第3章 基本图形的生成

计算机绘图显示有屏幕显示、打印机打印图样和绘图机输出图样等方式，其中用屏幕显示图样是计算机绘图的重要内容。常见的显示器为光栅图形显示器，可以看做像素的矩阵。像素是组成图形的基本元素，一般称为“点”。通过点亮一些像素，灭掉另一些像素，即在屏幕上产生图形。在光栅显示器上显示任何一种图形必须在显示器的相应像素点上画上所需颜色，即具有一种或多种颜色的像素集合构成图形。确定最佳接近图形的像素集合，并用指定属性写像素的过程称为图形的扫描转换或光栅化。对于一维图形，在不考虑线宽时，用一个像素宽的直、曲线来显示图形。二维图形的光栅化必须确定区域对应的像素集，并用指定的属性或图案进行显示，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。利用计算机编制图形软件时，编制基本图形元素是相当重要的，也是必需的。点是基本图形，本章主要讲述如何在指定的输出设备（如光栅图形显示器）上利用点构造其他基本二维几何图形（如点、直线、圆、椭圆、多边形域及字符串等）的算法与原理，并利用 Visual C++编程实现这些算法。

3.1 直　　线

数学上，理想的直线是由无数个点构成的集合，没有宽度。计算机绘制直线是在显示器所给定的有限个像素组成的矩阵中，确定最佳逼近该直线的一组像素，并且按扫描线顺序，对这些像素进行写操作，实现显示器绘制直线，即通常所说的直线的扫描转换，或称直线光栅化。

由于一图形中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的快。本节介绍一个像素宽直线的常用算法：数值微分画线法（DDA）、中点画线法、Bresenham 画线算法。

3.1.1 DDA（数值微分）画线算法

如图 3-1 所示，已知过端点 $p_0(x_0,y_0)$ 和 $p_1(x_1,y_1)$ 的直线段 p_0p_1；直线斜率为 $k=\dfrac{y_1-y_0}{x_1-x_0}$。DDA 算法原理：从左端点 x_0 开始，向 x 右端点步进画线，步长＝1（像素），利用 $y=kx+b$ 计算相应的 y 值，取像素点 $[x,\ \mathrm{int}(y)]$（int 为归整函数，请参阅后面的 Visual C++程序）作为当前点的坐标。考虑到计算每一个点的坐标需要做一次乘法和一次加法运算。设当前像素点 (x_i,y_i) 步长 Δx，则有 $x_{i+1}=x_i+\Delta x$，即

$$y_{i+1}=kx_{i+1}+b=kx_i+k\Delta x+b=y_i+k\Delta x \qquad (3.1)$$

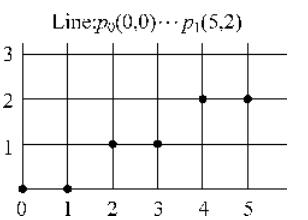

图 3-1　DDA 方法扫描转换连接两点

当 $\Delta x=1$，$y_{i+1}=y_i+k$，即当 x 每递增 1，y 递增 k（即直线斜率），这样计算过程由计算一个乘法和一个加法减少为一个加法。

上述算法仅适用于 $k\leqslant 1$ 的情形。在这种情况下，x 每增加 1，y 最多增加 1。当 $k\geqslant 1$ 时，必须把 x 和 y 地位互换，y 每增加 1，x 相应增加 $1/k$。

3.1.2 中点画线算法

中点画线的基本原理如图 3-2 所示。在画直线段的过程中，当前像素点为 $P(x_p,y_p)$，下一个像素点有两种选择，点 P_1 或点 P_2。P_1 与 P_2 中点 $M(x_{p+1},y_p+0.5)$，Q 为理想直线与 $x=x_p+1$ 线的交点。当 M 在 Q 的下方时，则下一个像素点应为 P_2；当 M 在 Q 的上方时，下一像素点应取为 P_1。

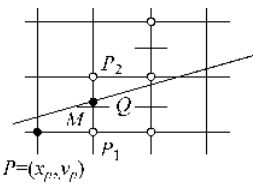

图 3-2 中点画线原理图

中点画线法的实现：令直线段 L 起点和终点分别为 (x_0,y_0) 和 (x_1,y_1)，方程式 $F(x,y)=ax+by+c=0$。其中，$a=y_0-y_1$，$b=x_1-x_0$，$c=x_0y_1-x_1y_0$；点与 L 的关系如下：

在直线上，$F(x,y)=0$；

在直线上方，$F(x,y)>0$；

在直线下方，$F(x,y)<0$。

把 M 代入 $F(x,y)$，判断 F 的符号，可知 Q 点在中点 M 的上方还是下方。为此构造判别式：

$$d=F(M)=F(x_p+1,\ y_p+0.5)=a(x_p+1)+b(y_p+0.5)+c \qquad (3.2)$$

根据当前像素计算 d 值来确定下一个像素。

当 $d<0$，L（Q 点）在 M 上方，取 P_2 为下一个像素。

当 $d>0$，L（Q 点）在 M 下方，取 P_1 为下一个像素。

当 $d=0$，选 P_1 或 P_2 均可，约定取 P_1 为下一个像素。

为提高运行效率，算法实现时，考虑 d 是 x_p、y_p 的线性函数，可采用增量计算。

当前像素点为 $P(x_p,y_p)$，当 $d<0$，L（Q 点）在 M 上方，取 P_2 为下一个像素。计算再下一个像素点：

$$\begin{aligned} d_1&=F(x_p+2,y_p+1.5)=a(x_p+2)+b(y_p+1.5)+c\\ &=[a(x_p+1)+b(y_p+0.5)+c]+a+b=d+a+b \end{aligned}$$

此时 d 增量为 $a+b$。

$d\geqslant 0$ 时，取 P_1 为下一个像素，计算再下一个像素点：

$$\begin{aligned} d_2&=F(x_p+2,y_p+0.5)=a(x_p+2)+b(y_p+0.5)+c\\ &=[a(x_p+1)+b(y_p+0.5)+c]+a=d+a \end{aligned}$$

d 增量为 a。

其中第一个像素点 (x_0,y_0) 相应 d 值为 d 的初始值 d_0。

$$\begin{aligned} d_0&=F(x_0+1,y_0+0.5)=a(x_0+1)+b(y_0+0.5)+c\\ &=(ax_0+by_0+c)+a+0.5b=F(x_0,y_0)+a+0.5b \end{aligned}$$

因为 $F(x_0,y_0)=0$，所以 $d_0=a+0.5b$。

考虑 d 的增量均为正数，用 $2d$ 代替 d 摆脱对浮点数的计算，相应 d_1 和 d_2 两个增量也分别改为 $2(a+b)$ 和 $2a$。

3.1.3 Bresenham 画线算法

Bresenham 算法克服上述画线算法设计中取整和乘积运算，使得直线扫描效率更高，是计算机图形学领域使用最广泛的直线扫描转换算法。原则是由误差项符号决定下一个像素取右边点还是右上方点。

设直线从起点(x_0,y_0)到终点(x_1,y_1)。直线方程 $y=kx+b$，$k=\frac{y_1-y_0}{x_1-x_0}=\frac{\Delta y}{\Delta x}$；讨论直线限于第一象限，如图 3-3 所示，设当前像素点为(x_i,y_i)，当直线光栅化时，x 每次都增加 1 个单元，下一个像素的列坐标为 x_i+1，行坐标为 y_i 或者递增 1 为 y_i+1，由 y 与 y_i 及 y_{i+1}的距离 d_1 及 d_2 的大小而定。计算公式为：

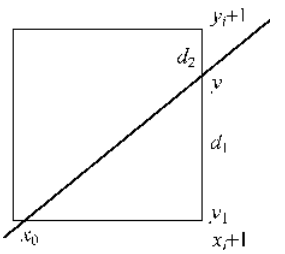

图 3-3　第一象限直线光栅化 Bresenham 算法

$$y = k(x+1)+b \tag{3.3}$$

$$d_1 = y - y_i = k(x_i+1)+b-y_i \tag{3.4}$$

$$d_2 = (y_i+1)-y = y_i+1-[k(x_i+1)+b] \tag{3.5}$$

如果 $d_1-d_2>0$，则 $y_{i+1}=y_i+1$，否则 $y_{i+1}=y_i$。

将式(3.3)、式(3.4)、式(3.5)代入 d_1-d_2，并以 $P_i=(d_1-d_2)\Delta x$，把 $k=\frac{\Delta y}{\Delta x}$代入上述等式，得：

$$P_i = 2x_i\Delta y - 2y_i\Delta x + 2\Delta y + (2b-1)\Delta x \tag{3.6}$$

d_1-d_2 是用以判断符号的误差项。由于在第一象限中，Δx 总大于 0，所以 P_i 仍旧可以用做判断符号的误差，且计算机仅包含整数运算，其中$(2b-1)\Delta x$ 的值与像素无关，在循环计算 P_i 时被消除。

直线在(x_{i+1},y_{i+1})处误差项，此时 $x_{i+1}-x_i=1$。

$$P_{i+1} = 2x_{i+1}\Delta y - 2y_{i+1}\Delta x + 2\Delta y + (2b-1)\Delta x$$

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

$$P_{i+1} = P_i + 2\Delta y - 2(y_{i+1}-y_i)\Delta x \tag{3.7}$$

取右上方像素点，$y_{i+1}-y_i=1$，式(3.7)为：

$$P_{i+1} = P_i + 2\Delta y - 2\Delta x$$

取右方像素点，$y_{i+1}-y_i=0$，式(3.7)为：

$$P_{i+1} = P_i + 2\Delta y$$

求误差的初值 P_0，可将 x_0、y_0 和 b 代入式(3.4)中的 x_i、y_i，得到：

$$P_0 = 2\Delta y - \Delta x$$

从线段起始点(x_0,y_0)开始，反复利用上述误差项公式，计算每一个 x 值对应的 y 的值。

综述上面的推导，第一象限内的直线 Bresenham 算法思想如下：

(1) 画点(x_0,y_0)，计算 $\Delta x=x_1-x_0$，$\Delta y=y_1-y_0$，误差项初值 $P_0=2\Delta y-\Delta x$，$i=0$。

(2) 求直线的下一点位置 $x_{i+1}=x_i+1$，如果 $P_i>0$，则 $y_{i+1}=y_i+1$，否则 $y_{i+1}=y_i$。

(3) 画点(x_{i+1},y_{i+1})。

(4) 求下一个误差 P_{i+1}，如果 $P_i>0$，则 $P_{i+1}=P_i+2\Delta y-2\Delta x$，否则 $P_{i+1}=P_i+2\Delta y$。

(5) $i=i+1$，如果 $i<\Delta x+1$ 则转步骤(2)，否则结束操作。

3.1.4 程序设计

1. 程序设计功能说明

为实现上述算法,本程序利用最基本的绘制元素(如点、直线等)绘制图形。图 3-4 所示为程序运行主界面,通过选择菜单及下拉菜单的各功能项分别完成各种对应算法的图形绘制。

图 3-4 基本图形生成的程序运行界面

2. 创建工程名称为"基本图形的生成"单文档应用程序框架

创建过程详见第 2 章的 2.2.3 节。

3. 编辑菜单资源

设计如图 3-4 所示的菜单项。在工作区的 ResourceView 标签中,单击 Menu 项左边"+",然后双击其子项 IDR_MAINFRAME,并根据表 3-1 中的定义编辑菜单资源。此时 VC 已自动建好程序框架。

表 3-1 菜单资源表

菜单标题	菜单项标题	标示符 ID
直线	DDA 算法生成直线	ID_DDALINE
	Bresenham 算法生成直线	ID_BRESENHAMLINE
	中点算法生成直线	ID_MIDPOINTLINE

4. 添加消息处理函数

利用 ClassWizard(建立类向导)为应用程序添加与菜单项相关的消息处理函数,ClassName 栏中选择 CMyView,根据表 3-2 建立如下的消息映射函数,ClassWizard 会自动完成有关的函数声明。

表 3-2 菜单项的消息处理函数

菜单项 ID	消 息	消息处理函数
ID_DDALINE	CONMMAN	OnDdaline
ID_MIDPOINTLINE	CONMMAN	OnMidpointline
ID_BRESENHAMLINE	CONMMAN	OnBresenhamline

5. 程序结构代码

在 CMyView.cpp 文件中相应位置添加如下代码:

```
// DDA算法生成直线
void CMyView:: OnDdaline()
{
CDC * pDC = GetDC();                                    //获得设备指针
int xa = 100,ya = 300,xb = 300,yb = 200,c = RGB(255,0,0);  //定义直线的两端点,直线颜色
  int x,y;
  float dx, dy, k;
  dx = (float)(xb - xa), dy = (float)(yb - ya);
  k = dy/dx, y = ya;
  if(abs(k)<1)
  {
  for (x = xa;x<= xb;x++ )
```

```
  {pDC->SetPixel(x,int(y+0.5),c);
  y=y+k;}
  }
  if(abs(k)>=1)
  {
  for(y=ya;y<=yb;y++)
  {pDC->SetPixel(int(x+0.5),y,c);
  x=x+1/k;}
  }
  ReleaseDC(pDC);
}
```

说明：

(1) 以上代码理论上通过定义直线的两端点，可得到任意端点之间的一直线，但由于一般屏幕坐标采用右手系坐标，屏幕上只有正的 x，y 值。

(2) 注意上述程序考虑到当 $k\leqslant 1$ 的情形，x 每增加 1，y 最多增加 1；当 $k>1$ 时，y 每增加 1，x 相应增加 $1/k$。在这个算法中，y 与 k 用浮点数表示，而且每一步都要对 y 进行四舍五入后取整。

```
//中点算法生成直线
void CMyView::OnMidpointline()
{
CDC* pDC=GetDC();
int xa=300, ya=200, xb=450, yb=300,c=RGB(0,255,0);
 float a, b, d1, d2, d, x, y;
  a=ya-yb, b=xb-xa, d=2*a+b;
  d1=2*a, d2=2*(a+b);
  x=xa, y=ya;
  pDC->SetPixel(x, y, c);
  while(x<xb)
  { if(d<0)   {x++, y++, d+=d2; }
    else      {x++, d+=d1;}
    pDC->SetPixel(x, y, c);
  }
ReleaseDC(pDC);
}
```

说明：

程序中只利用 d 的符号、d 的增量都是整数，只是初始值包含小数，用 2d 代替 d，使程序中仅包含整数的运算。

```
//Bresenham 算法生成直线
  void CMyView::OnBresenhamline()
  {
      CDC* pDC=GetDC();
      int x1=100, y1=200, x2=350, y2=100,c=RGB(0,0,255);
      int i,s1,s2,interchange;
      float x,y,deltax,deltay,e,temp;
      x=x1;
      y=y1;
```

```
        deltax = abs(x2 - x1);
        deltay = abs(y2 - y1);
        if(x2 - x1>= 0) s1 = 1; else s1 = - 1;
        if(y2 - y1>= 0) s2 = 1; else s2 = - 1;
        if(deltay>deltax){
        temp = deltax;
        deltax = deltay;
        deltay = temp;
        interchange = 1;
    }
    else interchange = 0;
    f = 2 * deltay - deltax;
    pDC ->SetPixel(x,y,c);
    for(i = 1;i<= deltax;i ++ ){
        if(f>= 0){
                if(interchange == 1) x += s1;
                else y += s2;
                pDC ->SetPixel(x,y,c);
                f = f - 2 * deltax;
        }
        else{
                if(interchange == 1) y += s2;
                else x += s1;
                f = f + 2 * deltay;
        }
    }
}
```

说明:

(1) 以上程序已经考虑到所有象限直线的生成。

(2) Bresenham 算法的优点如下:

① 不必计算直线的斜率,因此不做除法。

② 不用浮点数,只用整数。

③ 只做整数加减运算和乘 2 运算,而乘 2 运算可以用移位操作实现。

④ Bresenham 算法的运算速度很快。

3.2 圆

圆是绘图软件中基本元素,一般绘图软件中都包含绘制圆和一段圆弧的。下面给出画圆和一段曲线(圆弧)的算法。

3.2.1 直角坐标画圆算法

圆是离圆心(x_c, y_c)距离为半径 r 点的集合。

直角坐标系中圆的方程为:

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

由上式导出:

$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$$

当 $x-x_c$ 从 $-r$ 到 r 做加 1 递增时，就可以求出对应的圆周点的 y 坐标。但是这样求出的圆周上的点是不均匀的，$|x-x_c|$ 越大，对应生成圆周点之间的圆周距离也就越长。因此，所生成的圆不美观。

3.2.2 中点画圆算法

如图 3-5 所示，讨论圆心在原点，半径为 r 的圆的第二个八分圆。考虑生成的圆从 $(0,r)$ 到 $(r/\sqrt{2},r/\sqrt{2})$ 顺时针绘制最佳逼近该圆弧的像素序列。假设已经确定一点 $P=(x_p,y_p)$，怎样确定 P_1 或 P_2 哪一个为下一个像素点。

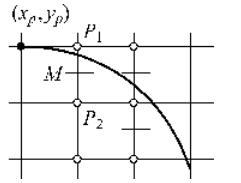

图 3-5 中点画圆法示意图

设函数为 $F(x,y)=x^2+y^2-r^2$ 的构造圆，圆上的点为 $F(x,y)=0$，圆外的点 $F(x,y)>0$，圆内的点 $F(x,y)<0$，M 为 P_1P_2 线段中点，即 $M=(x_p+1,y_p-0.5)$ 构造判别式：

$$d=F(M)=F(x_p+1,\ y_p-0.5)=(x_p+1)^2+(y_p-0.5)^2-r^2 \tag{3.8}$$

若 $d<0$，则应取 P_1 为下一像素，再下一像素的判别式为：

$$d=F(x_p+2,\ y_p-0.5)=(x_p+2)^2+(y_p-0.5)^2-r^2=d+2x_p+3$$

判别式增量为：

$$2x_p+3$$

若 $d\geqslant 0$，则应取 P_2 为下一像素，再下一像素的判别式为：

$$d=F(x_p+2,\ y_p-1.5)=(x_p+2)^2+(y_p-1.5)^2-r^2=d+2(x_p-y_p)+5$$

判别式增量为：

$$2(x_p-y_p)+5$$

这里第一个像素是 $(0,\ r)$，判别式 d 的初始值为：

$$d_0=F(1,\ r-0.5)=1.25-r$$

算法实现时，考虑初值 d_0 为浮点数，但是其增量均为整数，为减少计算量，提高生成速度，可以令 $d_0=1-r$，判别式的 $d<0$ 也相应变为 $d<0.25$。

3.2.3 Bresenham 画圆算法

设圆的半径为 r，考虑圆心在 $(0,0)$，从 $x=0$、$y=r$ 开始的顺时针方向的 1/8 圆周的生成过程。x 每步增加 1，从 $x=0$ 开始，到 $x=y$ 结束，即有 $x_{i+1}=x_i+1$；相应的，y_{i+1} 则在两种可能中选择：$y_{i+1}=y_i$ 或者 $y_{i+1}=y_i-1$。选择的原则是考察精确值 y 是靠近 y_i 还是靠近 y_i-1（见图 3-6），计算式为：

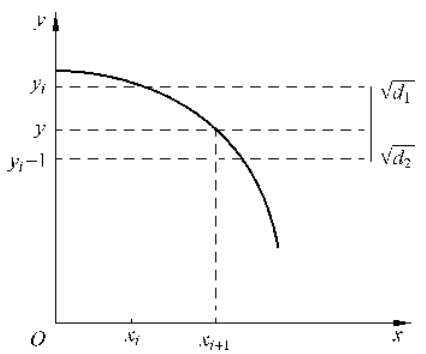

图 3-6 确定 y 的位置

$$y^2=r^2-(x_i+1)^2$$

$$d_1=y_i^2-y^2=y_i^2-r^2+(x_i+1)^2$$

$$d_2=y^2-(y_i-1)^2=r^2-(x_i+1)^2-(y_i-1)^2$$

令 $p_i=d_1-d_2$，并代入 d_1、d_2，则有

$$p_i=2(x_i+1)^2+y_i^2+(y_i-1)^2-2r^2 \tag{3.9}$$

p_i 称为误差项。如果 $p_i<0$，下一个像素点 $y_{i+1}=y_i$，

否则 $y_{i+1}=y_i-1$(其中 $p_i=0$ 选择 $y_{i+1}=y_i-1$ 为约定)。

p_i 的递归式为:

$$p_{i+1}=p_i+4x_i+6+2(y_i^2+1-y_i^2)-2(y_i+1-y_i) \tag{3.10}$$

p_i 的初值由式(3.9)代入 $x_i=0$,$y_i=r$,得:

$$p_0=2(0+1)^2+r^2+(r-1)^2=3-2r \tag{3.11}$$

根据上面的推导,圆周生成算法思想如下:

(1) 求误差初值,$p_0=3-2r$,$i=1$,画点$(0,r)$。

(2) 求下一个光栅位置,其中 $x_{i+1}=x_i+1$,如果 $p_i<0$ 则 $y_{i+1}=y_i$,否则 $y_{i+1}=y_i-1$。

(3) 画点(x_{i+1},y_{i+1})。

(4) 计算下一个误差,如果 $p_i<0$ 则 $p_{i+1}=p_i+4x_i+6$,否则 $p_{i+1}=p_i+4(x_i-y_i)+10$。

(5) $i=i+1$,如果 $x=y$ 则结束,否则返回步骤(2)。

程序设计步骤如下:

(1) 创建应用程序框架,以上面建立的单文档程序框架为基础。

(2) 编辑菜单资源。

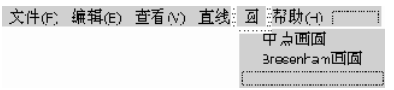

图 3-7　程序主菜单

在工作区的 ResourceView 标签中,单击 Menu 项左边"+",然后双击其子项 IDR_MAINFRAME,并根据表 3-3 中的定义添加编辑菜单资源。此时建好的菜单如图 3-7 所示。

表 3-3　菜单资源表

菜 单 标 题	菜单项标题	标示符 ID
圆	中点画圆	ID_MIDPOINTCIRCLE
	Bresenham 画圆	ID_BRESENHAMCIRCLE

(3) 添加消息处理函数。

利用 ClassWizard(建立类向导)为应用程序添加与菜单项相关的消息处理函数,ClassName 栏中选择 CMyView,根据表 3-4 建立如下的消息映射函数,ClassWizard 会自动完成有关的函数声明。

表 3-4　菜单项的消息处理函数

菜单项 ID	消　　息	消息处理函数
ID_MIDPOINTCIRCLE	CONMMAN	OnMidpointcircle
ID_BRESENHAMCIRCLE	CONMMAN	OnBresenhamcircle

(4) 程序结构代码,在 CMyView.cpp 文件中的相应位置添加如下代码。

```
void CMyView::OnMidpointcircle()
//中点算法绘制圆,如图 3-8 所示
{
  // TODO: Add your command handler code here
  CDC * pDC = GetDC();
  int xc = 300, yc = 300, r = 50, c = 0;
  int x,y;
  float d;
```

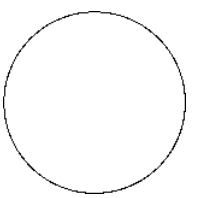

图 3-8　中点算法绘制圆

```
    x = 0; y = r; d = 1.25 - r;
    pDC->SetPixel((xc + x),(yc + y),c);
    pDC->SetPixel((xc - x),(yc + y),c);
    pDC->SetPixel((xc + x),(yc - y),c);
    pDC->SetPixel((xc - x),(yc - y),c);
    pDC->SetPixel((xc + y),(yc + x),c);
    pDC->SetPixel((xc - y),(yc + x),c);
    pDC->SetPixel((xc + y),(yc - x),c);
    pDC->SetPixel((xc - y),(yc - x),c);
    while(x<=y)
    {   if(d<0)   d += 2 * x + 3;
       else   { d += 2 * (x - y) + 5; y--;}
       x++;
      pDC->SetPixel((xc + x),(yc + y),c);
      pDC->SetPixel((xc - x),(yc + y),c);
      pDC->SetPixel((xc + x),(yc - y),c);
      pDC->SetPixel((xc - x),(yc - y),c);
      pDC->SetPixel((xc + y),(yc + x),c);
      pDC->SetPixel((xc - y),(yc + x),c);
      pDC->SetPixel((xc + y),(yc - x),c);
      pDC->SetPixel((xc - y),(yc - x),c);
    }
}
void CMyView::OnBresenhamcircle()  // Bresenham 算法绘制圆,如图 3-9 所示
{
    CDC * pDC = GetDC();
    int xc = 100, yc = 100, radius = 50, c = 0;
    int x = 0,y = radius,p = 3 - 2 * radius;
    while(x<y)
    {
       pDC->SetPixel(xc + x, yc + y, c);
       pDC->SetPixel(xc - x, yc + y, c);
       pDC->SetPixel(xc + x, yc - y, c);
       pDC->SetPixel(xc - x, yc - y, c);
       pDC->SetPixel(xc + y, yc + x, c);
       pDC->SetPixel(xc - y, yc + x, c);
       pDC->SetPixel(xc + y, yc - x, c);
       pDC->SetPixel(xc - y, yc - x, c);
       if (p<0)
          p = p + 4 * x + 6;
       else
       {
          p = p + 4 * (x - y) + 10;
          y -= 1;
       }
       x += 1;
    }
    if(x == y)
    pDC->SetPixel(xc + x, yc + y, c);
    pDC->SetPixel(xc - x, yc + y, c);
    pDC->SetPixel(xc + x, yc - y, c);
    pDC->SetPixel(xc - x, yc - y, c);
    pDC->SetPixel(xc + y, yc + x, c);
```

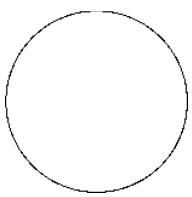

图 3-9　Bresenham 算法绘制圆

```
    pDC->SetPixel(xc - y, yc + x, c);
    pDC->SetPixel(xc + y, yc - x, c);
    pDC->SetPixel(xc - y, yc - x, c);
}
```


3.3 椭圆扫描转换中点算法

中点画圆算法可以推广到椭圆和一般二次曲线的生成。下面讨论椭圆的扫描转换中点算法,设椭圆为中心在坐标原点的标准椭圆,其方程为:

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

即
$$F(x,\ y)=b^2x^2+a^2y^2-a^2b^2=0$$

(1) 对于椭圆上的点,有 $F(x,\ y)=0$。

(2) 对于椭圆外的点,有 $F(x,\ y)>0$。

(3) 对于椭圆内的点,有 $F(x,\ y)<0$。

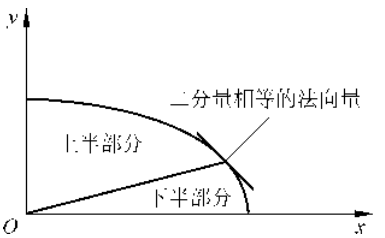

图 3-10 第一象限的椭圆弧

以弧上斜率为-1的点作为分界将第一象限椭圆弧分为上下两部分(如图 3-10 所示)。

法向量:

$$N(x,y)=\frac{\partial F}{\partial x}\mathrm{i}+\frac{\partial F}{\partial y}\mathrm{j}=2b^2x\mathrm{i}+2a^2y\mathrm{j}$$

$$b^2(x_i+1)<a^2(y_i-0.5)$$

而在下一个点,不等号改变方向,则说明椭圆弧从上部分转入下部分。

与中点绘制圆算法类似,一个像素确定后,在下面两个候选像素点的中点计算一个判别式的值,再根据判别式符号确定离椭圆最近的点。先看椭圆弧的上半部分,具体算法如下:

假设横坐标为 x_p 的像素中与椭圆最近点为$(x_p,\ y_p)$,下一对候选像素的中点应为$(x_p+1,\ y_p-0.5)$,判别式为:

$$d_1=F(x_p+1,y_p-0.5)=b^2(x_p+1)^2+a^2(y_p-0.5)^2-a^2b^2$$

若 $d_1<0$,表明中点在椭圆内,应取正右方像素点,再下一个像素点判别式为:

$$d_1'=F(x_p+2,y_p-0.5)=b^2(x_p+2)^2+a^2(y_p-0.5)^2-a^2b^2=d_1+b^2(2x_p+3)$$

判别式增量为 $b^2(2a_p+3)$。

若 $d_1\geqslant 0$,表明中点在椭圆外,应取右下方像素点,再下一个像素点判别式为:

$$\begin{aligned}d_1'&=F(x_p+2,y_p-1.5)=b^2(x_p+2)^2+a^2(y_p-1.5)^2-a^2b^2\\&=d_1+b^2(2x_p+3)+a^2(-2y_p+2)^2\end{aligned}$$

判别式增量为:

$$b^2(2x_p+3)+a^2(-2y_p+2)^2$$

判别式 d_1 的初始条件确定。椭圆弧起点为$(0,\ b)$,第一个中点为$(1,b-0.5)$,对应判别式为:

$$d_{10}=F(1,b-0.5)=b^2+a^2(b-0.5)^2-a^2b^2=b^2+a^2(-b+0.25)$$

在扫描转换椭圆的上半部分时,在每步迭代中需要比较法向量的两个分量来确定核实从上半部分转到下半部分。在下半部分算法有些不同,要从正上方和右下方两个像素中选

择下一个像素。在从上半部分转到下半部分时，还需要对下半部分的中点判别式进行初始化。即若上半部分所选择的最后一个像素点为(x_p, y_p)，则下半部分中点判别式应在$(x_p+0.5, y_p-1)$的点上计算。其在正下方与右下方的增量计算同上半部分。具体算法的实现请参考下面的程序设计。

对于一般椭圆方程：

$$\frac{(x-a_c)^2}{a^2}+\frac{(y-b_c)^2}{b^2}=1$$

其中，(a_c, b_c)为椭圆中心，为此先把(a_c, b_c)移到坐标原点，根据上述方法确定好标准椭圆各像素点，然后再平移到(a_c, b_c)。

对于长轴和短轴不与坐标轴平行的倾斜椭圆，可采用本书后面 4.2 节的方法绕中心点进行旋转变换，计算变换矩阵，再用本节所叙述生成椭圆方法得到各像素点集，然后用变换矩阵乘以这些点集即可绘制倾斜圆。

标准椭圆程序设计步骤如下。

(1) 创建应用程序框架，以上面建立的单文档程序框架为基础。

(2) 编辑菜单资源。

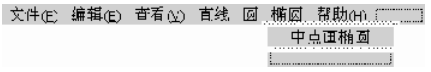

图 3-11　程序主菜单

在工作区的 ResourceView 标签中，单击 Menu 项左边“+”，然后双击其子项 IDR_MAINFRAME，并根据表 3-5 中的定义添加编辑菜单资源。此时建好的菜单如图 3-11 所示。

表 3-5　菜单资源表

菜 单 标 题	菜单项标题	标示符 ID
椭圆	中点画椭圆	ID_MIDPOINTELLISPE

(3) 添加消息处理函数。

利用 ClassWizard(建立类向导)为应用程序添加与菜单项相关的消息处理函数，ClassName 栏中选择 CMyView，根据表 3-6 建立如下的消息映射函数，ClassWizard 会自动完成有关的函数声明。

表 3-6　菜单项的消息处理函数

菜单项 ID	消　　息	消息处理函数
ID_MIDPOINTELLISPE	CONMMAN	OnMidpointellispe

(4) 程序结构代码如下：

```
void CMyView::OnMidpointellispe()      //中点算法绘制椭圆，如图 3-12 所示
{
    CDC* pDC = GetDC();
    int a = 200,b = 100,xc = 300,yc = 200,c = 0;
    int x,y;
    double d1,d2;
    x = 0;y = b;
    d1 = b * b + a * a * ( - b + 0.25);
    pDC->SetPixel(x + 300,y + 200,c);
```

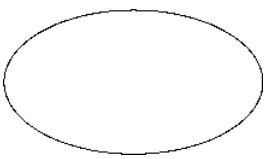

图 3-12　中点算法绘制椭圆