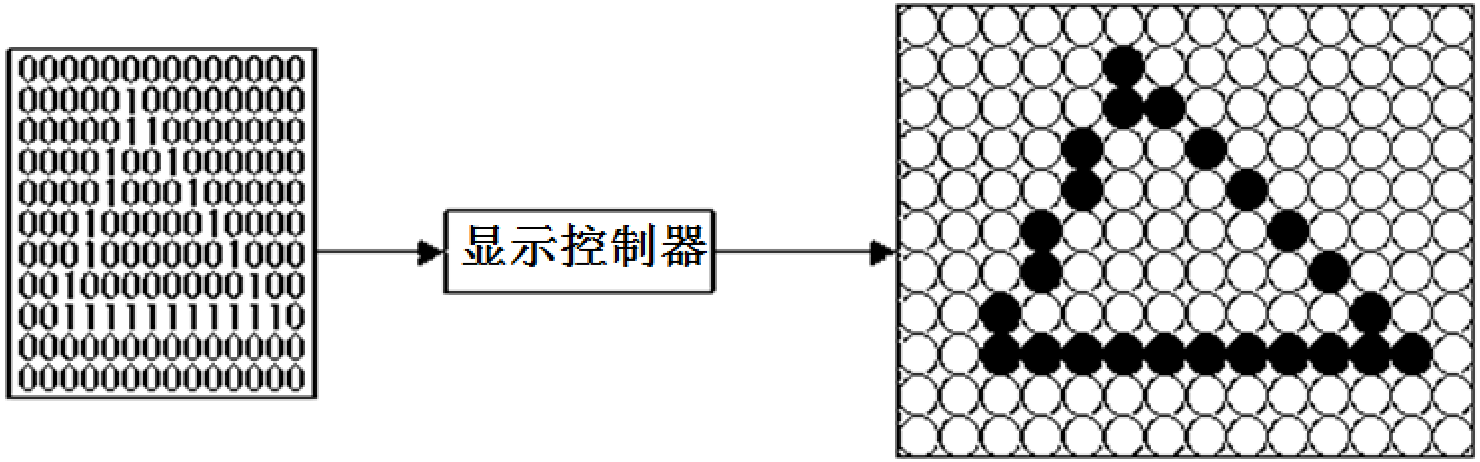
光栅扫描式显示器-帧缓存3

- 光栅内容来自帧缓存
- 分辨率：像素数量，如：1024*768
- 分辨率=帧缓存单元数量
- 像素颜色数量K取决于帧缓存位数n
- K与n关系？
- $K=2^n$



帧缓冲器 显示屏幕

光栅扫描式显示器-帧缓存

- **分辨率M*N、颜色数K与帧缓存大小V的关系**

$$V \geq M \times N \times \lceil \log_2 K \rceil$$

- 分辨率是1024×1024的显示器若要显示8种颜色，需要帧缓存V为多少KB?
- 注意单位：KB
- V=1024*1024*$\log_2 8$ (Bit)
- V=1024*1024*$\log_2 8$/8/1024 (KByte)=375K

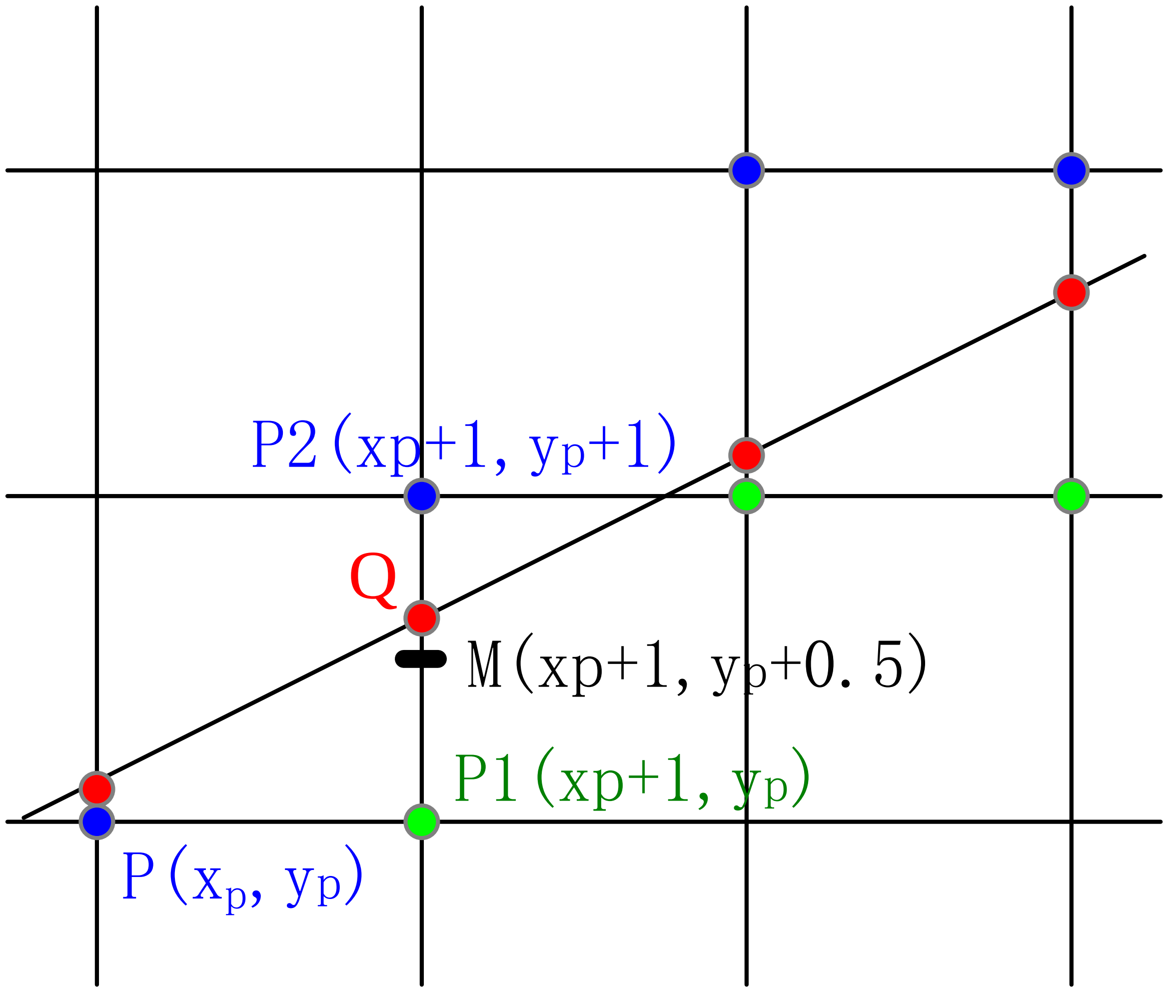
问题

- 问题1：每个像素用24位表示、分辨率为1024*1024的显示器，至少需要的帧缓存容量为（ C ）。

  A. 1M　　B. 2M C. 3M D. 8M

中点画线法

- 如果当前我们选择**P**，则下一像素必然选择**P1**或 **P2**



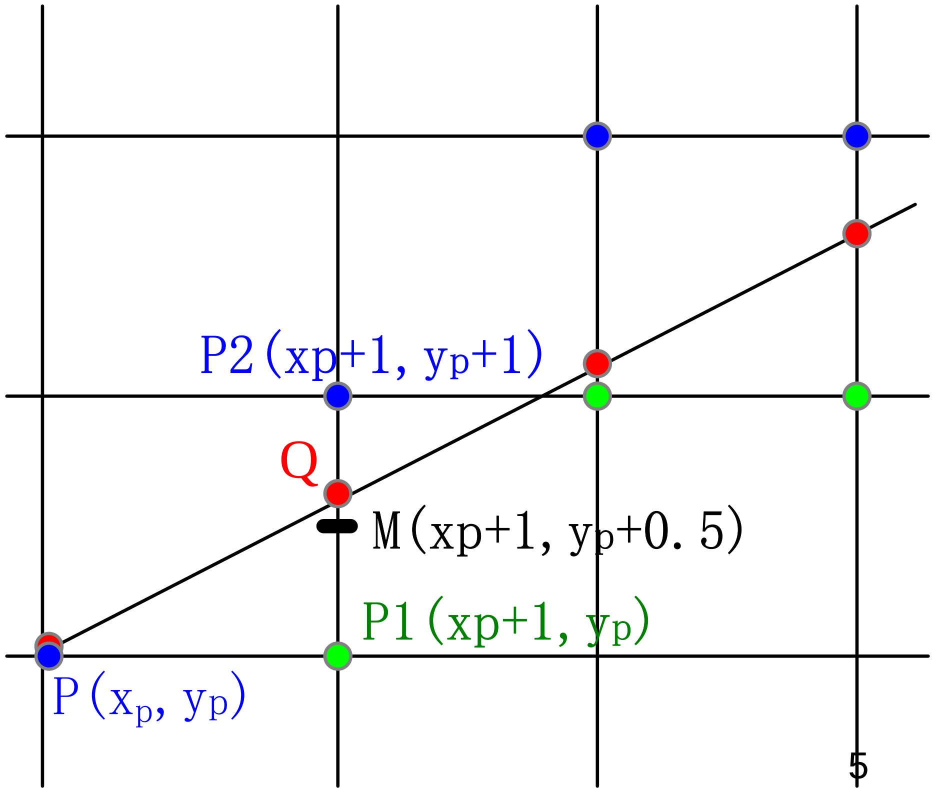
中点画线法

- 怎样选择: **P1** 或 **P2**

  中点画线法基本原理:

  - 如果线段从中点**M**的下方通过，则选择 **P1**
  - 如果线段从中点**M**的上方通过，则选择 **P2**

- 怎样表示 “线段从中点**M**的下方或上方通过”?



中点画线法---构造判别式（ 0≤k≤1 ）

- 直线方程:

  $F(x,y) = ax+by+c=0$

  其中：

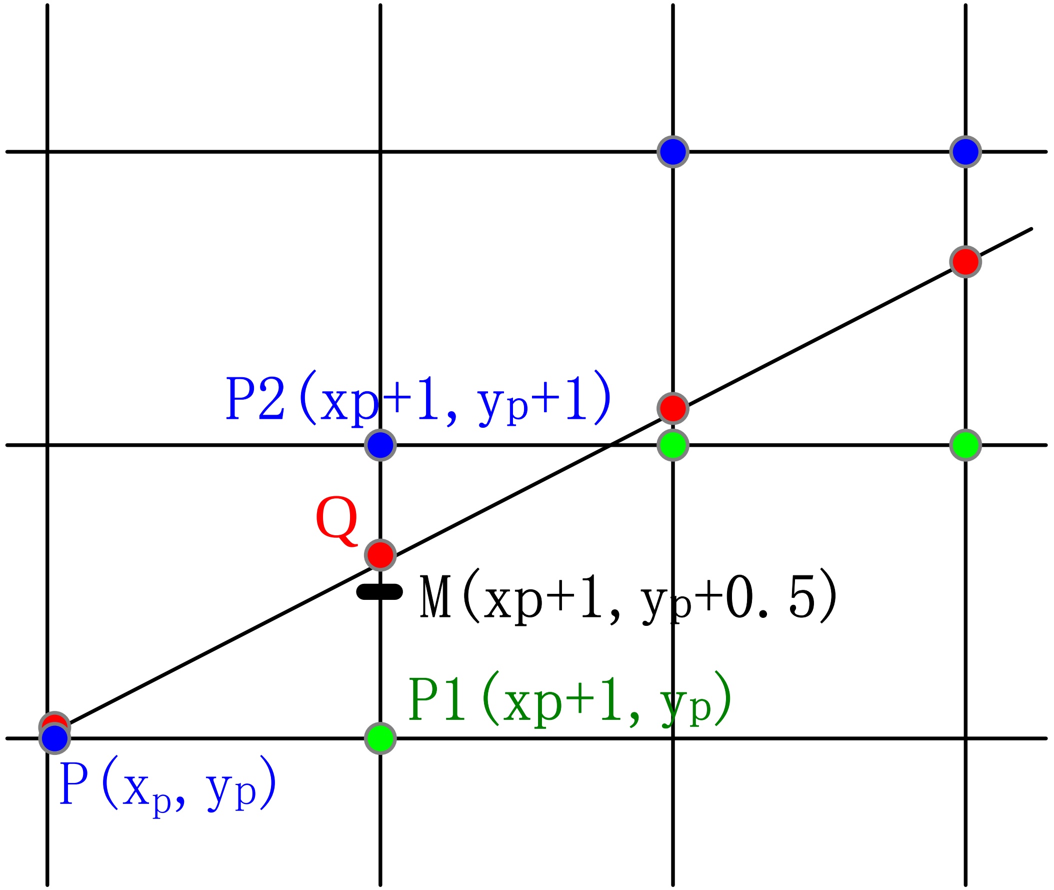
  $a = y_0 - y_1;$

  $b = x_1 - x_0;$

  $c = x_0y_1 - x_1y_0;$

  点在线上方:$F(x, y)>0;$

  点在线下方:$F(x, y)<0;$



- 中点M:

$$d = F(M) = F(x_p+1, y_p+0.5) = a(x_p+1) + b(y_p+0.5) + c$$

中点画线法---构造判别式（ $0 \le k \le 1$ ）

●判别式：$\boldsymbol{d} = F(M) = F(x_p+1, y_p+0.5) = a(x_p+1) + b(y_p+0.5) + c$

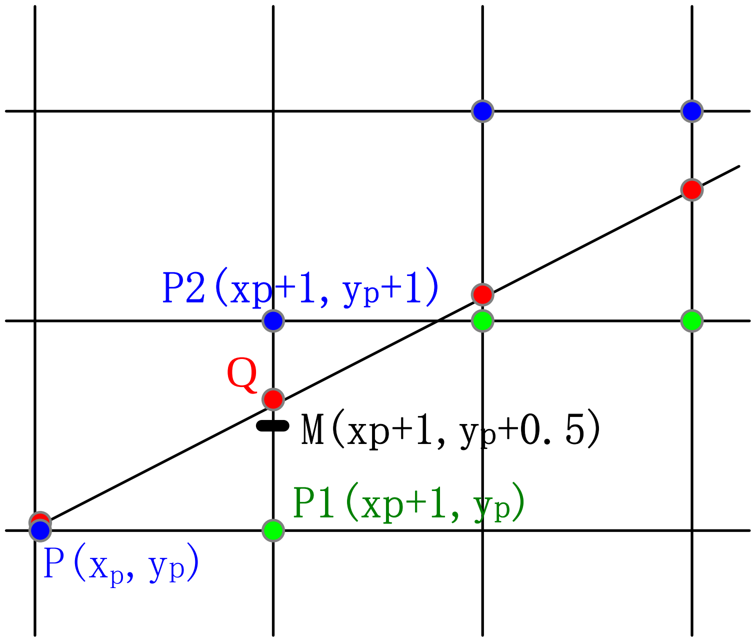
下一步?

当d<0，M在直线(Q点)下方，取右上方P_2；

当d>0，M在直线(Q点)上方，取右方P_1；

当d=0，选P_1或P_2均可，约定取P_1；

能否采用增量算法呢?

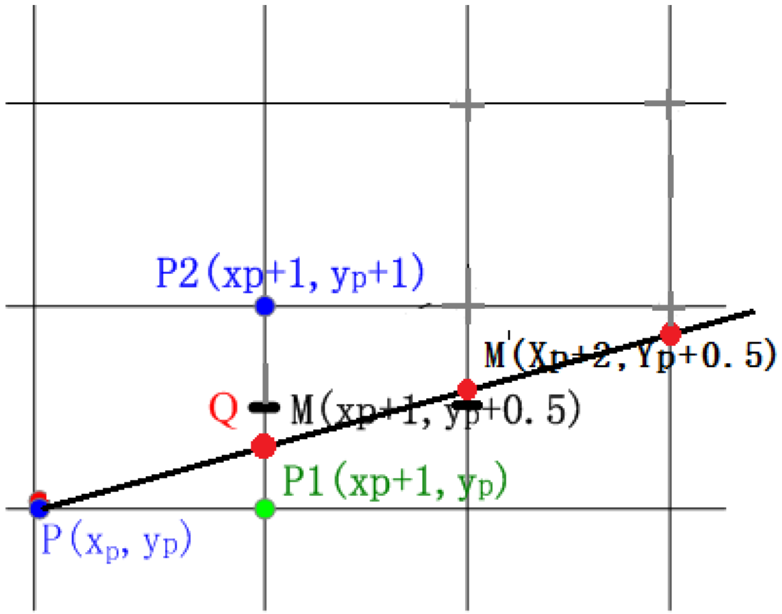


中点画线法---判别式的增量算法

- 若$d \geq 0$ ---->M在直线上方---->取**P1**;
- 此时再下一个像素的判别式为

$$d_1 = F(x_p+2, y_p+0.5)$$
$$= a(x_p+2)+b(y_p+0.5)+c$$
$$= a(x_p+1)+b(y_p+0.5)+c+a$$
$$= d+a;$$

增量为a



中点画线法---判别式的增量算法

- 若$d<0$ ----> M在直线下方---->取**P2**;
- 此时再下一个象素的判别式为

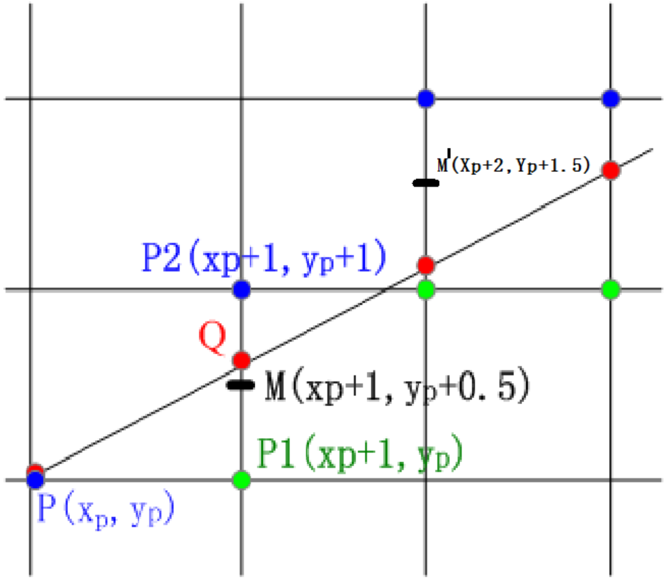
d_2= F(x_p+2, y_p+1.5)

=$a(x_p+2)+b(y_p+1.5)+c$

= $a(x_p+1)+b(y_p+0.5)+c+a+b$

=$d+a+b$；

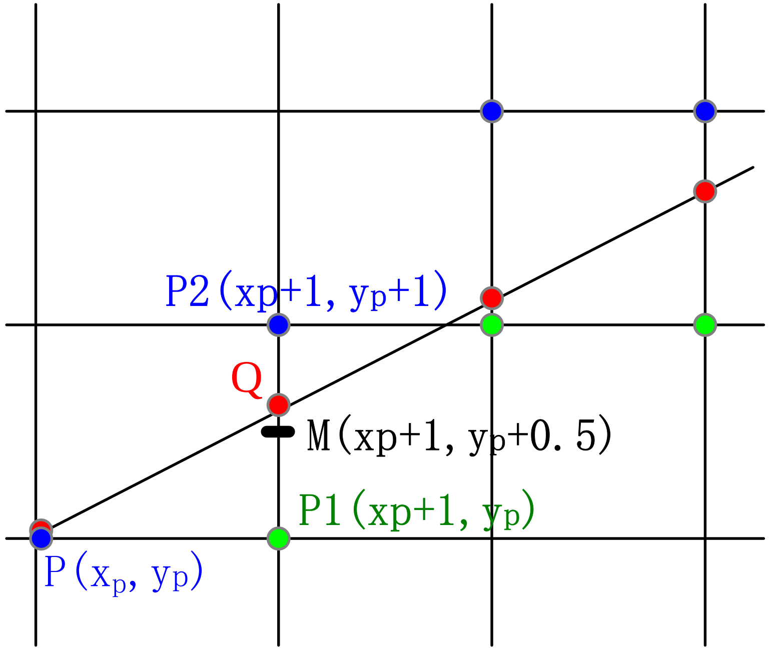
增量为$a+b$



中点画线法---判别式的初始值

- **判别式d的初始值**

  画线从(x_0, y_0)开始，d的初值



$$d_0=F(x_0+1, y_0+0.5)$$
$$= a(x_0 +1)+b(y_0 +0.5)+c$$
$$= F(x_0, y_0)+a+0.5b = \boldsymbol{a+0.5b}$$

由于只用*d* 的符号判断，为了只包含整数运算，可以用**2*d***代替***d***来摆脱小数，提高效率。

中点画线法

```
void MidPLine(int x0,int y0,int x1,int y1){
   a=y0-y1, b=x1-x0, d=2*a+b;
   x=x0, y=y0;
   setpixel(x, y, color);
   while (x<=x1)
   {  if (d<0)       {x++; y++; d=d+2*a+2*b; }
      else       {x++; d =d +2 *a;}
      setpixel (x, y, color);
    }
}
```

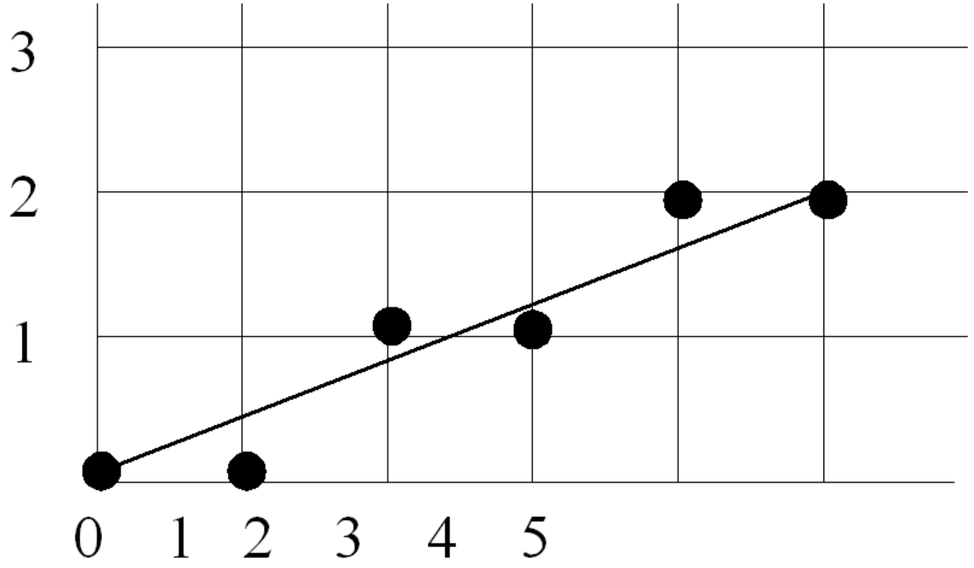
中点画线法

例：用中点画线法P0(0,0) P1(5,2)

a=y0-y1= -2 b=x1-x0=5

d0=2a+b=1   2a= -4   2(a+b)=6

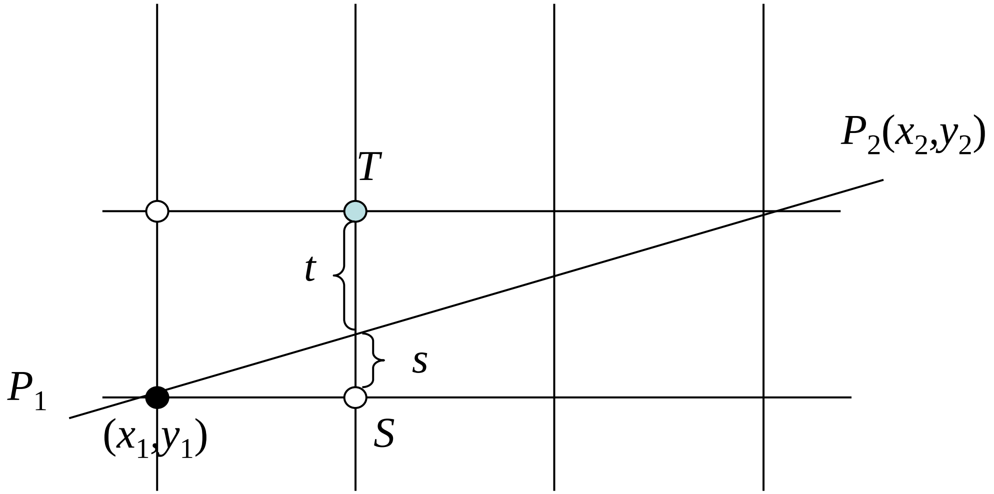
i	xi	yi	d
1	0	0	1
2	1	0	-3
3	2	1	3
4	3	1	-1
5	4	2	5

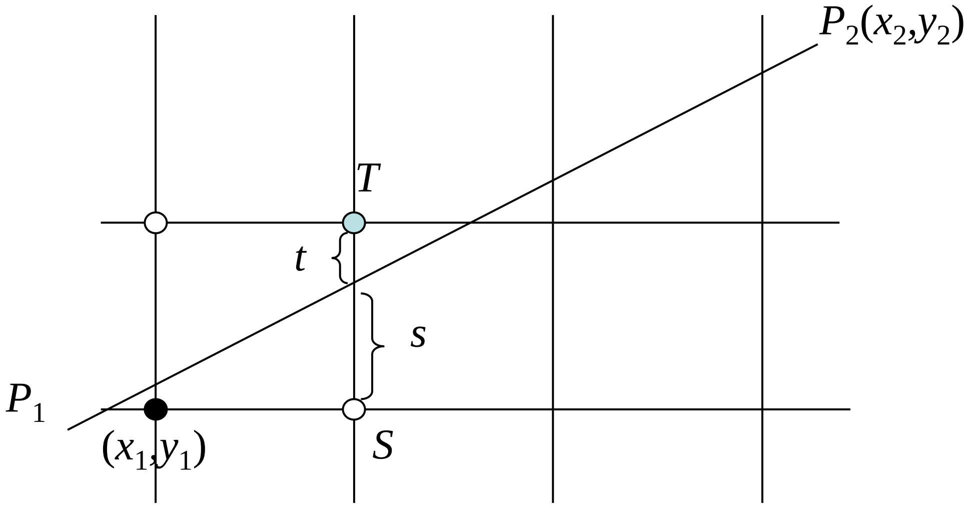


Bresenham画线算法

- 假定直线段的斜率：0≤k≤1
- **Bresenham画线算法基本原理**：

　首先从像素$P_1(x_1, y_1)$开始，每次在水平方向朝$P_2(x_2, y_2)$移动一个像素，由于受斜率k的限制，当s<t 时,下一个像素点选右边的点S，当s>t时选右上的点T。





Bresenham画线算法---判别式构造

- 假设所绘制的直线方程为：$y=mx+b$
- 当$x=x_i+1$时，直线对应的y坐标是：

  $y=mx_{i+1}+b=m(x_i+1)+b$
- 点S到直线的距离为：$s=y - y_i$；
- 点T到直线的距离为：$t=y_i+1-y$；
- 通过考虑两个距离之差即***s-t***的值来确定应选哪个点：
- 当**s<t**时，直线离S点近，选S;
- 当**s>=t**时，直线离T点近，选T点；
- 而**s-t**$=(y-y_i)-[(y_i+1)-y]=2y-2y_i-1=2m(x_i+1)+2b-2y_i-1$

Bresenham画线算法---迭代公式

令$\Delta x=x_2-x_1$，$\Delta y=y_2-y_1$，则$m=\Delta y/\Delta x$， 用$\Delta y/\Delta x$代替m得

$$s-t=2\frac{\Delta y}{\Delta x}(x_i+1)+2b+2y_i-1$$

两边同乘以Δx后得：

$$\Delta x(s-t)=2\Delta y(x_i+1)+\Delta x(2b-2y_i-1)$$

$$=2\Delta y\cdot x_i-2\Delta x\cdot y_i+2\Delta y+\Delta x(2b-1)$$

令$d_i=\Delta x(s-t)$，由于$\Delta x>0$，所以d_i和$s-t$符号相同，因此可得：

$$d_i=2\Delta y\cdot x_i-2\Delta x\cdot y_i+C$$

其中， $C=2\Delta y+\Delta x(2b-1)$，它是个常量。

同理，通过使下标增1后，得到d_{i+1}为

$$d_{i+1}=2\Delta y\cdot x_{i+1}-2\Delta x\cdot y_{i+1}+C$$

Bresenham画线算法---迭代公式

两者相减得到

$d_{i+1} - d_i = 2\Delta y \cdot (x_{i+1} - x_i) - 2\Delta x \cdot (y_{i+1} - y_i)$

$= 2\Delta y - 2\Delta x\,(y_{i+1} - y_i)$

改写为

$$d_{i+1} = d_i + 2\Delta y - 2\Delta x\,(y_{i+1} - y_i)$$

- 当$d_i \geq 0$时，即$s-t \geq 0$，该选择T， $y_{i+1} = y_i + 1$，

  $d_{i+1} = d_i + 2(\Delta y - \Delta x)$；

- 当$d_i < 0$时，即$s-t < 0$，该选择S， $y_{i+1} = y_i$，

  $d_{i+1} = d_i + 2\Delta y$。

因此得到的判别式迭代公式总结为

$$d_{i+1} = \begin{cases} d_i + 2(\Delta y - \Delta x) & d_i \geq 0 \\ d_i + 2\Delta y & d_i < 0 \end{cases}$$

Bresenham画线算法---起点的判别式

由于$d_i=\Delta x(s-t)$，所以对于直线**起点的判断式d**有

$$d_i=\Delta x[2m(x_i+1)+2b-2y_i-1]$$

$$=\Delta x[2(mx_i+b-y_i)+2m-1]$$

由于起点在直线上，所以满足$mx_i+b-y_i=0$，可得

$$\mathbf{d_1=2m\Delta x-\Delta x=2\Delta y-\Delta x}$$

上面讨论的是直线斜率0≤k≤0的情况。对于一般情况可做如下处理。

当斜率的绝对值$|k|>1$时，将x, y和dx, dy互换，即以y方向作为计长方向，y方向总是增1（或减1），x向是否增减1，根据以上给出的判别式决定。根据上面的讨论，当$d_i\geq 0$时，x增1（或减1），$d_i<0$时，x不变。

Bresenham算法

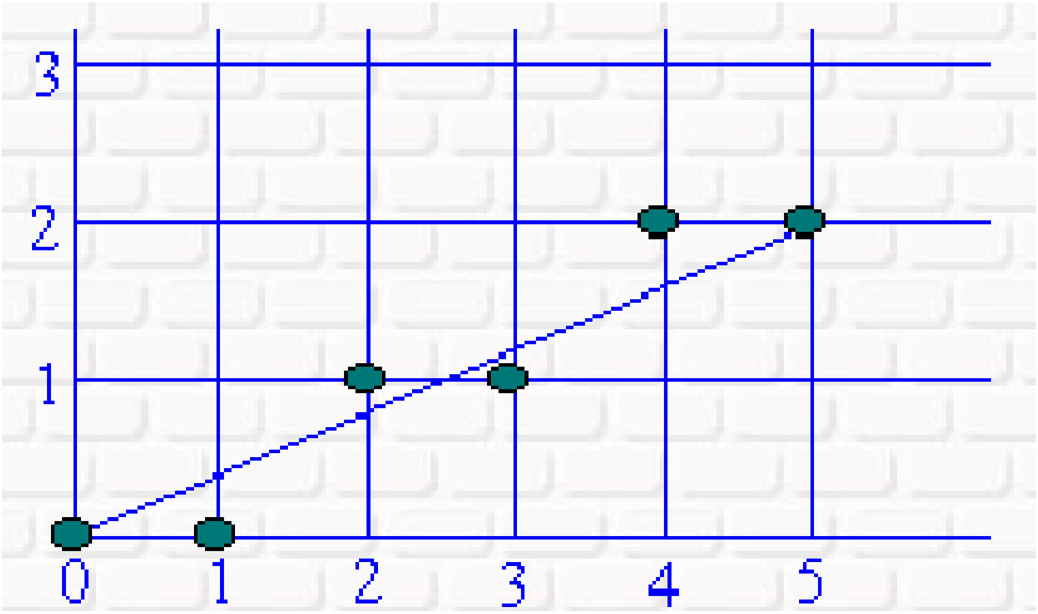
```
void BresLine(int x1,int y1,int x2,int y2){
{
   int x, y, dx, dy, d;
   dx = x2-x1; dy = y2-y1; curx = x1; cury = y1;
   dS = 2* dy, dT = 2 * (dy – dx);d = dS-dx;
   while (curx <=x2)
    {
          if (d < 0) d += dS;
          else { cury = cury + 1; d += dT;}
          setpixel(curx,  cury, color);
          curx = curx + 1;
    }
}
```

Bresenham算法

例， 两点P0（0,0）和P1（5,2）的直线段。

x	y	d
0	0	-1
1	0	3
2	1	-3
3	1	1
4	2	-5
5	2	-1

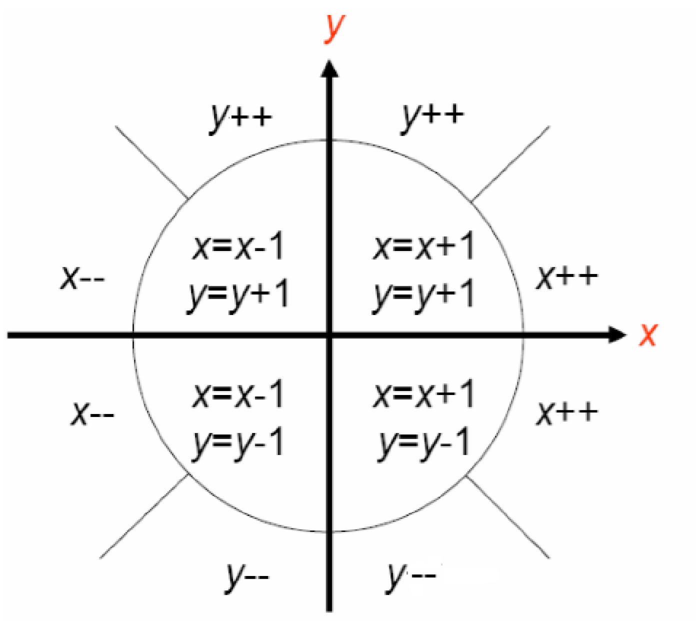
dx=5;dy=2;dS=4;dT=-6;d0=-1



通用Bresenham画线算法

对于一般情况可做如下处理：

当斜率的绝对值$|k|>1$时，将x, y和dx, dy互换，即以y方向作为计长方向，y方向总是增1（或减1），x向是否增减1，根据以上给出的判别式决定。根据上面的讨论，当$d_i \ge 0$时，x增1（或减1），$d_i<0$时，x不变。



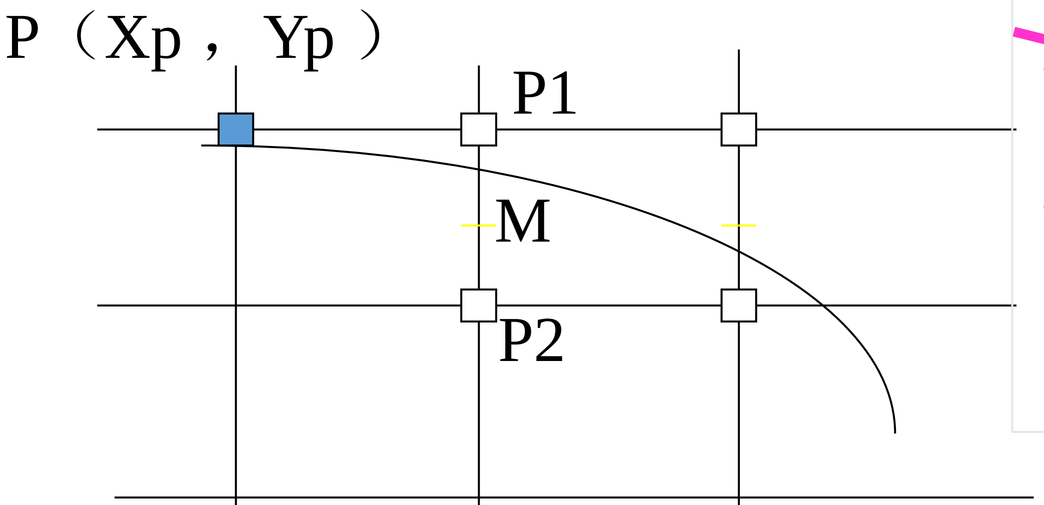
通用Bresenham算法

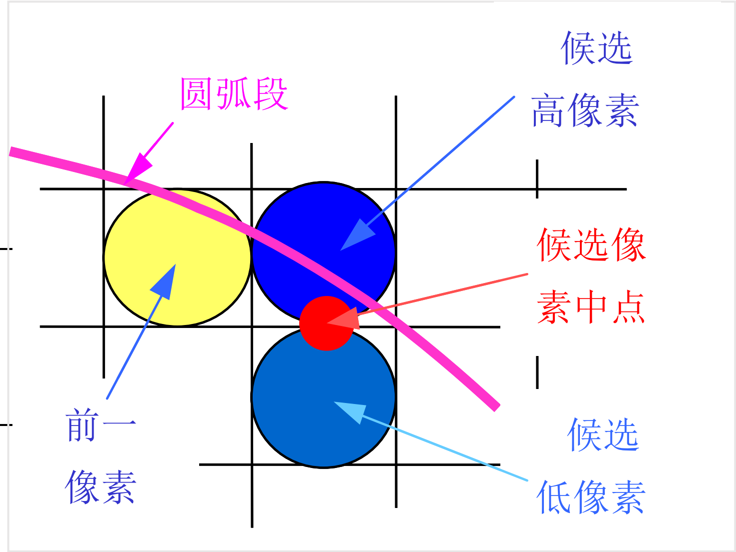
```
//通用Bresenham算法:
// (x1,y1)→(x2,y2)，整数点
//Sign(x) = 1,0,-1 当 x>0,=0,<0
//变量初始化

x=x1;  y=y1;
dx=abs(x2-x1); dy=abs(y2-y1);
s1=Sign(x2-x1); s2=Sign(y2-y1);

//根据直线斜率的符号，交互dx和dy
if  dy > dx  then
    Temp＝dx;  dx=dy;  dy=Temp;
    Interchange=1;
else
    Interchange=0;
end if

// 误差补偿
e=2*dy-dx;
```

```
//主循环
for  i=1  to  dx
     setpixel(x,y);
     while(e>0)
            if(Interchange==1) then
                      x=x+s1;
            else
                      y=y+s2;
            end if
            e=e-2*dx;
     end while
      if(Interchange==1) then
            y=y+s2;
     else
            x=x+s1;
     end if
            e=e+2dy;
     next i;
Finish
```

中点画圆算法

- 利用圆的对称性，只须讨论1/8圆。





- P为当前点亮象素，那么，下一个点亮的象素可能是P1（X_p+1，Y_p）或P2（X_p +1，Y_p -1)。

- **核心问题：采样&像素计算**

中点画圆法

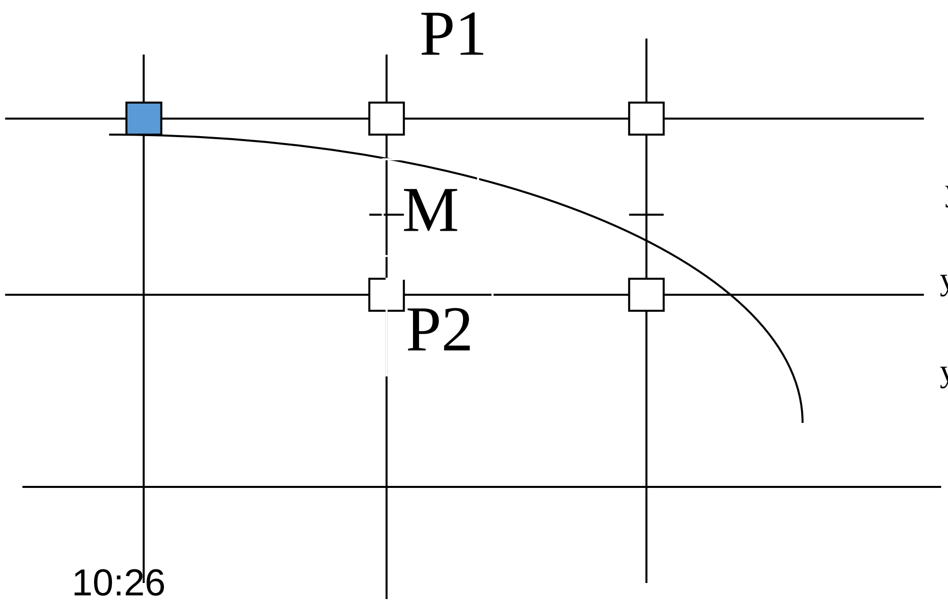
- 构造函数：F（X，Y）=X^2 + Y^2 - R^2；则

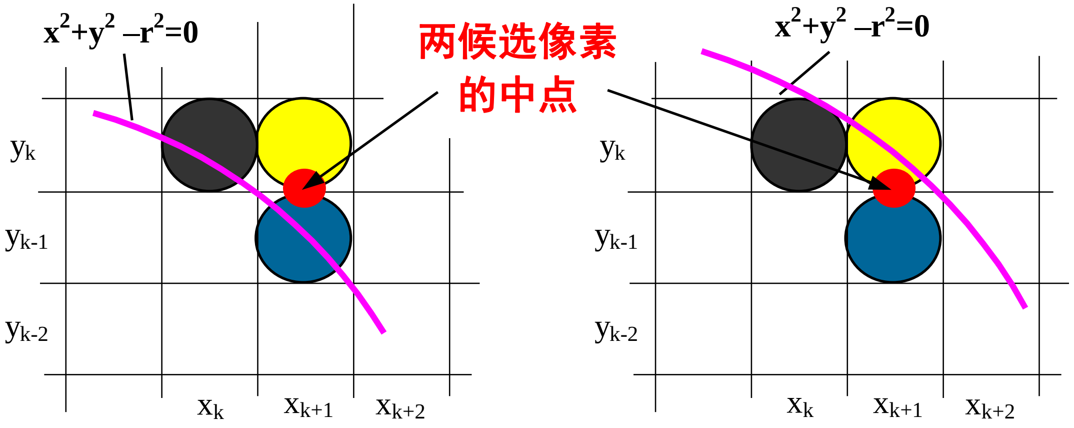
  F（X，Y）= 0　（X，Y）在圆上；

  F（X，Y）< 0　（X，Y）在圆内；

  F（X，Y）> 0　（X，Y）在圆外。

- 设M为P1、P2间的中点，M=(Xp+1,Yp-0.5)





沿圆路径取样位置 x_{k+1} 处候选像素间的中点

中点画圆法

为此，可采用如下**判别式** $d = F(M) = F(x_p + 1, y_p - 0.5)$

$$= (x_p + 1)^2 + (y_p - 0.5)^2 - r^2$$

假定当前判别式*d*为已知，

(1) 若***d*<0**，则**P1**被选为新的点亮象素，那么再下一个象素的判别式为

$$d_1 = F(M) = F(x_p + 2, y_p - 0.5)$$

$$= (x_p + 2)^2 + (y_p - 0.5)^2 - r^2$$

$$= d + 2x_p + 3$$

故***d*的增量为2x_p + 3**。

(2) 若***d*>=0**，则**P2**被选为新的点亮象素，则再下一个象素的判别式为

$$d_1 = F(M) = F(x_p + 2, y_p - 1.5)$$

$$= (x_p + 2)^2 + (y_p - 1.5)^2 - r^2$$

$$= d + (2x_p + 3) + (-2y_p + 2)$$

即***d*的增量为2(x_p - y_p) + 5**。

中点画圆法

- 首点的坐标为(0, *r*)，因此*d*的初值为

$$d_0 = F(0+1, r-0.5)$$
$$= 1 + (r-0.5)^2 - r^2$$
$$= 1.25 - r$$

```
void MidPoint_Circle (int x0, int y0, int r, int color) // (x0,y0)为圆心，r为半径
    {
    int x=0;
    int y=r;
    int d=1- r; //是d=1.25 – r取整后的结果
    Cirpot (x0, y0, x, y, color);
    while ( x<y)
    {
        if (d<0)  d+=2*x+3;
        else
         {
             d+= 2(x-y) +5;
             y --;
         }
        x ++ ;
       Cirpot ( x0, y0, x, y, color);
      }
    }
```

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

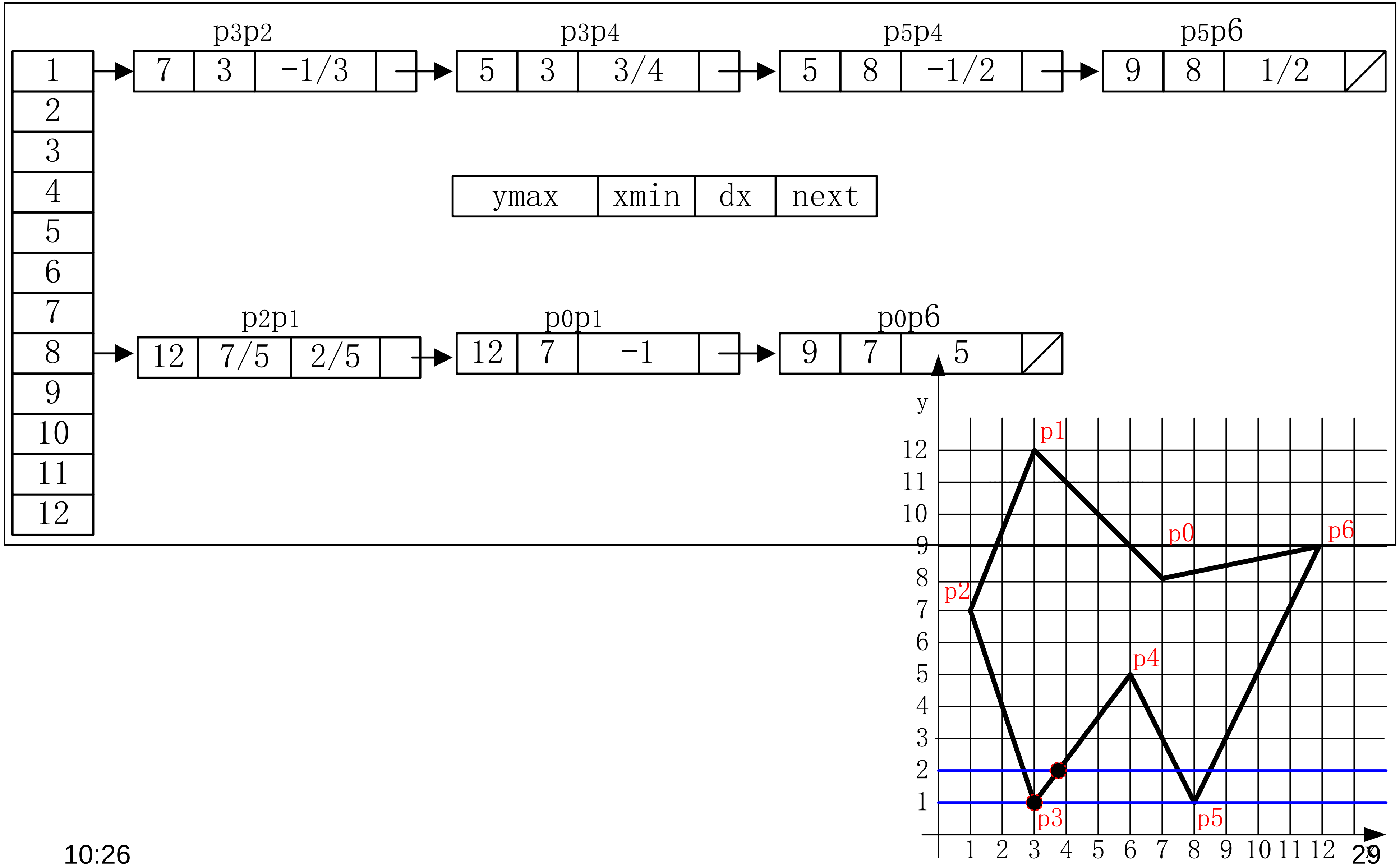
多边形扫描转换算法

- 为了实现上述思想，算法中需要采取灵活的数据结构。
  - 边表ET (Edge Table)：记录多边形信息
  - 活动边链表AET (Active Edge Table) ：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

- 边结点的数据结构
  - y_{max}：边的上端点的y坐标
  - x_{min}：边的下端点x坐标，在活动边链表中，表示扫描线与边的交点的x坐标
  - $dx=1/k$：边的斜率k的倒数
  - next：指向下一条边结点的指针

边表(Edge Table, ET)



多边形$P_0P_1P_2P_3P_4P_5P_6P_0$

构建边表方法

step1: 计算多边形所有顶点的最大y坐标为y_{max};

step2: 构造指针数组ET[y_{max}+1];

step3: 按顺序处理顶点序列中的每条边：

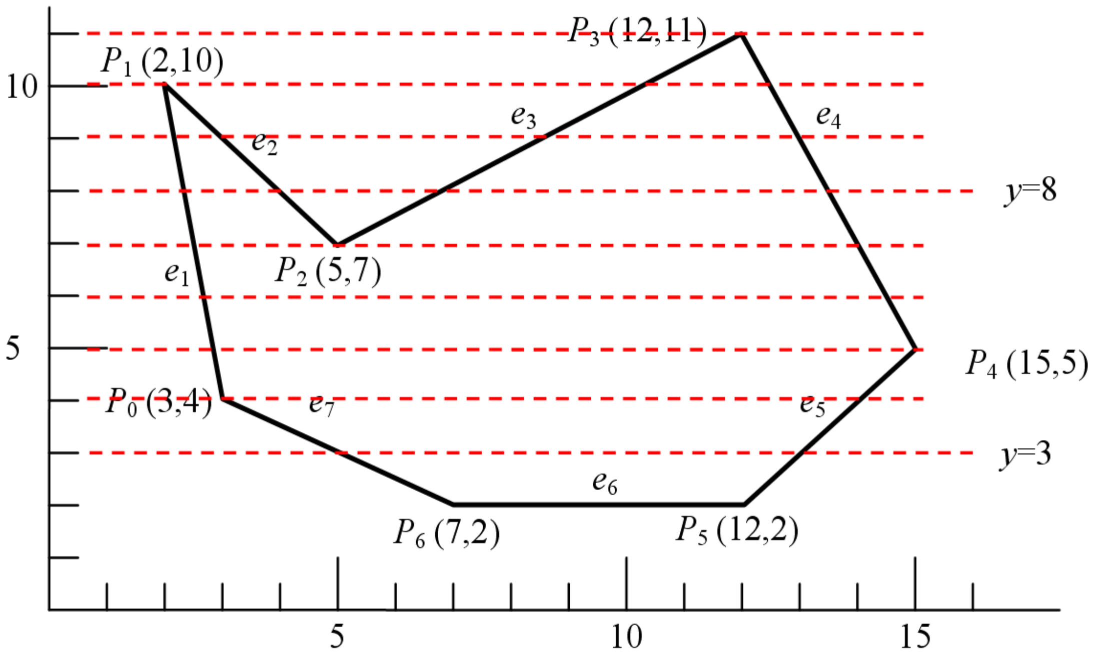
step3.1: 确定边的下端点(min)和上端点(max);

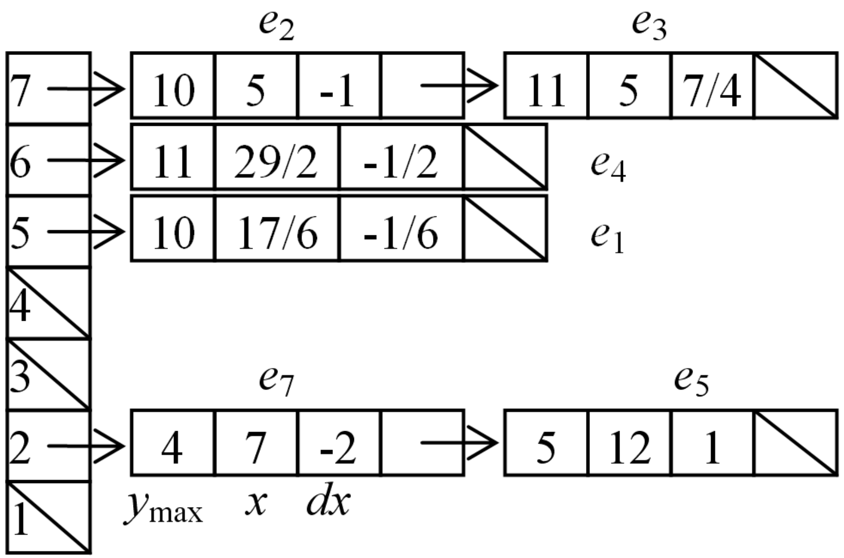
step3.2: 对下端点(min)进行奇异点预处理：若为非极值点,将该点上提一个像素单位，即将该点的y坐标加1，重新计算x坐标，将新的（x,y）作为下端点(min)。

step3.3: 构造边结点：x_{min}，dx (1/k)，y_{max}。

step3.4:将边结点插入到ET[y_{min}]链表中,要求ET[y_{min}]链表中结点按x_{min}由小到大排序，若x_{min}相等，则按照dx(1/k)由小到大排序。

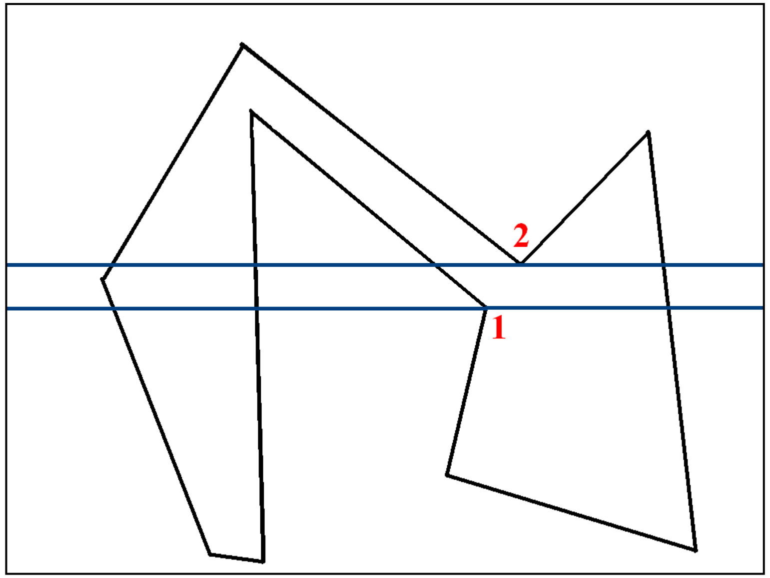
多边形扫描转换实例



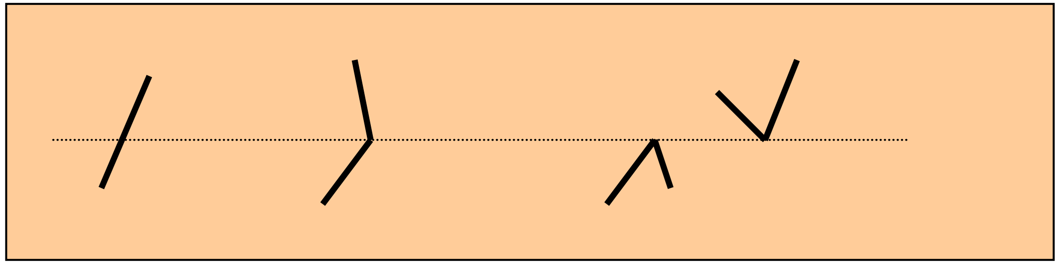


边表ET

奇异点处理

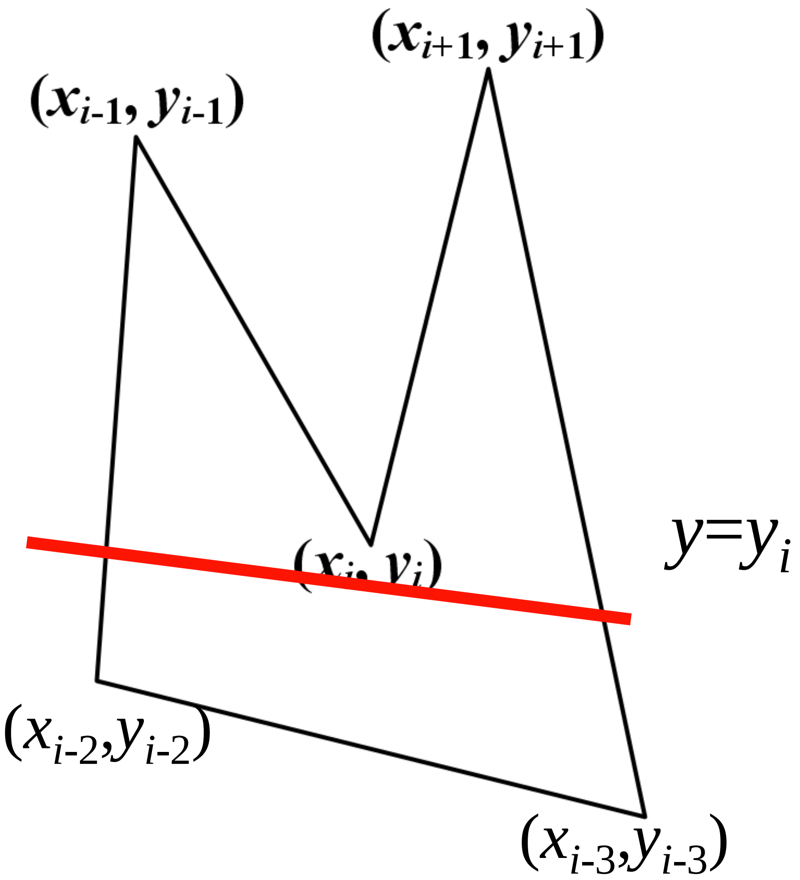


奇异点示意图



- 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

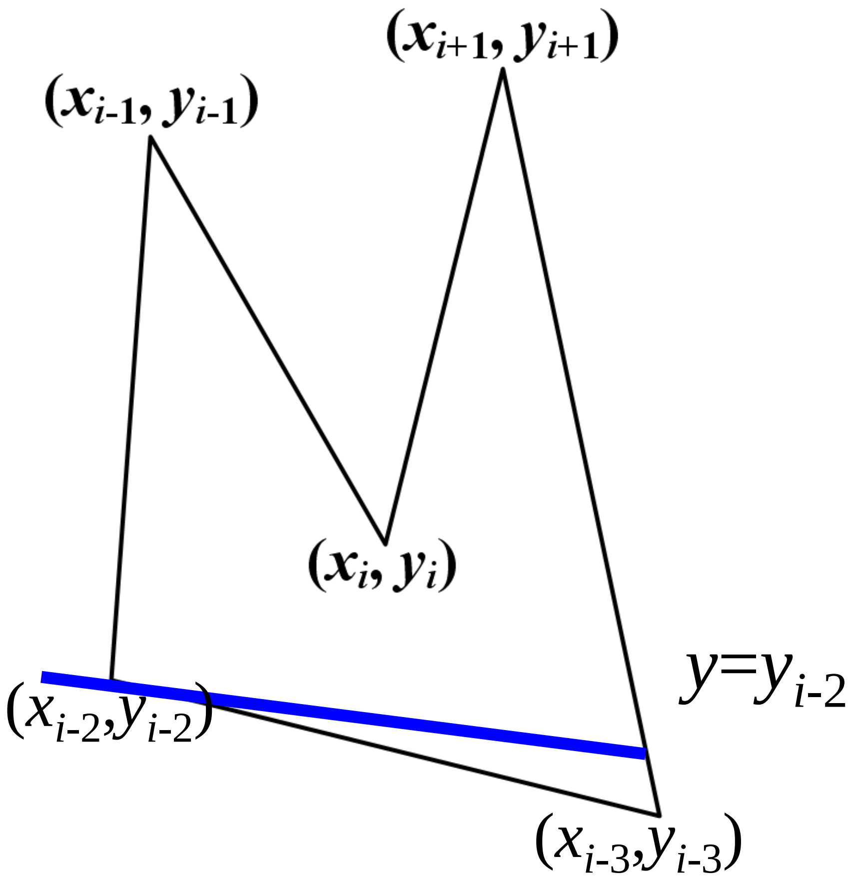


- **极值点**：相邻三个顶点的y坐标满足如下条件：

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

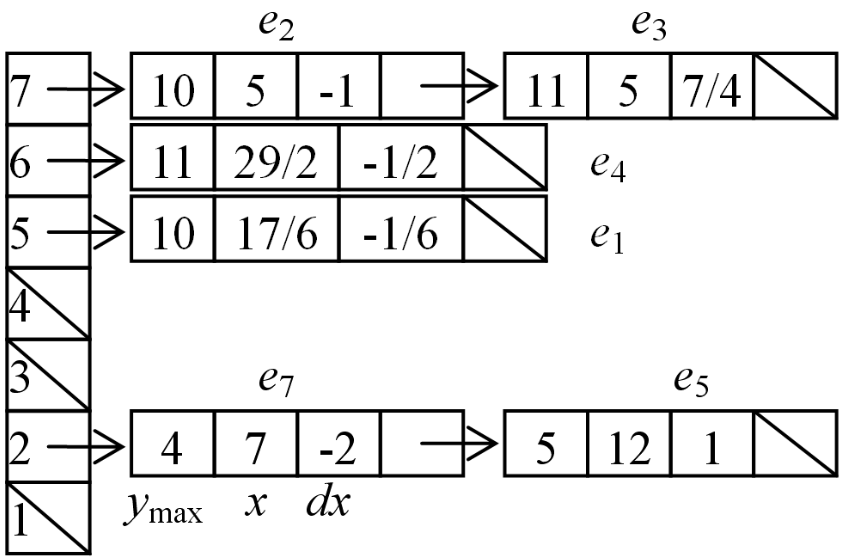
  即相邻三个顶点位于扫描线的同一侧

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

多边形扫描转换算法

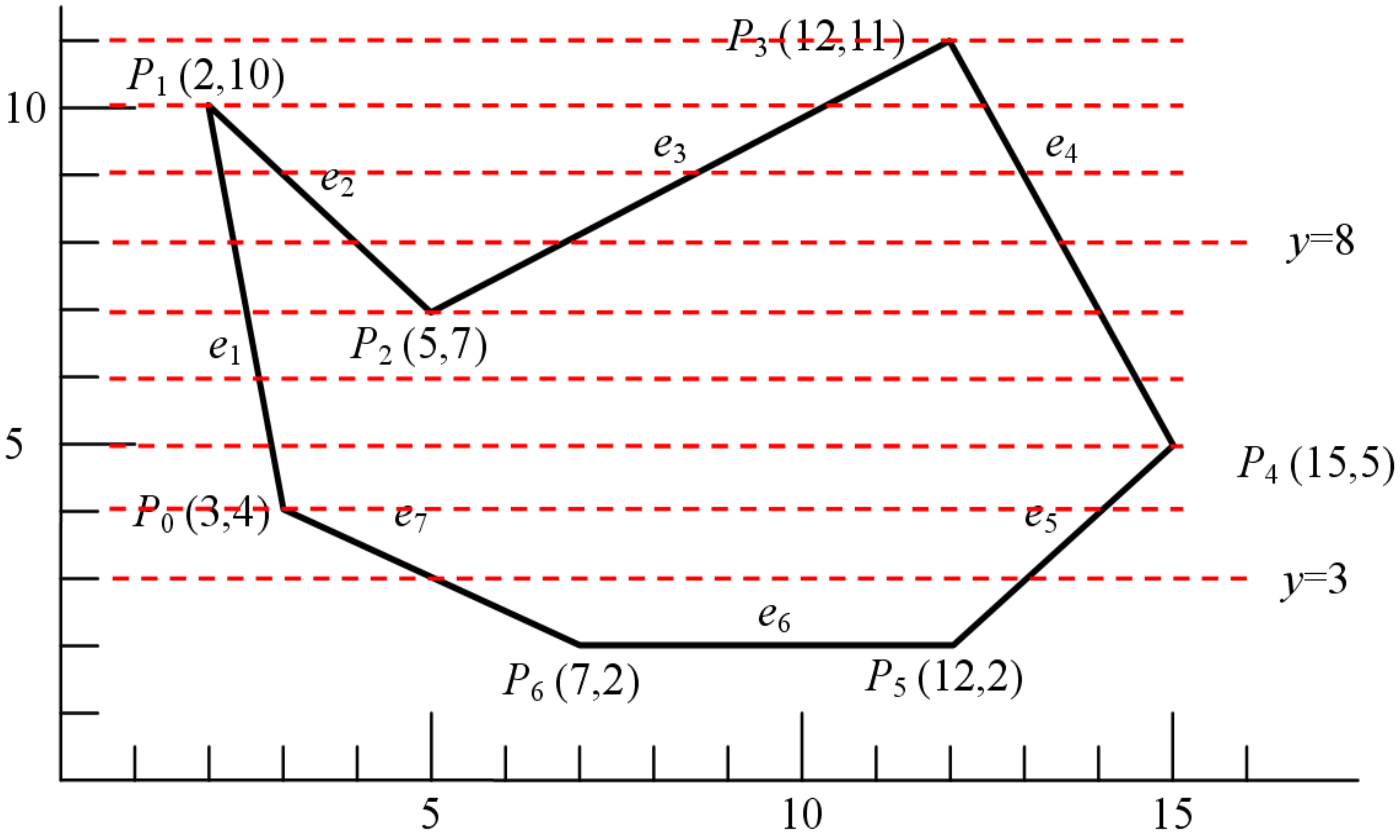



```
void FillPolygonbyAET(Polygon *P, int p
{  step1: 构建边表ET;
   step2: AET=NULL;
   step3:
    for(y = ymin;y <= ymax;y++){
      step3.1:   if  (ET[y] != NULL)  {
          取出y链表中的所有边结点插入到AET中;
          AET中的各边结点按照x值(x值相等时，按dx值)递增排序;
          删除ET[y]链表的所有结点并释放结点空间; (该步骤不用)
      }
      …
   }
}/*end of FillPolygonPbyP() */
```

多边形扫描转换算法

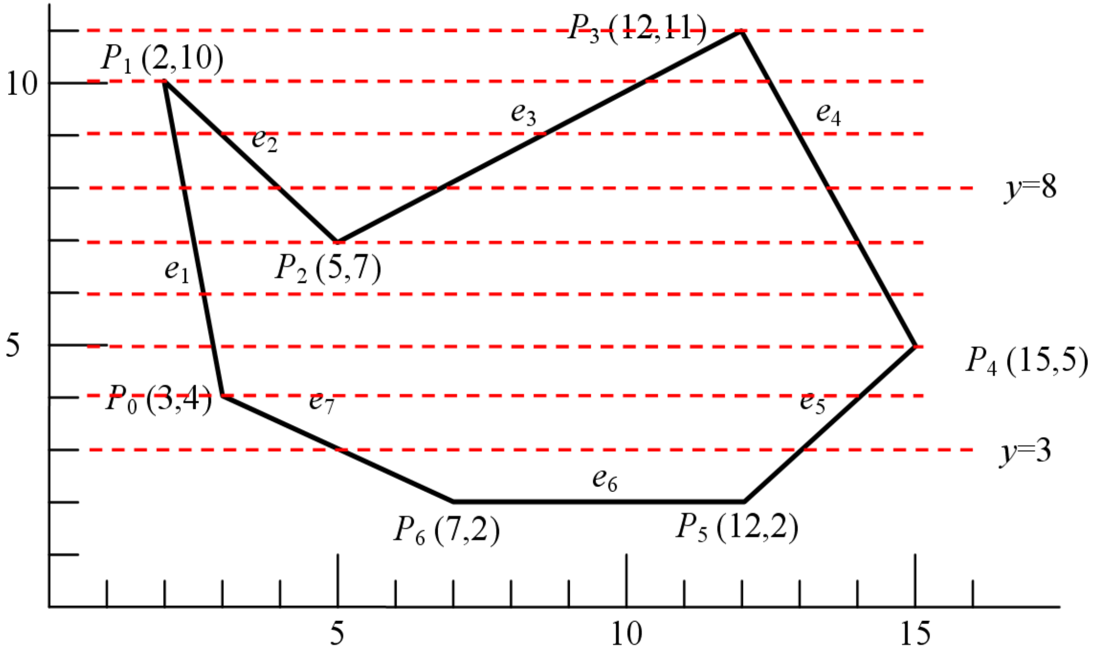
```
void FillPolygonbyAET(Polygon *P, int polygonColor)
{  ....
   step3.2:  if (AET != NULL) {
    step3.2.1:
     /* AET中的边结点依次两两配对，对每一对边结点间的像素着色*/
     p = AET;
     while (p){
        对点(p->xmin, y)和(p->next->xmin, y)中的像素进行着色;
         p = p->next->next;
      }
     step3.2.2:
      将AET中满足y == ymax的边删除 ;
    }
  step3.3: AET表中所有结点的x域: x = x + dx;
  }
}/*end of FillPolygonPbyP()  */
```

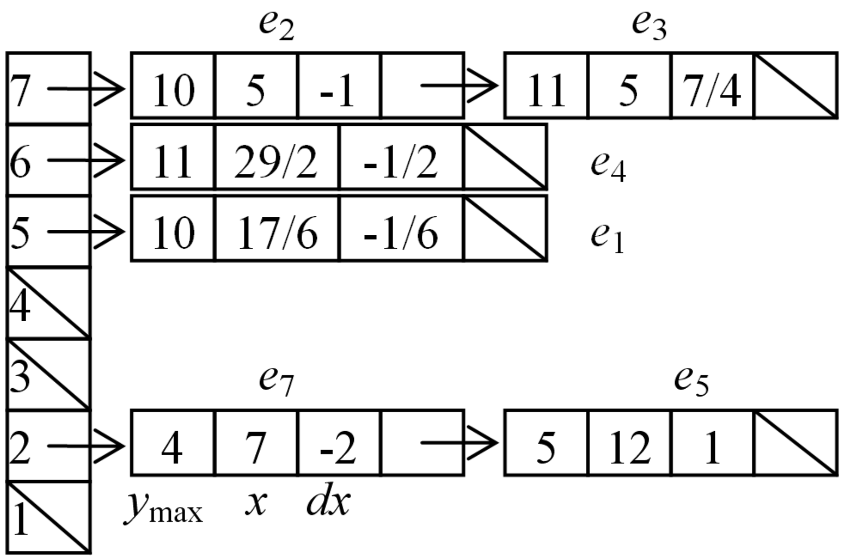
多边形扫描转换实例1



P3 (12,11)
P1 (2,10)
10
e3
e4
e2
y=8
P2 (5,7)
e1
5
P4 (15,5)
P0 (3,4)
e7
e5
y=3
e6
P6 (7,2)
P5 (12,2)
5
10
15

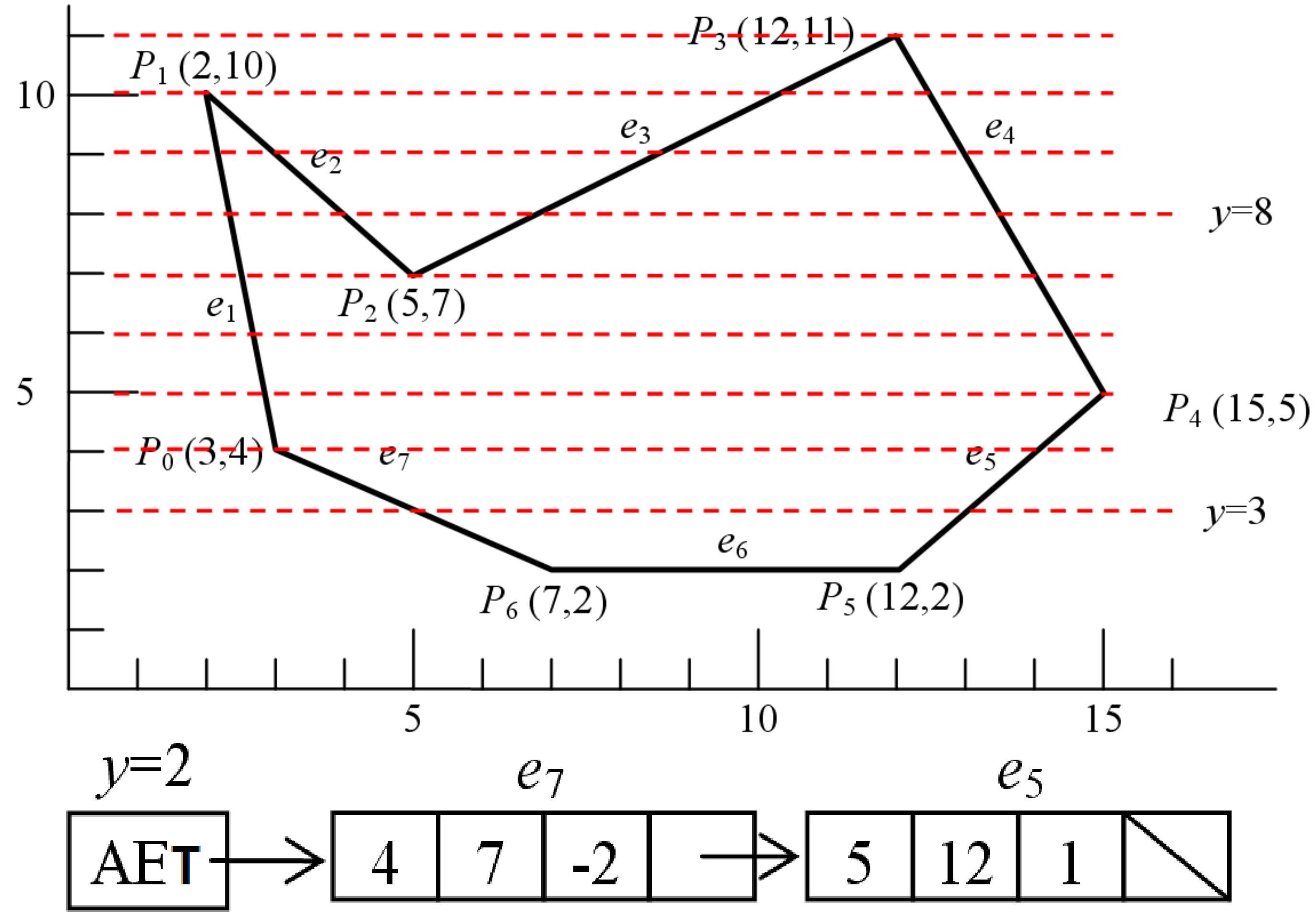
多边形扫描转换实例1





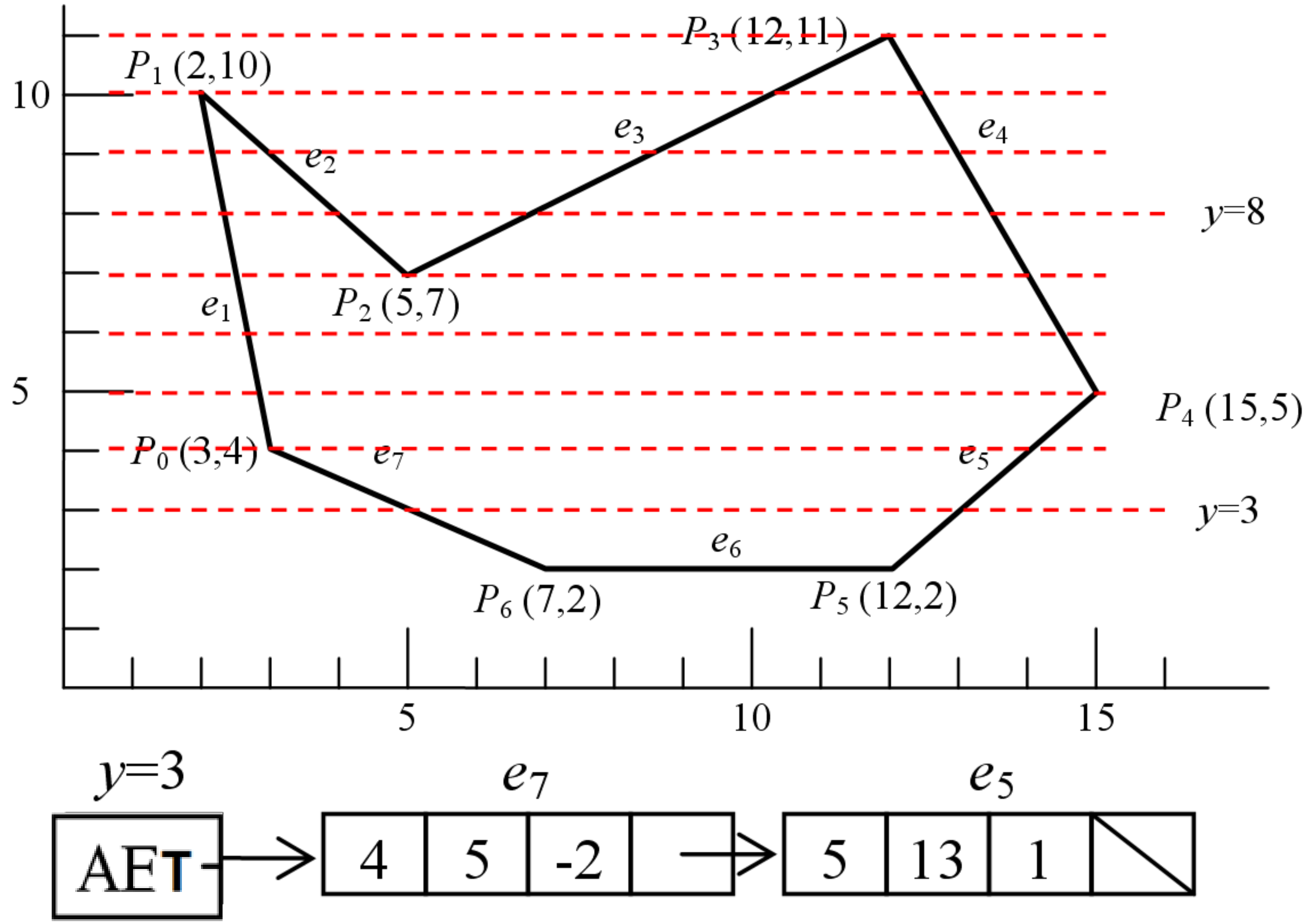
边表ET

多边形扫描转换实例1



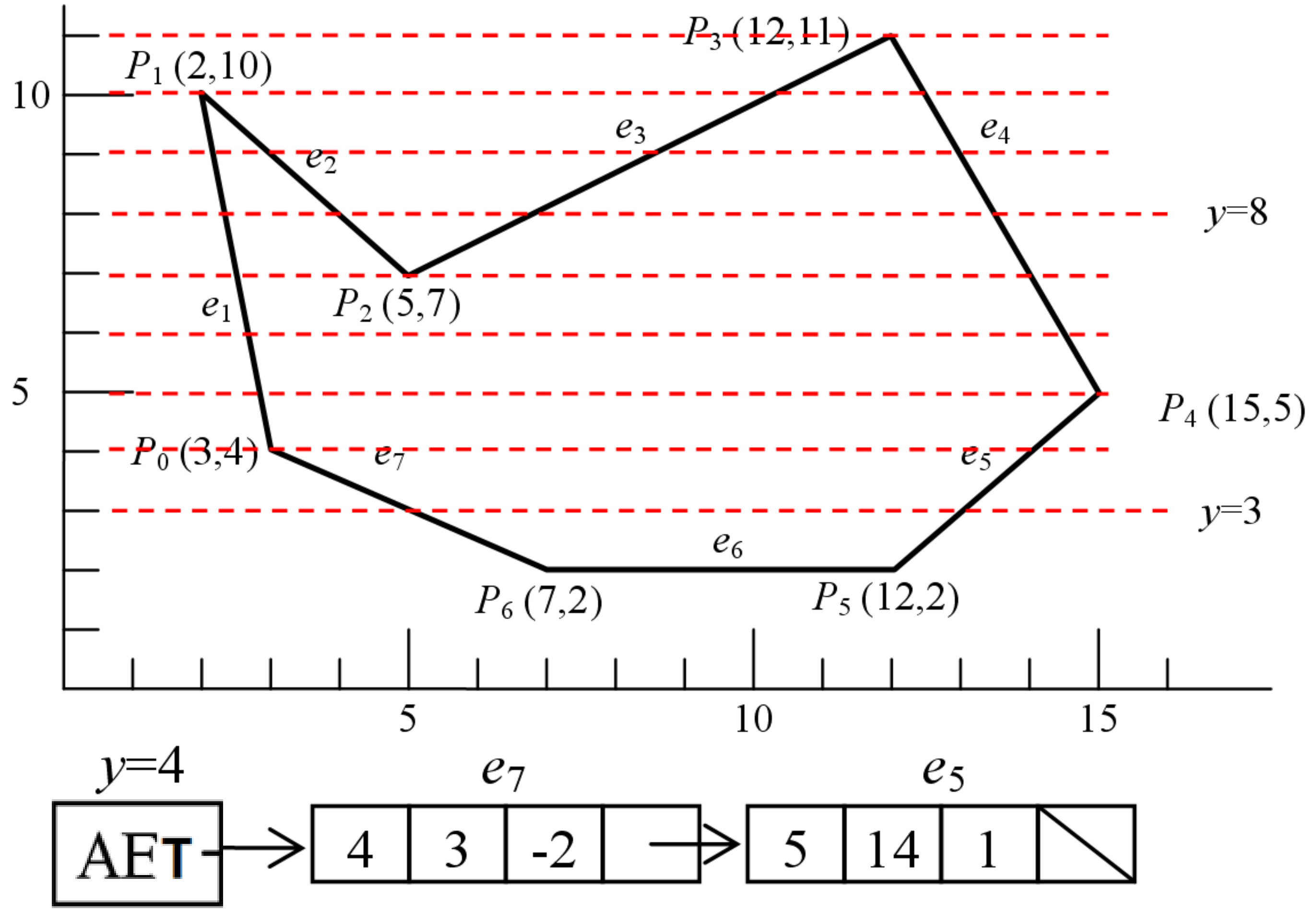
P_3 (12,11)
P_1 (2,10)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15
y=2
e_7
e_5
AET
4
7
-2
5
12
1

多边形扫描转换实例1



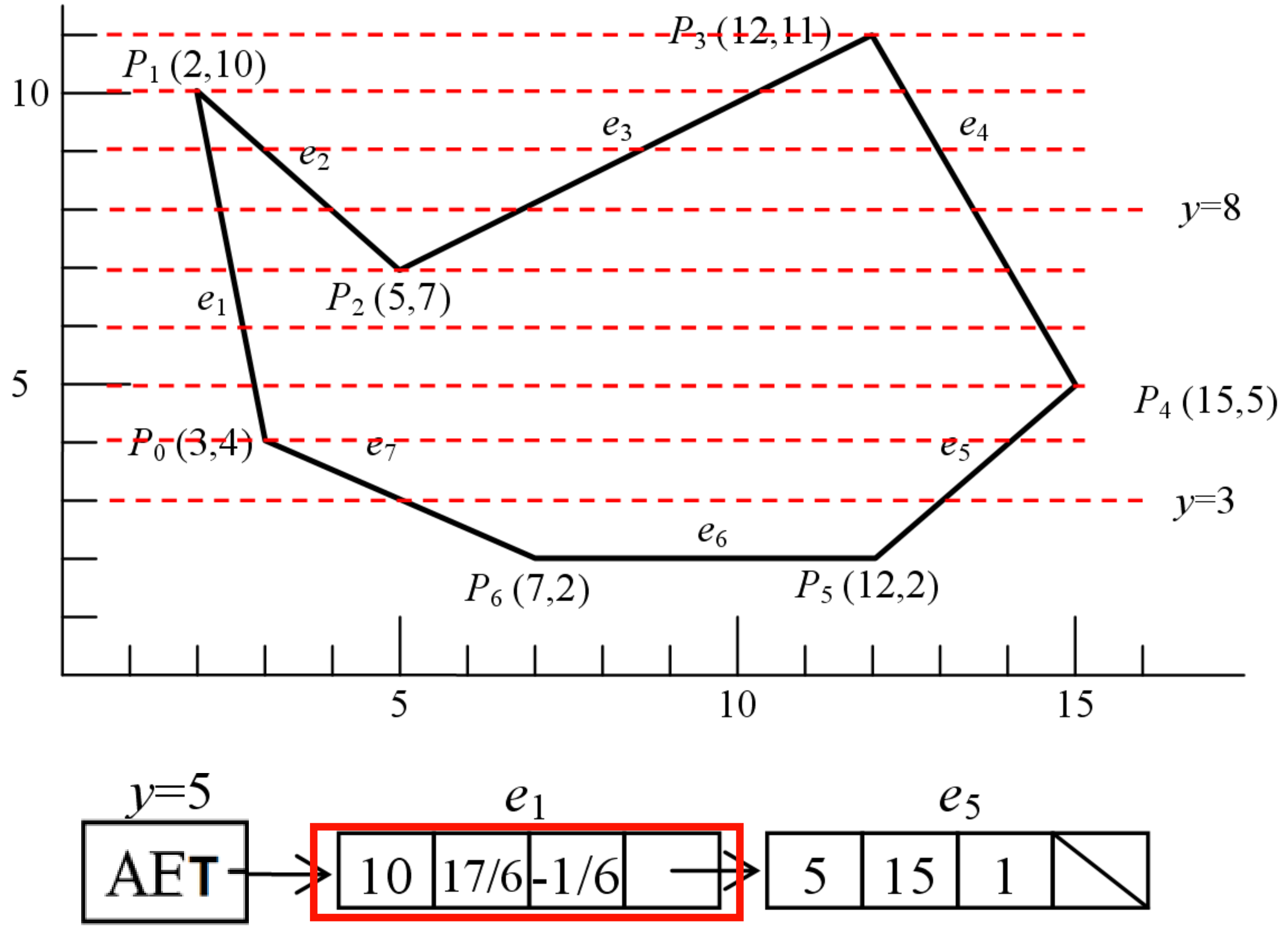
P3 (12,11)
P1 (2,10)
10
e3
e4
e2
y=8
e1
P2 (5,7)
5
P4 (15,5)
P0 (3,4)
e7
e5
y=3
e6
P6 (7,2)
P5 (12,2)
5
10
15
y=3
e7
e5
AET
4
5
-2
5
13
1

多边形扫描转换实例1



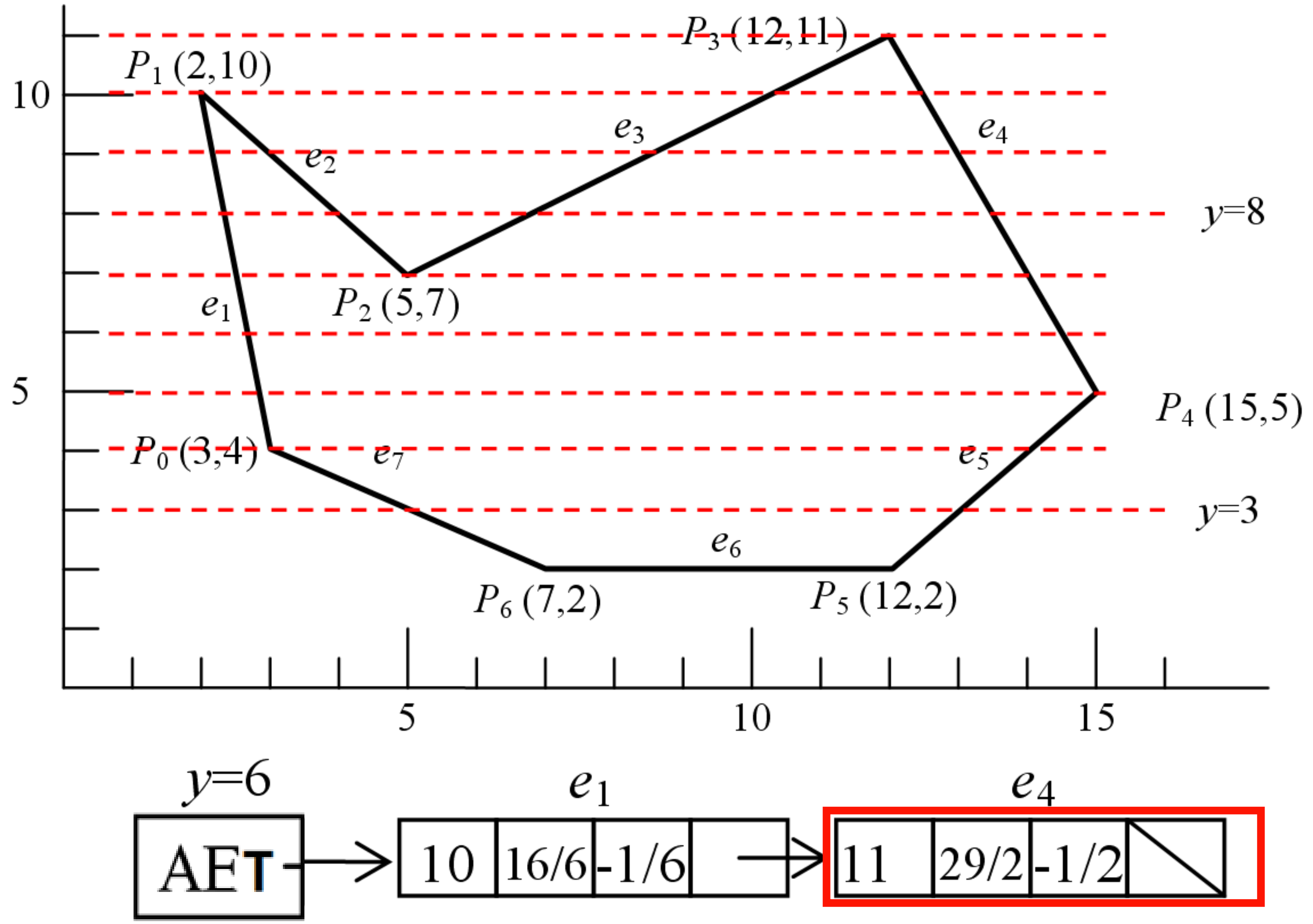
P_3 (12,11)
P_1 (2,10)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15
y=4
e_7
e_5
AET
4
3
-2
5
14
1

多边形扫描转换实例1



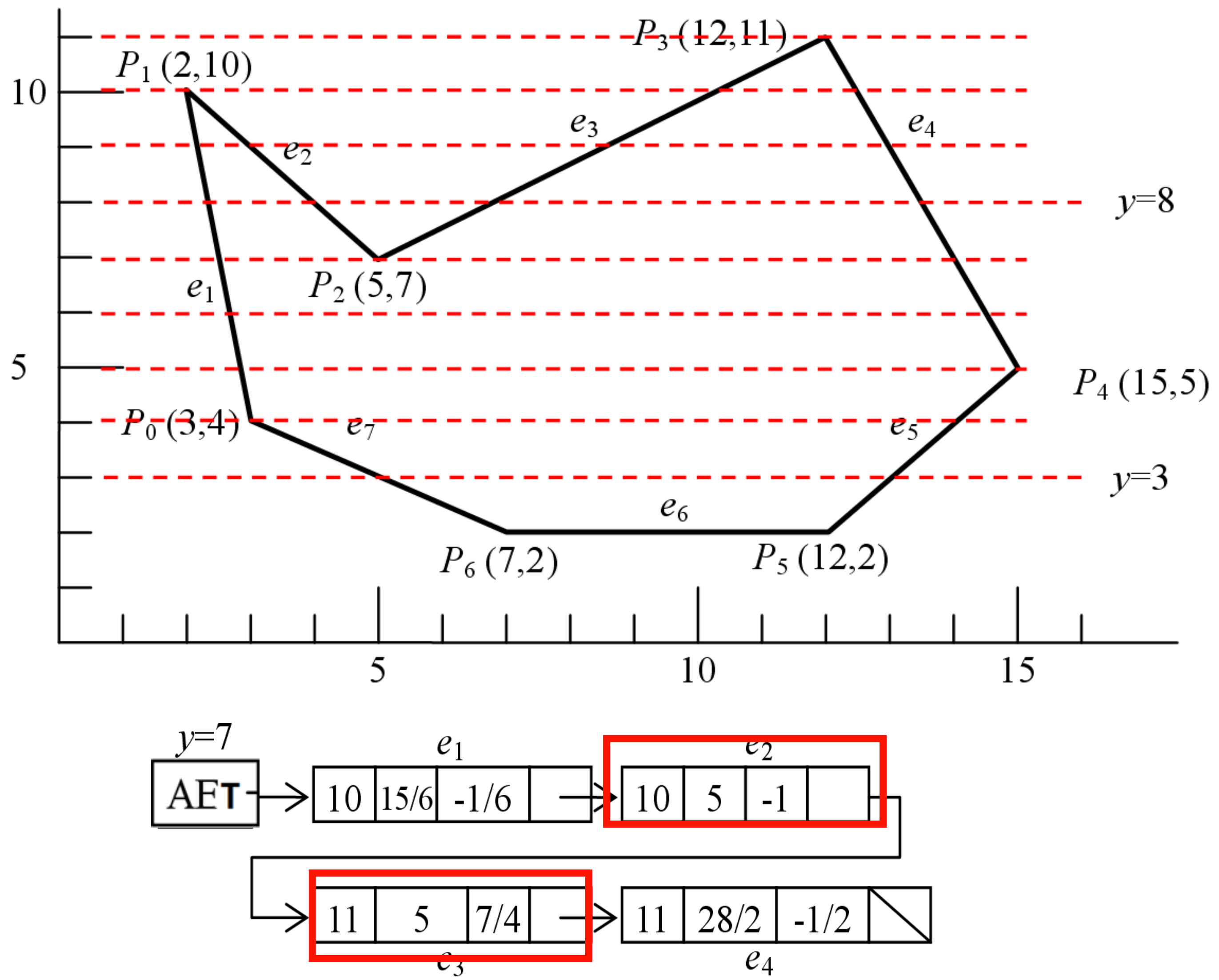
P_3 (12,11)
P_1 (2,10)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15
y=5
AET
e_1
10
17/6
-1/6
e_5
5
15
1

多边形扫描转换实例1



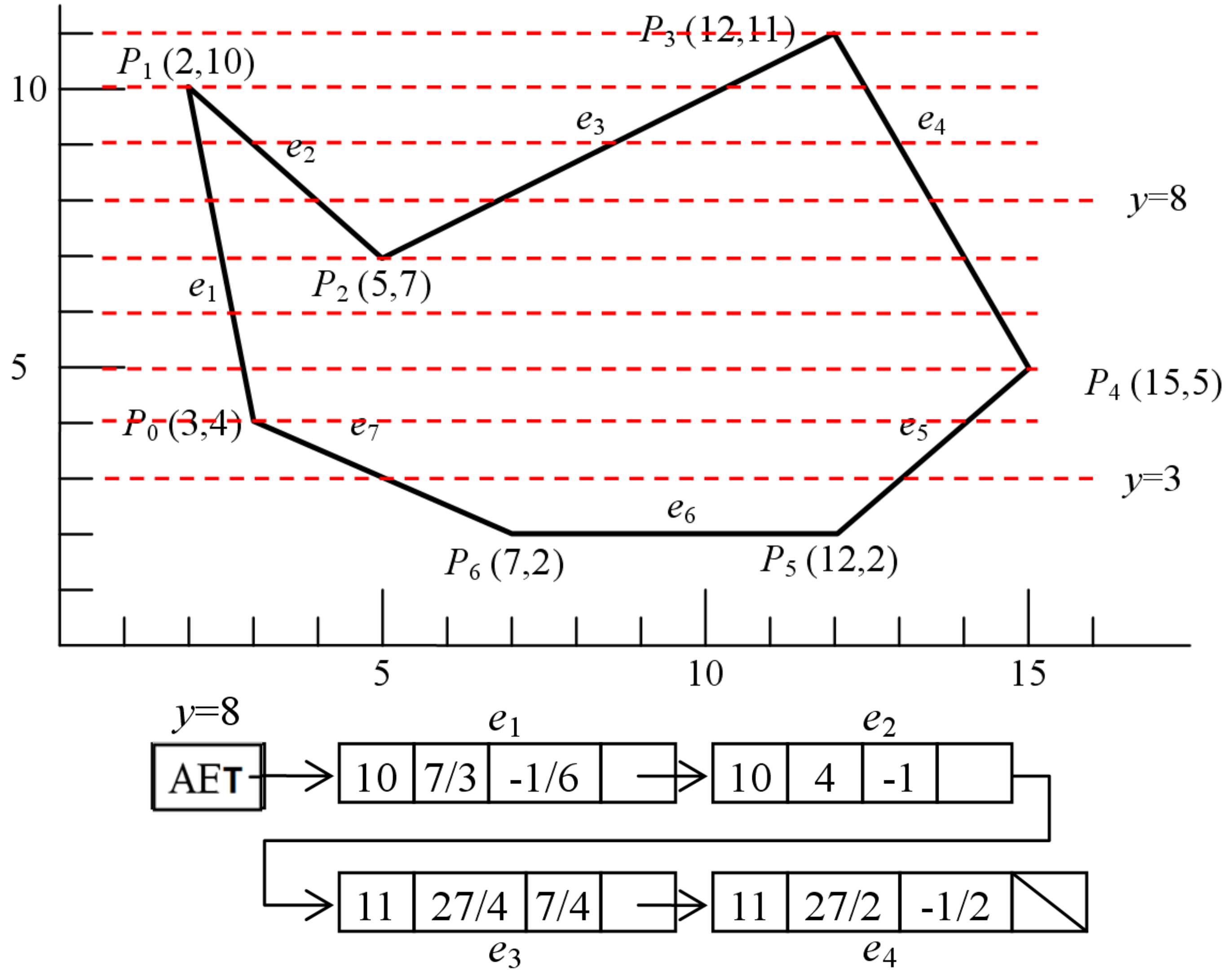
P_3 (12,11)
P_1 (2,10)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15
y=6
e_1
e_4
AET
10
16/6
-1/6
11
29/2
-1/2

多边形扫描转换实例1



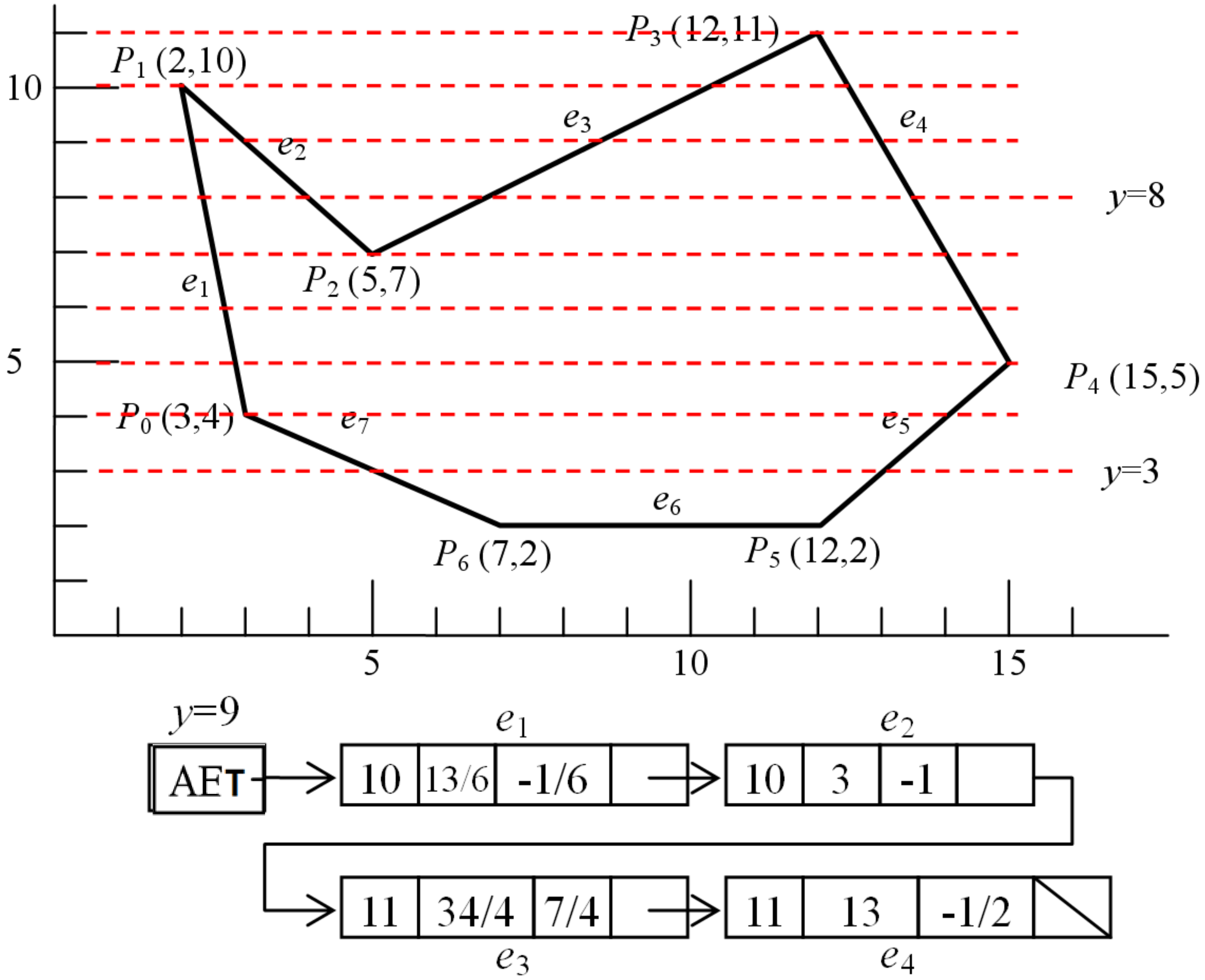
P_3 (12,11)
P_1 (2,10)
10
e_3
e_4
e_2
y=8
e_1
P_2 (5,7)
5
P_4 (15,5)
P_0 (3,4)
e_7
e_5
y=3
e_6
P_6 (7,2)
P_5 (12,2)
5
10
15
y=7
AET
e_1
10
15/6
-1/6
e_2
10
5
-1
11
5
7/4
e_3
11
28/2
-1/2
e_4

多边形扫描转换实例1



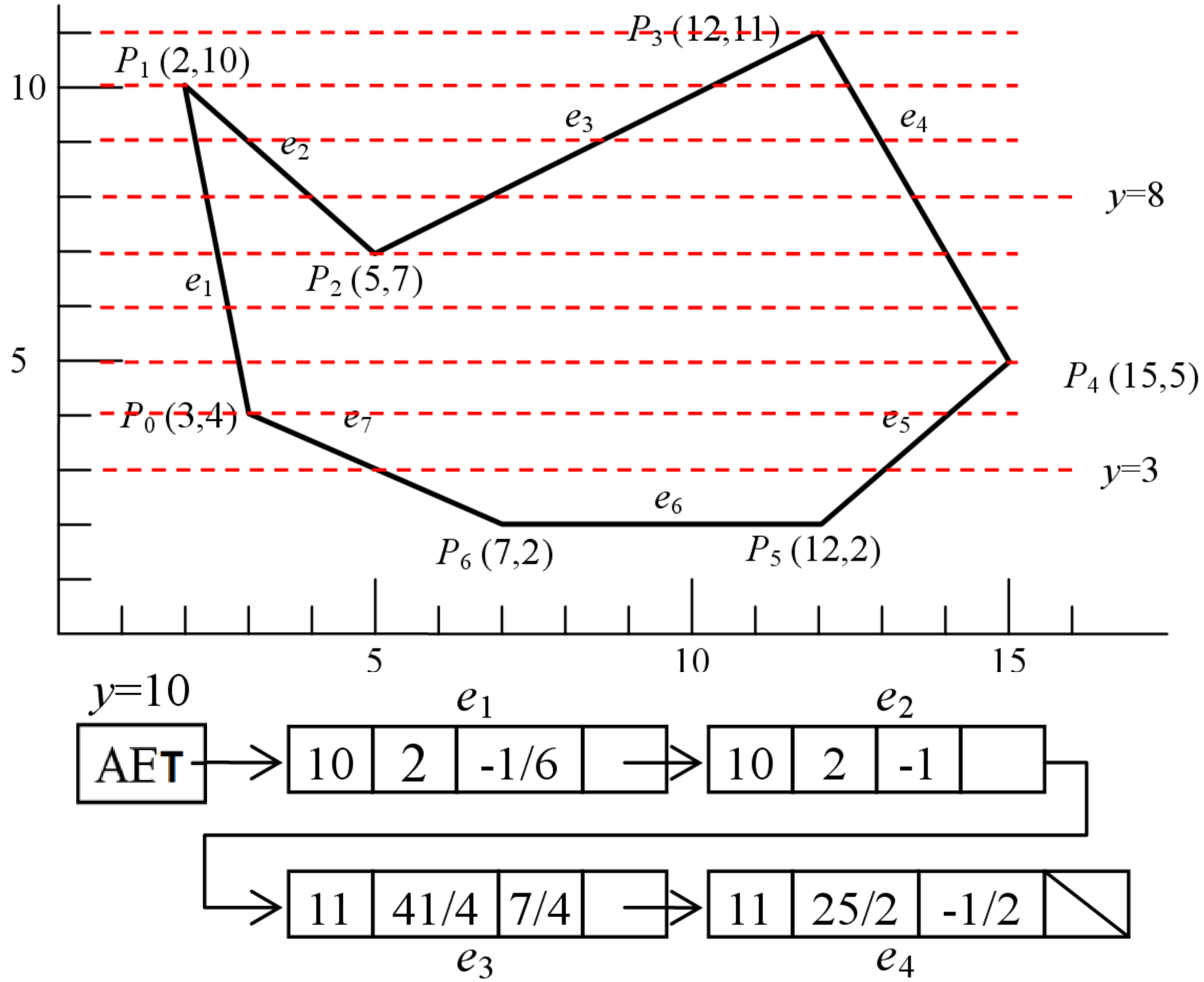
P3 (12,11)
P1 (2,10)
10
e3
e4
e2
y=8
e1
P2 (5,7)
5
P4 (15,5)
P0 (3,4)
e7
e5
y=3
e6
P6 (7,2)
P5 (12,2)
5
10
15
y=8
e1
e2
AET
10 7/3 -1/6
10 4 -1
11 27/4 7/4
11 27/2 -1/2
e3
e4

多边形扫描转换实例1



P3 (12,11)
P1 (2,10)
10
e3
e4
e2
y=8
P2 (5,7)
e1
5
P4 (15,5)
P0 (3,4)
e7
e5
y=3
e6
P6 (7,2)
P5 (12,2)
5
10
15
y=9
AET
e1
10
13/6
-1/6
e2
10
3
-1
11
34/4
7/4
e3
11
13
-1/2
e4

多边形扫描转换实例1



P_3 (12,11)
P_1 (2,10)
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=10
e_1
e_2
AET
10 2 -1/6
10 2 -1
11 41/4 7/4
11 25/2 -1/2
e_3
e_4

多边形扫描转换与区域填充区别

- 联系：光栅图形面着色
- 区别：
- 1. 基本思想

  前者：顶点表示转换成点阵表示

  后者：改变区域内填充颜色，没有改变表示方法
- 2. 边界要求

  前者：要求扫描线与多边形边界交点个数为偶数

  后者：区域封闭
- 3. 基本条件

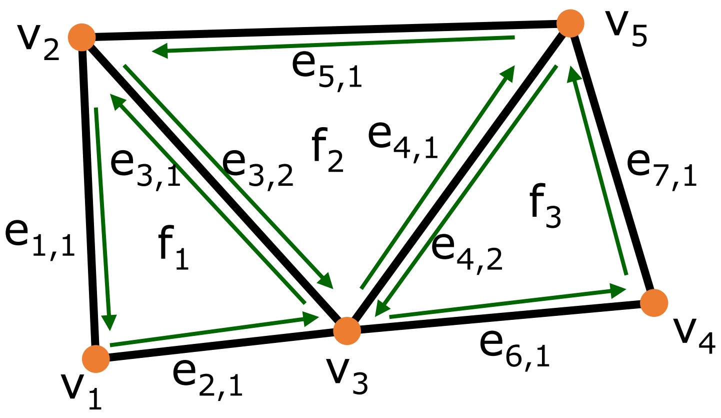
  前者：从边界顶点信息出发

  后者：区域内种子点

半边结构(Half-Edge Structure)

- 每条边被记为两条半边，记录每条半边：
  - 起始顶点的指针
  - 邻接面的指针(如果为边界，指针为NULL)
  - 下一条半边(逆时针方向)
  - 相邻的半边
  - 前一条半边(可选)
- 面：边界上的一条半边
- 顶点
  - 坐标值
  - 指向以此顶点为起始端点的半边

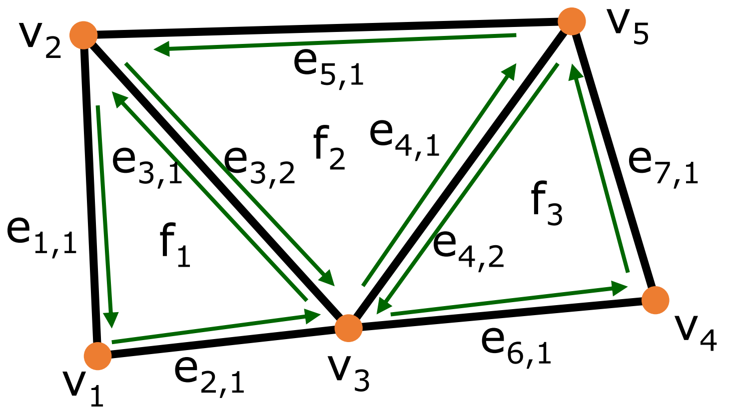
半边结构的实例



顶点	坐标	以此为起点的半边
v_1	(x_1,y_1,z_1)	$e_{2,1}$
v_2	(x_2,y_2,z_2)	$e_{1,1}$
v_3	(x_3,y_3,z_3)	$e_{4,1}$
v_4	(x_4,y_4,z_4)	$e_{7,1}$
v_5	(x_5,y_5,z_5)	$e_{5,1}$

面	半边
f_1	$e_{1,1}$
f_2	$e_{3,2}$
f_3	$e_{4,2}$

半边结构的实例



半边	起点	相邻半边	面	下条半边	前条半边
$e_{3,1}$	v_3	$e_{3,2}$	f_1	$e_{1,1}$	$e_{2,1}$
$e_{3,2}$	v_2	$e_{3,1}$	f_2	$e_{4,1}$	$e_{5,1}$
$e_{4,1}$	v_3	$e_{4,2}$	f_2	$e_{5,1}$	$e_{3,2}$
$e_{4,2}$	v_5	$e_{4,1}$	f_3	$e_{6,1}$	$e_{7,1}$

关于半边结构

- 半边结构讨论：
  - 优势：查询时间 O(1)， 操作时间（通常）O(1)
  - 缺点：只能表示可定向流形，信息冗余
- 关于半边结构更多信息

Bezier样条和B样条的由来

- 法国雷诺汽车公司的P.E. Bezier 提出了一种以逼近为基础的新的参数曲线表示方法，称为Bezier曲线。
- 后来又经过 Gordon、Forrest 和 Riesenfeld 等人的拓展，提出了 B 样条曲线。
- 这两种曲线都能较好地将函数逼近同几何表示结合起来，使得外形设计师使用相关软件时可以象使用作图工具一样得心应手。

Bezier曲线的定义和性质

1．定义

给定空间n+1个点的位置矢量P_i（i=0,1,2,…,n）

则Bezier曲线可定义为：

$$p(t)=\sum_{i=0}^{n}P_i B_{i,n}(t), \qquad t\in[0,1]$$

其中，P_i构成该Bezier曲线的特征多边形，$B_{i,n}(t)$是n次Bernstein基函数：

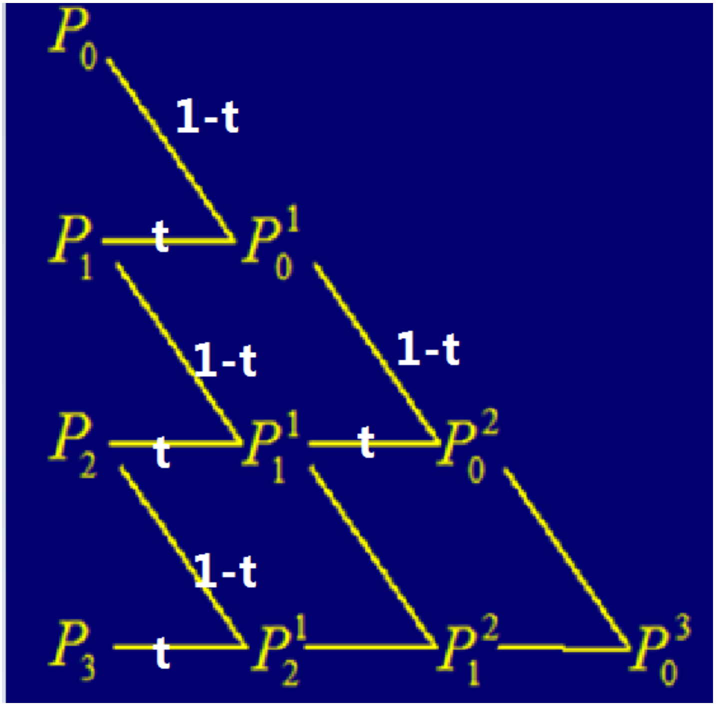
$$B_{i,n}(t)=C_n^i t^i(1-t)^{n-i}=\frac{n!}{i!(n-i)!}t^i\cdot(1-t)^{n-i}$$

$$(i=0,1,\cdots,n)$$

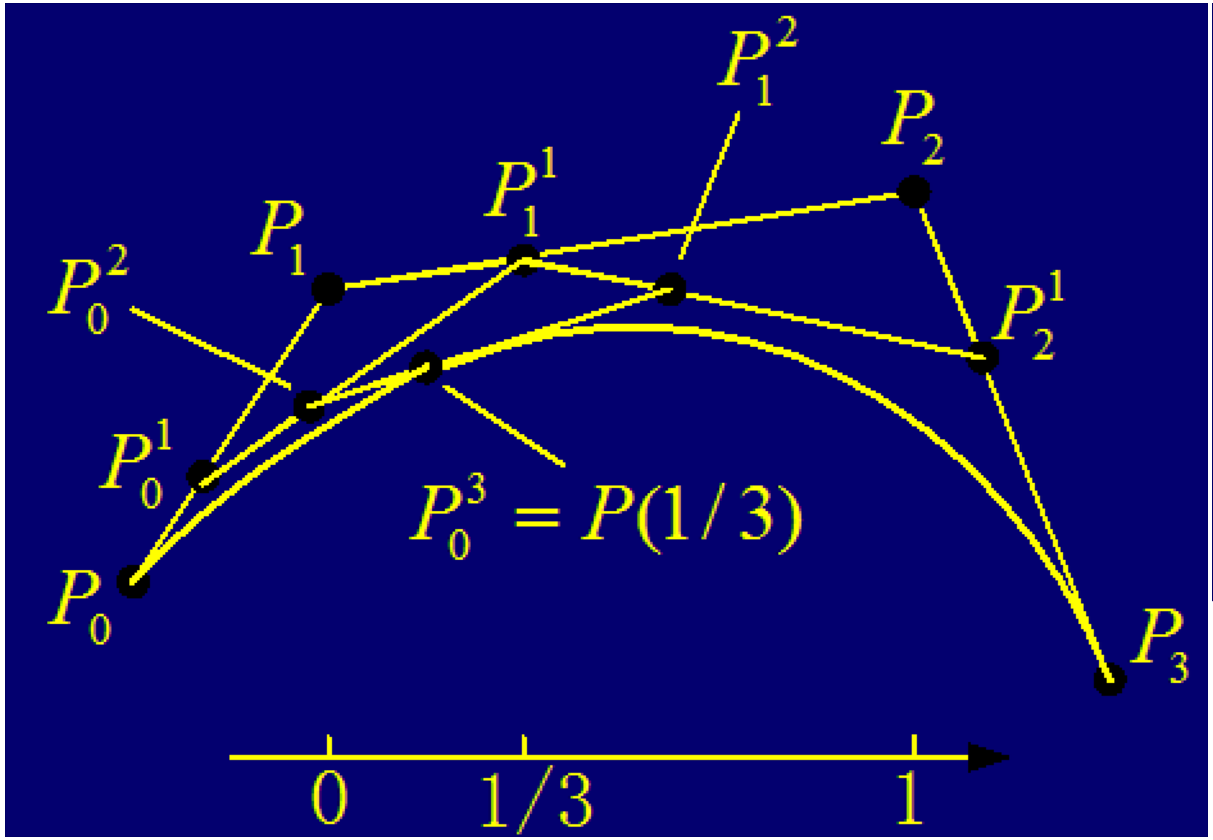
$0^0=1$, $0!=1$

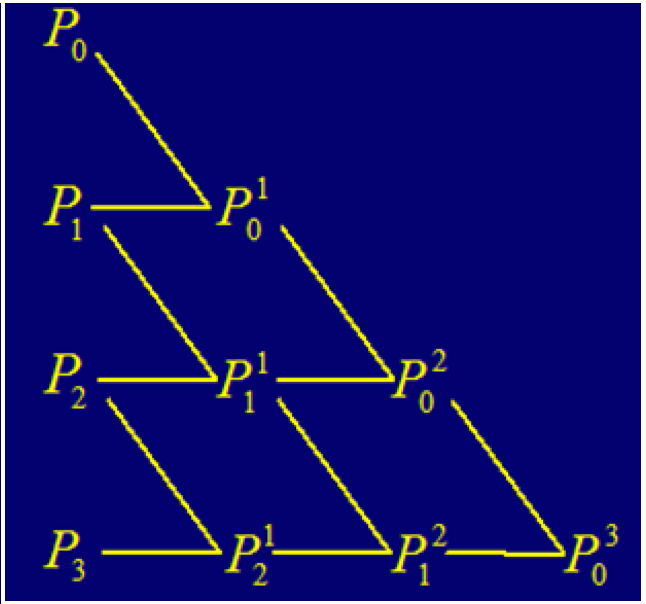
Bezier曲线的递推算法

当n=3时，de casteljau算法递推出的P^k_i呈直角三角形，对应结果如图所示。从左向右递推，最右边点P^3_0即为曲线上的点。教材P192页。



Bezier曲线的递推算法





Bezier样条的缺憾

- 阶次缺乏灵活性　m个控制点，生成的曲线的阶次只能为(m-1次)；
- 控制性较差　当控制点数较多时，曲线的阶次将较高，此时，控制点对曲线形状的控制将明显减弱；或者只能添加额外的拼接条件。
- 不能局部修改　曲线的混合函数值在开区间 (0 , 1) 内均不为零。因此，曲线在 ($0 < t < 1$) 区间内的任何一点均要受到全部控制点的影响。　（而在外形设计中，局部修改往往是要随时进行的）

B样条曲线

- 将不同 Bezier 曲线拼接的问题，转换为同一个样条函数内部的分段问题，并且希望新的曲线：
  - 易于进行局部修改；
  - 更逼近特征多边形；
  - 是低阶次曲线。

B样条曲线

- 曲线参数方程的定义
- B样条曲线的分类和特点
  - 均匀B样条
    - 均匀二次、三次B样条
  - 非均匀B样条
    - NURBS
- OpenGL绘制B样条曲线面

B样条曲线参数方程

$$p(t)=\sum_{k=0}^{n} P_k B_{k,m}(t)$$

De Boor-Cox 递推定义:

$$B_{k,1}(t)=\begin{cases}1 & \text{若}\ t_k \le t < t_{k+1}\\ 0 & \text{其它}\end{cases}$$

$$B_{k,m}(t)=\frac{t-t_k}{t_{k+m-1}-t_k}B_{k,m-1}(t)+\frac{t_{k+m}-t}{t_{k+m}-t_{k+1}}B_{k+1,m-1}(t)$$

- **n+1个控制点**，获得的**B**样条曲线为**m-1次**（**m**阶），且曲线在连接点处具有**m-2阶连续性**。

NURBS的特点

- 自由曲线面和其它简单曲线面的统一表达。
- 具有仿射变换不变性。
- 灵活、局部性：少量的控制点调节出宽广平滑的表面。权因子或者节点值的修改，便于更灵活的控制曲线面形状。
- 广泛适应性。尤其适合流线型表面（人的身体、生活用品、果实 ）和工业产品表面（如汽车、飞机整体或零件） 等。
- CAM领域的广泛应用。具有一系列强有力的几何造型的配套技术(包括节点插入、细分、升阶等)。

- 不太适合用nurbs建模的对象：山或行星等有较多棱角的物体，脸或手指等需要有许多细节的物体，建立简单模型（如盒子）需要的面比多边形建模复杂的多，需要更多的存储空间。
- 反求曲线面上点的参数值时，计算量较大，有时存在数值不稳定问题。

1、帧缓存容量的计算
2、中点画线法算法
3、Bresenham画线算法
4、中点画圆算法
5、多边形扫描转换算法
6 二维三维几何变换：给定顶点，及平移或缩放或旋转或错切或反射矩阵，计算变换后的坐标。
7 Cohen—Sutherland算法—顶点的编码方法
8 Sutherland—Hodgcman
9 半边表示—给定网格，填写半边的表格
10 计算Bezier曲线上的点的坐标
 1 根据定义计算
 2 de Casteljau算法
11 三次均匀B样条上点的坐标的计算，教材P196（7，31）和P197（矩阵形式)