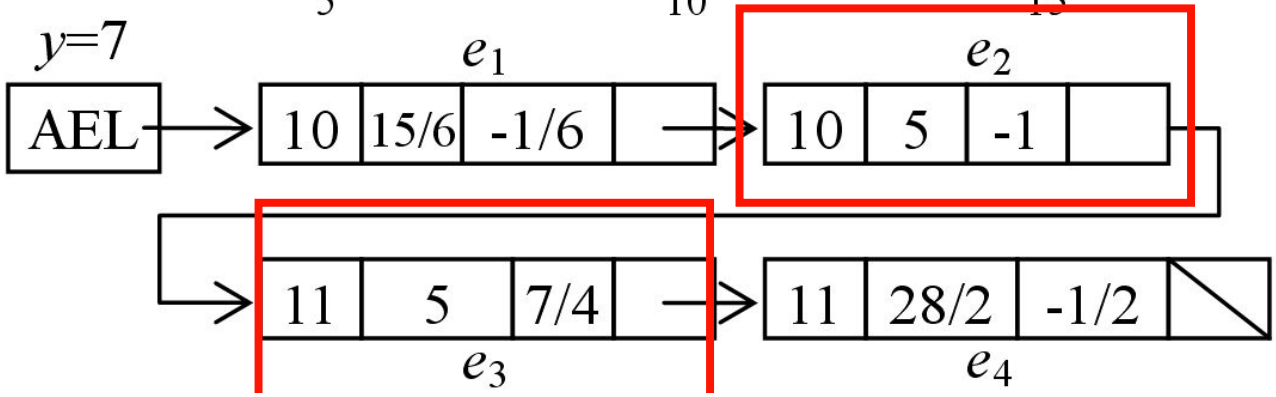

P_1 (2,10)
P_3 (12,11)
10
e_2
e_3
e_4
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15
y=5
e_1
e_5
AEL
10
17/6
-1/6
5
15
1

多边形扫描转换实例

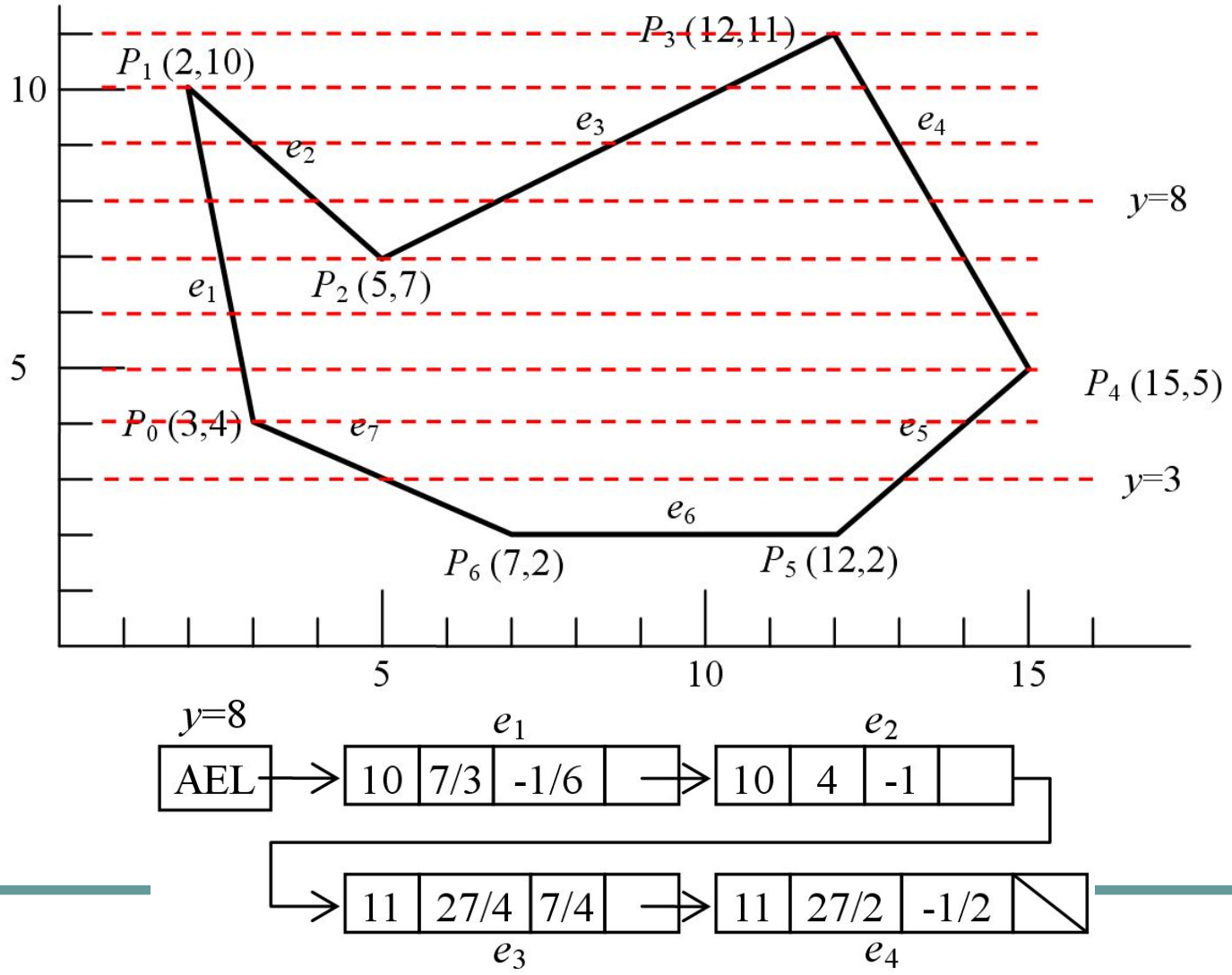

P_3 (12,11)
P_1 (2,10)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15
y=6
e_1
e_4
AEL
10
16/6
-1/6
11
29/2
-1/2

多边形扫描转换实例

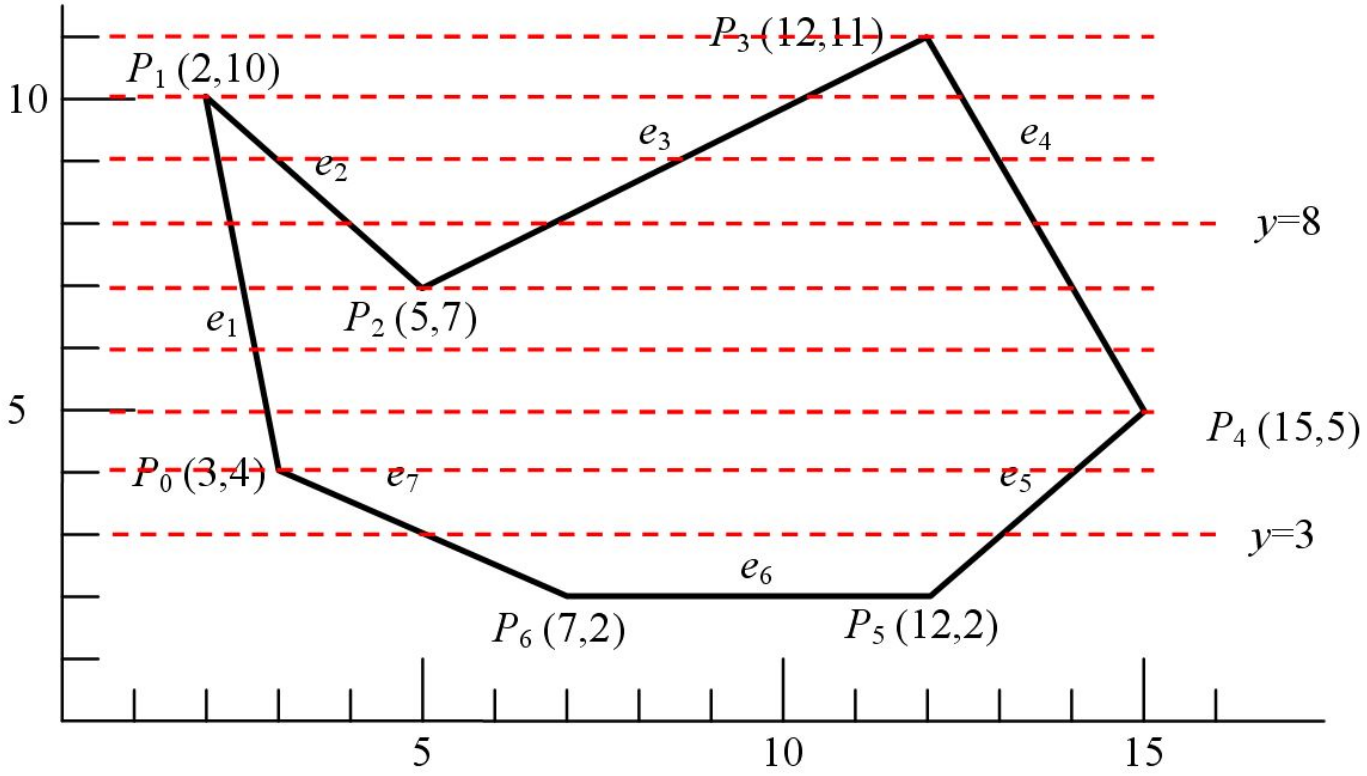

P_1 (2,10)
P_3 (12,11)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15

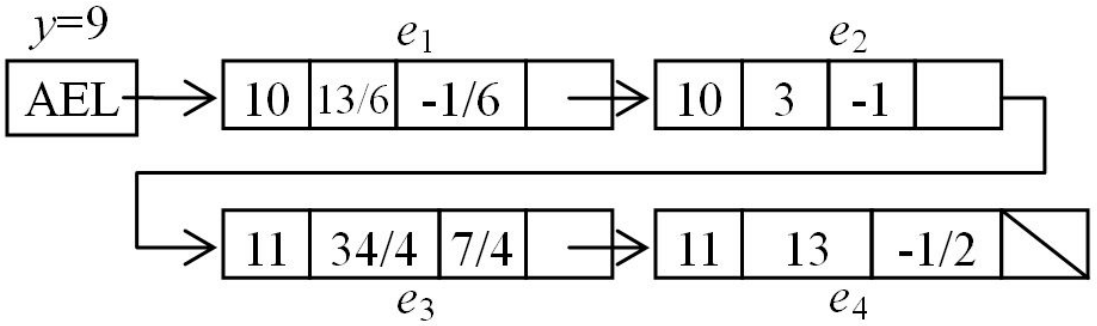

y=7
AEL
e_1
10
15/6
-1/6
e_2
10
5
-1
11
5
7/4
e_3
11
28/2
-1/2
e_4

多边形扫描转换实例

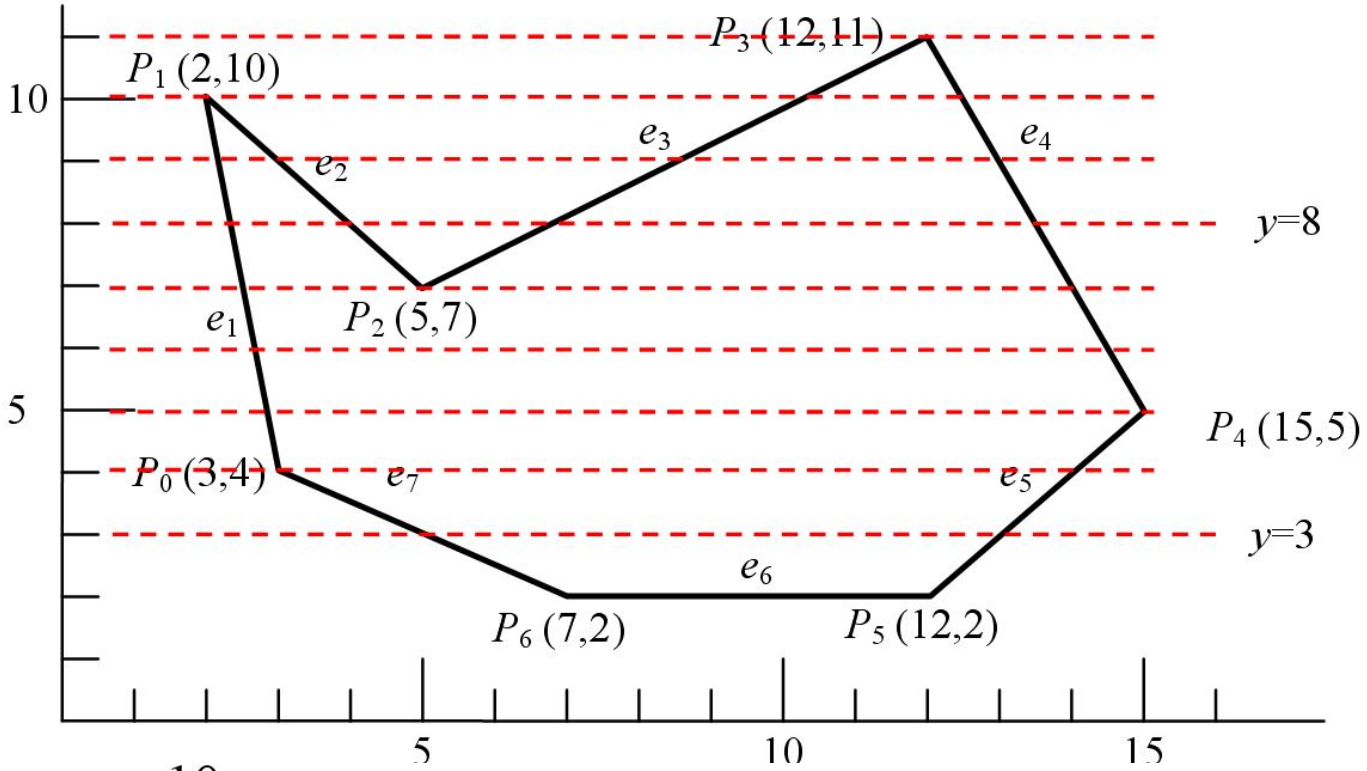

P_1 (2,10)
P_3 (12,11)
10
e_2
e_3
e_4
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15
y=8
e_1
e_2
AEL
10 7/3 -1/6
10 4 -1
11 27/4 7/4
11 27/2 -1/2
e_3
e_4

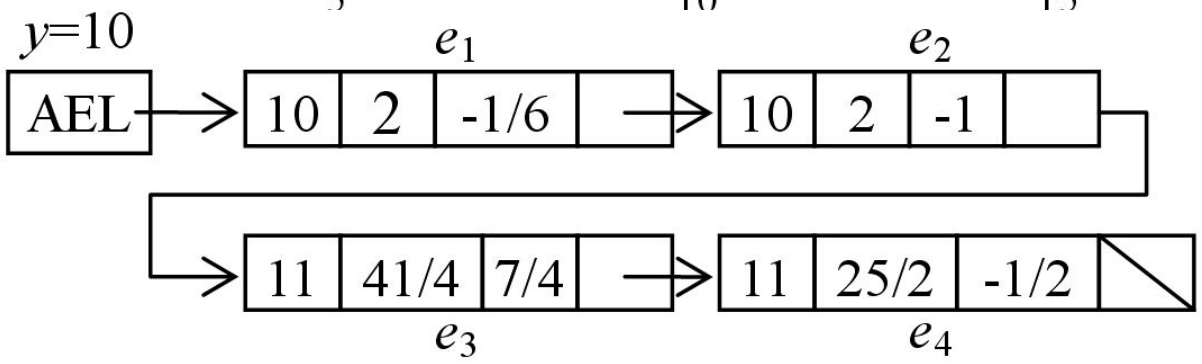
多边形扫描转换实例


P_3 (12,11)
P_1 (2,10)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15

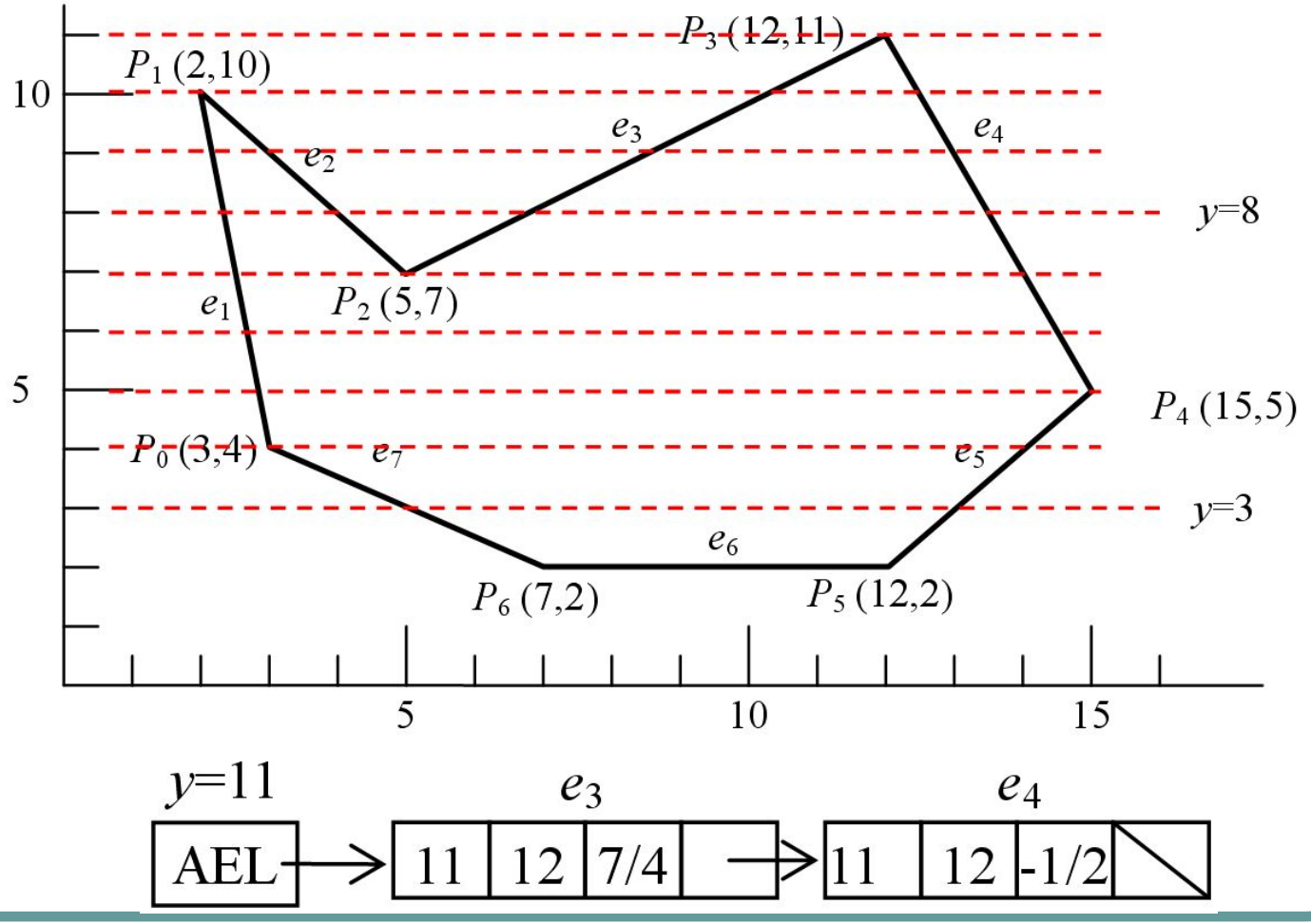

y=9
AEL
e_1
10
13/6
-1/6
e_2
10
3
-1
11
34/4
7/4
e_3
11
13
-1/2
e_4

多边形扫描转换实例


P_1 (2,10)
P_3 (12,11)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15


y=10
e_1
e_2
AEL
10
2
-1/6
10
2
-1
11
41/4
7/4
11
25/2
-1/2
e_3
e_4

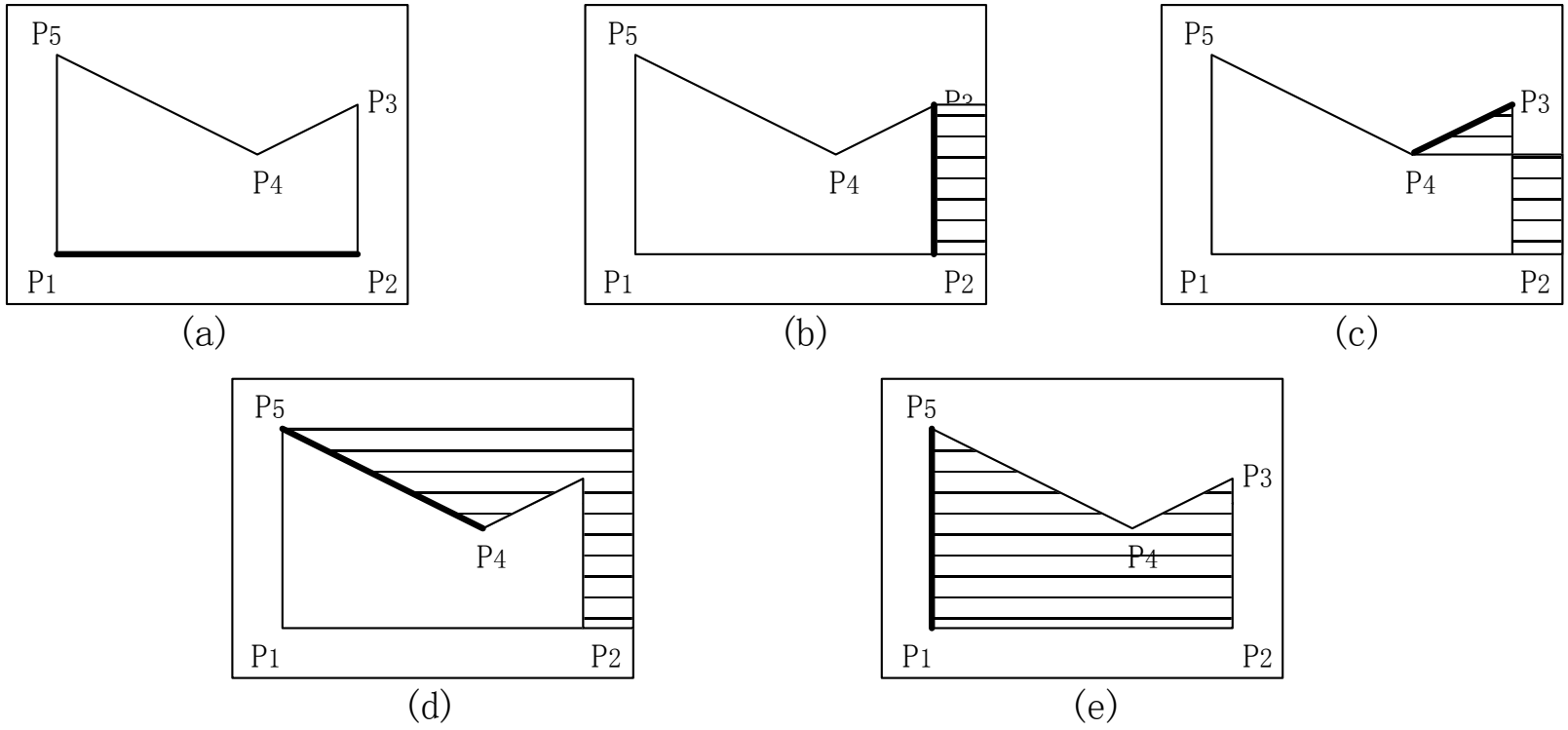
多边形扫描转换实例


P_1 (2,10)
P_2 (5,7)
P_3 (12,11)
P_4 (15,5)
P_5 (12,2)
P_6 (7,2)
P_0 (3,4)
e_1
e_2
e_3
e_4
e_5
e_6
e_7
y=8
y=3
10
5
5
10
15
y=11
e_3
e_4
AEL
11
12
7/4
11
12
-1/2

多边形扫描转换

- 优点
  - 充分利用多边形的区域、扫描线和边的连贯性，避免了反复求交的大量运算
- 不足
  - 算法的数据结构和程序结构复杂
  - 对各种表的维持和排序开销太大，适合软件实现而不适合硬件实现

边填充算法(Edge Fill Algorithm)

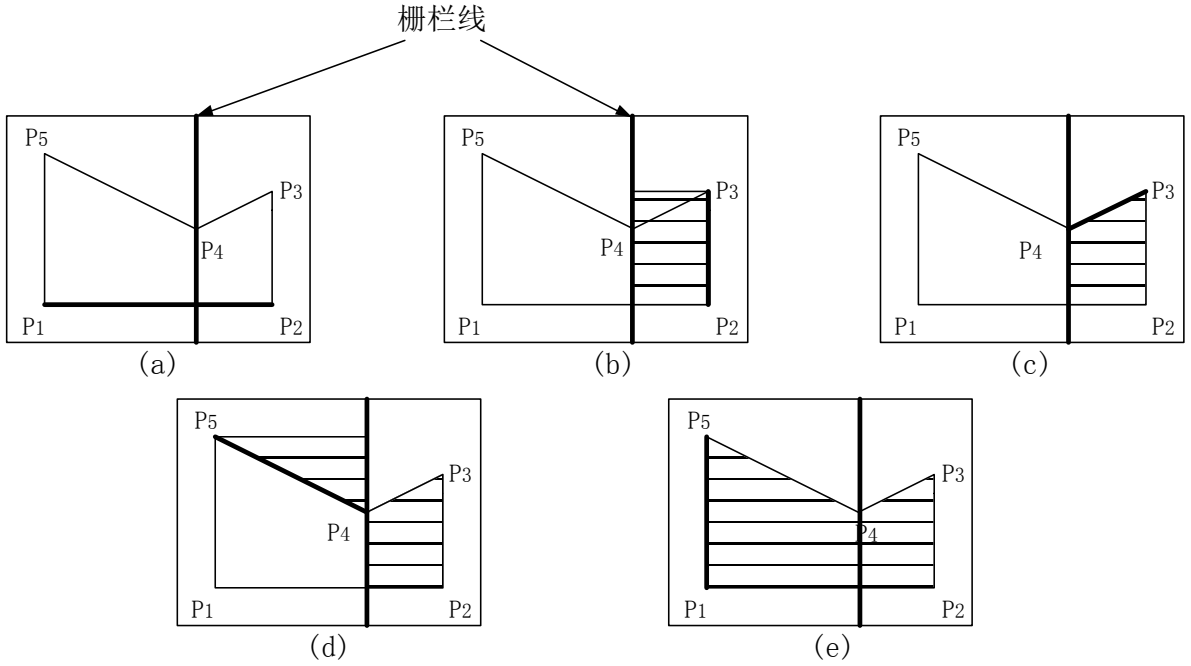


- 和填充顺序、填充方向有关吗？
- 该算法的优缺点？

边填充算法(Edge Fill Algorithm)

- **优点：**
  - 最适合于有帧缓存的显示器
  - 可按任意顺序处理多边形的边
  - 仅访问与该边有交点的扫描线上右方的像素，算法简单
- **缺点：**
  - 对复杂图形，每一像素可能被访问多次，输入/输出量大
  - 图形输出不能与扫描同步进行，只有全部画完才能打印

栅栏填充算法(Fence Fill algorithm)



- 引入栅栏的目的？

多边形的扫描转换与区域填充的比较

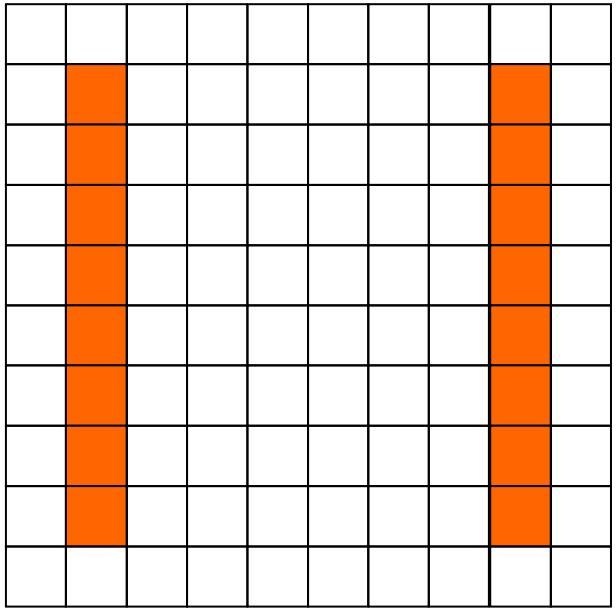
多边形扫描转换与区域填充比较

- 基本思想不同
  - 多边形扫描转换将多边形顶点表示转换为点阵表示，扫描过程利用了多边形的各种连贯性
  - 区域填充只改变区域的颜色，不改变区域的表示方法。填充过程利用了区域的连贯性

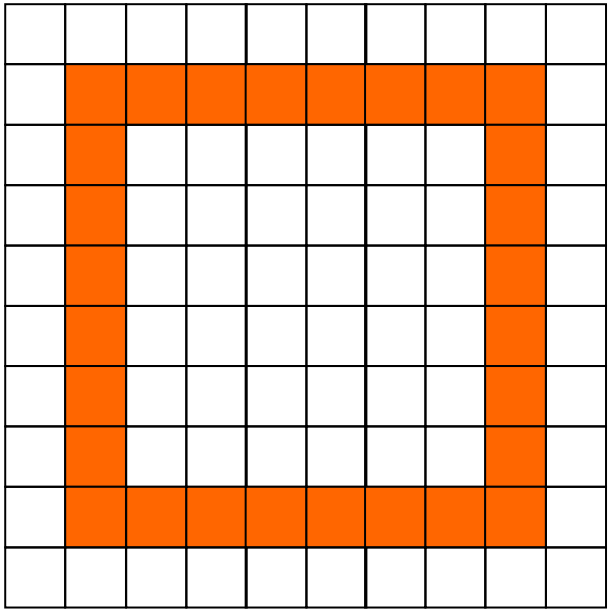
多边形扫描转换与区域填充比较

- 对边界的要求不同
  - 多边形扫描转换只要求每一条扫描线与多边形有偶数个交点
  - 区域填充中
    - 四连通区域必须是封闭的八连通边界
    - 八连通区域必须是封闭的四连通边界

多边形扫描转换与区域填充比较



多边形扫描转换允许边界



区域填充允许边界

多边形扫描转换与区域填充比较

- 出发点不同
  - 区域填充：知道需要区域内一个种子点(复杂计算)
  - 多边形扫描转换：没有要求

扫描线种子填充算法 ???

- 利用扫描线的连贯性
- 减少递归次数

扫描线种子填充算法

- 种子像素入栈
- 当栈非空时，重复以下步骤：
  - 栈顶像素出栈
  - 沿扫描线对出栈像素的左右像素进行填充，直到遇到边界像素为止
  - 将上述区间内最左、最右像素记为xl和xr
  - 在区间$[x_l,x_r]$中检查与当前扫描线相邻的上下两条扫描线是否全为边界像素、或已填充的像素，若为非边界、未填充的像素，则把每一区间的最右像素取为种子像素入栈

走样与反走样

走样(antialiasing)

图形的锯齿状：图形信号连续，光栅显示系统中，离散表示。

用离散量(像素)表示连续的量(图形)而引起的失真，叫混淆或叫走样(aliasing)

光栅图形走样：

- 阶梯状边界；
- 图形细节失真；
- 狭小图形遗失：动画序列中时隐时现，产生闪烁。

图形反走样技术（antialiasing）

1.从硬件角度提高分辨率

- 高分辨率显示器

- 显示器点距减少一倍
- 帧缓存容量增加到原来的4倍
- 输带宽提高4倍
- 扫描转换花4倍时间
- 代价高

图形反走样技术（antialiasing）

2.从软件角度替高分辨率

- 高分辨率计算，低分辨率显示
- 像素细分技术，相当于后置滤波

1	1
1	1

1	2	1
2	4	2
1	2	1

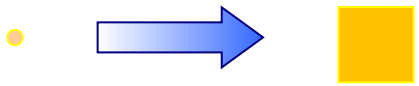
- 只能减轻，不能消除

图形反走样技术（antialiasing）

3.区域采样技术

- 改变边或直线的外观，模糊淡化阶梯
- 相当于图像的前置滤波

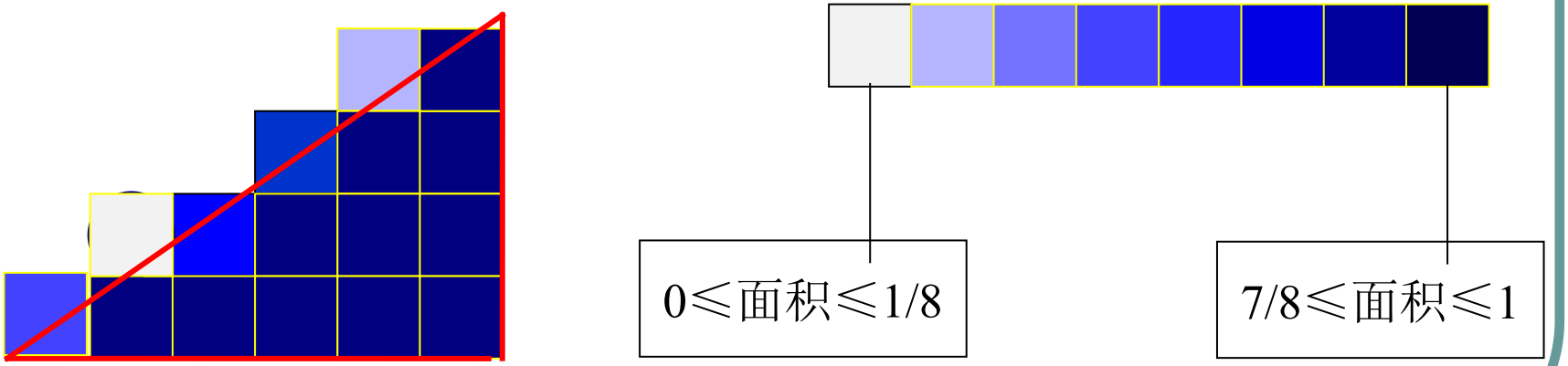
直线有宽度



有限
区域

图形反走样技术（antialiasing）

根据相交的面积值决定像素显示的亮度级别



总结

- 基本概念
- 区域填充
  - 四连通区域和八连通区域
  - 连通区域的种子填充算法
- 多边形的扫描转换
  - 逐点判断算法
  - 扫描线算法
    - 区域、扫描线、边的连贯性
    - 奇异点的处理
    - 数据结构与算法实现
- 多边形的扫描转换与区域填充的比较
- 走样与反走样

上机作业（1）

- Bresenham直线段扫描转换
  - 窗口中用鼠标输入若干点
  - 用Bresenham算法依次画出直线段，形成多边形
- 扫描线填充算法
  - 在该多边形中填充颜色
- 要求：使用指定算法，效率较高，界面库不限
- 2-3次上机实践（第9周开始），当面展示实现结果

谢谢！