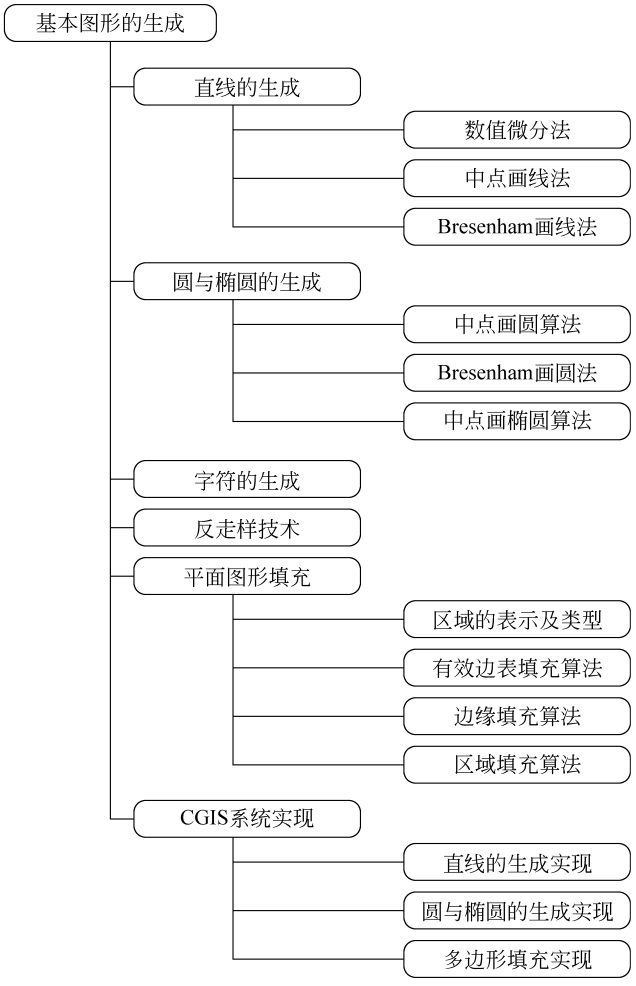
第 3 章　基本图形的生成与显示

内容结构



学习重点

- 直线生成的数值微分法、中点画线法及 Bresenham 画线法的算法描述；
- 多边形填充的有效边表填充算法、边缘填充算法及区域填充算法描述；
- 直线的生成及多边形填充算法的系统实现。

目标要求

- 知识目标：熟练掌握直线生成的几种算法描述，掌握有效边表填充算法和区域填充算法，理解栅栏填充算法；
- 技能目标：熟练案例系统的操作流程，掌握直线的生成算法和多边形填充算法的

VC 实现技术；

- 拓展目标：小组合作，完成本章的相关探究任务。

计算机图形学中首先要解决的问题是基本图形的生成与显示，包括图形基本元素（图元）的扫描转换，如直线、圆弧与椭圆的生成，图元属性如线宽的生成，二维图形的填充，字符的表示及输入输出等。对图形的扫描转换一般分为两步：先确定最佳接近图形的像素集，然后用指定图形的颜色或其他属性写像素。这个过程就称为扫描转换或光栅化。对于一维图形，在不考虑线宽时，用 1 个像素宽的直/曲线（像素序列）来显示图形；二维图形的光栅化，必须确定区域所对应的像素集，并用指定的属性或图案来显示，即区域填充。

复杂的图形系统，都是由一些最基本的图形元素组成的，点是最基本的图元。本章主要介绍如何在指定的输出设备上利用点构造其他二维几何图形（如直线、圆、椭圆、多边形及字符等）的算法原理及实现方法。

3.1 直线的生成

直线的扫描转换就是在屏幕像素点中用指定颜色点最佳逼近于理想直线的像素点集的过程。用计算机绘制三维立体图形时，首先要将三维立体图形投影到二维平面上，而绘制二维图形时要用到大量的直线段，绘制曲线和各种复杂的图形时也要用一组短小的直线来逼近。因此，直线的生成算法是图形生成技术的基础。由于一幅图中有可能包含成千上万条直线，所以要求绘制直线的算法应尽可能快，即尽量使用加减法实现，避免乘、除、开方和三角等复杂运算。

在图形设备上输出一条直线，是通过在应用程序中对每一条直线端点坐标进行描述，由输出设备将一对端点间的路径加以描绘实现的，如图 3-1 所示。对于水平线或垂直线，只要有了驱动设备使之动作的指令，就能准确地画出。但是对于任意斜率的直线，就要考虑算法了。因为大多数图形设备都只提供驱动 x 方向和 y 方向动作的信号。

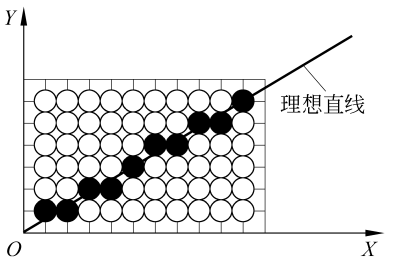


图 3-1 直线的扫描转换

在实际绘图时，有时只要绘制单像素宽的直线，有时则需要绘制以理想直线为中心的不同线宽的直线，还需要用不同颜色和线型来画线。本节介绍一个像素宽直线的常用算法：数值微分算法、中点画线算法、Bresenham 画线算法。

3.1.1 数值微分法

数值微分算法即 DDA（Digital Differential Analyzer）算法，这是一种基于直线的微分方程来生成直线的方法，即通过对 x 和 y 各增加一个小增量，计算下一步的 x、y 值。

1. 算法的基本原理

设 (x_1, y_1) 和 (x_2, y_2) 分别为理想直线的端点坐标，直线的方程为 $y=kx+b$，由直线的

微分方程得$\frac{dy}{dx}=k$(直线的斜率),即

$$\frac{\Delta y}{\Delta x}=\frac{y_2-y_1}{x_2-x_1} \tag{3-1}$$

可通过计算由 x 方向的增量 Δx 引起 y 的改变来生成直线。由 $y_{i+1}=y_i+\Delta y$(y_i 为直线上某步的初值),则有:

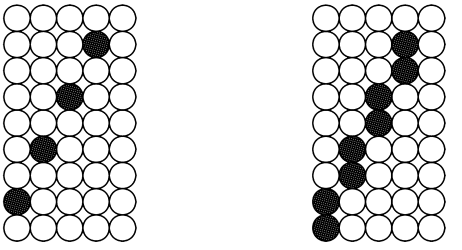
$$y_{i+1}=y_i+\frac{y_2-y_1}{x_2-x_1}\cdot\Delta x \tag{3-2}$$

也可通过计算由 y 方向的增量 Δy 引起 x 的改变来生成直线。若设 $x_{i+1}=x_i+\Delta x$,则由式(3-1)可得:

$$x_{i+1}=x_i+\frac{x_2-x_1}{y_2-y_1}\cdot\Delta y \tag{3-3}$$

式(3-2)和式(3-3)是递推的。

本算法的基本思想是:选定$|x_2-x_1|$和$|y_2-y_1|$中较大者作为步进方向,假设$|x_2-x_1|$较大,取该方向上的 Δx 为一个像素单位长,即每次递增一个像素,然后利用式(3-2)计算相应的 y 值,把每次计算出的(x_{i+1},y_{i+1})经取整后顺序输出到显示器,则得到扫描转换后的直线。之所以取$|x_2-x_1|$和$|y_2-y_1|$中较大者为步进方向,看图 3-2 就会明白,图中直线$|y_2-y_1|$步进方向较大,而图 3-2(a)取$|x_2-x_1|$为步进方向,导致像素点分布太散。



(a) 取Δx为一个像素单位　　(b) 取Δy为一个像素单位

图 3-2　取不同的像素单位

2. 算法实例

【例 3-1】 利用 DDA 算法绘制直线 P_1P_2,其端点坐标为 $P_1(0,0)$和 $P_2(-3,5)$。

【解】 因为$|y_2-y_1|=5$,$|x_2-x_1|=3$,所以取 y 方向为步进方向,直线的斜率 $k=-5/3$,当 $\Delta y=1$ 时,Δx 为 $1/k$ 即 $-3/5$,即 x 方向上递增 $-3/5$。递推公式为 $x_{i+1}-x_i=-3/5$,为了取得整数像素点,对所求得的 x 值进行四舍五入取整来修正。由此,计算出直线各点像素值如图 3-3(a)所示,生成的直线坐标点如图 3-3(b)所示。

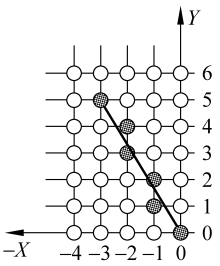
3.1.2　中点画线法

1. 算法原理

为了讨论方便,这里假定直线斜率 $k\in[0,1]$,其他情况可参照以下讨论进行处理。中点画线算法的基本原理如图 3-4 所示,若直线在 x 方向上增加一个单位,则在 y 方向上的增量只能在 0～1 之间。假设 x 坐标为 x_p 的各像素点中,与直线最近者已经确定为 $P(x_i,y_i)$,用小实心圆表示。那么下一个与直线最近的像素只能是正右方的 $P_d(x_i+1,y_i)$,或右

x	y	x四舍五入取整
0	0	0
−0.6	1	−1
−1.2	2	−1
−1.8	3	−2
−2.4	4	−2
−3.0	5	−3

(a) 各像素点的值



(b) 直线的各坐标点

图 3-3　DDA 算法画直线实例

上方的 $P_u(x_i+1, y_i+1)$，用小空心圆表示。以 M 表示 P_d 和 P_u 的中点，则 M 的坐标为 $(x_i+1, y_i+0.5)$。又假设 Q 是理想直线与垂直线 $x=x_i+1$ 的交点。显然，若 M 在 Q 的下方，则 P_u 离直线近，下一像素点应点亮 P_u；若 M 在 Q 的上方，则 P_d 离直线近，下一像素点应点亮 P_d。

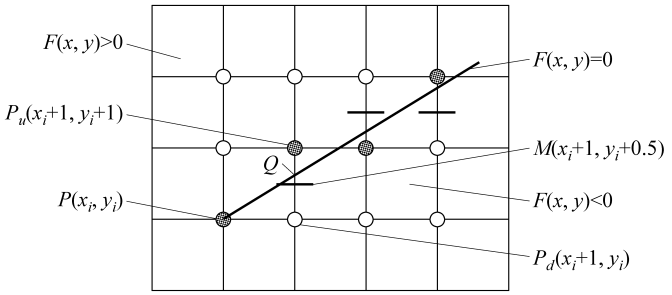


图 3-4　中点画线算法原理

2. 算法公式推导

设 $P_1(x_1, y_1)$ 和 $P_2(x_2, y_2)$ 分别为理想直线上的两点坐标，直线的隐函数方程为

$$F(x,y) = y - kx - b = 0 \tag{3-4}$$

其中，直线的斜率 $k=\dfrac{\Delta y}{\Delta x}=\dfrac{y_2-y_1}{x_2-x_1}$，$\Delta x=x_2-x_1$ 为直线水平方向位移，$\Delta y=y_2-y_1$ 为直线垂直方向位移。如图 3-4 所示，理想直线将平面划成 3 个区域：对于直线上的点，$F(x,y)=0$；对于直线上方的点，$F(x,y)>0$；对于直线下方的点，$F(x,y)<0$。假设直线的斜率为 $0\leqslant k\leqslant 1$，则 $|\Delta x|\geqslant|\Delta y|$，所以确定 x 方向为主位移方向。按照中点画线算法原理，x 方向上每次加 1，y 方向上加不加 1，只要把 M 的坐标代入 $F(x,y)$ 中，通过判断 $F(x,y)$ 的符号来决定。

1) 构造中点判别式

从 $P(x_i, y_i)$ 点走第一步后，为了进行下一像素点的选取，需将 P_u 和 P_d 的中点 $M(x_i+1, y_i+0.5)$ 代入隐函数方程式(3-4)，构造中点判别式 d：

$$d = F(x_M, y_M) = F(x_i+1, y_i+0.5) = y_i + 0.5 - k(x_i+b) - b \tag{3-5}$$

当 $d<0$ 时，中点 M 在直线的下方，P_u 点离直线距离近，下一像素点应点亮 P_u，即 y 方向上走一步；当 $d>0$ 时，中点 M 在直线的上方，P_d 点离直线距离近，下一像素点应点亮 P_d，即 y 方向上不走步；当 $d=0$ 时，中点 M 在直线上，P_u、P_d 与直线的距离相等，点亮 P_u

或 P_d 均可，约定取 x 方向的 P_d，如图 3-4 所示。

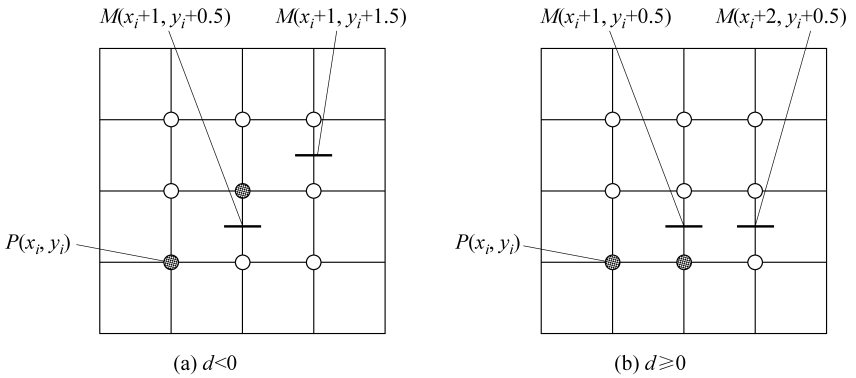
因此有：

$$y_{i+1}=\begin{cases}y_i+1, & d<0\\ y_i, & d\geqslant 0\end{cases} \tag{3-6}$$

2）中点判别式的递推公式

在图 3-4 中，根据当前点 $P(x_i,y_i)$确定下一点是点亮 P_u 还是点亮 P_d 时，使用了中点判别式 d。为了能够继续判断直线上的每一个点，需要给出中点判别式的递推公式及其初始值。

在主位移 x 方向上已走一步的情况下，考虑沿主位移方向再走一步，应该选择哪个中点代入中点判别式来决定下一步该点亮的像素，如图 3-5 所示，分两种情况讨论。



(a) $d<0$　　(b) $d\geqslant 0$

图 3-5　中点判别式的递推

① 当 $d<0$ 时，下一步进行判断的中点坐标为 $M(x_i+2,y_i+1.5)$，如图 3-5(a)所示。所以下一步中点判别式为

$$\begin{aligned}d_{i+1}&=F(x_i+2,y_i+1.5)=y_i+1.5-k(x_i+2)-b\\&=y_i+0.5-k(x_i+1)-b+1-k=d_i+1-k\end{aligned} \tag{3-7}$$

② 当 $d\geqslant 0$ 时，下一步的中点坐标为 $M(x_i+2,y_i+0.5)$，如图 3-5(b)所示。所以下一步中点判别式为

$$\begin{aligned}d_{i+1}&=F(x_i+2,y_i+0.5)=y_i+0.5-k(x_i+2)-b\\&=y_i+0.5-k(x_i+1)-b-k=d_i-k\end{aligned} \tag{3-8}$$

3）中点判别式的初始值

设直线的起点坐标为 $P_0(x_0,y_0)$，x 为主位移方向。因此，第一个中点是$(x_0+1,y_0+0.5)$，相应的 d 的初始值为

$$\begin{aligned}d_0&=F(x_0+1,y_0+0.5)=y_0+0.5-k(x_0+1)-b\\&=y_0-kx_0-b-k+0.5\end{aligned}$$

其中，因为(x_0,y_0)在直线上，所以 $y_0-kx_0-b=0$，则：

$$d_0=0.5-k \tag{3-9}$$

3. 算法实例

【例 3-2】 利用中点画线算法绘制直线 P_1P_2，其端点坐标为 $P_1(0,0)$和 $P_2(5,3)$。

【解】 首先计算斜率 $k=0.6$，$d_0=0.5-0.6=-0.1$。

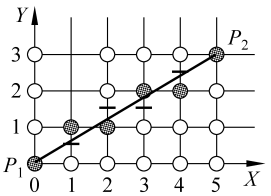
当 $d<0$ 时，$d_{i+1}=d_i+1-k=d_i+0.4$；

当 $d\geqslant 0$ 时，$d_{i+1}=d_i-k=d_i-0.6$。

先绘制初始点(0,0)，按照中点画线算法原理得出其余各像素点的坐标如图 3-6(a)所示，生成的直线如图 3-6(b)所示。

x	y	d
0	0	−0.1
1	1	0.3
2	1	−0.3
3	2	0.1
4	2	−0.5
5	3	−0.1

(a) 各像素点值



(b) 直线坐标点

图 3-6　中点画线算法实例

3.1.3　Bresenham 画线法

Bresenham 画线算法是计算机图形学领域使用最广泛的直线生成算法。该算法是为数字绘图仪设计的，由于它也适用于光栅图形显示器，所以后来被广泛应用于直线的扫描转换及其他一些应用中。与中点画线算法相似，该算法由误差项符号决定下一像素取正右方点还是右上方点。

1. 算法原理

为了讨论方便，这里仍假定直线斜率 $k\in[0,1]$，其他情况可参照以下讨论进行处理。Bresenham 画线算法的原理如图 3-7 所示，过各行各列像素中心构造一组虚拟网格线，按直线从起点到终点的顺序计算直线与各垂直网格线的交点，然后确定该列像素中与此交点最近的像素。如果在 X 方向增加一个单位长度，在 Y 方向是否增加一个单位根据计算的误差项 d 来决定。d 为所画直线与网格垂直线的交点到本网格水平底线的距离。

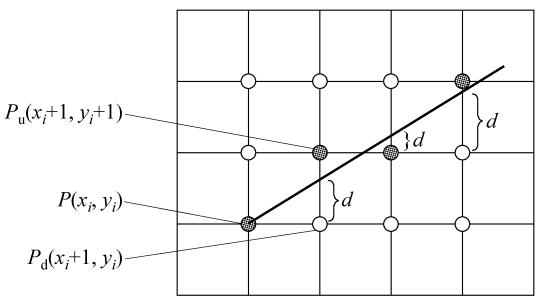


图 3-7　Bresenham 画线算法的原理

2. 算法公式推导

假定直线的当前点是 $P(x_i,y_i)$，直线的斜率为 $0\leqslant k\leqslant 1$，则以 x 方向为主位移方向，如图 3-7 所示，下一点只能在 $P_u(x_{i+1},y_{i+1})$ 和 $P_d(x_{i+1},y_i)$ 两点中选取，为了确定下一像素点的选取，需判断误差项 d 的值，

1) 构造误差判别式

设直线的函数方程为 $y=kx+b$，则当 x 每增加 1，d 的值相应地增加 k(直线的斜率)，

即 $d_{i+1}=d_i+k$。其中 $k=dy/dx=(y_2-y_1)/(x_2-x_1)$，一旦 $d\geqslant 1$ 时，就把它减去 1，这样保持 d 始终在 0～1 之间。当 $d>0.5$ 时，P_u 点离直线距离近，下一像素点应点亮 P_u，即 y 方向上走一步；当 $d<0.5$ 时，P_d 点离直线距离近，下一像素应点亮 P_d，即 y 方向上不走步；当 $d=0.5$ 时，P_u、P_d 与直线的距离相等，点亮 P_u 或 P_d 均可，约定取 P_d。

因此

$$y_{i+1}=\begin{cases}y_i+1, & d>0.5\\ y_i, & d\leqslant 0.5\end{cases} \tag{3-10}$$

令 $e=d-0.5$，则

$$y_{i+1}=\begin{cases}y_i+1, & e>0\\ y_i, & e\leqslant 0\end{cases} \tag{3-11}$$

2）误差判别式的递推公式

在图 3-7 中，根据当前点 $P(x_i,y_i)$确定下一点是点亮 P_u 还是点亮 P_d 时，使用了误差判别式 e。为了能够继续判断直线上的每一个点，需要给出误差判别式的递推公式和初始值。

在主位移 X 方向上已走一步的情况下，考虑沿主位移方向再走一步，应该由下一个误差判别式的值来决定下一步该点亮的像素。这里分两种情况讨论。

① 当 $d_i>0.5$ 时，$d_{i+1}=d_i+k-1$。

② 当 $d_i\leqslant 0.5$ 时，$d_{i+1}=d_i+k$。

令 $e=d-0.5$，则以上两种情况变为

① 当 $e_i>0$ 时，

$$e_{i+1}=e_i+k-1。 \tag{3-12}$$

② 当 $e_i\leqslant 0$ 时，

$$e_{i+1}=e_i+k。 \tag{3-13}$$

3）误差判别式的初始值

设直线的起点坐标为 $P_0(x_0,y_0)$，因为直线的起始点在像素中心，所以 d 的初始值为 $d_0=0$，令 $e=d-0.5$，则 $e_0=-0.5$。

3. 算法的实现

设直线的起点为 $P_1(x_1,y_1)$，终点为 $P_2(x_2,y_2)$，并且该直线的斜率为 $k\in[0,1]$，则进行该直线扫描转换的 Bresenham 算法实现如下：

```
Bresenham_Line(x1,y1,x2,y2,color)
{  float x,y,dx,dy;
   float k,e;
   dx=x2-x1;dy=y2-y1;
   k=dy/dx;
   e=-0.5;x=x1;y=y1;
   for(;x<=x2;x++)
   {    SetPixel(x,y,color);
        e=e+k;
        if(e>0)  {y=y+1;e=e-1;}
   }
}
```

4. 算法的优化

上述 Bresenham 算法在计算直线斜率与误差项时，要用到小数与除法，由于算法中只用到 e 的符号，为了便于硬件计算，可以改用整数以避免除法，因此可将式(3-12)和式(3-13)两端同时乘以 $2dx$，令 $f=2\times e\times dx$，则由 $e_0=-0.5$，得 $f_0=-dx$；式(3-12)和式(3-13)将变成

当 $f_i>0$ 时，

$$f_{i+1}=f_i+2\times dy-2\times dx \tag{3-14}$$

当 $f_i\leqslant 0$ 时，

$$f_{i+1}=f_i+2\times dy \tag{3-15}$$

便可获得整数 Bresenham 算法。以下是 x 步进方向大于 y 步进方向的整数 Bresenham 算法伪代码：

```
INT_Bresenham_Line(x1,y1,x2,y2,color)
{   float x,y,dx,dy;
    float k,f;
    dx=x2-x1;dy=y2-y1;
    f=-dx;x=x1;y=y1;
    for(;x<=x2;x++)
    {   SetPixel(x,y,color);
        f=f+2*dy;
        if(f>0){y=y+1;f=f-2*dx;}
    }
}
```

5. 算法实例

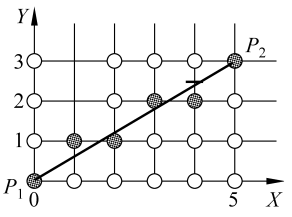
【例 3-3】 利用 Bresenham 算法绘制直线 P_1P_2，其端点坐标为 $P_1(0,0)$和 $P_2(5,3)$。

【解】 按照以上整数 Bresenham 算法计算出各值：

$x_1=0, y_1=0, x_2=5, y_2=3, dx=x_2-x_1=5, dy=y_2-y_1=3-0=3, f_0=-dx=-5$，由上述算法原理可计算出各像素点的坐标如图 3-8(a)所示，对应生成的直线如图 3-8(b)所示，生成图形的对应像素与中点画线算法生成的图形一样。

x	y	f
0	0	1
1	1	−3
2	1	3
3	2	−1
4	2	5
5	3	1

(a) 各像素点值



(b) 直线坐标点

图 3-8　Bresenham 画线算法实例

3 种直线生成算法的比较：DDA 算法直观、易于实现，但是 x、y 和 k 必须用浮点数表示，而且每一步都需要对 x 和 y 进行舍入取整，不利于硬件实现。中点算法可以替换成整数运算，便于硬件实现。Bresenham 算法具有不计算斜率，不用浮点数，只做整数的加减运

算或乘 2 运算，而乘 2 运算可以用移位操作来实现，因此 Bresenham 算法是速度最快的。

3.2 圆与椭圆的生成

为讨论方便，本节只考虑中心在原点，半径为整数 R 的圆 $x^2+y^2=R^2$。对于中心不在原点的圆，可先通过平移变换，转化为中心在原点的圆，再进行扫描转换，把所得的像素坐标加上一个位移量即得像素坐标。

在进行圆的扫描转换时，只要能生成 8 分圆，那么圆的其他部分可通过一系列的简单反射变换得到。如图 3-9 所示，假设已知一个圆心在原点的圆上一点 (x,y)，根据对称可得到另外 7 个 8 分圆上的对应点 (y,x)、$(y,-x)$、$(x,-y)$、$(-x,-y)$、$(-y,-x)$、$(-y,x)$、$(-x,y)$。因此，只需讨论其中一个 8 分圆的扫描转换。

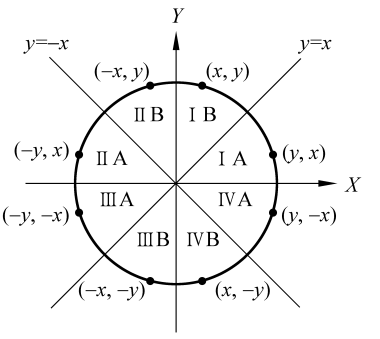


图 3-9 圆的对称性

3.2.1 简单画圆法

算法原理：利用圆的参数方程，直接计算离散点的值。

圆的参数方程为：

$$\begin{cases} x = x_0 + R \cdot \cos\theta \\ y = y_0 + R \cdot \sin\theta \end{cases} \tag{3-16}$$

式中，x_0、y_0 为圆心坐标；R 为圆的半径。

对以上方程进行离散化，设置固定步长，如 θ 角度分成 360 份，步长为 1，则可以生成圆弧。此方法简单，但计算效率不高，参与运算的有小数和函数，不易用硬件实现。

3.2.2 中点画圆法

根据圆的对称性，下面讨论中心在原点，半径为 R 的 1/8 圆弧，即图 3-9 所示的 IB 区域，实现从 $x=0$ 到 $y=x$ 顺时针确定离理想圆弧最近的像素集合。

1. 算法原理

如图 3-10 所示，假设圆弧的当前点是 $P(x_i,y_i)$，那么沿主位移 x 方向走一步，下一像素点只能是正右方的 $P_u(x_i+1,y_i)$ 或者右下方的 $P_d(x_i+1,y_i-1)$。P_u 和 P_d 的中点为 $M(x_i+1,y_i-0.5)$，显然，若 M 在圆内，则 P_u 离圆弧近，应该点亮 P_u；否则点亮 P_d。

2. 算法公式推导

设圆的隐函数方程为 $F(x,y)=x^2+y^2-R^2$，理想圆弧将平面划分成 3 个区域：对于圆弧上的点，$F(x,y)=0$；对于圆弧外的点，$F(x,y)>0$；对于圆弧内的点，$F(x,y)<0$。

1) 构造中点判别式

从 $P(x_i,y_i)$ 点走第一步后，为了进行下一像素点的选取，需将 P_u 和 P_d 的中点为 $M(x_i+1,y_i-0.5)$ 代入隐式方程，构造中点偏差判别式 d。

$$d = F(x_M,y_M) = F(x_i+1,y_i-0.5) = (x_i+1)^2+(y_i-0.5)^2-R^2 \tag{3-17}$$

当 $d<0$ 时，中点 M 在圆弧内，P_u 离圆弧近，下一像素应该点亮 P_u，即 y 方向不走步；

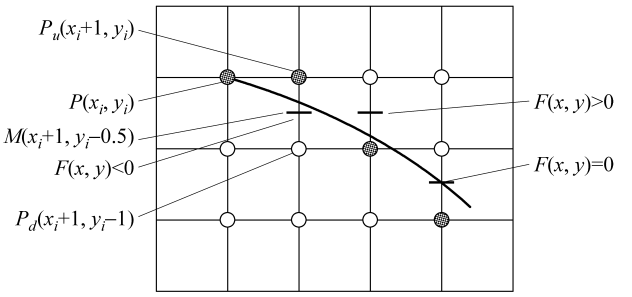


图 3-10　中点画圆算法原理

当 $d>0$ 时，中点 M 在圆弧外，P_d 离圆弧近，下一像素应该点亮 P_d，即 y 方向下走一步；当 $d=0$ 时，中点 M 在直线上，P_u、P_d 与圆弧的距离相等，点亮 P_u 或 P_d 均可，约定取 P_d，如图 3-10 所示。

因此

$$y_{i+1}=\begin{cases} y_i, & d<0 \\ y_i-1, & d\geqslant 0 \end{cases} \tag{3-18}$$

2）中点判别式的递推公式

在图 3-10 中，根据当前点 $P(x_i, y_i)$确定下一点是点亮 P_u 还是点亮 P_d 时，使用了中点判别式 d。为了能够继续判断圆弧上的每一个点，需要给出中点判别式的递推公式及其初始值。

在主位移 x 方向上已走一步的情况下，考虑沿主位移方向再走一步，应该选择哪个中点代入中点判别式来决定下一步该点亮的像素，如图 3-11 所示，分两种情况讨论。

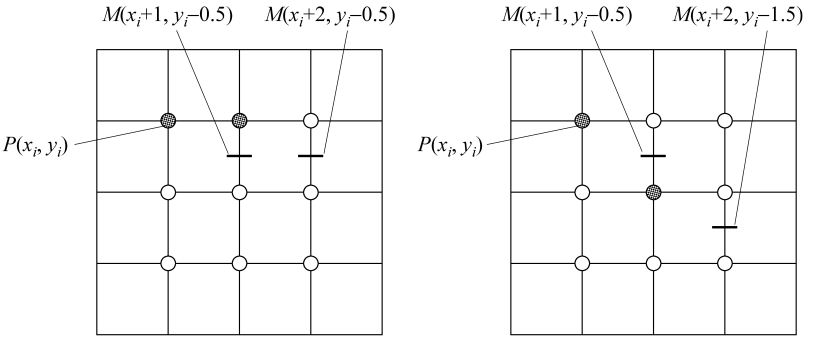


图 3-11　中点偏差判别式的递推

① 当 $d<0$ 时，下一步进行判断的中点坐标为 $M(x_i+2, y_i-0.5)$。所以下一步中点偏差判别式为

$$\begin{aligned} d_{i+1}&=F(x_i+2, y_i-0.5)=(x_i+2)^2+(y_i-0.5)^2-R^2 \\ &=(x_i+1)^2+(y_i-0.5)^2-R^2+2x_i+3=d_i+2x_i+3 \end{aligned} \tag{3-19}$$

② 当 $d\geqslant 0$ 时，下一步进行判断的中点坐标为 $M(x_i+2, y_i-1.5)$。所以下一步中点偏差判别式为