第三章 图元生成

一、概述

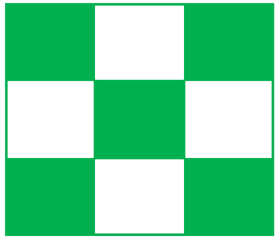
图元是各种图形的组成元素，通用图像系统中的图元有点、圆、椭圆、区域等，图元的属性有线宽、线型、填充模式等。在栅格图形系统中，图元的生成算法就是确定图元中每一个点在帧缓存中的位置，并填写其亮度（颜色）。图元的生成算法要求生成的图形是连通的、快速的.

最基本的图元：点的生成： setpixel(x,y);

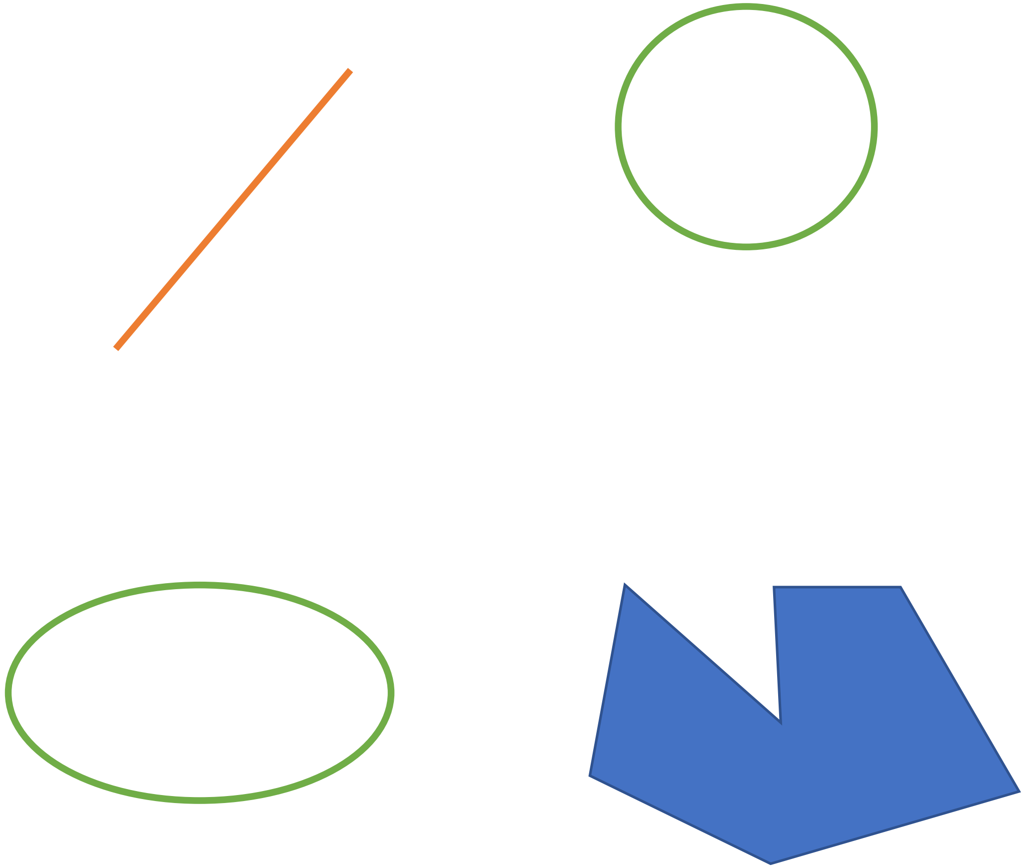
setcolor(c);

一、概述

- 连通：8连通



- 快速：无乘法、无浮点运算



二、直线的生成算法

- 1、数学描述

  已知端点坐标： p_1：（x_1,y_1），p_2:(x_2,y_2)， 绘制一条从p_1到p_2的直线。

$$y = \frac{y_2 - y_1}{x_2 - x_1}x + \frac{y_1.x_2 - y_2.x_1}{x_2 - x_1}$$
$$= mx + b$$

其中$m = \frac{y_2-y_1}{x_2-x_1} = \frac{\Delta y}{\Delta x}$， $b = \frac{y_1.x_2-y_2.x_1}{x_2-x_1}$

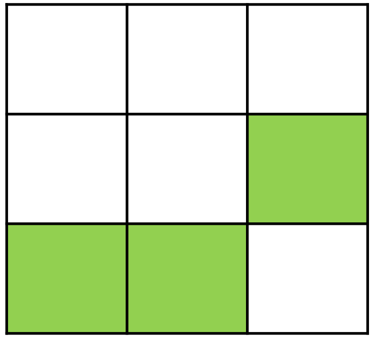
要求： 八连通，无乘法运算，无浮点运算

二、直线的生成算法

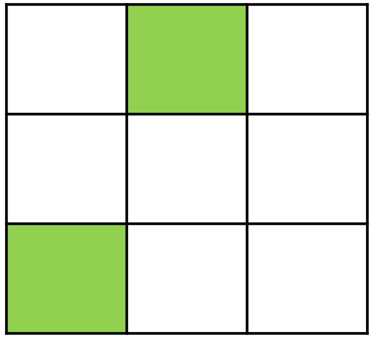
- 1、数学描述

```
for(x=x1,x<x2.,x++)
    {
        y=round(mx+b);
        setpixel(x,y);
    }
```

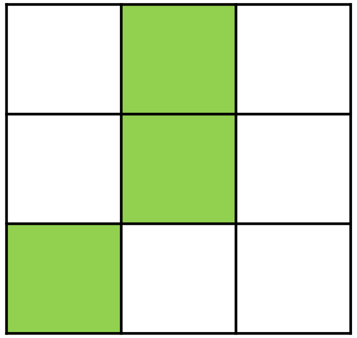
存在问题:
有乘法运算,
有浮点运算,
不能保证8连通



|m| <1, x++



|m| >1, x++



|m| >1, y++

二、直线的生成算法

- 2、DDA算法（Digital Differential Analyzer）
  - 思路：消除乘法

$$y_k = mx_k + b$$

$$y_{k+1} = m(x_{k+1}) + b$$

$$y_{k+1} - y_k = m$$

$$y_{k+1} = y_k + m$$

二、直线的生成算法

- 2、DDA算法（Digital Dirrerential Analyzer）

  - 算法
  - 特点：
    - 无乘法运算
    - 有浮点运算，
    - 误差累积。

step1： 判别x_1, x_2大小，令$x_1<x_2$；

step2：判别$|m|$的大小，设置$k=0$；

step3：如果$|m|<1$, 以x为步长， $x_{k+1}=x_k+1$，计算y_{k+1}；

$$y_{k+1}=round(y_k+m);$$

setpixel（x_{k+1}， y_{k+1}）；

如果$|m|>1$, 以y为步长， $y_{k+1}=y_k+1$；计算x_{k+1}：

$$x_{k+1}=round(x_k+\frac{1}{m});$$

setpixel（x_{k+1}， y_{k+1}）；

Step4： $k=k+1$; 回到$step3$；直到$y_{k+1}=y_2$或者$x_{k+1}=x_2$。

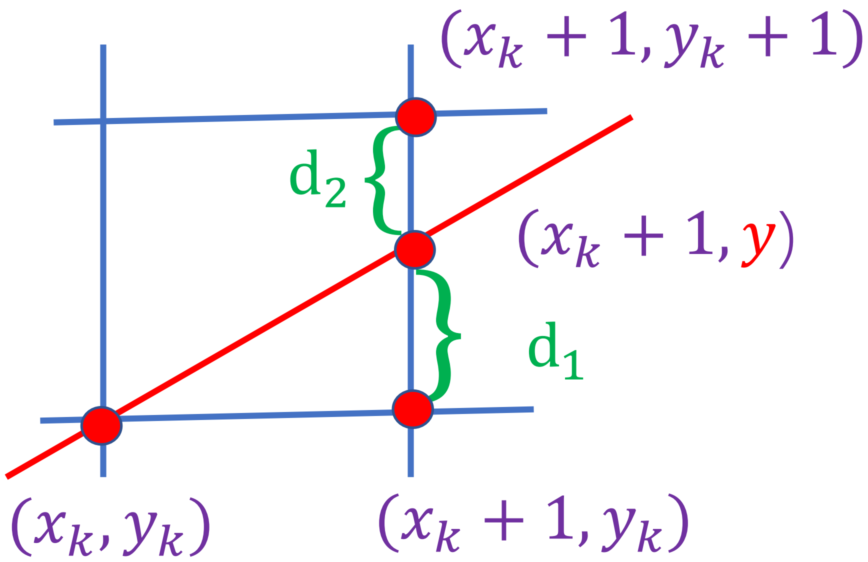
二、直线的生成算法

3、Bresenham算法（以$x_1 < x_2$， $|m| < 1$为例）

- 思路：y_{k+1}只有两种可能，增1或者不变，即：

$$y_{k+1} = \begin{cases} y_k \\ y_k + 1 \end{cases}$$

- 定义：$\begin{cases} d_1 = y - y_k \\ d_2 = y_k + 1 - y \\ y = m(x_k + 1) + b \end{cases}$



- 判别参数：$p_k = \Delta x(d_1 - d_2)$

$$y_{k+1} = \begin{cases} y_k & p_k < 0 \\ y_k + 1 & p_k > 0 \end{cases}$$

如果p_k的计算只有整数和加法运算，则该直线算法可以避免乘法和浮点计算。

二、直线的生成算法

3、Bresenham算法（以x_1<x_2， |m| <1为例）

- 判别参数p_k的计算：

$$p_k = \Delta x(d_1 - d_2)$$

$$d_1 - d_2 = 2y - 2y_k - 1 = 2m(x_k + 1) - 2y_k + 2b - 1$$

其中：$m = \frac{\Delta y}{\Delta x}$

则：$p_k = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$　　其中$c = 2\Delta y + \Delta x(2b - 1)$

$$p_{k+1} = 2\Delta y \cdot (x_k + 1) - 2\Delta x \cdot y_{k+1} + c$$

$$p_{k+1} - p_k = 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

当$p_k < 0$，有$y_{k+1} = y_k$，　$p_{k+1} = p_k + 2\Delta y$

当$p_k > 0$，有$y_{k+1} = y_k + 1$，　$p_{k+1} = p_k + 2\Delta y - 2\Delta x$

其中：$p_0 = 2\Delta y - \Delta x$

二、直线的生成算法

- 3、Bresenham算法（以$x_1<x_2$， $|m|<1$为例）

  - 算法

step1：计算$\Delta x = x_2 - x_1$，$\Delta y = y_2 - y_1$，$2\Delta y$

step2：设置$k = 0$; $p_0 = 2\Delta y - \Delta x$

step3：对于任意的k,

if $p_k < 0$，$p_{k+1} = p_k + 2\Delta y$；$y_{k+1} = y_k$；

else $p_{k+1} = p_k + 2\Delta y - 2\Delta x$；$y_{k+1} = y_k + 1$；

setpixel(x_k+1, y_{k+1})；

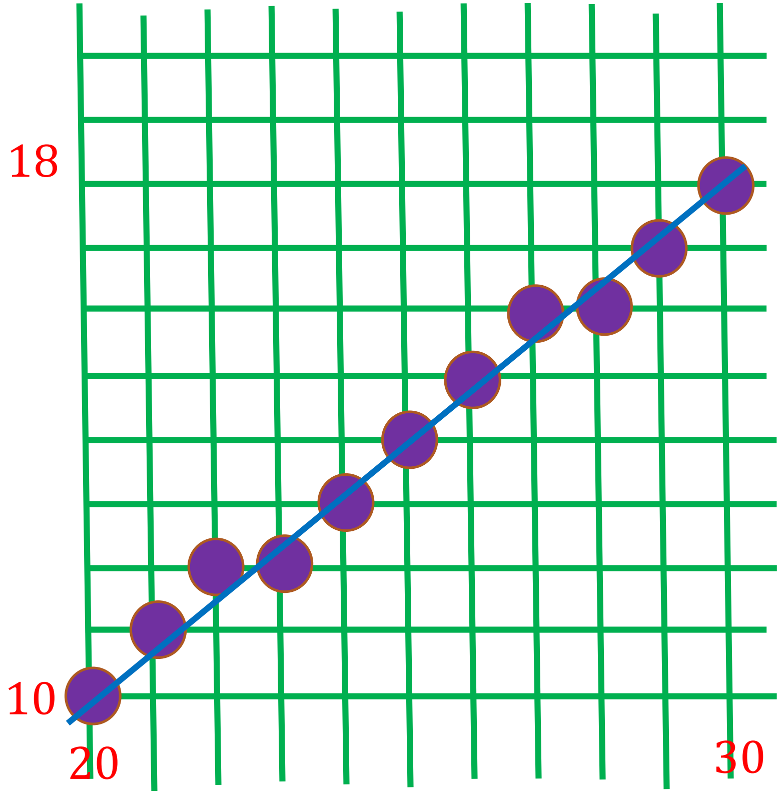
step4：$k = k + 1$; $x_k = x_k + 1$；回到step3，直到 $x_k = x_2$。

  - 特点：无乘法、无浮点运算

二、直线的生成算法

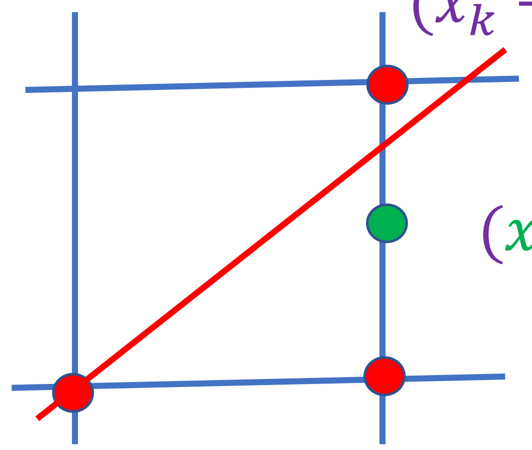
- 3、Bresenham算法（以$x_1<x_2$， |m| <1为例）
  - 例子：已知：$p_1(20,10)$，$p_2(30，18)$，绘制一条从p_1到p_2的直线。

k	p_k	x_{k+1}	y_{k+1}
0	6	21	11
1	2	22	12
2	-2	23	12
3	14	24	13
4	10	25	14
5	6	26	15
6	2	27	16
7	-2	28	16
8	14	29	17
9	10	30	18



二、直线的生成算法

- 4、中点算法（Mid Point Line Algrithm)



  - 思路：如果中点（ $x_k+1, y_k+\frac{1}{2}$）在线下，
    则： $y_{k+1} = y_k + 1$
    否则： $y_{k+1} = y_k$

  - 如果一个判别参数 p_k 可以不用浮点和乘法运算就可以判断中点（ $x_k+1, y_k+\frac{1}{2}$） 在线上或线下，则该直线算法可以避免乘法和浮点计算。

二、直线的生成算法

- 4、中点算法（Mid Point Line Algrithm)

- 判别参数p_k的定义：

$a = y_2 - y_1$

$b = x_1 - x_2$

$c = y_1 x_2 - x_1\, y_2$

定义:直线函数：$f_{line}(x,y) = ax + by + c$

$$f_{line}(x,y) = \begin{cases} < 0 & (x,y)\text{在直线下（上方）方} \\ = 0 & (x,y)\text{在直线上} \\ > 0 & (x,y)\text{在直线上方} \end{cases}$$

定义判别参数为中点的直线函数值：$p_k = f_{line}\left(x_k + 1, y_k + \frac{1}{2}\right)$

$$\begin{cases} \text{如果}p_k < 0\text{，中点在线下，} & y_{k+1} = y_k + 1 \\ \text{如果}p_k > 0\text{，中点在线上，} & y_{k+1} = y_k \end{cases}$$

二、直线的生成算法

- 4、中点算法（Mid Point Line Algrithm)
  - 判别参数p_k的计算：

$$p_k = a(x_k + 1) + b(y_k + \frac{1}{2}) + c$$

$$p_{k+1} = a(x_k + 2) + b(y_{k+1} + \frac{1}{2}) + c$$

$$p_{k+1} - p_k = \begin{cases} a + b & p_k < 0\text{，中点在线下，} y_{k+1} = y_k + 1 \\ a & p_k > 0\text{，中点在线上，} y_{k+1} = y_k \end{cases}$$

$$p_0 = f_{line}\,(x_1 + 1, y_1 + \frac{1}{2}) = a + 0.5b$$

  - 算法（略，参见Bresenham算法）
  - 特点：同Bresenham算法

三、圆的生成算法

- 1、数学描述
  - 已知圆心（0，0）和半径R，绘制一个圆：$x^2+y^2=R^2$

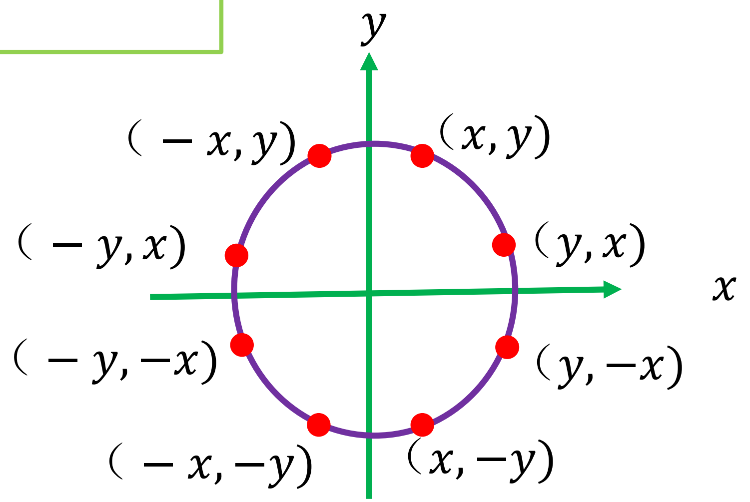
```
for(x=-R, x<R, x++)
    {
        y=round(sqrt（R^2 − x^2））；
        setpixel(x,y);
        setpixel(x,-y);
    }
```

存在问题：
有根号运算，

不能保证8连通

$$\begin{cases} x = R\cos\theta \\ y = R\sin\theta \end{cases}$$

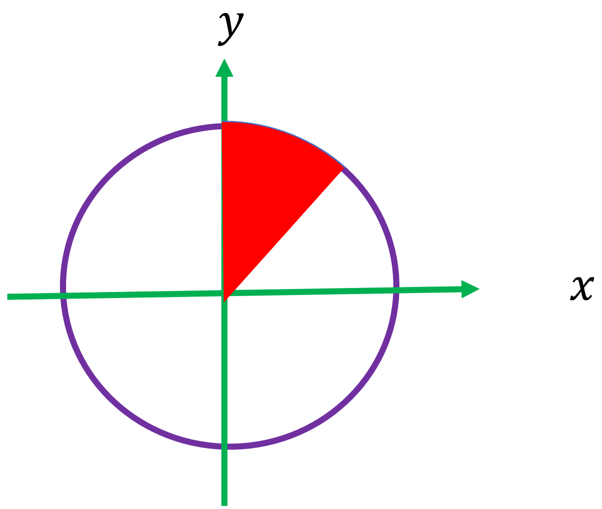
- 八分之一对称

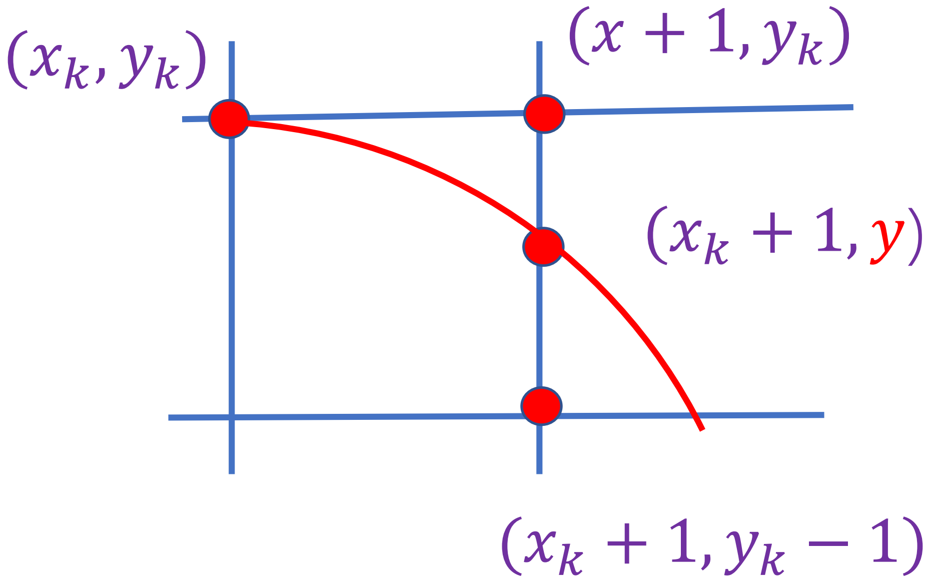


三、圆的生成算法

- 2、Bresenham算法
  - 思路：在八分之一圆上有|m| <1， $x_{k+1} = x_k + 1$, y_{k+1}只有两种可能，

    减1或者不变，即：$y_{k+1} = \begin{cases} y_k \\ y_k - 1 \end{cases}$



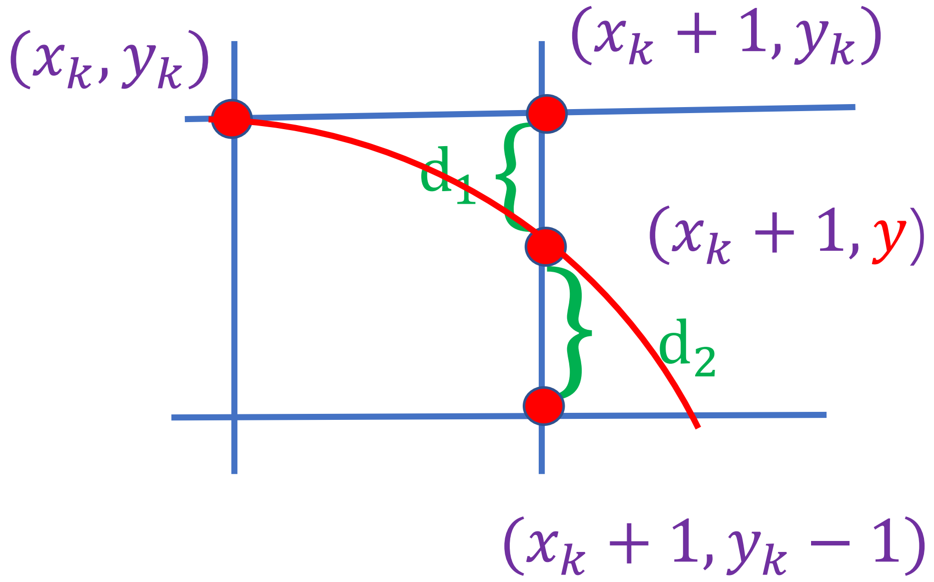


三、圆的生成算法

- 2、Bresenham算法
  - 原理: $y_{k+1}=\begin{cases} y_k & d_1<d_2 \\ y_k-1 & else \end{cases}$

$$\begin{cases} \mathrm{d}_1=y_k-y \\ d_2=y-y_k+1 \\ y=\sqrt{R^2-(x_k+1)^2} \end{cases}$$

  - 判别参数的定义: $\begin{cases} d_1'=y_k^2-y^2 \\ d_2'=y^2-(y_k-1)^2 \\ p_k=d_1'-d_2' \end{cases}$



三、圆的生成算法

- 2、Bresenham算法
  - 判别参数p_k的计算

$$p_k = d_1' - d_2' = y_k^2 - 2y^2 + (y_k - 1)^2$$

$$= 2y_k^2 - 2y_k + 2(x_k + 1)^2 - 2R^2 + 1$$

$$p_{k+1} = 2y_{k+1}^2 - 2y_{k+1} + 2(x_k + 2)^2 - 2R^2 + 1$$

$$p_{k+1} - p_k = \begin{cases} 4x_k + 6 & p_k < 0 \quad y_{k+1} = y_k \\ 4(x_k - y_k) + 10 & else \quad y_{k+1} = y_k - 1 \end{cases}$$

初始点（0， R）

$$p_0 = R^2 - 2(R^2 - 1) + (R - 1)^2$$

$$\boxed{p_0 = 3 - 2R}$$

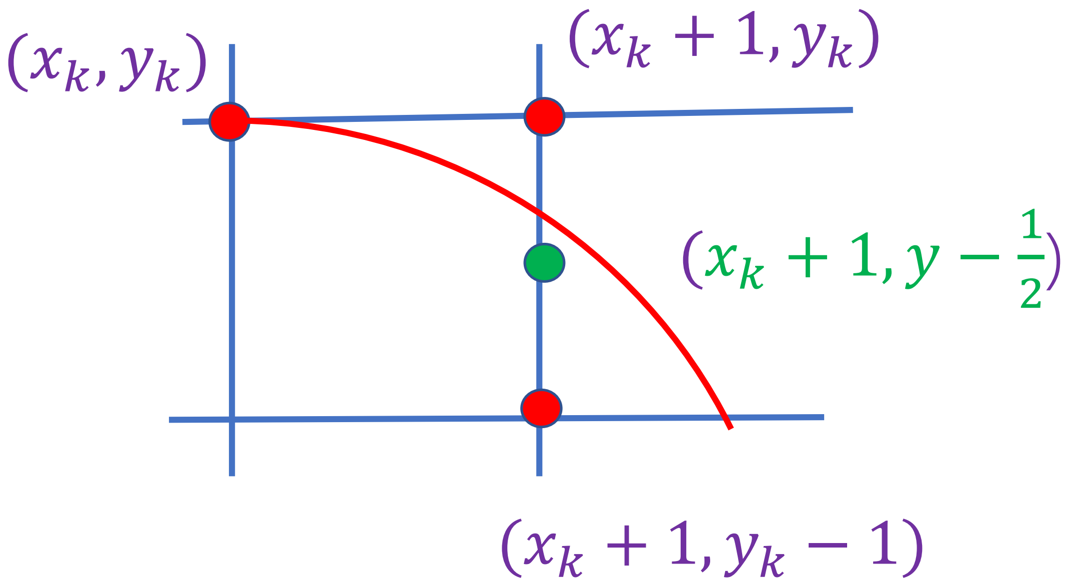
  - 算法（略，参见中点算法）
  - 特点：无浮点运算

三、圆的生成算法

- 3、中点算法
  - 思路：如果中点（$x_k+1, y_k-\frac{1}{2}$）在圆内，

    则： $y_{k+1}=y_k$

    否则： $y_{k+1}=y_k$-1



三、圆的生成算法

- 3、中点算法

- 判别参数p_k的定义

定义: 圆函数：$f_{circle}(x, y) = x^2 + y^2 - R^2$

$$f_{circle}(x, y) = \begin{cases} < 0 & (x, y)\text{在圆内} \\ = 0 & (x, y)\text{在圆上} \\ > 0 & (x, y)\text{在圆外} \end{cases}$$

定义判别参数为中点的圆函数值：$p_k = f_{circle}\left(x_k + 1, y_k - \frac{1}{2}\right)$

$$\begin{cases} \text{如果}p_k < 0\text{，中点在圆内，} & y_{k+1} = y_k \\ \text{否则，中点在圆外，} & y_{k+1} = y_k - 1 \end{cases}$$

三、圆的生成算法

- 3、中点算法
  - 判别参数p_k的计算：

$$p_k = f_{circle}\left(x_k + 1, y_k - \frac{1}{2}\right) = (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - R^2$$

$$p_{k+1} = f_{circle}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right) = (x_k + 2)^2 + (y_{k+1} - \frac{1}{2})^2 - R^2$$

$$p_{k+1} - p_k = \begin{cases} 2x_k + 3 & p_k < 0\text{，中点圆内，} y_{k+1} = y_k \\ 2x_k - 2y_k + 5 & else\text{，中点在圆外，} y_{k+1} = y_k - 1 \end{cases}$$

初始点（0，R）

$$p_0 = f_{circle}\,(1, R - \frac{1}{2}) = \frac{5}{4} - R \approx 1 - R$$

三、圆的生成算法

- 3、中点算法

  - 算法

  - 特点：无浮点运算

step1：设置（x_0, y_0）=（0，R）

step2：设置$k = 0$; $p_0 = 1 - R$

step3：对于任意的k,

if $p_k < 0$，setpixel(x_k+1，y_k)及其7个对称点,并且计算：

$$p_{k+1} = p_k + 2x_k + 3;$$

x_{k+1}= x_k+1，y_{k+1}= y_k;

else setpixel(x_k+1，$y_k - 1$)及其7个对称点，并且计算：

$$p_{k+1} = p_k + 2x_k - 2y_k + 5;$$

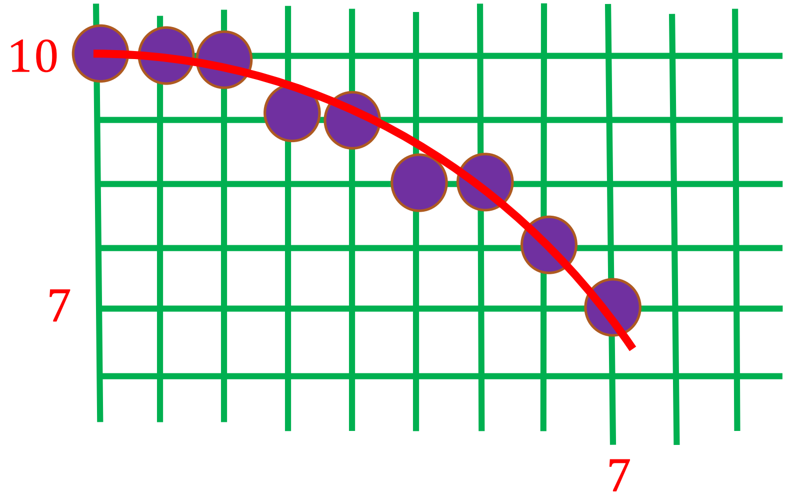
x_{k+1}= x_k+1，y_{k+1}= $y_k - 1$;

step4：$k = k + 1$,回到step3,直到 x_k= y_k。

三、圆的生成算法

- 3、中点算法
  - 例子：生成半径为10的圆

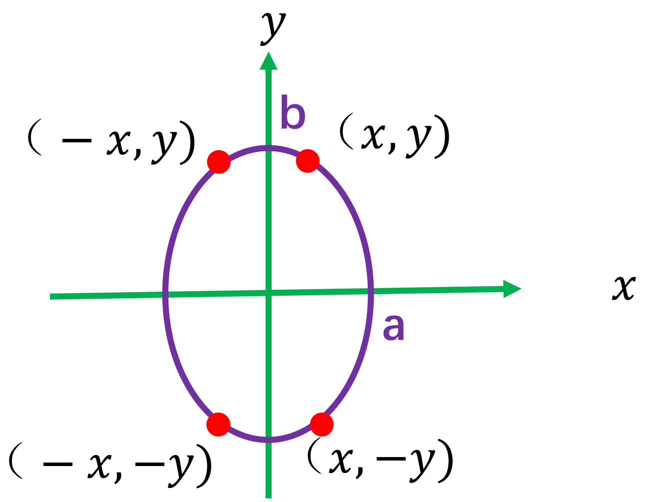
k	p_k	x_{k+1}	y_{k+1}
0	-9	1	10
1	-6	2	10
2	-1	3	10
3	**6**	4	**9**
4	-3	5	9
5	**8**	6	**8**
6	**5**	7	**7**



四、椭圆的生成算法

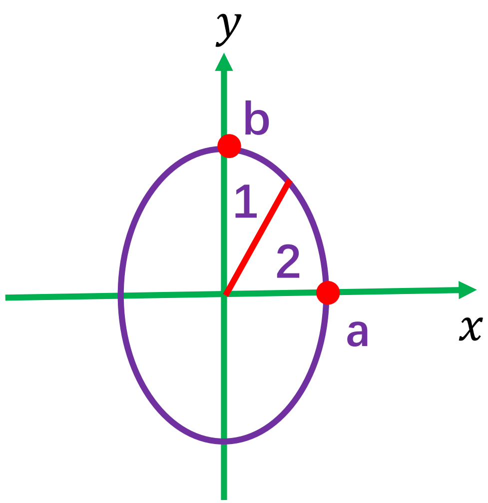
- 1、数学描述
  - 已知中心（0，0），长半轴a，短半轴为b，绘制一个椭圆：

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$$

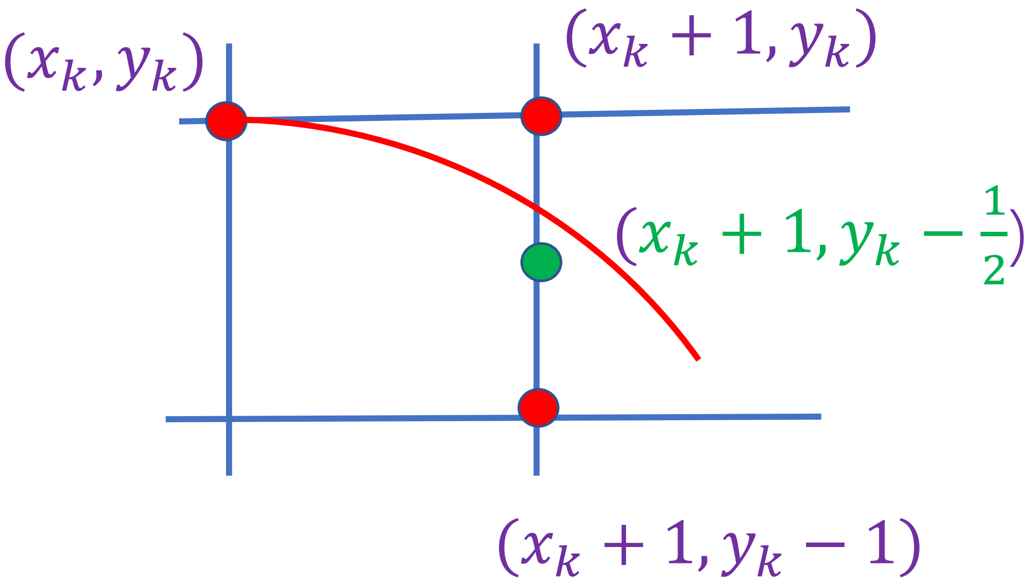


  - 四分之一对称

四、椭圆的生成算法



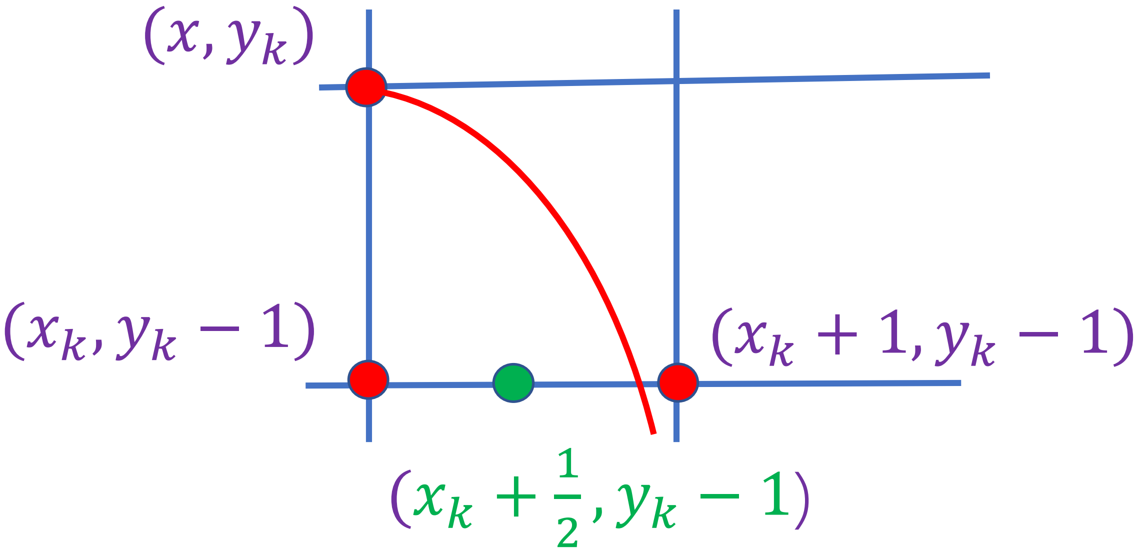
- 2、中点算法
  - 思路：将第一象限的四分之一椭圆分成两个区域
    - 在区域1内有|m| <1， $x_{k+1}=x_k+1$，y_{k+1}只有两种可能，减1或者不变，即：



$$y_{k+1}=\begin{cases} y_k & \text{中点（}x_k+1, y_k-\frac{1}{2}\text{）在椭圆内} \\ y_k-1 & else \end{cases}$$

四、椭圆的生成算法

- 2、中点算法
  - 在区域2内有$|m|>1$，$y_{k+1}=y_k-1$，x_{k+1}只有两种可能，曾1或者不变，即



$$x_{k+1}=\begin{cases} x_k & 中点（x_k+\frac{1}{2}, y_k-1）在椭圆外 \\ x_k+1 & else \end{cases}$$

$$m=\frac{dy}{dx}=-\frac{2b^2x}{2a^2y}$$

$|m|=1$时，有：$b^2x=a^2y$

四、椭圆的生成算法

- 2、中点算法
  - 区域1判别参数p_k的定义

定义:椭圆函数：$f_{ellipse}(x,y)=b^2x^2+a^2y^2-a^2b^2$

$$f_{ellipse}(x,y)=\begin{cases}<0 & (x,y)\text{在椭圆内}\\ =0 & (x,y)\text{在椭圆上}\\ >0 & (x,y)\text{在椭圆外}\end{cases}$$

定义判别参数为中点的椭圆函数值：$p_k=f_{ellipse}\left(x_k+1,y_k-\frac{1}{2}\right)$

$$\begin{cases}\text{如果}p_k<0\text{，中点在椭圆内，} & y_{k+1}=y_k\\ \text{否则，中点在椭圆外，} & y_{k+1}=y_k-1\end{cases}$$

四、椭圆的生成算法

- 3、中点算法

  - 区域1判别参数p_k的计算：

$$p_k = f_{ellipse}\left(x_k + 1, y_k - \frac{1}{2}\right) = b^2(x_k + 1)^2 + a^2(y_k - \frac{1}{2})^2 - a^2b^2$$

$$p_{k+1} = f_{ellipse}\left(x_k + 2, y_{k+1} - \frac{1}{2}\right) = b^2(x_k + 2)^2 + a^2(y_{k+1} - \frac{1}{2})^2 - a^2b^2$$

$$\boxed{\begin{aligned} &p_{k+1} - p_k \\ &= \begin{cases} 2b^2x_k + 3b^2 & p_k < 0\text{，中点椭圆内，} y_{k+1} = y_k \\ 2b^2x_k - 2a^2y_k + 3b^2 + 2a^2 & else\text{，中点在椭圆外，} y_{k+1} = y_k - 1 \end{cases} \end{aligned}}$$

初始点（0，b）：

$$\boxed{p_0 = f_{ellipse}\left(1, b - \frac{1}{2}\right) = b^2 - a^2b + \frac{1}{4}a^2}$$

四、椭圆的生成算法

- 2、中点算法

  - 区域2判别参数q_k的定义

定义:椭圆函数：$f_{ellipse}(x,y)=b^2x^2+a^2y^2-a^2b^2$

$$f_{ellipse}(x,y)=\begin{cases}<0 & (x,y)\text{在椭圆内}\\ =0 & (x,y)\text{在椭圆上}\\ >0 & (x,y)\text{在椭圆下}\end{cases}$$

定义判别参数为中点的椭圆函数值：$q_k=f_{ellipse}\left(x_k+\frac{1}{2},y_k-1\right)$

$$\begin{cases}\text{如果}q_k<0\text{，中点在椭圆内，} & x_{k+1}=x_k+1\\ \text{否则，中点在椭圆外，} & x_{k+1}=x_k\end{cases}$$

四、椭圆的生成算法

- 3、中点算法

  - 区域2判别参数p_k的计算：

$$q_k = f_{ellipse}\left(x_k+\frac{1}{2}, y_k-1\right) = b^2(x_k+\frac{1}{2})^2 + a^2(y_k-1)^2 - a^2b^2$$

$$q_{k+1} = f_{ellipse}\left(x_{k+1}+\frac{1}{2}, y_k-2\right) = b^2(x_{k+1}+\frac{1}{2})^2 + a^2(y_k-2)^2 - a^2b^2$$

$$\boxed{q_{k+1}-q_k = \begin{cases} 2b^2x_k - 2a^2y_k + 2b^2 + 3a^2 & q_k < 0\text{，中点椭圆内，}x_{k+1} = x_k+1 \\ -2a^2y_k + 3a^2 & else\text{，中点在椭圆外，}x_{k+1} = x_k \end{cases}}$$

初始点：区域1的终点，记为(x_0, y_0)

$$\boxed{q_0 = f_{ellipse}\ (x_0+\frac{1}{2}, y_0-1)}$$

四、椭圆的生成算法

- 3、中点算法
  - 算法

step1：设置（x_0, y_0）=（$0, b$）

step2：设置$k = 0$; $p_0 = b^2 - a^2 b - \frac{1}{4} a^2$

step3：对于任意的k，；

if $p_k < 0$，setpixel(x_k+1，y_k)及其4个对称点；

$p_{k+1} = p_k + 2b^2 x_k + 3b^2$；

$y_{k+1} = y_k$；

else setpixel(x_k+1，$y_k - 1$)及其4个对称点；

$p_{k+1} = p_k + 2b^2 x_k - 2a^2 y_k + 3b^2 + 2a^2$；

$y_{k+1} = y_k - 1$；

step4：$k = k + 1$；$x_{k+1} = x_k + 1$；回到step3直到 $b^2 x_{k+1} \geq a^2 y_{k+1}$。

四、椭圆的生成算法

- 3、中点算法
  - 算法

step5：设置（x_0,y_0）=（x_{k+1},y_{k+1}）

step6：设置$k=0$；$q_0=b^2(x_k+\frac{1}{2})^2+a^2(y_k-1)^2-a^2b^2$

step7：对于任意的k，；

if $q_k<0$，setpixel(x_k+1，y_k-1)及其4个对称点；

$q_{k+1}=q_k+2b^2x_k-2a^2y_k+2b^2+3a^2$；

$x_{k+1}=x_k+1$；

else setpixel(x_k，y_k-1)及其4个对称点；

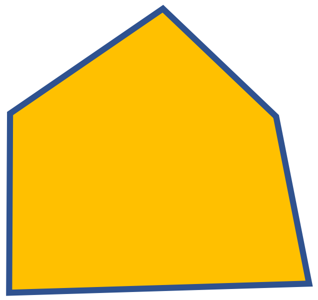
$q_{k+1}=q_k+-2a^2y_k+3a^2$；

$x_{k+1}=x_k$；

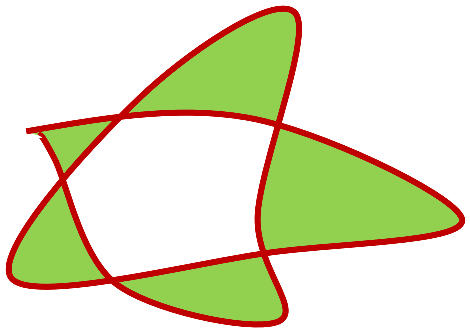
step8：$k=k+1$；$y_{k+1}=y_k-1$；回到step7直到$x_{k+1}=a$。

五、区域填充

- 1、问题描述
  - 区域：一组相邻而又相连的的像素，具有相同的属性。
  - 区域填充：根据边或边界点生成区域的过程；
  - 方法：
    - 扫描转换法：按扫描线顺序确定每个点是否在区域内——边定义的多边形；
    - 种子填充法：已知种子点，遍历区域内与种子点相连的点——边界点定义的多边形



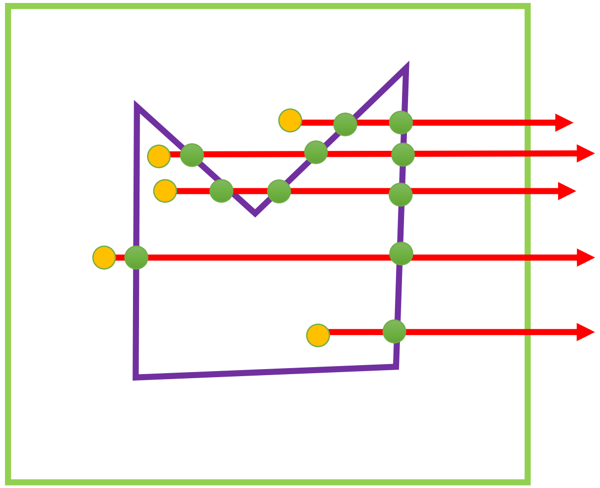
边定义的多边形



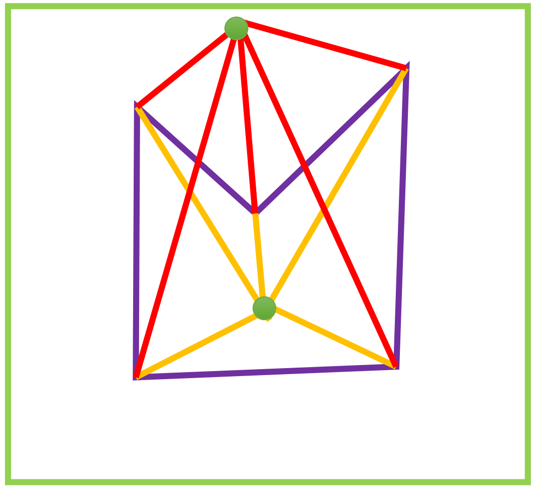
边界点定义的多边形

五、区域填充

- 2、扫描线转换法
  - 简单测试法
    - 射线法：交点数为奇数，被测点在区域内；交点数为偶数，被测点在区域外；
    - 弧长法：$\sum\theta = 2\pi$，被测点在区域内；$\sum\theta = 0$，被测点在区域外；
  - 特点：算法简单，计算量大。



射线法



弧长法

五、区域填充

- 2、扫描线转换法
  - 扫描线算法：求交→排序→填充
    - 思路：空间相关性

同一扫描线上的点有相关性，交点之间直接填充；

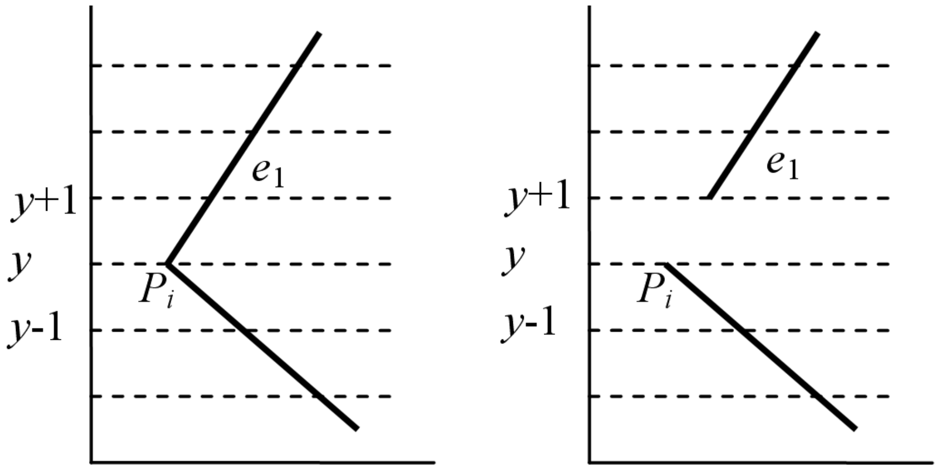
相邻扫描线上的交点有相关性，简化交点求解。

- 奇异点：

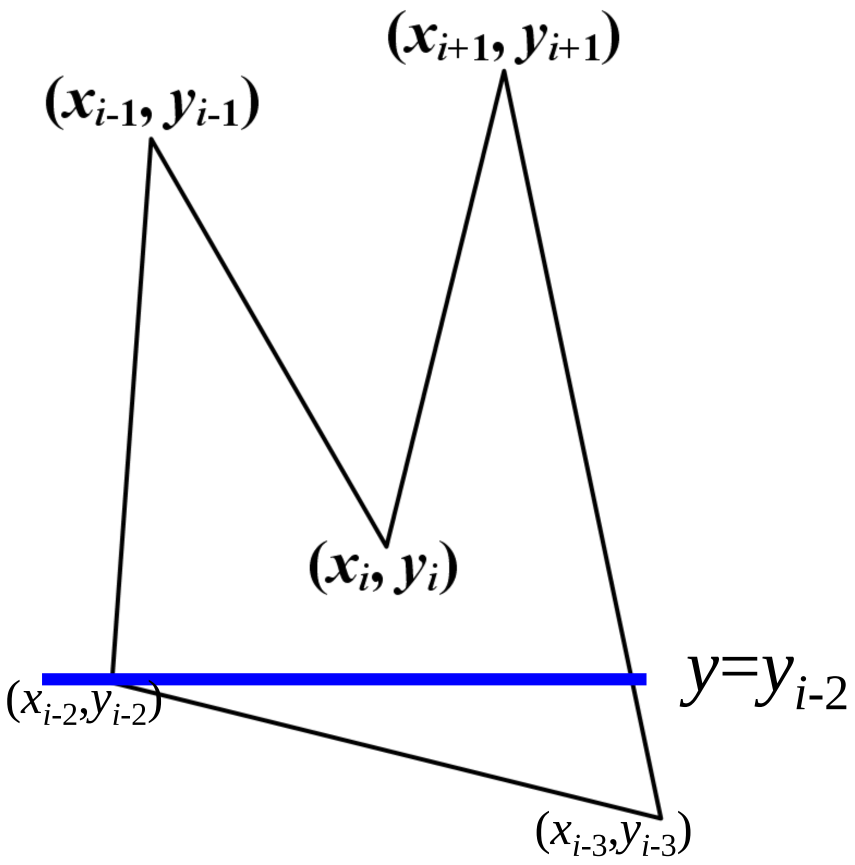
单调变换，交点数为1（跳一格）；

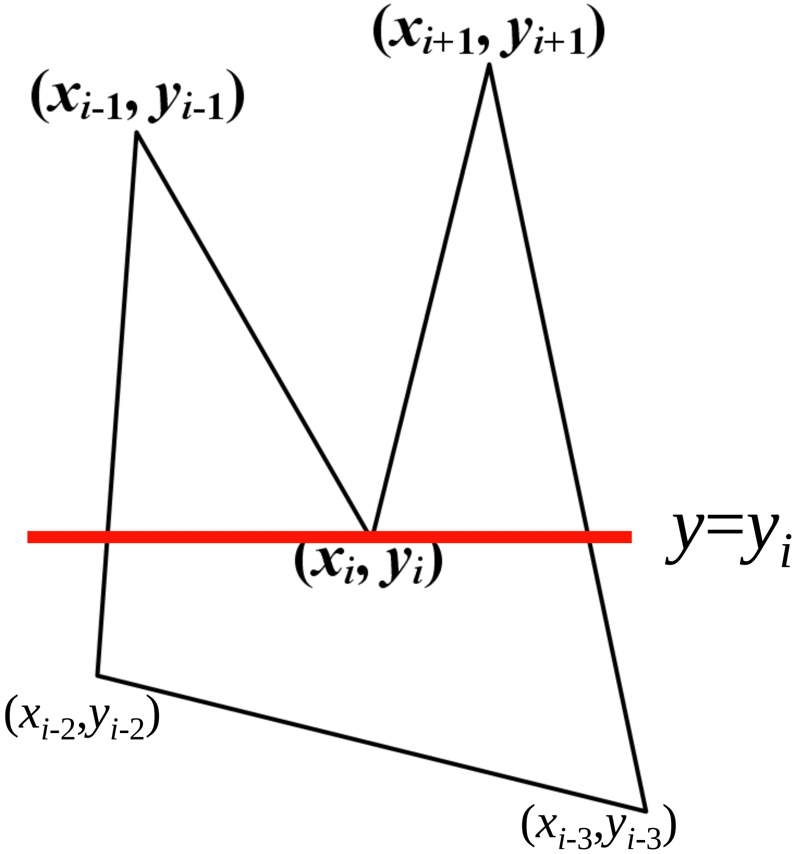
反转：交点数为2；

平行线：交点数为0



跳一格



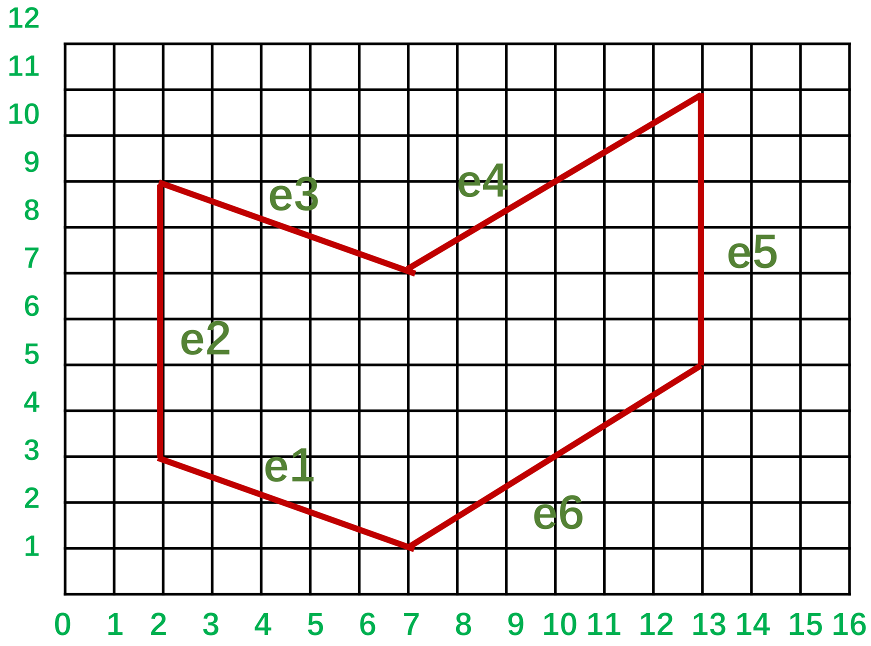


- 单调变换：相邻三个顶点的y坐标满足如下条件：
  $(y_{i-3}-y_{i-2})(y_{i-1}-y_{i-2})<0$
  即相邻三个顶点位于扫描线的两侧
- 反转：相邻三个顶点的y坐标满足如下条件：
  $(y_{i-1}-y_i)(y_{i+1}-y_i)\geq 0$
  即相邻三个顶点位于扫描线的同一侧

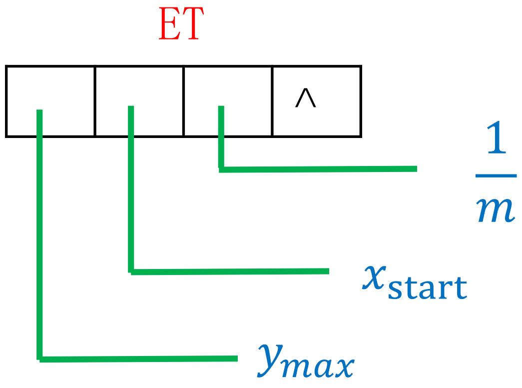
五、区域填充

- 2、扫描线转换法
  - 扫描线算法
    - 算法

step1：建立边表ET（Eage Table），记录每条边的基本信息：y从小到大，第一次遇到一条边的最大y值，x值，斜率的倒数以及一个指针；



eg：填充由顶点：
(7,1)、(2,3)、(2,9)、(7,7)、(13,11)、(13,5)
构成的多边形

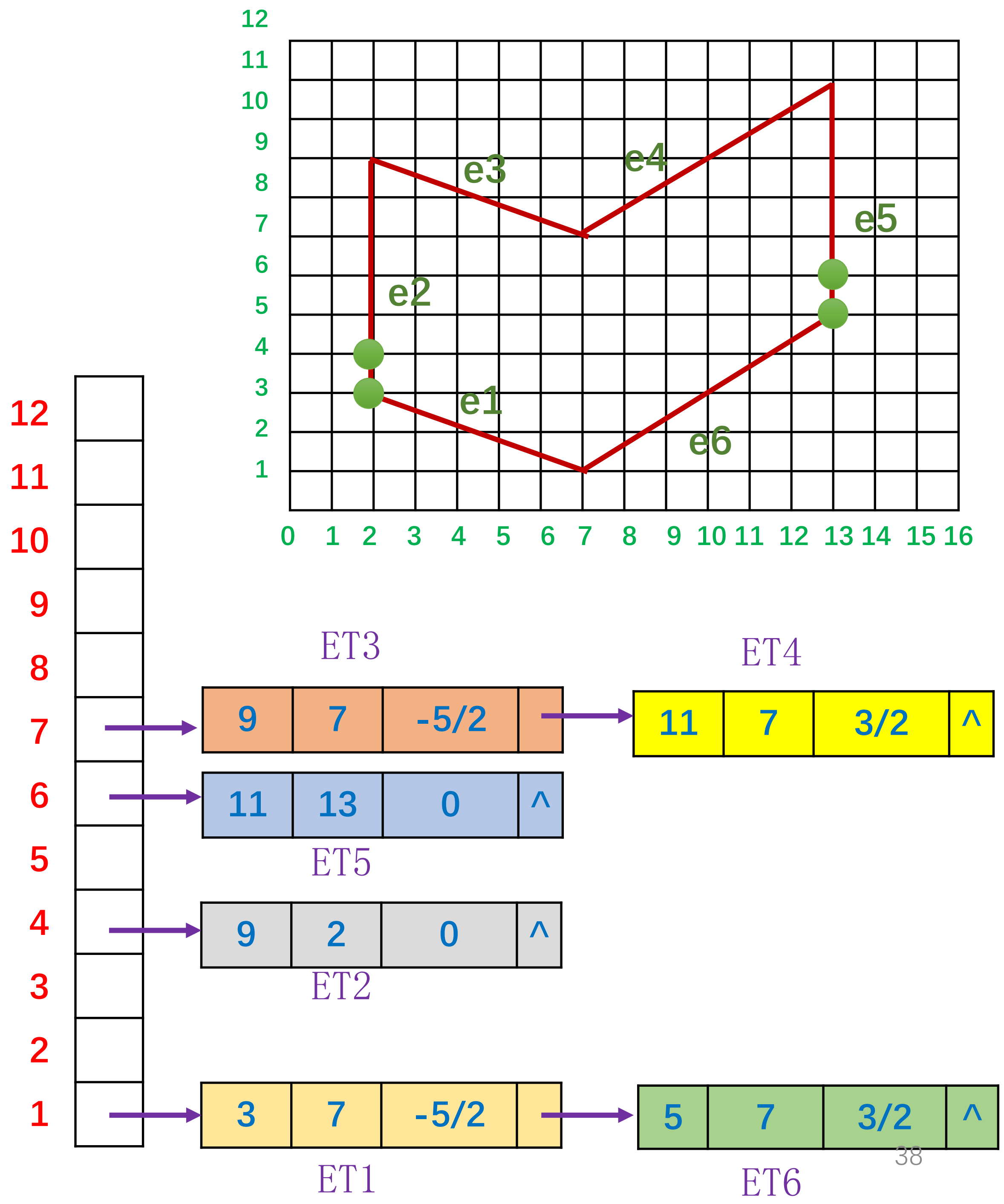


ET1：	3	7	-5/2	^
ET2：	9	2	0	^
ET3：	9	7	-5/2	^
ET4：	11	7	3/2	^
ET5：	11	13	0	^
ET6：	5	7	3/2	^

五、区域填充

- 2、扫描线转换法
  - 扫描线算法
    - 算法

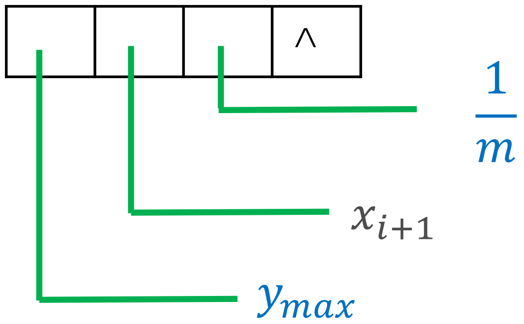
step2：建立有序边表SET（Sorted Eage Table），以Y边从小到大建立一个链表，依次存放第一次出现的边的初始信息；



五、区域填充

- 2、扫描线转换法
  - 扫描线算法
    - 算法

AET



step3：建立活性边表AET（Active Eage Table）：

当$y = y_{i+1}$时，$x_{i+1}=x_i+\frac{1}{m}$;

当$y < y_{max}$时，保持该边，更新x_i值;

否则，移除该边。

Step4：填充AET中相邻两条边的x_{i+1}之间的点。

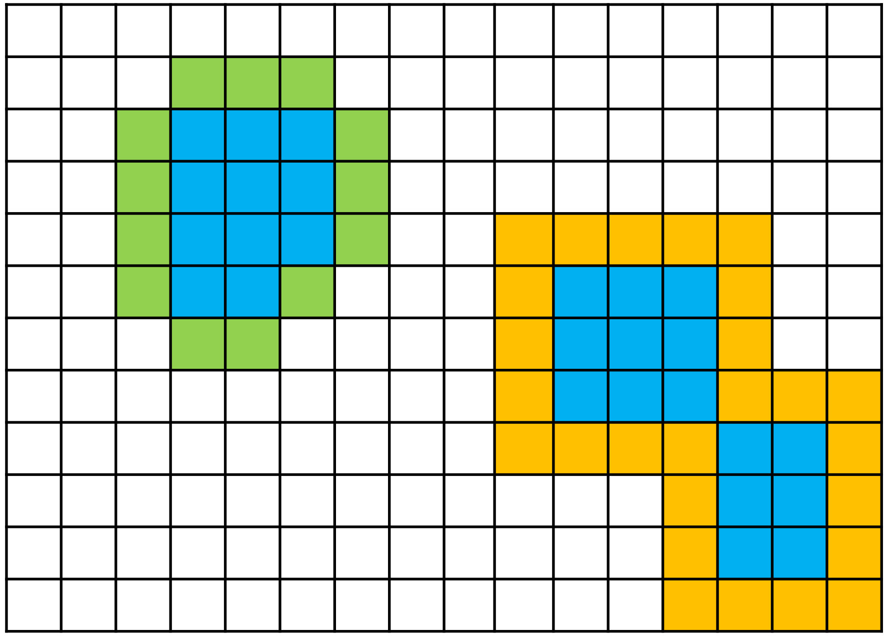
AET中的边数总是偶数

	y_{max}	x_{i+1}	$\frac{1}{m}$		→	y_{max}	x_{i+1}	$\frac{1}{m}$		→					→				
y=1	3	7	-5/2	^	→	5	7	3/2	^										
y=2	3	9/2	-5/2	^	→	5	17/2	3/2	^										
y=3	3	2	-5/2	^	→	5	10	3/2	^										
y=4	9	2	0	^	→	5	23/2	3/2	^										
y=5	9	2	0	^	→	5	13	3/2	^										
y=6	9	2	0	^	→	11	13	0	^										
y=7	9	2	0	^	→	9	7	-5/2	^	→	11	7	3/2	^	→	11	13	0	^
y=8	9	2	0	^	→	9	9/2	-5/2	^	→	11	17/2	3/2	^	→	11	13	0	^
y=9	9	2	0	^	→	9	2	-5/2	^	→	11	10	3/2	^	→	11	13	0	^
y=10	11	23/2	3/2	^	→	11	13	0	^										
y=11	11	13	3/2	^	→	11	13	0	^										

ET1	3	7	-5/2	^	
ET2	9	2	0	^	
ET3	9	7	-5/2	^	
ET4	11	7	3/2	^	
ET5	11	13	0	^	
ET6	5	7	3/2	^	

五、区域填充

- 3、种子点填充算法
  - 边界复杂的情况下，用交互方法确定种子点。如果边界是八连通的，用四方向法；如果边界是四连通的，用八方向法。



五、区域填充

- 3、种子点填充算法
  - 边界填充算法（四连通）；

step1：种子点如栈；

step2：如果栈非空，

栈顶像素出栈；

将出栈像素设置填充色；

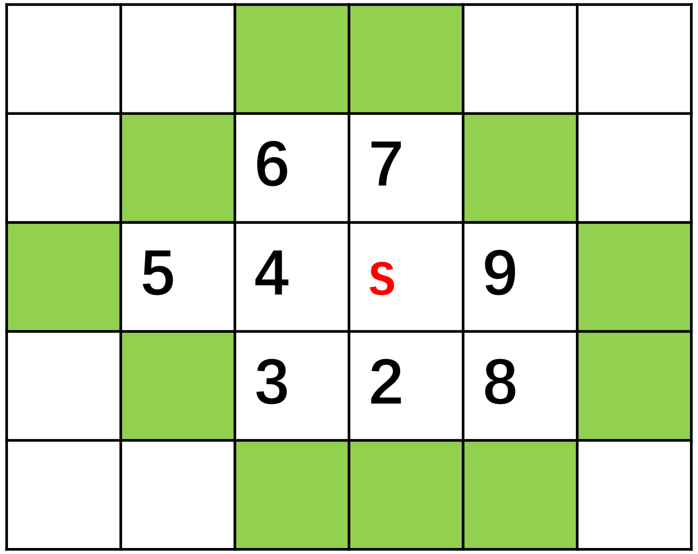
按右上左下顺序检查四邻域；

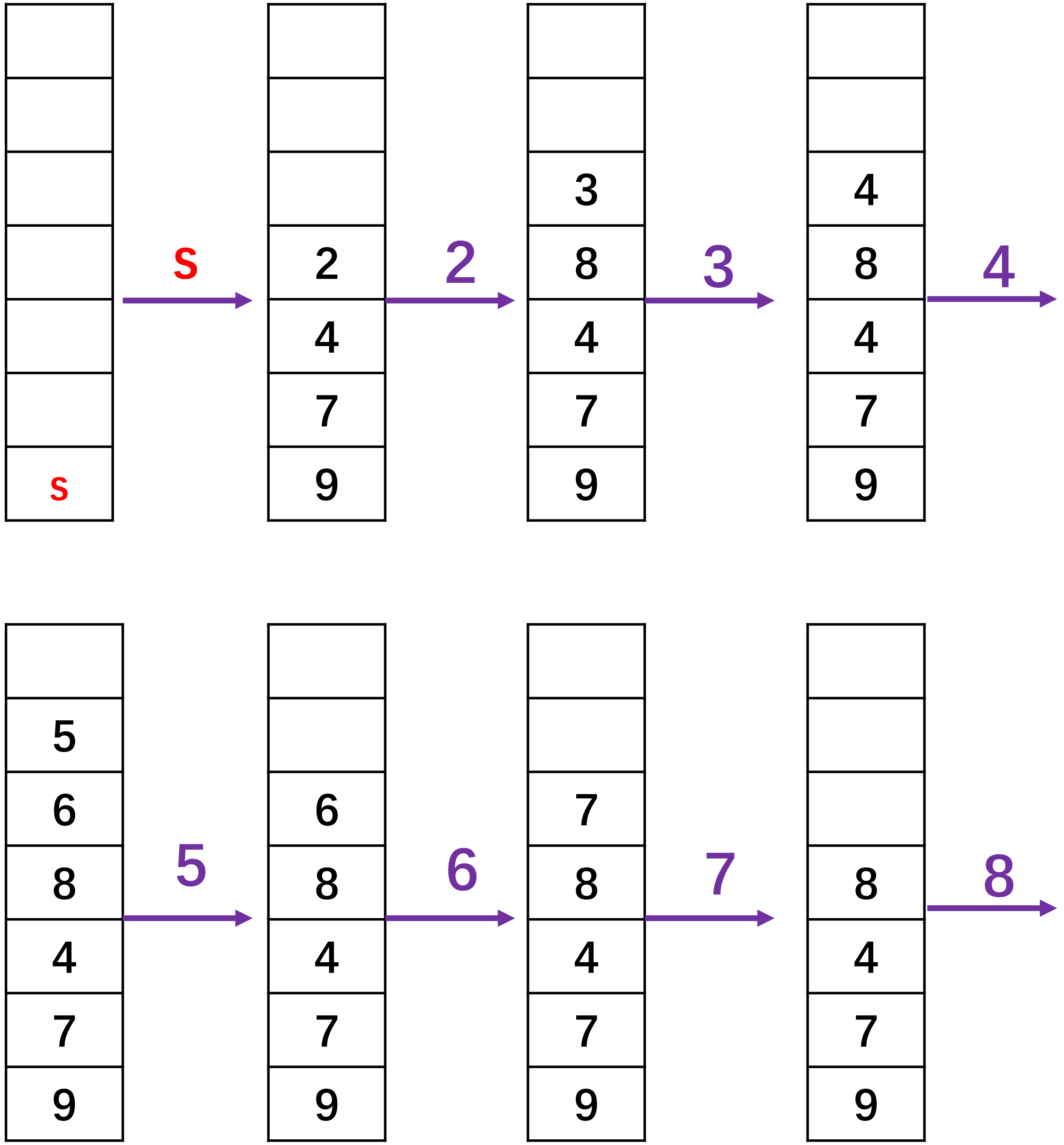
不在边界上且未填色，则入栈。

steps：栈空结束。

五、区域填充

- 3、种子点填充算法
  - 边界填充算法（四连通）；





五、区域填充

- 3、种子点填充算法
  - 扫描线种子点填充（漫水法）；

step1：种子点如栈；

Step2：如果栈非空，

栈顶像素出栈；

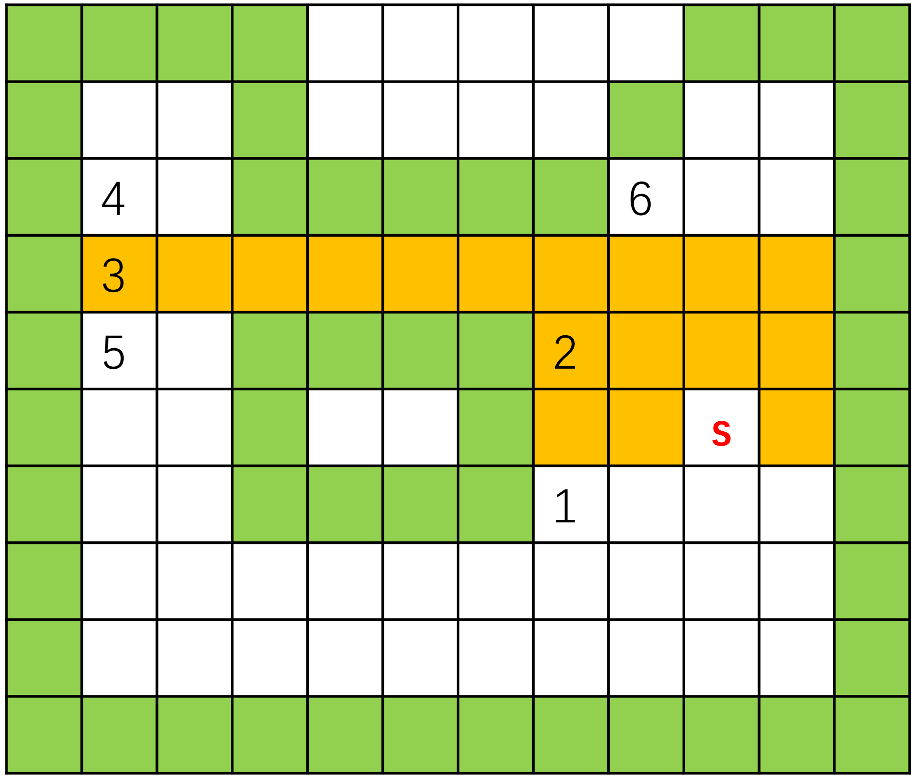
栈顶像素沿扫描线向左右填充至边界；

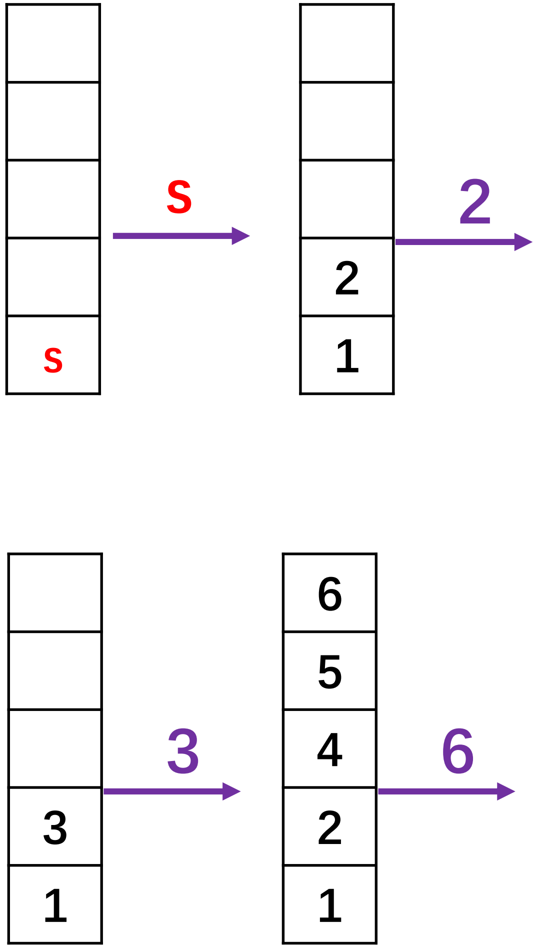
上下两行每区段最左边未填充像素入栈。

steps：栈空结束。

五、区域填充

- 3、种子点填充算法
  - 扫描线种子点填充（漫水法）

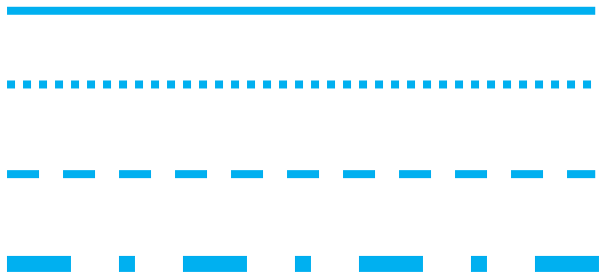




扫地机器人路线

六、线型和线宽

- 1、线型



线型

1	0

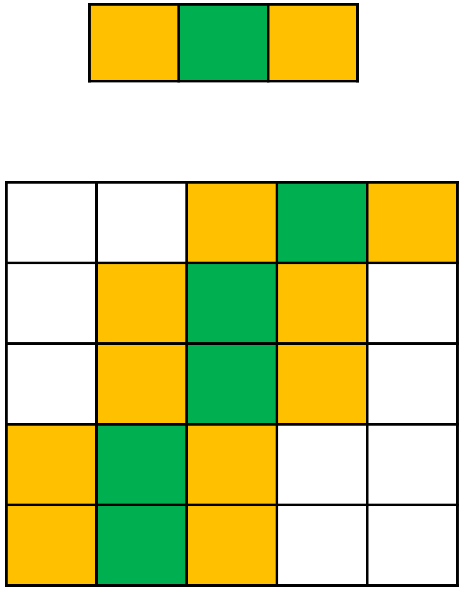
1	1	0

1	1	1	0	0	1	0	0

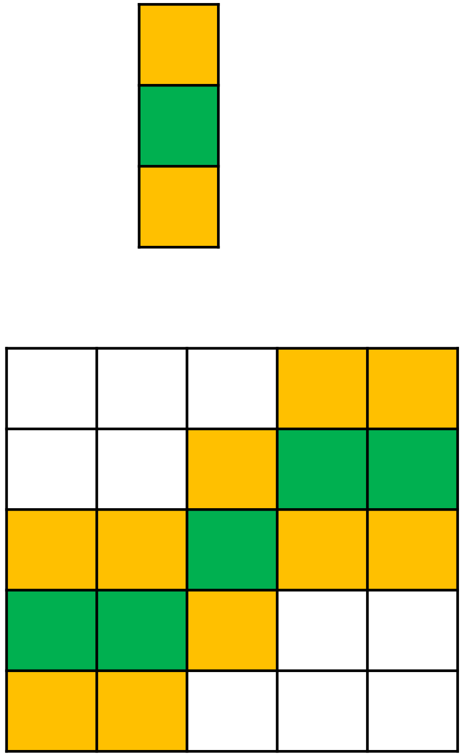
位串

六、线型和线宽

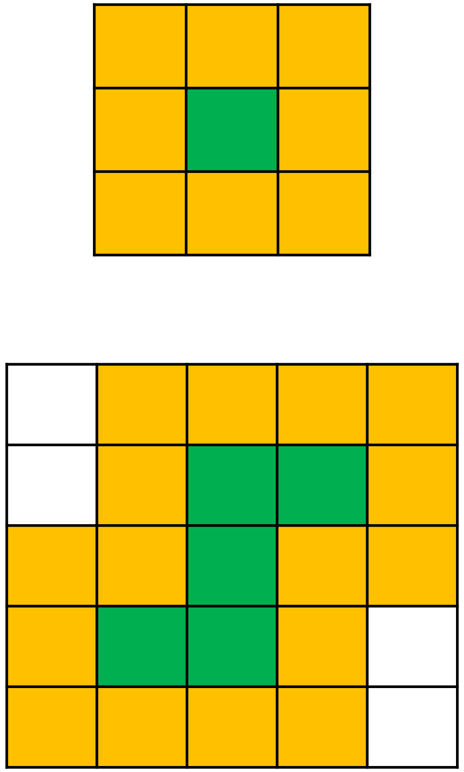
- 2、线宽



|m| >1 水平刷



|m| <1 垂直刷

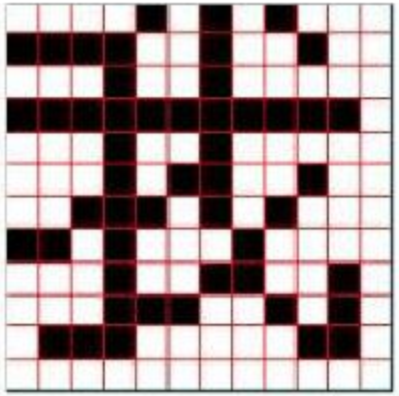


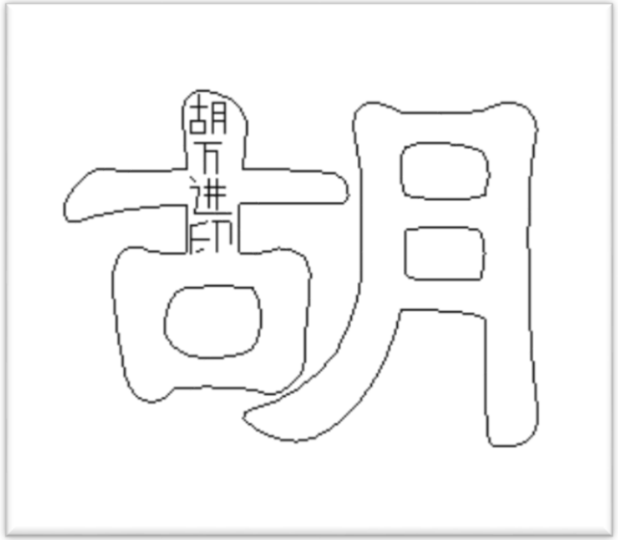
方型刷

七、字符生成

前 5 行：

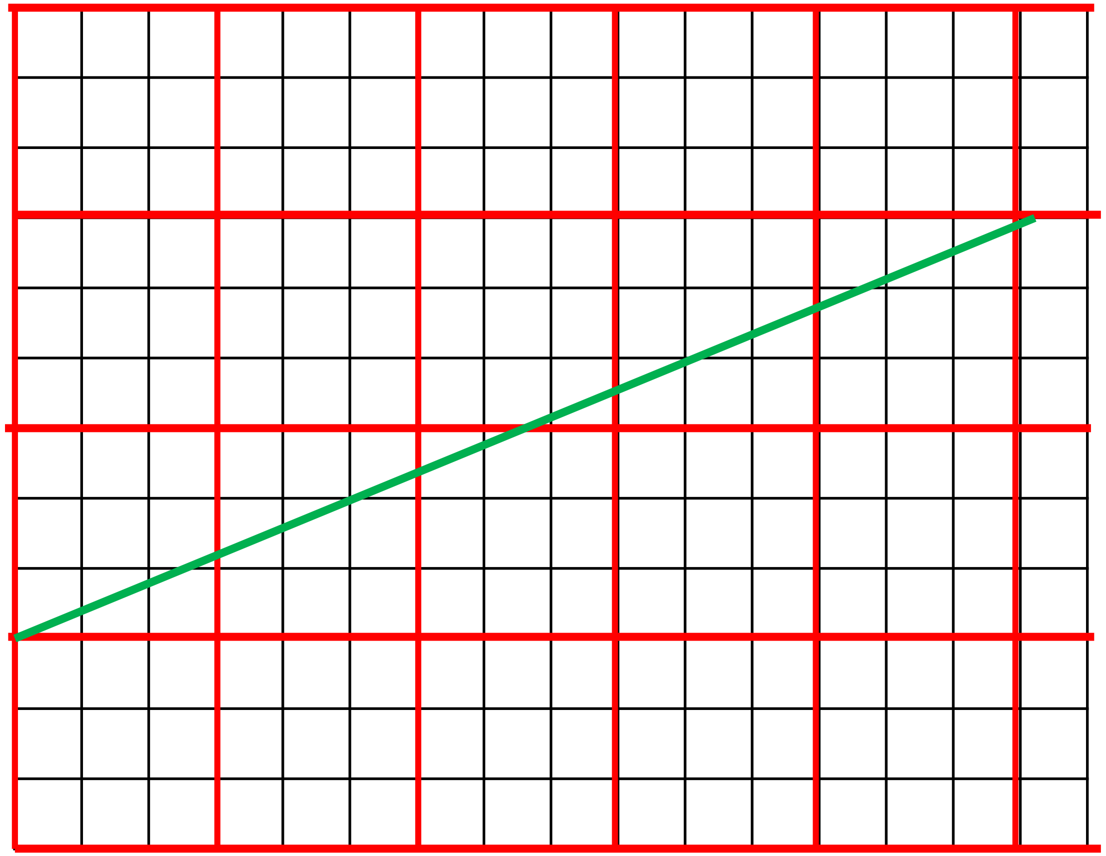
1：00001010 1000 → 0x0a 0x80
2：11110010 0100 → 0xf2 0x40
3：00010010 0000 → 0x12 0x00
4：11111111 1110 → 0xff 0xe0
5：00010010 0000 → 0x12 0x00





八、反走样

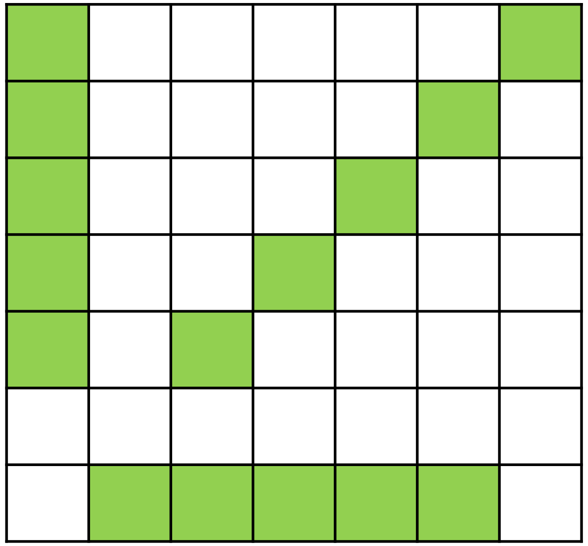
- 1、过取样:
  - 把每一个像素分成若干子像素，在子像素层绘制图形，再统计每个像素中图形经过的子像素数目确定该像素的亮度



八、反走样

- 2、子像素加权

1	2	1
2	4	2
1	2	1



- 3、滤波
- 4、线亮校正

The End