第 3 章　基本图元的扫描转换

本章学习目标

- 了解扫描转换的基本概念。
- 掌握绘制像素点函数的用法。
- 掌握直线的扫描转换算法。
- 熟悉圆的扫描转换算法。
- 了解椭圆的扫描转换算法。
- 掌握 Wu 反走样算法。

直线、圆和椭圆是二维场景中常用的 2D 图元。尽管 MFC 的 CDC 类已经提供了相关的绘图函数，但直接使用这些成员函数仍然无法完全满足计算机图形学的绘图要求，如对基本图形进行反走样处理，绘制颜色光滑过渡的直线等。图形的光栅化(rasterization)就是在像素点阵中确定最佳逼近于理想图元的像素点集，并用指定颜色显示这些像素点集的过程。当光栅化与按扫描线顺序绘制图形的过程结合在一起时，也称为扫描转换(scan conversion)。本章从基本图元的生成原理出发，使用绘制像素点函数研究其扫描转换算法。

3.1　直线的扫描转换

光栅扫描显示器是画点设备，因此不能直接从一像素点到另一像素点绘制一段直线。直线扫描转换的结果是一组在几何上距离理想直线(ideal line)最近的离散像素点集。图 3-1 中，像素使用放大了很多倍的小圆表示。白色空心圆表示未选择(或称为未点亮)的像素点，黑色实心圆表示已选择(或称为已点亮)的像素点。假定从像素点 P 开始，向右方递增一个单位，理想直线与像素网格的交点为 Q，如图 3-2 所示。Q 点在光栅扫描显示器上是从上下像素点对(P_u 和 P_d)中选择一个来显示的。选择的依据是比较 P_u 和 P_d 到直线的距离(用 n_u 和 n_d 来表示)，取距离 Q 点最近的像素点来表示 Q 点。观察相似三角形，也可以使用 d_u 和 d_d 来近似表示 P_u 和 P_d 到直线的距离。这里，有 $d_u+d_d=1$，其中，下标“u”代表 up，下标“d”代表 down。

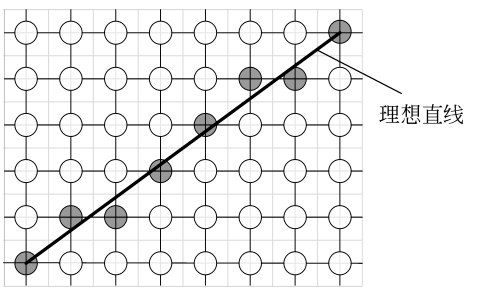


图 3-1　直线的扫描转换

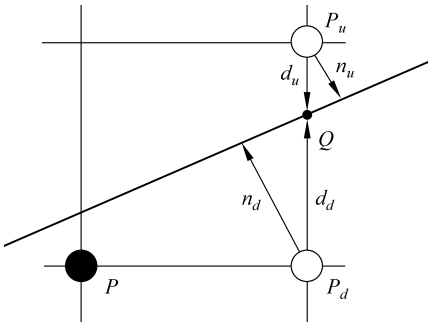


图 3-2　选取离直线最近的像素点

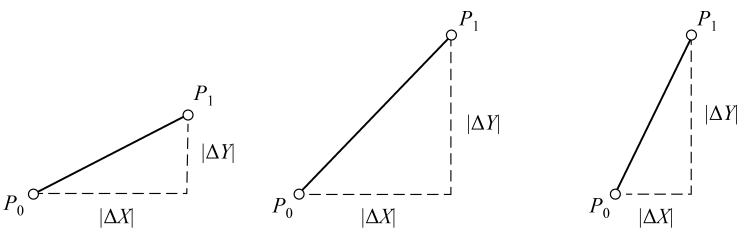
计算机图形学要求直线的绘制速度要快，即尽量使用加减法，避免乘、除、开方、三角等复杂运算。有多种算法可以对直线进行扫描转换，如 DDA 算法、Bresenham 算法、中点算法等。本章重点介绍 Bresenham 算法和中点算法。对于直线，中点算法与 Bresenham 算法产生同样的像素点集。

给定直线的起点坐标为 $P_0(x_0,y_0)$，终点坐标为 $P_1(x_1,y_1)$，用斜截式表示的直线方程为

$$y=kx+b \tag{3-1}$$

其中，直线的斜率为 $k=\dfrac{\Delta y}{\Delta x}=\dfrac{y_1-y_0}{x_1-x_0}$，$\Delta x=x_1-x_0$ 为水平方向位移，$\Delta y=y_1-y_0$ 为垂直方向位移，b 为 y 轴上的截距。

扫描转换算法中，常根据$|\Delta x|$和$|\Delta y|$的大小来确定绘图的主位移方向。在主位移方向上执行的是±1 操作，另一个方向上是否±1，需要建立误差项来判定。如果$|\Delta x|>|\Delta y|$，则取 x 方向为主位移方向，如图 3-3(a)所示；如果$|\Delta x|=|\Delta y|$，取 x 方向为主位移方向或取 y 方向为主位移方向皆可，如图 3-3(b)所示；如果$|\Delta x|<|\Delta y|$，则取 y 方向为主位移方向，如图 3-3(c)所示。



(a) x方向　　(b) x方向或y方向　　(c) y方向

图 3-3　主位移方向

除特别声明外，以下给出的直线扫描转换算法是针对斜率满足 $0\leqslant k\leqslant 1$ 的情形，其他斜率情况下，可以根据直线的对称性类似推导。第一象限包含两个八分象限(octant)。斜率满足 $0\leqslant k\leqslant 1$ 的情形位于第 1 个八分象限内，斜率 $k>1$ 的情形位于第 2 个八分象限内，如图 3-4 所示。

说明：在计算机图形学中，“直线”这个术语表示一段直线，而不是数学意义上两端无限延伸的直线。

3.1.1　DDA 算法

数值微分法(digital differential analyzer，DDA)是用数值方法求解微分方程的一种算法[19]。式(3-1)的微分表示为

$$\frac{\mathrm{d}y}{\mathrm{d}x}=k \tag{3-2}$$

其有限差分近似解为

$$\begin{cases}x_{i+1}=x_i+\Delta x\\ y_{i+1}=y_i+\Delta y=y_i+k\Delta x\end{cases} \tag{3-3}$$

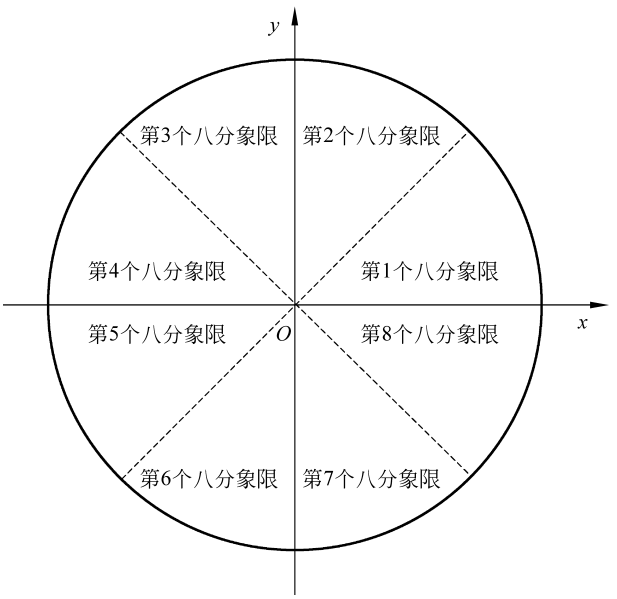


图 3-4　八分象限示意图

式(3-3)表示直线上的像素点 P_{i+1} 与像素点 P_i 的递推关系。可以看出，x_{i+1} 和 y_{i+1} 的值可以根据 x_i 和 y_i 的值推算出来，这说明 DDA 算法是一种增量算法。在一个迭代算法中，如果每一步的 x，y 值是用前一步的值加上一个增量来获得的，那么，这种算法就称为增量算法(incremental algorithm)。

当直线的斜率满足 $0 \leqslant k \leqslant 1$ 时，有 $\Delta x \geqslant \Delta y$，所以 x 方向为主位移方向。取 $\Delta x = 1$，有 $\Delta y = k$。DDA 算法简单表述为

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = y_i + k \end{cases} \tag{3-4}$$

图 3-5 中，$P_i(x_i, y_i)$ 为理想直线的起点扫描转换后的像素点。$Q(x_i + 1, y_i + k)$ 为理想直线与下一列垂直网格线的交点。从 P_i 像素点出发，沿主位移 x 方向上递增一个单位，下一列上只有 1 像素点被选择，候选像素点为 $P_u(x_i + 1, y_i + 1)$ 或 $P_d(x_i + 1, y_i)$。最终选择哪个像素点，可以通过对直线斜率进行圆整计算来决定。

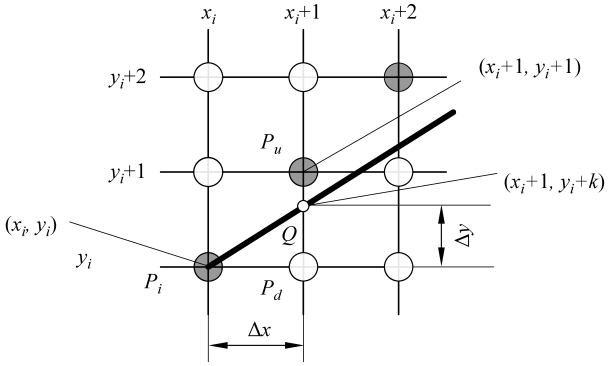


图 3-5　DDA 算法原理示意图

式(3-4)中，x 方向的增量为 1，y 方向的增量为 k。x 是整型变量，y 和 k 是浮点型变量。DDA 算法使用宏命令 ROUND(y_{i+1}) ＝int(y_{i+1}＋0.5)，来选择 P_d 像素点(y_{i+1}＝y_i)，或者选择 P_u 像素点(y_{i+1}＝y_i＋1)。这种圆整计算是为了选择距离直线最近的像素点，即

$$y_{i+1}=\begin{cases}y_i+1, & k\geqslant 0.5\\ y_i, & k<0.5\end{cases} \tag{3-5}$$

DDA 算法中，斜率涉及浮点数运算，而且对 y 取整要花费时间，这不利于硬件实现。为此 Bresenham 提出了一个只使用整数运算就能完成直线绘制的经典算法。

3.1.2 Bresenham 算法

1965 年，Bresenham 为数字绘图仪开发了一种绘制直线的算法[20]，如图 3-6 所示。该算法同样适用于光栅扫描显示器，被称为 Bresenham 算法。Bresenham 算法是一个只使用整数运算的经典算法，能够根据前一个已知坐标(x_i，y_i)进行增量运算得到下一个坐标(x_{i+1}，y_{i+1})，而不必进行取整操作。

1. Bresenham 算法原理

Bresenham 算法在主位移方向上每次递增一个单位。另一个方向的增量为 0 或 1，取决于像素点与理想直线的距离，这一距离称为误差项，用 d 表示。

图 3-7 中，直线位于第一个八分象限内，0≤k≤1，因此 x 方向为主位移方向。$P_i(x_i,y_i)$ 点为当前像素，$Q(x_i+1,y_i+d)$为理想直线与下一列垂直网格线的交点。假定直线的起点为 P_i，该点位于网格点上，所以 d_i 的初始值为 0。

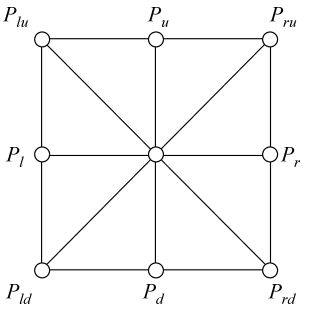


图 3-6　数字化绘图仪画笔运动路径

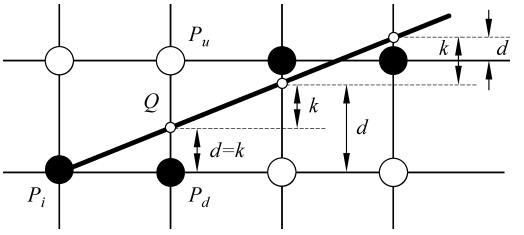


图 3-7　Bresenham 算法原理

沿 x 方向递增一个单位，即 x_{i+1}＝x_i＋1。下一个候选像素点是 $P_d(x_i+1,y_i)$或 P_u(x_i＋1，y_i＋1)。究竟是选择 P_u 还是 P_d，取决于交点 Q 的位置，而 Q 点的位置是由直线的斜率决定的。Q 点与像素点 P_d 的误差项为 d_{i+1}＝k。当 d_{i+1}＜0.5 时，像素点 P_d 距离 Q 点近，选取 P_d；当 d_{i+1}＞0.5 时，像素点 P_u 距离 Q 点近，选取 P_u；当 d_{i+1}＝0.5 时，像素点 P_d 与 P_u 到 Q 点的距离相等，选取任一像素点均可，约定选取 P_u。

因此

$$y_{i+1}=\begin{cases}y_i+1, & d_{i+1}\geqslant 0.5\\ y_i, & d_{i+1}<0.5\end{cases} \tag{3-6}$$

算法的关键在于计算递推计算误差项 d_i。从 d_0＝0 开始，沿 x 方向递增一个单位，有

$d_{i+1}=d_i+k$。一旦 $d_{i+1}\geqslant 0$，y 方向上走了一步，就将 d 减去 1。由于只需要检查误差项的符号，令 $e_{i+1}=d_{i+1}-0.5$，以消除小数的影响。式(3-6)改写为

$$y_{i+1}=\begin{cases}y_i+1, & e_{i+1}\geqslant 0\\ y_i, & e_{i+1}<0\end{cases} \tag{3-7}$$

取 e 的初始值为 $e_0=-0.5$。沿 x 方向每递增一个单位，有 $e_{i+1}=e_i+k$。当 $e_{i+1}\geqslant 0$ 时，下一像素点更新为(x_i+1,y_i+1)，同时将 e_{i+1}更新为 $e_{i+1}-1$；否则，下一像素点更新为(x_i+1,y_i)。

2. 整数 Bresenham 算法原理

虽然当前点的 x 坐标和 y 坐标均使用了加 1 或减 1 的整数运算，但是在递推计算直线误差项 e 时，仍然使用了浮点型变量 k，除法也参与了运算。按照 Bresenham 的说法，使用整数运算可以加快算法的速度。应对算法进行修正，以避免除法运算。由于 Bresenham 算法中只用到误差项的符号，而 Δx 在第一个八分象限内恒为正，可以进行如下替换 $e=2\Delta x\times e$。改进的整数 Bresenham 算法如下：

e 的初值为 $e_0=-\Delta x$，沿 x 方向每递增一个单位，有 $e_{i+1}=e_i+2\Delta y$。当 $e_{i+1}\geqslant 0$ 时，下一像素点更新为(x_i+1,y_i+1)，同时将 e_{i+1}更新为 $e_{i+1}-2\Delta x$；否则，下一像素点更新为(x_i+1,y_i)。

3. 通用整数 Bresenham 算法原理

以上整数 Bresenham 算法绘制的是第一个八分象限内的直线。在绘制图形时，要求编程实现能绘制任意斜率的通用直线。根据对称性，可以设计通用整数 Bresenham 算法。图 3-8 中，x 和 y 是加 1 还是减 1，取决于直线所在的象限。例如，对于第一个八分象限$(0\leqslant k\leqslant 1)$，$x$ 方向为主位移方向。Bresenham 算法的原理为 x 每次加 1，y 根据误差项决定是加 1 或者加 0；对于第二个八分象限，y 方向为主位移方向。Bresenham 算法的原理为 y 每次加 1，x 是否加 1 需要使用误差项来判断。使用通用整数 Bresenham 算法绘制从原点发出的 360 条射线，效果如图 3-9 所示。

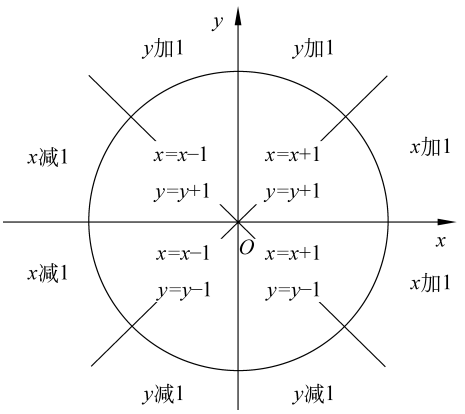


图 3-8　通用整数 Bresenham 算法判别条件

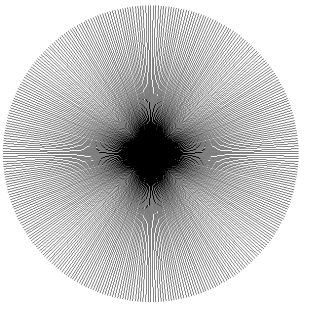


图 3-9　直线算法的校核图

3.1.3　中点算法

1967 年 Pitteway 等提出了一个中点算法[21]。1984 年，Aken 等对中点算法进行了改

进[22]。中点算法是基于隐函数方程设计的，使用像素网格中点来判断如何选取距离理想直线最近的像素点。

1. 中点算法原理

每次沿主位移方向上递增一个单位，另一个方向上增量为 1 或 0，取决于中点误差项的值。

由式(3-1)得到理想直线的隐函数方程为

$$F(x,y)=y-kx-b=0 \tag{3-8}$$

理想直线将平面划分成三个区域：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，$F(x,y)<0$。

考查斜率位于第一个八分象限内的理想直线。假定直线上的当前像素点为 $P_i(x_i,y_i)$，Q 点是直线与网格线的交点。沿着主位移 x 方向上递增一个单位，即执行 $x_{i+1}=x_i+1$，下一像素点将从 $P_u(x_i+1,y_i+1)$和 $P_d(x_i+1,y_i)$两个候选像素点中选取。连接像素点 P_u 和像素点 P_d 的网格中点为 $M(x_i+1,y_i+0.5)$，如图 3-10 所示。显然，若中点 M 位于理想直线的下方，则像素点 P_u 距离直线近；否则，像素点 P_d 距离直线近。

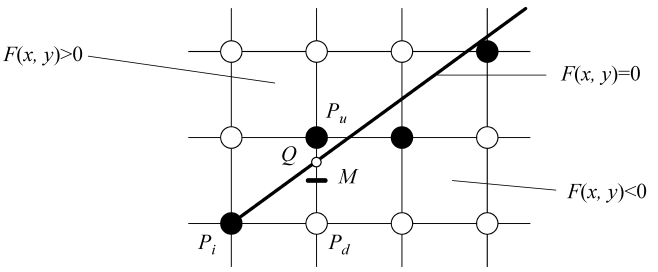


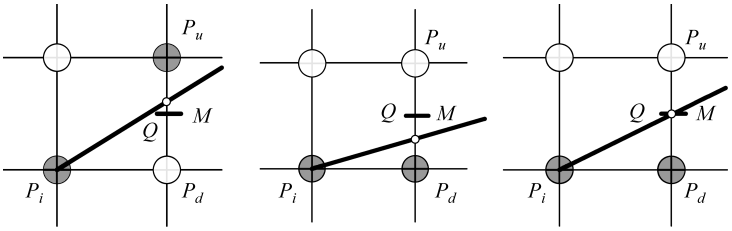
图 3-10　直线中点算法原理

2. 构造中点误差项

从 $P_i(x_i,y_i)$像素点出发，沿主位移方向选取直线上的下一像素点时，需要将连接 P_u 和 P_d 两个候选像素点连线的网格中点 M 代入隐函数方程(3-8)，构造中点误差项 d

$$d_i=F(x_i+1,y_i+0.5)=y_i+0.5-k(x_i+1)-b \tag{3-9}$$

当 $d_i<0$ 时，中点 M 位于直线的下方，像素点 P_u 距离直线近，下一像素点应选取 P_u，即 y 方向上增量为 1；当 $d_i>0$ 时，中点 M 位于直线的上方，像素点 P_d 距离直线近，下一像素点应选取 P_d，即 y 方向上增量为 0；当 $d_i=0$ 时，中点 M 位于直线上，像素点 P_u、P_d 与直线的距离相等，选取任一像素点均可，约定选取 P_d，如图 3-11 所示。



(a) $d_i<0$　　(b) $d_i>0$　　(c) $d_i=0$

图 3-11　中点算法分析

因此

$$y_{i+1}=\begin{cases}y_i+1, & d_i<0\\ y_i, & d_i\geqslant 0\end{cases} \tag{3-10}$$

3. 中点误差项的递推

图 3-11 中，根据当前像素点 P_i 确定下一像素点是选取 P_u 还是选取 P_d 时，使用了中点误差项 d 进行判断。为了能够继续光栅化直线上的后续像素点，需要给出中点误差项的递推公式。

在主位移 x 方向上已递增一个单位的情况下，考虑沿 x 方向再递增一个单位，应该选择哪个网格中点来计算误差项，分两种情况讨论。

当 $d_i<0$ 时，下一步进行判断的中点为 $M_u(x_i+2,y_i+1.5)$，如图 3-12(a)所示。中点误差项的递推公式为

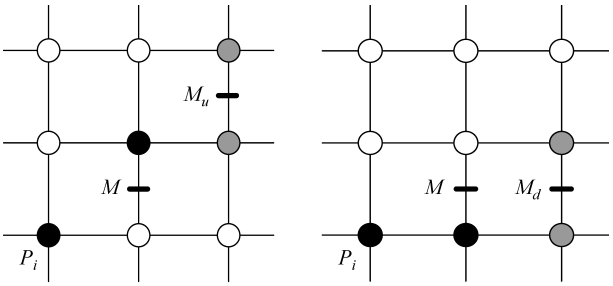
$$\begin{aligned}d_{i+1}&=F(x_i+2,y_i+1.5)=y_i+1.5-k(x_i+2)-b\\&=y_i+0.5-k(x_i+1)-b+1-k=d_i+1-k\end{aligned} \tag{3-11}$$

所以，中点误差项的增量为 $1-k$。

当 $d_i\geqslant 0$ 时，下一步进行判断的中点为 $M_d(x_i+2,y_i+0.5)$，如图 3-12(b)所示。中点误差项的递推公式为

$$\begin{aligned}d_{i+1}&=F(x_i+2,y_i+0.5)=y_i+0.5-k(x_i+2)-b\\&=y_i+0.5-k(x_i+1)-b-k=d_i-k\end{aligned} \tag{3-12}$$

所以，中点误差项的增量为 $-k$。



(a) $d_i<0$　　(b) $d_i\geqslant 0$

图 3-12　中点的递推

4. 中点误差项的初始值

直线的起点坐标扫描转换后的像素点为 $P_0(x_0,y_0)$。从像素点 P_0 出发沿主位移 x 方向递增一个单位，第一个参与判断的中点是 $M(x_0+1,y_0+0.5)$。代入中点误差项计算公式(3-9)，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，因为像素点 $P_0(x_0,y_0)$ 位于直线上，所以 $y_0-kx_0-b=0$，则

$$d_0=0.5-k \tag{3-13}$$

5. 中点算法整数化

上述的中点算法有一个缺点：在计算中点误差项 d 时，其初始值与递推公式中分别包含有小数 0.5 和斜率 k。由于中点算法只用到 d 的符号，可以使用正整数 $2\Delta x$ 乘以 d 来摆脱小数运算。

$$e_i = 2\Delta x d_i$$

整数化处理后，中点误差项的初始值为

$$e_0 = \Delta x - 2\Delta y \tag{3-14}$$

当 $e_i < 0$ 时，中点误差项的递推公式为

$$e_{i+1} = e_i + 2\Delta x - 2\Delta y \tag{3-15}$$

所以，中点误差项的增量为 $2\Delta x - 2\Delta y$。

当 $e_i \geqslant 0$ 时，表示选择 P_d，中点误差项的递推公式为

$$e_{i+1} = e_i - 2\Delta y \tag{3-16}$$

所以，中点误差项的增量为 $-2\Delta y$。

20 世纪 70 年代，由于计算机运算速度受限，完全整数的光栅化算法是计算机图形学研究者追求的一个目标。现有的研究成果已经证明：端点采用整数坐标没有什么益处，因为现在的 CPU 可以按照与处理整数同样的速度处理浮点数。

例 3-1 绘制起点为红色，终点为蓝色的颜色渐变直线。

如果直线上每像素点的颜色由两个端点的颜色经过线性插值得到，则可以实现直线颜色的光滑渐变。假设直线位于第一个八分象限内，基于整数中点算法开发颜色渐变直线算法。

RGB 宏有红色分量 byteR、绿色分量 byteG 和蓝色分量 byteB。每个分量占 1 字节，数值范围为 0～255。编程时，常将颜色分量规范化到[0,1]闭区间，使用时乘以 255 即可。这样，红色对应的颜色为(1,0,0)，绿色对应的颜色为(0,1,0)。蓝色对应的颜色为(0,0,1)。注意，沿主位移方向，当 x 坐标从 x_0 执行加 1 操作到达 x_1 时，byteR 分量从 1 减小到 0，byteG 分量保持不变，byteB 分量从 0 增加到 1。令 $\Delta x = x_1 - x_0$，则 byteR 的步长增量 incrR 为 $-1/\Delta x$，绿色分量的步长增量 incrG 为 0，蓝色分量的步长增量 incrB 为 $1/\Delta x$。

由于颜色分量的运算包含浮点数运算，所以使用直线的浮点数中点算法来编程实现。

3.2 圆的扫描转换

圆的扫描转换是在屏幕像素点阵中确定最佳逼近于理想圆的像素点集的过程。绘制圆可以使用简单方程画圆算法或极坐标画圆算法，但这些算法涉及开方运算或三角运算，效率很低。1977 年，Bresenham 开发了一种绘制圆弧算法[23]，假设笔式绘图仪递增地沿着圆弧前进，能为圆心在原点的圆产生所有像素点。这里介绍一种运用中点准则推导的中点画圆算法，它也能产生一组优化的像素。中点画圆算法本质上与 Bresenham 绘制圆弧算法一致，但更容易理解。本节主要讲解仅包含加减运算的顺时针绘制圆弧的中点算法，根据对称性可以绘制整圆。

3.2.1 简单方程画圆算法

1. 直角坐标算法

圆心位于原点，半径为 R 的圆的直角坐标方程为

$$x^2 + y^2 = R^2 \tag{3-17}$$

可以解得 $y=f(x)$的表达式为

$$y = \pm\sqrt{R^2 - x^2} \tag{3-18}$$

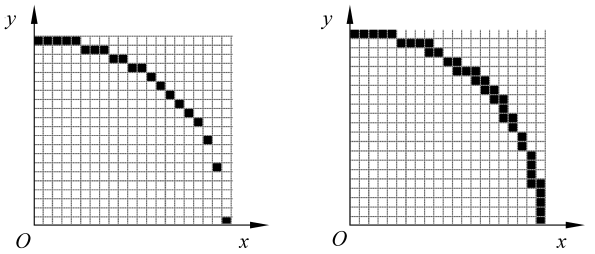
当 x 从 0 逐像素递增到 R 时，根据式(3-18)可以计算出每一步的 y，从而顺时针绘制出第一象限内的 1/4 圆弧，效果如图 3-13(a)所示。注意，当 x 靠近 R(图中取 $R=20$)时，圆上的像素会有较大的间断，因为圆的斜率在此处变得无穷大。

2. 极坐标算法

圆心位于原点，半径为 R 的圆的极坐标方程为

$$\begin{cases} x = R\cos\theta \\ y = R\sin\theta \end{cases} \tag{3-19}$$

当 θ 从 0°一度一度地递增到 90°时，根据式(3-19)可以计算出每一步的 x 和 y，从而逆时针绘制出第一象限内的 1/4 圆弧，效果如图 3-13(b)所示。极坐标画圆算法与直角坐标方程算法相比，该方法避免像素出现大的间断。



(a) 直角坐标方程　　(b) 极坐标方程

图 3-13　简单方程画圆弧效果图

3.2.2　中点画圆算法

1. 八分圆弧

圆心位于原点、半径为 R 的圆的隐函数方程为

$$F(x,y) = x^2 + y^2 - R^2 = 0 \tag{3-20}$$

圆将平面划分成 3 个区域：对于圆上的点，$F(x,y)=0$；对于圆外的点，$F(x,y)>0$；对于圆内的点，$F(x,y)<0$，如图 3-14 所示。

根据圆的对称性，可以用 4 条对称轴 $x=0$，$y=0$，$x=y$，$x=-y$ 将圆划分 8 等份，如图 3-15 所示。只要绘制出第一象限内编号为②的八分圆弧(以下简称为圆弧)，根据对称性就可以生成其他 7 个八分圆弧，这称为八分法画圆算法。假定圆弧②上的任意点为 (x,y)，可以顺时针方向确定另外 7 个点：(y,x)、$(y,-x)$、$(x,-y)$、$(-x,-y)$、$(-y,-x)$、$(-y,x)$、$(-x,y)$。

2. 中点算法原理

从图 3-15 中所示的圆弧②可以看出，y 是 x 的单调递减函数。假设圆弧起点 $x=0$，$y=R$ 精确地落在像素点上，中点算法要从 $x=0$ 绘制到 $x=y$，顺时针方向确定最佳逼近于圆弧的像素点集。此段圆弧上各个点的切线斜率 k 处处满足 $|k|<1$，即 $|\Delta x|>|\Delta y|$，所以

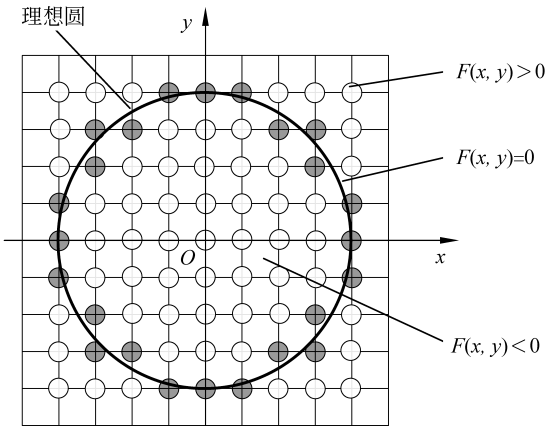


图 3-14　圆的扫描转换

x 方向为主位移方向。中点算法原理表述为：x 方向上每次加 1，y 方向上减不减 1 取决于中点误差项的值。

假定圆弧上当前像素点是 $P_i(x_i, y_i)$，Q 点是圆弧与网格线的交点。下一像素点将从 $P_u(x_i+1, y_i)$ 和 $P_d(x_i+1, y_i-1)$ 两个候选像素点中选取，如图 3-16 所示。连接像素点 P_u 和像素点 P_d 的网格中点为 $M(x_i+1, y_i-0.5)$。显然，若 M 点位于理想圆弧的下方，则像素点 P_u 离圆弧近；若 M 点位于理想圆弧的上方，则像素点 P_d 离圆弧近。

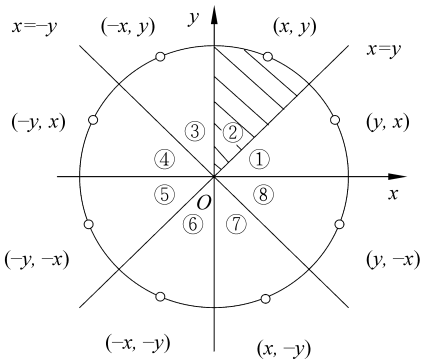


图 3-15　圆的对称性

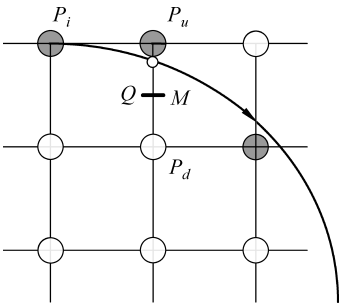


图 3-16　圆的中点算法原理

3. 构造中点误差项

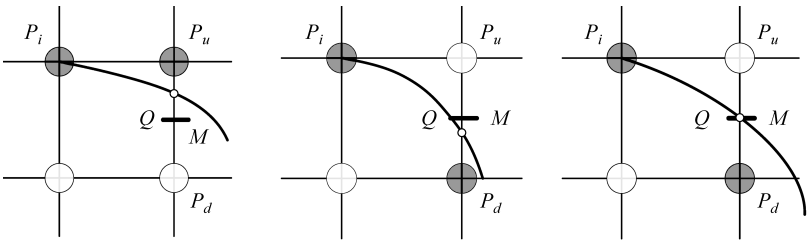
从 $P_i(x_i, y_i)$ 像素出发选取下一像素点时，需将 P_u 和 P_d 两个候选像素点连线的网格中点 $M(x_i+1, y_i-0.5)$ 代入隐函数方程，构造中点误差项 d_i

$$d_i = F(x_i+1, y_i-0.5) = (x_i+1)^2 + (y_i-0.5)^2 - R^2 \tag{3-21}$$

当 $d_i<0$ 时，中点 M 位于圆弧内，下一像素点应选取 P_u，即 y 方向上不减 1；当 $d_i>0$ 时，中点 M 位于圆弧外，下一像素点应选取 P_d，即 y 方向上减 1；当 $d_i=0$ 时，中点 M 位于圆弧上，像素点 P_u、P_d 与圆弧的距离相等，选取任一像素点均可，约定选取 P_d，如图 3-17 所示。

因此

$$y_{i+1} = \begin{cases} y_i, & d_i < 0 \\ y_i - 1, & d_i \geqslant 0 \end{cases} \tag{3-22}$$



(a) $d_i<0$　　(b) $d_i>0$　　(c) $d_i=0$

图 3-17　中点算法分析

4. 递推公式

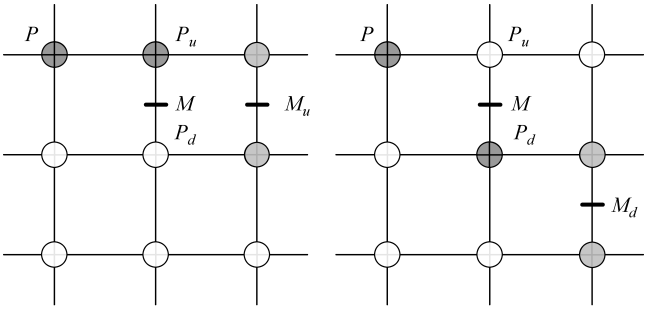
图 3-17 中，根据当前点 P_i 确定了下一像素是选取 P_u 还是 P_d 时，使用了中点误差项 d_i。为了能够继续判断圆弧上的后续像素点，需要给出中点误差项的递推公式和初始值。

1）中点误差项的递推公式

在主位移 x 方向上已递增一个单位的情况下，考虑沿主位移方向上再递增一个单位，应该选择哪个中点来计算误差项，以判断下一步要选取的像素，分两种情况讨论。

当 $d_i<0$ 时，下一步的中点坐标为 $M_u(x_i+2,y_i-0.5)$，如图 3-18(a)所示。中点误差项的递推公式为

$$\begin{aligned} d_{i+1} &= F(x_i+2,y_i-0.5)=(x_i+2)^2+(y_i-0.5)^2-R^2 \\ &=(x_i+1)^2+(y_i-0.5)^2-R^2+2x_i+3=d_i+2x_i+3 \end{aligned} \tag{3-23}$$



(a) $d_i<0$　　(b) $d_i\geqslant 0$

图 3-18　中点的递推

当 $d_i\geqslant 0$ 时，下一步的中点坐标为 $M_d(x_i+2,y_i-1.5)$，如图 3-18(b)所示。中点误差项的递推公式为

$$\begin{aligned} d_{i+1} &= F(x_i+2,y_i-1.5)=(x_i+2)^2+(y_i-1.5)^2-R^2 \\ &=(x_i+1)^2+(y_i-0.5)^2-R^2+2x_i+3+(-2y_i+2) \\ &=d_i+2(x_i-y_i)+5 \end{aligned} \tag{3-24}$$

2）中点误差项的初始值

圆弧的起点扫描转换后的像素为 $P_0(0,R)$。若沿主位移 x 方向递增一个单位，第一个参与判断的中点为 $M(1,R-0.5)$，相应的中点误差项 d 的初始值为

$$d_0=F(1,R-0.5)=1+(R-0.5)^2-R^2=1.25-R \tag{3-25}$$

5. 整数中点画圆算法

由于使用的只是 d 的符号，可以通过一些简单的变换来摆脱小数。定义 $e_i=d_i-0.25$，初始值 $d_0=1.25-R$ 对应于 $e_0=1-R$。误差项 $d_i<0$ 对应于 $e_i<-0.25$。由于 e_i 始终是整数，可以将 $e_i<-0.25$ 等价为 $e_i<0$。基于整数中点画圆算法扫描转换 $R=20$ 的圆，放大效果如图 3-19 所示。

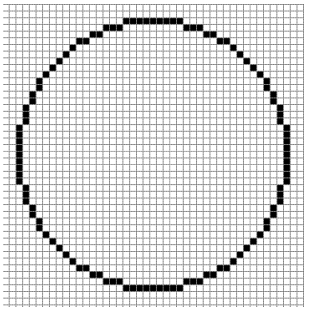


图 3-19　中点画圆算法扫描转换效果图

3.3　椭圆的扫描转换

椭圆的扫描转换是在屏幕像素点阵中选取最佳逼近于理想椭圆的像素点集的过程。椭圆是长半轴与短半轴不相等的圆。椭圆的扫描转换与圆的扫描转换有相似之处，但也有不同，主要区别是椭圆弧上存在改变主位移方向的临界点[22]。本节主要讲解顺时针绘制四分椭圆弧的中点算法，根据对称性可以绘制完整椭圆。

1. 四分椭圆弧

中心在原点、长半轴为 a、短半轴为 b 的轴对称椭圆方程为

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1 \tag{3-26}$$

隐函数表示为

$$F(x,y)=b^2x^2+a^2y^2-a^2b^2=0 \tag{3-27}$$

椭圆将平面划分成 3 个区域：对于椭圆上的点，$F(x,y)=0$；对于椭圆外的点，$F(x,y)>0$；对于椭圆内的点，$F(x,y)<0$，如图 3-20 所示。

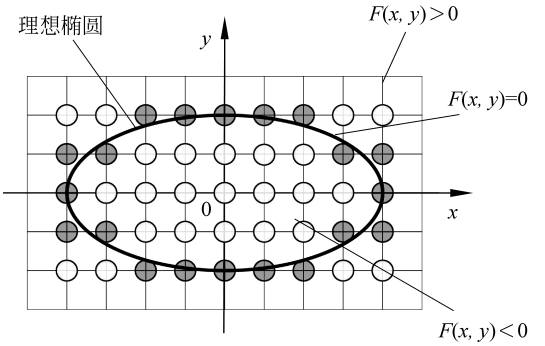


图 3-20　椭圆的扫描转换

考虑到椭圆的对称性，可以用对称轴 $x=0$，$y=0$，将椭圆四等分。只要绘制出第一象限内的四分椭圆弧（以下简称为椭圆弧），如图 3-21 阴影区域所示，根据对称性就可以生成其他 3 个椭圆弧，这称为四分法画椭圆算法。已知第一象限内的一点(x,y)，可以顺时针确定另外 3 个对称点：$(x,-y)$、$(-x,-y)$和$(-x,y)$。

2. 临界点分析

在处理第一象限的四分椭圆弧时，进一步以法向量（normal vector）的两个分量相等的点把其划分为两个区域：区域Ⅰ和区域Ⅱ，该点称为临界点，如图 3-22 所示。特别地，在临界点处，曲线的斜率为-1。相比圆的绘制，椭圆弧需要计算临界点的位置。椭圆上任一点(x,y)处的法向量 $N(x,y)$为

$$N(x,y)=\frac{\partial F}{\partial x}i+\frac{\partial F}{\partial y}j=2b^2xi+2a^2yj \tag{3-28}$$

式中，法向量的 x 方向的分量为 $N_x=2b^2x$，法向量的 y 方向的分量为 $N_y=2a^2y$，i 和 j 是沿 x 轴向和沿 y 轴向的标准单位向量。

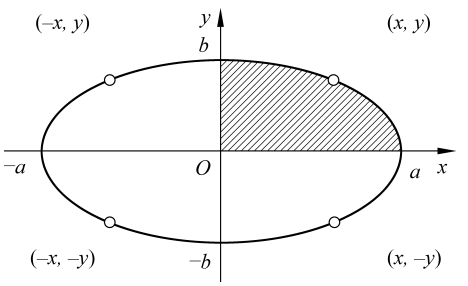


图 3-21　椭圆的对称性

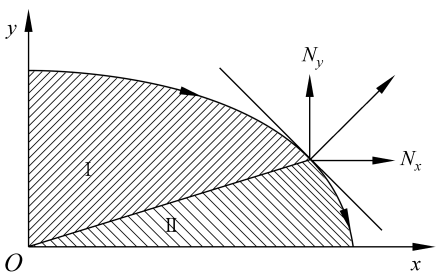


图 3-22　椭圆的临界点

从曲线上一点的法向量角度看，在区域Ⅰ内，$N_x<N_y$；在临界点处，$N_x=N_y$；在区域Ⅱ内，$N_x>N_y$。显然，在临界点处，法向量分量的大小发生了变化。

从曲线上的斜率角度看，在临界点处，$\frac{dy}{dx}=-1$。区域Ⅰ内，有$\frac{dy}{dx}>-1$，即 $dx>dy$，所以 x 方向为主位移方向；在临界点处，有 $dx=dy$；在区域Ⅱ内，有$\frac{dy}{dx}<-1$，即 $dy>dx$，所以 y 方向为主位移方向。显然，在临界点处，主位移方向发生了改变。

3. 中点算法原理

在区域Ⅰ，x 方向上每次递增一个单位，y 方向上减 1 或减 0 取决于中点误差项的值；在区域Ⅱ，y 方向上每次递减一个单位 1，x 方向上加 1 或加 0 取决于中点误差项的值。

先考虑图 3-23 所示区域Ⅰ的 AC 段椭圆弧。此时中点算法要从起点 $A(0,b)$到临界点 $C(a^2/\sqrt{a^2+b^2},b^2/\sqrt{a^2+b^2})$顺时针方向确定最佳逼近于该段椭圆弧的像素点集。由于 x 方向为主位移方向，假定当前点是 $P_i(x_i,y_i)$，下一步将从正右方的像素 $P_u(x_i+1,y_i)$和右下方的像素 $P_d(x_i+1,y_i-1)$两个候选像素中选取。

再考虑图 3-23 所示区域Ⅱ的 CB 段椭圆弧。此时中点画椭圆算法要从临界点 $C(a^2/\sqrt{a^2+b^2},b^2/\sqrt{a^2+b^2})$到终点 $B(a,0)$顺时针方向确定最佳逼近于该段椭圆弧的像素点集。由于 y 方向为主位移方向，假定当前点是 $P_i(x_i,y_i)$，下一步将从正下方的像素 P_l

(x_i, y_i-1)和右下方的像素 $P_r(x_i+1, y_i-1)$两个候选像素中选取。这里，下标“l”代表 left，下标“r”代表 right。

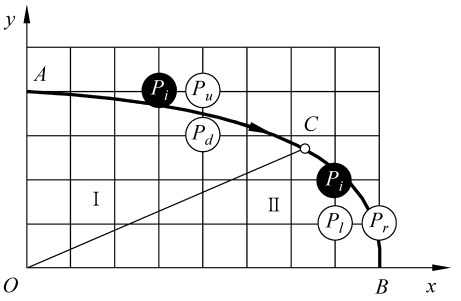


图 3-23　椭圆弧的中点算法原理

4. 构造区域Ⅰ的中点误差项

从当前点 P_i 出发选取下一像素时，需将 P_u 和 P_d 两个候选像素连线的网格中点 $M(x_i+1, y_i-0.5)$代入隐函数方程，构造中点误差项 d_{1i}

$$d_{1i}=F(x_i+1, y_i-0.5)=b^2(x_i+1)^2+a^2(y_i-0.5)^2-a^2b^2 \tag{3-29}$$

当 $d_{1i}<0$ 时，中点 M 位于椭圆弧内，下一像素应选取 P_u，即 y 方向上不减 1；当 $d_{1i}>0$ 时，中点 M 位于椭圆弧外，下一像素应选取 P_d，即 y 方向上减 1；当 $d_{1i}=0$ 时，中点 M 位于椭圆上，像素 P_u 和 P_d 与椭圆弧的距离相等，选取任一像素均可，约定选取 P_d，如图 3-24 所示。

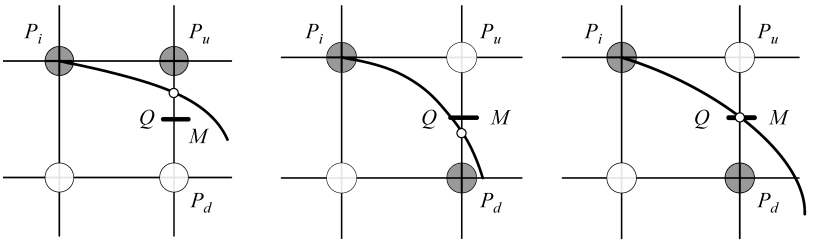


图 3-24　区域Ⅰ内像素点的选取

因此

$$y_{i+1}=\begin{cases} y_i, & d_{1i}<0 \\ y_i-1, & d_{1i}\geqslant 0 \end{cases} \tag{3-30}$$

5. 区域Ⅰ内中点误差项的递推

图 3-24 中，根据当前点 P_i 选取 P_u 还是 P_d，使用了中点误差项 d_{1i}。为了能够继续选取椭圆弧上的后续像素，需要给出中点误差项 d_{1i} 的递推公式和初始值。

1）中点误差项 d_1 的递推公式

在主位移 x 方向上已递增一个单位的情况下，考虑沿主位移方向再递增一个单位，应该选取哪个中点来计算误差项，以判断下一步要选取的像素，分两种情况讨论。

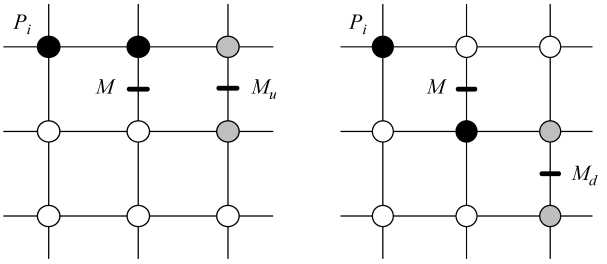
当 $d_{1i}<0$ 时，下一步的中点坐标为 $M_u(x_i+2, y_i-0.5)$，如图 3-25(a)所示。中点误差

项的递推公式为

$$\begin{aligned} d_{1(i+1)} &= F(x_i+2, y_i-0.5) = b^2(x_i+2)^2 + a^2(y_i-0.5)^2 - a^2b^2 \\ &= b^2(x_i+1)^2 + a^2(y_i-0.5)^2 - a^2b^2 + b^2(2x_i+3) \\ &= d_{1i} + b^2(2x_i+3) \end{aligned} \tag{3-31}$$

当 $d_{1i} \geqslant 0$ 时，下一步的中点坐标为 $M_d(x_i+2, y_i-1.5)$，如图 3-25(b)所示。中点误差项的递推公式为

$$\begin{aligned} d_{1(i+1)} &= F(x_i+2, y_i-1.5) = b^2(x_i+2)^2 + a^2(y_i-1.5)^2 - a^2b^2 \\ &= b^2(x_i+1)^2 + a^2(y_i-0.5)^2 - a^2b^2 + b^2(2x_i+3) + a^2(-2y_i+2) \\ &= d_{1i} + b^2(2x_i+3) + a^2(-2y_i+2) \end{aligned} \tag{3-32}$$



(a) $d_{1i}<0$　　(b) $d_{1i} \geqslant 0$

图 3-25　区域Ⅰ内中点的递推

2）中点误差项 d_{1i} 的初始值

在区域Ⅰ内，椭圆弧的起点扫描转换后的像素为 $P_0(0,b)$。沿主位移 x 方向递增一个单位，第一个参与判断的中点是 $M(1, b-0.5)$，相应的中点误差项 d_{1i} 的初始值为

$$\begin{aligned} d_{10} &= F(1, b-0.5) = b^2 + a^2(b-0.5)^2 - a^2b^2 \\ &= b^2 + a^2(-b+0.25) \end{aligned} \tag{3-33}$$

6. 构造区域Ⅱ的中点误差项

在区域Ⅱ内，主位移方向发生变化，由 x 方向转变为 y 方向。从区域Ⅰ椭圆弧的终止点 $P_i(x_i, y_i)$ 出发选取下一像素时，需将 $P_l(x_i, y_i-1)$ 和 $P_r(x_i+1, y_i-1)$ 的中点 $M(x_i+0.5, y_i-1)$ 代入隐函数方程，构造中点误差项 d_{2i}

$$d_{2i} = F(x_i+0.5, y_i-1) = b^2(x_i+0.5)^2 + a^2(y_i-1)^2 - a^2b^2 \tag{3-34}$$

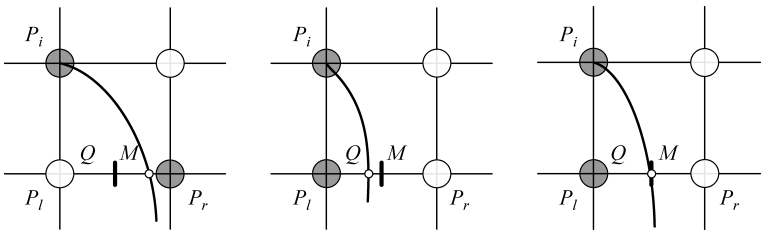
当 $d_{2i}<0$ 时，中点 M 位于椭圆弧内，下一像素点应选取 P_r，即 x 方向上加 1；当 $d_{2i}>0$ 时，中点 M 位于椭圆弧外，下一像素点应选取 P_l，即 x 方向上不加 1；当 $d_{2i}=0$ 时，中点 M 位于椭圆弧上，P_l、P_r 与椭圆弧的距离相等，选取任一像素均可，约定选取 P_l，如图 3-26 所示。

因此

$$x_{i+1} = \begin{cases} x_i+1, & d_{2i} < 0 \\ x_i, & d_{2i} \geqslant 0 \end{cases} \tag{3-35}$$

7. 区域Ⅱ内中点误差项的递推

图 3-26 中，根据 P_i 确定下一像素点是选取 P_l 还是 P_r 时，使用了中点误差项 d_{2i}。为了能够继续选取椭圆弧上的后续像素，需要给出中点误差项 d_{2i} 的递推公式和初始值。



(a) $d_{2i}<0$　　(b) $d_{2i}>0$　　(c) $d_{2i}=0$

图 3-26　下半部分像素点的选取

1）中点误差项 d_2 的递推公式

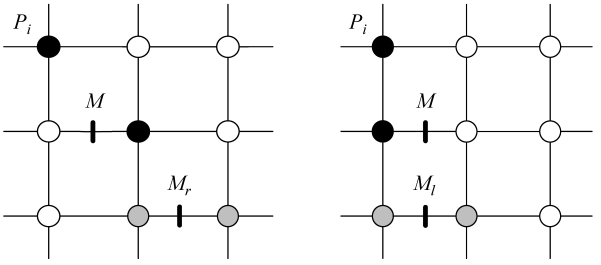
在主位移 y 方向上已递增一个单位的情况下，考虑沿主位移方向上再递增一个单位，应该选择哪个中点来计算误差项，以判断下一步要选取的像素，分两种情况讨论。

当 $d_{2i}<0$ 时，下一步的中点坐标为 $M_r(x_i+1.5, y_i-2)$，如图 3-27(a)所示。中点误差项的递推公式为

$$\begin{aligned} d_{2(i+1)} &= F(x_i+1.5, y_i-2) = b^2(x_i+1.5)^2 + a^2(y_i-2)^2 - a^2b^2 \\ &= b^2(x_i+0.5)^2 + a^2(y_i-1)^2 - a^2b^2 + b^2(2x_i+2) + a^2(-2y_i+3) \\ &= d_{2i} + b^2(2x_i+2) + a^2(-2y_i+3) \end{aligned} \tag{3-36}$$

当 $d_{2i}\geqslant 0$ 时，下一步的中点坐标为 $M_l(x_i+0.5, y_i-2)$，如图 3-27(b)所示。中点误差项的递推公式为

$$\begin{aligned} d_{2(i+1)} &= F(x_i+0.5, y_i-2) = b^2(x_i+0.5)^2 + a^2(y_i-2)^2 - a^2b^2 \\ &= b^2(x_i+0.5)^2 + a^2(y_i-1)^2 - a^2b^2 + a^2(-2y_i+3) \\ &= d_{2i} + a^2(-2y_i+3) \end{aligned} \tag{3-37}$$



(a) $d_{2i}<0$　　(b) $d_{2i}\geqslant 0$

图 3-27　区域Ⅱ内中点的递推

2）中点误差项 d_2 的初始值

对于区域Ⅰ内椭圆弧上一点 $P_i(x_i, y_i)$，如果在当前中点 $M_{\mathrm{I}}(x_i+1, y_i-0.5)$ 处，假定图 3-28 中 $P_i(x_i, y_i)$ 点为区域Ⅰ内椭圆弧上的最后一像素，$M_{\mathrm{I}}(x_i+1, y_i-0.5)$ 是 P_u 和 P_d 像素的中点。满足法向量的 x 方向分量小于法向量的 y 方向分量

$$b^2(x_i+1) < a^2(y_i-0.5) \tag{3-38}$$

而在下一个中点处，不等号改变方向，则说明椭圆弧从区域Ⅰ转入了区域Ⅱ。在区域Ⅱ内，中点转换为 $M_{\mathrm{II}}(x_i+0.5, y_i-1)$，用于判断选取 P_l 和 P_r 像素，所以区域Ⅱ内椭圆弧中点

误差项 d_{2i} 的初始值为

$$d_{2i}=b^2(x+0.5)^2+a^2(y-1)^2-a^2b^2 \tag{3-39}$$

基于中点画椭圆算法对 $a=30$，$b=20$ 的椭圆扫描转换，放大效果如图 3-29 所示。

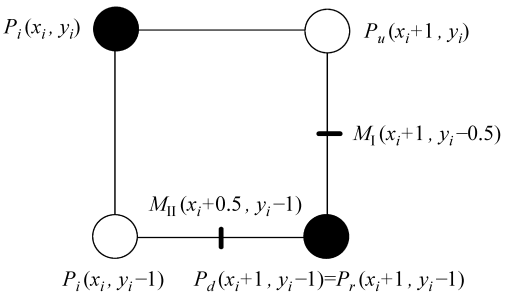


图 3-28　区域Ⅰ与区域Ⅱ的切换

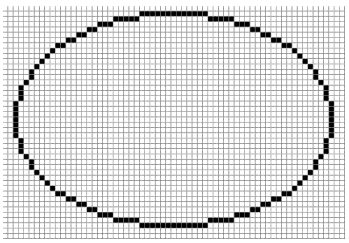


图 3-29　中点画椭圆算法扫描转换效果图

3.4　反走样技术

3.4.1　反走样现象

扫描转换算法在处理非水平、非垂直且非 45°的直线时会出现锯齿或台阶边界，如图 3-30 所示。这是由于光栅扫描显示器上显示的图像是由一系列亮度相同而面积不为零的离散像素构成。这种由离散量表示连续量而引起的失真称为走样(aliasing)。用于减轻走样现象的技术称为反走样(anti-aliasing，AA)，游戏中也称为抗锯齿。真实像素面积不为零，走样是连续图形离散为图像后引起的失真，是数字化的必然产物。走样是光栅扫描显示器的一种固有现象，只能减轻，不可避免。

图形边界是用直线表示的。图 3-31 中，理想直线扫描转换后得到一组距离直线最近的黑色像素点集。每当前一列选取的像素和后一列所选的像素位于不同行时，在显示器上就会出现一个锯齿，发生了走样。显然，只有绘制水平线、垂直线和 45°斜线时，才不会发生走样。

图 3-30　锯齿形边界

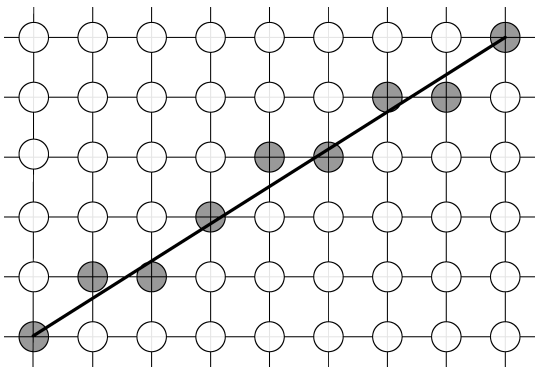
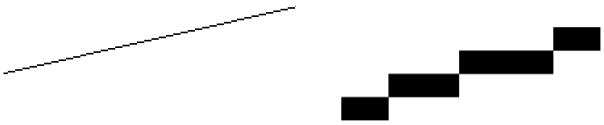


图 3-31　直线的走样现象

Windows 的附件“画图”软件绘制直线时没有进行反走样处理，如图 3-32 所示。Microsoft Office 的 Word 软件绘制直线时使用了反走样技术，如图 3-33 所示。从图 3-33(b)中可以看出，Word 软件使用了多行像素来绘制斜线，并且相邻像素的亮度等级发生了变化；而“画图”软件只使用一行像素来绘制斜线，并且像素的亮度等级保持不变。



(a) 原始图　　(b) 放大图

图 3-32　“画图”软件绘制的斜线

(a) 原始图　　(b) 放大图

图 3-33　Word 中绘制的斜线

3.4.2　反走样技术分类

反走样可以从硬件方面考虑，也可以从软件方面考虑。图 3-34 中，从硬件角度把显示器的分辨率提高了一倍。由于每个锯齿在 x 方向和 y 方向只有原分辨率的一半，所以走样现象有所减弱。虽然如此，硬件反走样技术由于受制造工艺与生产成本的限制，不可能将分辨率做得很高，很难达到理想的反走样效果。通常讲的反走样技术主要指软件反走样算法。

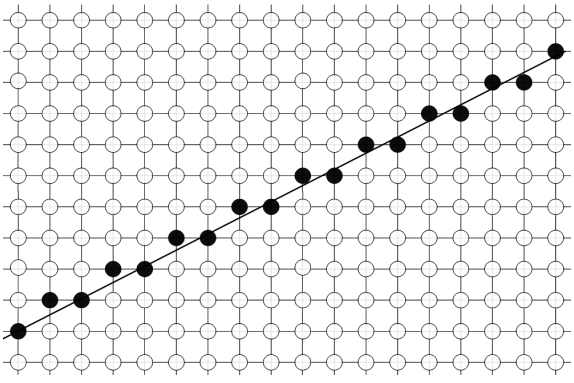
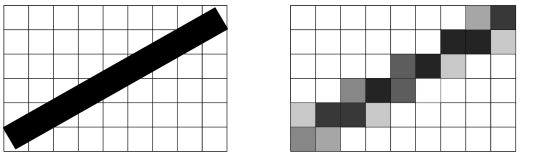


图 3-34　显示器的分辨率提高一倍的效果图

软件反走样技术主要有两种。第一种技术是超采样算法[24,25]，即在高分辨率下进行采样，在低分辨率下进行绘制。第二种技术是将像素作为一个有限区域，通过加权进行绘制，该算法的实质是利用人眼视觉特性，通过加权平均的方法，调节像素的亮度等级以产生模糊的边界，从而达到较好地消除“锯齿”的视觉效果。加权参数可以选择距离、面积和体积等。尽管这种方法可以减小锯齿边的影响，但不能处理非常小的物体所造成的“细节”闪烁走样。

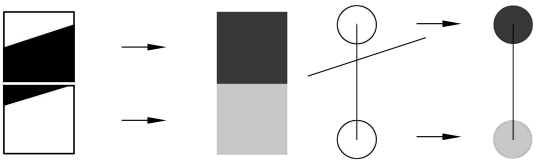
3.4.3 反走样简化模型

屏幕上所绘制的直线段不是数学意义上无宽度的理想线段，而是宽度至少为一像素单位的线段。对于主位移为 x 方向的直线，图 3-35(a)所示的黑色线条是放大的直线，其宽度为 1 像素。可以看出，该矩形在屏幕像素的每一列上通常覆盖 2 像素。也就是说，在直线所经过的每一列上，不应该只在上下两个候选像素中选择 1 像素来显示，而应该同时选择 2 像素来显示，且应该根据直线所覆盖的像素面积的大小来计算上下 2 像素的亮度值，见图 3-35(b)。直线覆盖的像素区域越大，其亮度值越小，如图 3-36(a)所示。在算法设计上，直线简化为宽度为零的理想直线，像素简化为距离为 1 个单位的小圆，如图 3-36(b)所示。图 3-35(b)简化后可用图 3-37 表示。总而言之，反走样算法是根据相邻 2 像素到理想直线的距离对亮度级别进行调节，使得外观上表现出光滑边界，向背景色融合。



(a) 直线矩形　　(b) 上下像素的亮度值变化图

图 3-35　单像素宽度的直线



(a) 正方形像素　　(b) 圆像素

图 3-36　宽度直线的像素模型

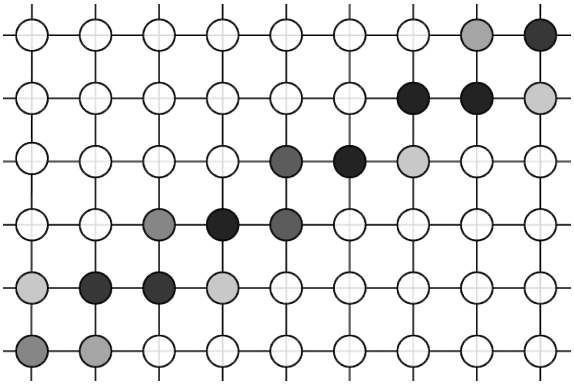


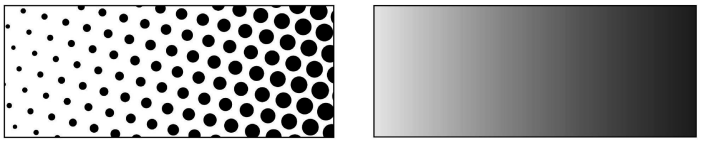
图 3-37　反走样示意图

3.5 Wu反走样算法

自从1977年Crow在计算机图形学领域中提出走样问题并给出一种解决方法以来，已经有很多反走样算法相继问世。Wu Xiaolin于1991年提出了另一种Wu反走样算法。Wu算法是对距离进行加权的反走样算法[26]。

3.5.1 算法原理

Wu反走样算法采用空间混色原理对走样现象进行修正。空间混色原理指出[27]，人眼对某一区域颜色的识别是取这个区域颜色的平均值。图3-38(a)所示的半色调点图，通过调整点的大小与间距可以在人脑中产生连续的明暗色调，如图3-38(b)所示。Wu反走样算法原理是对于理想直线上的任一点，同时用两个不同亮度等级的相邻像素来表示。



(a) 半色调点图　　(b) 连续色调

图3-38　半色调点混合为连续色调

图3-39所示的理想直线与每一列的交点，光栅化后可用与交点距离最近的上下2像素共同显示，但分别设置为不同的亮度。假定背景色为白色，直线的颜色为黑色。若像素距离交点越近，该像素的颜色就越接近直线的颜色，其亮度就越小；若像素距离交点越远，该像素的颜色就越接近背景色，其亮度就越大，但上下像素的亮度之和应等于1。

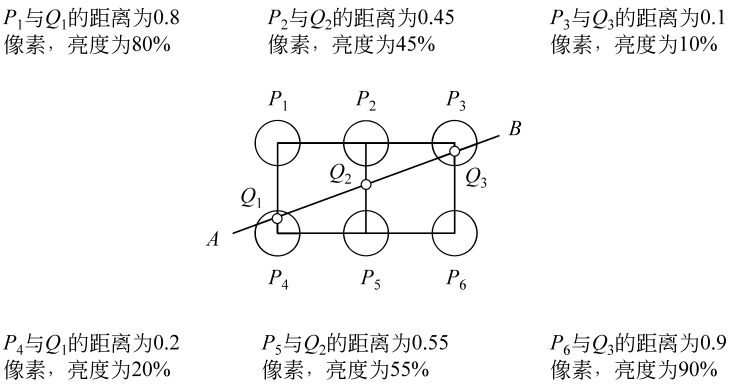


图3-39　2像素共同表示一个点

对于每一列而言，可以将下方像素 P_d 与交点 Q 之间的距离 e 作为加权参数，对上下像素的亮度等级进行调节。由于上下像素的间距为1个单位，容易知道，上方的像素 P_u 与交点的距离为 $1-e$。例如，像素 P_1 距离 Q_1 点0.8像素，该像素的亮度等级为80%；像素 P_4 距离 Q_1 点0.2像素，该像素的亮度等级为20%。同理，像素 P_2 距离 Q_2 点0.45像素，该像素的亮度等级为45%；像素 P_5 距离 Q_2 点0.55像素，该像素的亮度等级为55%；像素 P_3