第一章 基本图形的生成

计算机图形学已成为计算机领域发展最快的领域，同时计算机图形软件也相应得到快速发展。计算机绘图显示有屏幕显示、打印机打印屏幕上的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。

计算机上常见的显示器为光栅图形显示器，光栅图形显示器可以看作像

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，在显示器的相应像素点上画上所需颜色的像素点，即具有一种或多种颜色的像素集合构成图形。确定最佳逼近图形的像素集合，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）上利用点构造其它基本二维几何图形（点、直线、圆、椭圆、多边形域、字符串等）的算法与原理，并利用 Visual C++编程实现这些算法。**1.1 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

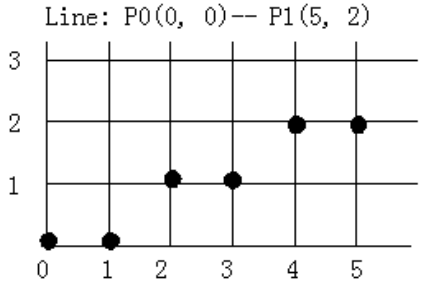


图 1-1　DDA 方法扫描转换连接两点

显示器绘制直线，即通常所用说直线的扫描转换，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。本节我们介绍一个像素宽直线的常用算法：数值微分法（**DDA**）、中点画线法、**Bresenham** 算法。

1.1.1 DDA (数值微分)算法

一．DDA 算法原理

如图 **1-1** 所示，已知过端点 $p_0(x_0,y_0), p_1(x_1,y_1)$ 的直线段 p_0p_1；直线斜率为 $k=\frac{y_1-y_0}{x_1-x_0}$，从 x 的左端点 x_0 开始，向 x 右端点步进画线，步长**=1(**个像素**)**，

计算相应的 y 坐标 $y = kx + B$；取像素点 $(x, round(y))$ 作为当前点的坐标。计算 $y_{i+1} = kx_{i+1} + B = kx_1 + B + k\Delta x = y_1 + k\Delta x$ 当□ $x = 1$, $y_{i+1} = y_i + k$ 即：当 x 每递增 1，y 递增 k(即直线斜率)；

注意：上述分析的算法仅适用于□k□≤1 的情形。在这种情况下，x 每增加 1,y 最多增加 1。当 $k \geq 1$ 时，必须把 x，y 地位互换，y 每增加 1，x 相应增加 1/k（请参阅后面的 Visual C++程序）。

1.1.2 生成直线的中点画线法

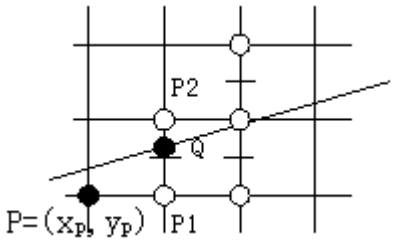


图 1-2 中点画线法每步迭代涉及的像素和中点示意图

中点画线法的基本原理，如图 1-2 所示，在画直线段的过程中，当前像素点为 p，下一个像素点有两种选择，点 p_1 或 p_2 。M 为 p_1 与 p_2 中点，Q 为理想直线与 $x=x_p+1$ 垂线的交点。当 M 在 Q 的下方，则 P_2 应为下一个像素点；M 在 Q 的上方，应取 P_1 为下一点。

中点画线法的实现。令直线段 $L(p_0(x_0,y_0), p_1(x_1, y_1))$，其方程式 $F(x,y)=ax+by+c=0$。

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：

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

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，取 P_1 为下一个像素；

其中 d 是 x_p，y_p的线性函数。

1.1.3 Bresenham 算法

一．Bresenham 算法原理

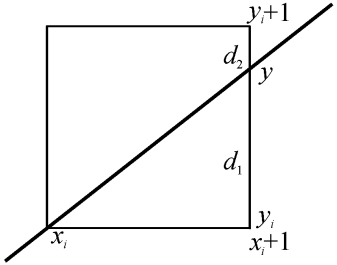


图 1-3 第一象限直线光栅化 Bresenham 算法

Bresenham 算法是计算机图形学领域使用最广泛的直线扫描转换算法。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_1, y_1)到终点(x_2, y_2)。直线可表示为方程 $y = mx+b$，其中 b=y1-m*x1 ,m=（y2-y1）/(x2-x1)=dy/dx； 此处的讨论直线方向限

于 1*a* 象限(图 1-3)，当直线光栅化时，*x* 每次都增加 1 个单元，设 x 像素为 (x_i, y_i)。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

$$y=m(x_i+1)+b \qquad (1.1)$$

$$d_1 = y - y_i \qquad (1.2)$$

$$d_2 = y_i + 1 - y \qquad (1.3)$$

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

式(1.1)、(1.2)、(1.3)代入 d_1-d_2，再用 dx 乘等式两边，并以 $P_i=(d_1-d_2)\,dx$ 代入上述等式，得

$$P_i = 2x_i dy-2y_i dx+2dy+(2b-1)\,dx \qquad (1.4)$$

d_1-d_2 是用以判断符号的误差。由于在 1a 象限，dx 总大于 0，所以 P_i 仍旧可以用作判断符号的误差。P_{i+1} 为

$$P_{i+1} = P_i+2dy-2(y_{i+1}-y_i)\,dx \qquad (1.5)$$

求误差的初值 P_1，可将 x_1、y_1 和 b 代入式(2.4)中的 x_i、y_i 而得到

$$P_1 = 2dy-dx$$

综述上面的推导，第 1*a* 象限内的直线 Bresenham 算法思想如下：

- 1. 画点(x_1，y_1)，$dx=x_2-x_1$，$dy=y_2-y_1$，计算误差初值 $P_1=2dy-dx$，$i=1$；
- 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 $P_{i+1}=P_i+2dy-2dx$，否则

$P_{i+1}=P_i+2dy$;

l 5. $i=i+1$；如果 $i<dx+1$ 则转步骤 2；否则结束操作。

1.1.4 程序设计

一、程序设计功能说明

为编程实现上述算法，本程序利用最基本的绘制元素（如点、直线等），绘制图形。如图 1-4 所示，为程序运行主界面，通过单击菜单及下拉菜单的各功能项分别完各种对应算法的图形绘制。

图 **1-4**

二、创建“工程名称为基本图形的生成”单文档应用程序框架

（1）启动 VC，选择【文件】|【新建】菜单命令，并在弹出的新建对话框中单击【工程】标签。

（2）选择【MFC AppWizard(exe)】，在【工程名称】编辑框中输入 “基

本图形的生成”作为工程名称，单击【确定】按钮，出现 Step 1 对话框。

（3）选择【单个文档】选项，单击【下一个】出现 Step 2 对话框。

（4）接受缺省选项，单击【下一个】，在一下出现的 Step 3- Step 5 对话框，接受缺省选项，单击【下一个】按钮。

（5）在 Step 6 对话框中单击【完成】按钮，即完成“基本图形的生成”应用程序的所有选项，随后出现工程信息对话框（记录以上步骤各选项选择情况），如图 1-5 所示，单击【确定】完成应用程序框架的创建。


新建工程信息
AppWizard will create a new skeleton project with the following specifications:
Application type of 基本图形的生成:
Single Document Interface Application targeting:
Win32
Classes to be created:
Application: CMyApp in 基本图形的生成.h and 基本图形的生成.cpp
Frame: CMainFrame in MainFrm.h and MainFrm.cpp
Document: CMyDoc in 基本图形的生成Doc.h and 基本图形的生成Doc.cpp
View: CMyView in 基本图形的生成View.h and 基本图形的生成View.cpp
Features:
+ Initial toolbar in main frame
+ Initial status bar in main frame
+ Printing and Print Preview support in view
+ Uses shared DLL implementation (MFC42.DLL)
+ ActiveX Controls support enabled
+ Localizable text in:
中文[中国]
工程目录:
D:\HOF\计算机图形学BOOK\基本图形的生成
确定
取消

图 1-5

2． 编辑菜单资源

设计如图 1-4 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 1.1 表中的定义编辑菜单资源。此时 VC 已自动建好的程序框架如图 1-5 所示。

表 1.1 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

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 1.2 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 1.2 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

4. 程序结构代码，在 CMyView.cpp 文件中相应位置添加如下代码：

代码见纸书

// DDA 算法生成直线

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，屏幕坐标与窗口坐标之间转换知识请参考第三章。

2．注意上述程序考虑到当 **k** ≤**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。

//中点算法生成直线代码见纸书

说明：

1. 其中 **d** 是 x_p，y_p 的线性函数，为了提高运算效率程序中采用增量计算。具体算法如下：若当前像素处于 **d>0** 情况，则取正右方像素 **P1**（x_p+1，y_p），判断下一个像素点的位置，应计算 $d_1=F(x_p+2, y_p+0.5)=a(x_p+2)+b(y_p+0.5)=d+a$；其中增量为 **a**；若 **d<0** 时，则取右上方像素 **P2**（x_p+1，y_p+1）。判断再下一像素，则要计算 $d_2=F(x_p+2,y_p+1.5)=a(x_p+2)+b(y_p+1.5)+c=d+a+b$，增量为 **a**＋**b**。

2. 画线从(x_0, y_0)开始，d 的初值 $d_0=F(x_0+1, y_0+0.5)=F(x_0, y_0)+a+0.5b$ 因 $F(x_0, y_0)=0$，则 $d_0=a+0.5b$。

3. 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

代码见纸书

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。

1.2 圆

给出圆心坐标(x_C, y_C)和半径 r，逐点画出一个圆周的公式有下列两种：**1.2.1 直角坐标法** $(x-x_C)^2 + (y-y_C)^2 = r^2$ 由上式导出：

$y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$ 当 $x-x_C$ 从$-r$ 到 r 作加 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。**1.2.1** 中点画圆法

如图 **1-7** 所示，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 $F(x,y)<0$，构造判别式

$$d=F(M)=F(x_p+1,y_p-0.5)=(x_p+1)^2+(y_p-0.5)^2$$

若 $d<0$ 则应取 **P1** 为下一像素，而且再下一像素的判别式为

$$d=F(x_p+2,y_p-0.5)=(x_p+2)^2+(y_p-0.5)^2-R^2=d+$$

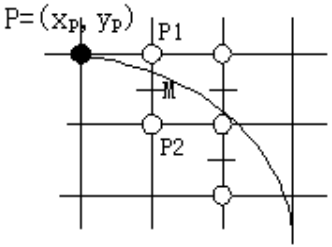


图 1-7 中点画圆法

$d \geq 0$ 若 则应取 P2 为下一像素，而且下一像素的判别式为

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

我们讨论按顺时针方向生成第二个八分圆，则第一个像素是（0,R），判别式 d 的初始值为 $d_0 = F(1, R - 0.5) = 1.25 - R$

1.2.3 圆的 Bresenham 算法

设圆的半径为 *r*。先考虑圆心在(0, 0)，并从 *x*=0、*y*=*r* 开始的顺时针方向的 1/8 圆周的生成过程。在这种情况下，*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 $y_i - 1$(图 1-8)，计算式为

$$y^2 = r^2 - (x_i+1)^2 d_1 = y_i^2 - y^2 = y_i^2 - r^2 + (x_i+1)^2 d_2 = y^2 - (y_i - 1)^2 =$$

$r^2-(x_i+1)^2-(y_i-1)^2$

令 $p_i=d_1-d_2$，并代入 d_1、d_2，则有

$$p_i = 2(x_i+1)^2 + y_i^2 + (y_i-1)^2-2r^2 \qquad (1.6)$$

p_i 称为误差。如果 $p_i<0$ 则 $y_{i+1}=y_i$，否则 $y_{i+1}=y_i-1$。

p_i 的递归式为

$$p_{i+1} = p_i + 4x_i + 6 + 2(y_i^2+1- y_i^2) - 2(y_{i+1}-y_i) \qquad (1.7)$$

p_i 的初值由式(1.6)代入 $x_i=0$，$y_i=r$ 而得

$$p_1 = 3-2r \qquad (1.8)$$

根据上面的推导，圆周生成算法思想如下：

1. 求误差初值，$p_1=3-2r$，$i=1$，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。

二．程序设计

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

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 1.3 表中的添加编辑菜单资源。

此时建好的菜单如图 1-9 所示。

表 1.3 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

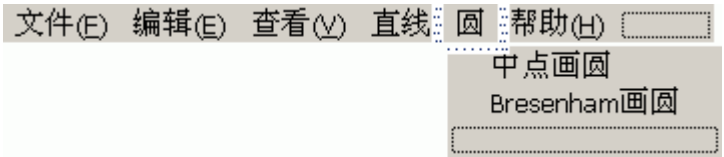


图 1-9

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 1.4 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 1.4 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 文件中相应位置添加如下代码：

1.3 椭圆的扫描转换 1.3.1 椭圆的特征

$$F(x,y)=b^2x^2+a^2y^2-a^2b^2=0$$

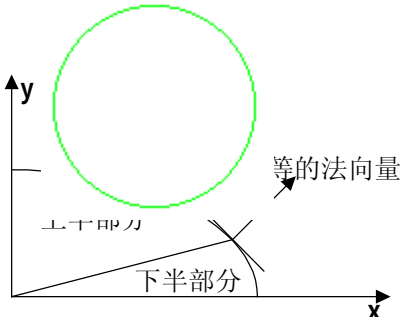


图 1-9 第一象限的椭圆弧

- 对于椭圆上的点，有 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。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 1-9 所示）。

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

引理：若在

当前点，法向量的 y 分量比 x 分量大，即

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

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。

3.3.2 椭圆的中点

Bresenham 算法算法原理推导上半部分的椭圆绘制公式

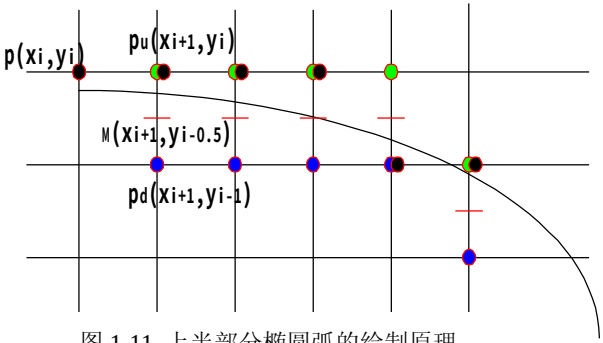


图 1-11 上半部分椭圆弧的绘制原理

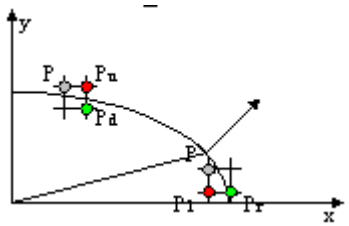
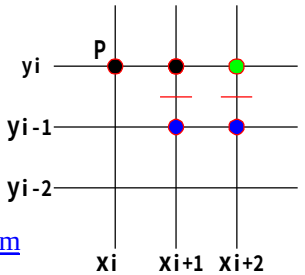


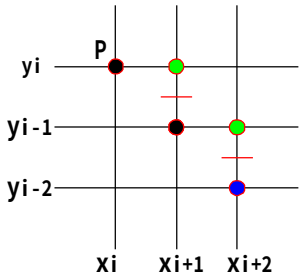
图 1-10 Bresenham 椭圆绘制算法原理

判别式• 若 $d_1 \leqslant 0$，取 $P_u(x_i+1, y_i)$

• 若 $d_1>0$，取



(a) d<=0的情况



(b) d>0的情况

 CAD 教育网制作www.cadedu.com

$P_d(x_i+1,y_i-1)$

误差项的递推

$$d_1 = F(x_i+2, y_i-0.5) = b^2(x_i+2)^2 + a^2(y_i-0.5)^2 - a^2b^2$$
$$= b^2(x_i+1)^2 + +a^2(y_i-0.5)^2 - a^2b^2 + b^2(2x_i+3)$$
$$= d_1 + b^2(2x_i+3)$$

$d_1 \leqslant 0$： $d_1 > 0$：

$$d_1 = F(x_i+2, y_i-1.5) = b^2(x_i+2)^2 + a^2(y_i-1.5)^2 - a^2b^2$$
$$= b^2(x_i+1)^2 + a^2(y_i-0.5)^2 - a^2b^2 + b^2(2x_i+3) + a^2(-2y_i+2)$$
$$= d_1 + b^2(2x_i+3) + a^2(-2y_i+2)$$

推导椭圆弧下半部分的绘制公式

$$d_2 = F(x_i + 0.5, y_i - 1) = b^2(x_i + 0.5)^2 + a^2(y_i - 1)^2 - a^2b^2$$

原理判别式

判别式的初始值

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

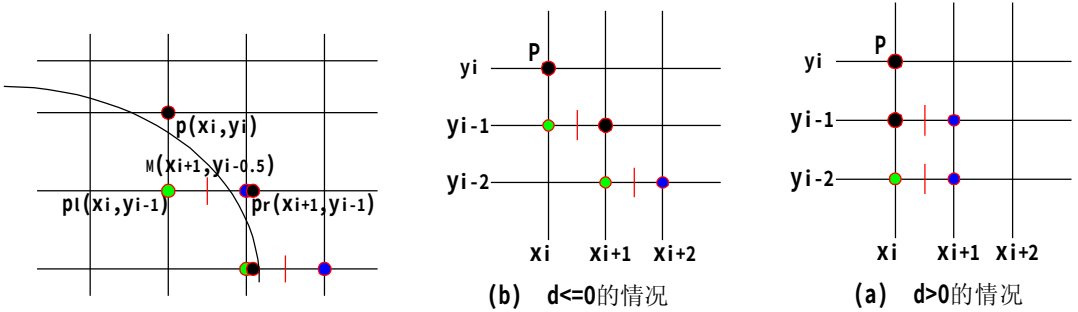


图 1-13 下半部分椭圆弧的绘制原理

- 若 $d_2>0$，取 $P_l(x_i,y_i-1)$
- 若 $d_2\leqslant 0$，取 $P_r(x_i+1,y_i-1)$

误差项的递推

$d_2\leqslant 0$：

$$\begin{aligned} d_2 &= F(x_i+0.5,y_i-2)=b^2(x_i+0.5)^2+a^2(y_i-2)^2-a^2b^2 \\ &= b^2(x_i+0.5)^2+a^2(y_i-1)^2-a^2b^2+a^2(-2y_i+3) \\ &= d_2+a^2(-2y_i+3) \end{aligned}$$

$d_2>0$：

$$\begin{aligned} d_2 &= F(x_i+1.5,y_i-2)=b^2(x_i+1.5)^2+a^2(y_i-2)^2-a^2b^2 \\ &= b^2(x_i+0.5)^2+a^2(y_i-1)^2-a^2b^2+b^2(2x_i+2)+a^2(-2y_i+3) \\ &= d_2+b^2(2x_i+2)+a^2(-2y_i+3) \end{aligned}$$

算法步骤：

1.输入椭圆的长半轴 a 和短半轴 b。

2.计算初始值 $d=b^2+a^2(-b+0.25)$、x=0、y=b。

3.绘制点(x,y)及其在四分象限上的另外三个对称点。4.判断 d 的符号。若 d ≤0，则先将 d 更新为 $d+b^2(2x+3)$，再将(x,y)更新为(x+1,y)；否则先将 d 更新为 $d+b^2(2x+3)+a^2(-2y+2)$，再将(x,y)更新为(x+1,y-1)。

5.当 $b^2(x+1)<a^2(y-0.5)$时，重复步骤 3 和 4。否则转到步骤 6。

$$d = b^2(x+0.5)^2 + a^2(y-1)^2 - a^2b^2$$

6.用上半部分计算的最后点(x,y)来计算下半部分中 d 的初值：7.绘制点(x,y)及其在四分象限上的另外三个对称点。

8.判断 d 的符号。

若 d≤0，则先将 d 更新为 $b^2(2x_i+2)+a^2(-2y_i+3)$，再将(x,y)更新为(x+1,y-1)；否则先将 d 更新为 $d+a^2(-2y_i+3)$，再将(x,y)更新为(x,y-1)。

9.当 y>0 时，重复步骤 7 和 8。否则结束。

二．程序设计

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

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 1.5 表中的添加编辑菜单资源。此时建好的菜单如图 1-14 所示。

表 1.5 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

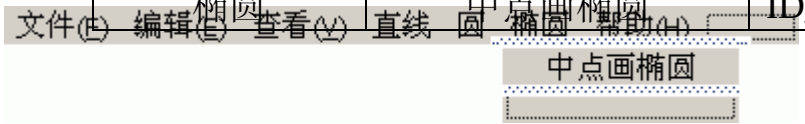


图 1-14

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 1.6 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 1.6 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

1.4 多边形的扫描转换与区域填充

在计算机图形学中，多边形有两种重要的表示方法：**顶点**表示和**点阵**表示。顶点表示是用多边形的顶点序列来表示多边形，特点直观、几何意义强、占内存少，易于进行几何变换，但由于它没有明确指出哪些像素在多边形内故不能直接用于面着色。点阵表示是用位于多边形内的像素集合来刻画多边形。这种表示丢失了许多几何信息，但便于帧缓冲器表示图形，是面着

色所需要的图形表示形式。光栅图形的一个基本问题是把多边形的顶点表示转换为点阵表示。这种转换称为**多边形的扫描转换**。

1.4.1 多边形的扫描转换

多边形可分为凸多边形、凹多边形、含内环的多边形。

① 凸凸多边形：任意两顶点间的连线均在多边形内

② 凹多边形　　任意两顶点间的连线均在连线有不在多边内

③ 含内环的多边形　　形内的部分

2.4.1.1 扫描线算法

扫描线多边形区域填充算法是按扫描线顺序，计算扫描线与多边形的相交区间，再用要求的颜色显示这些区间的像素。区间的端点可以通过计算扫描线与多边形边界线的交点获得。对于一条扫描线，多边形的填充过程可以分为四个步骤：

（1）求交：计算扫描线与多边形各边的交点；

（2）排序：把所有交点按 x 值递增顺序排序；

（3）配对：第一个与第二个，第三个与第四个等等；每对交点代表扫描线与多边形的一个相交区间，

（4）填色：把相交区间内的像素置成多边形颜色，把相交区间外的像素置成背景色。

具体实现方法：为多边形的每一条边建立一边表；为了提高效率，在处理一条扫描线时，仅对与它相交的多边形的边进行求交运算。我们把与当前

扫描线相交的边称为活性边，并把它们按与扫描线交点递增的顺序存放在一个链表中，称此链表为活性边表。另外使用增量法计算时，我们需要知道一条边何时不再与下一条扫描线相交，以便及时把它从扫描线循环删除出去。为了方便活性边表的建立与更新，我们为每一条扫描线建立一个新边表（NET）,存放在该扫描线第一次出现的边。为使程序简单、易读，这里新边表的结点应保存其对应边如下信息：当前边的边号、边的较低端点（xmin，ymin）与边的较高端点（xmax,ymax）和从当前扫描线到下一条扫描线间 x 的增量□ x。

相邻扫描线间 **x** 的增量□ **x** 的计算，假定当前扫描线与多边形某一条边的交点的 **x** 坐标为 **x**，则下一条扫描线与该边的交点不要重计算，只要加一个增量△**x**。设该边的直线方程为：**ax+by+c=0**；若 **y**＝y_i，**x**=x_i；则当 **y** = y_{i+1}时，

$$x_{i+1} = \frac{1}{a}(-b \cdot y_{i+1} - c_i) = x_i - \frac{b}{a};$$

其中 $\Delta x = -b/a$ 为常数，

扫描线与多边形顶点相交的处理方法如图所示。

1．扫描线与多边形相交的边分处扫描线的两侧，则记为一个交点，如点 P_5，P_6。

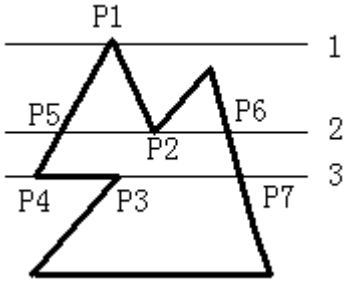


图 1-15 扫描线与多边形相交，特殊情况的处理

2．扫描线与多边形相交的边分处扫描线同侧，且 $y_i<y_{i-1}$，$y_i<y_{i+1}$，则计 **2** 个交点**(**填色**)**，如 P_2，若 $y_i>y_{i-1}$，$y_i>y_{i+1}$，则计 **0** 个交点**(**不填色**)**，如 P_1 。

3．扫描线与多边形边界重合 **(**当要区分边界和边界内区域时需特殊处理**)**，则计 **1** 个交点。

具体实现时，只需检查顶点的两条边的另外两个端点的 **y** 值。按这两个 **y** 值中大于交点 **y** 值的个数是 **0,1,2** 来决定。

算法步骤：

(1)初始化：构造边表；

(2)对边表进行排序，构造活性边表；

(3)对每条扫描线对应的对活性边表中求交点；

(4)判断交点类型，并两两配对。

(5)对符合条件的交点之间用画线方式填充；

(6)下一条扫描线，直至满足扫描结束条件。

1.4.2 区域填充算法

这里的**区域**指已表示成点阵形式的填充图形，是像素的集合。区域有两种表示形式：内点表示和边界表示。内点表示，即区域内的所有像素有相同颜色；边界表示，即区域的边界点有相同颜色。**区域填充**指先将区域的一点赋予指定的颜色，然后将该颜色扩展到整个区域的过程。

区域填充算法要求区域是连通的。区域可分为 **4 向连通区域和 8 向连通区域**。**4** 向连通区域指的是从区域上一点出发，可通过四个方向，即上、下、左、右移动的组合，在不越出区域的前提下，到达区域内的任意像素；

8 向连通区域指的是从区域内每一像素出发，可通过八个方向，即上、下、左、右、左上、右上、左下、右下这八个方向的移动的组合来到达。

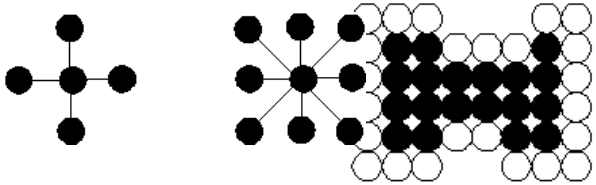


图 1-16 四连通区域和八连通

图 1-17 区域的内点表示和边界

1.1.2.1 区域填充的递归算法

上面讨论的多边形填充算法是按扫描线顺序进行的。种子填充算法则是假设在多边形内有一像素已知，由此出发利用连通性填充区域内的所有像素。一般采用多次递归方式。

1.4.2.2 区域填充的扫描线算法

算法的基本过程如下：给定种子点(x,y)，首先填充种子点所在扫描线上给定区域的一个区段，然后确定与这一区段相连通的上、下两条扫描线上位于给定区域内的区段，并依次保存下来。反复这个过程，直到填充结束。

区域填充的扫描线算法可由下列三个步骤实现：

(1)初始化：确定种子点元素（**x**，**y**）。

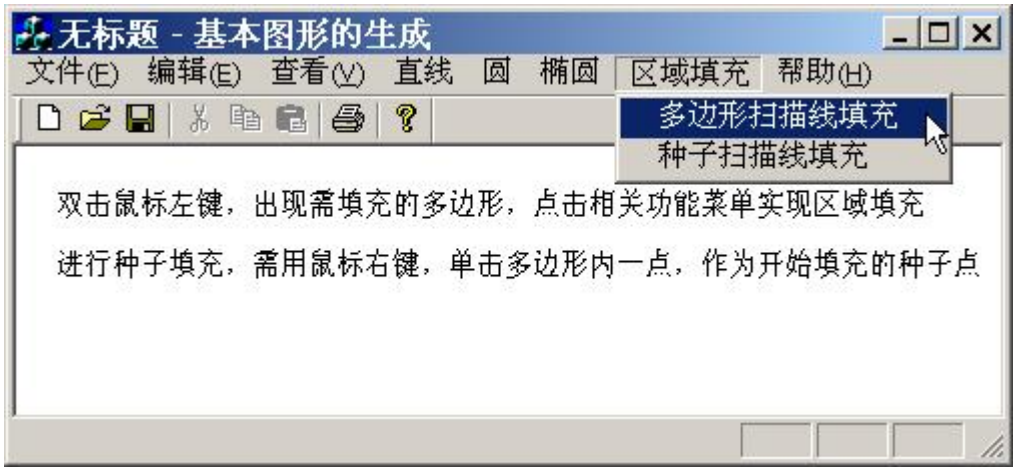
(2)判断种子点（**x**，**y**）是否满足非边界、非填充色的条件，满足条件，以 **y** 作为当前扫描线沿当前扫描线向左、右两个方向填充，直到边界。

(3) 确定新的种子点：检查与当前扫描线 **y** 上、下相邻的两条扫描线上的像素。若存在非边界、未填充的像素，则返回第（**2**）步进行扫描填充。直至区域所有元素均为填充色，程序结束。

扫描线填充算法提高了区域填充的效率。

二．程序设计

1. 创建应用程序框架，以上述单文档程序框架为基础。



2. 编辑菜单资源

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 1.7 表中的添加编辑菜单资源。此时建好的菜单如图 1-18 所示。

表 1.7 菜单资源表

菜单标题	菜单项标题	标示符 ID
区域填充	多边形扫描填充	ID_MIDPOINTELLISPE

图 1-18

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 1.8 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 1.8 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. 添加程序结构代码

（1）在“基本图形的生成 View.h”适当位置添加以下黑体字部分代码

代码见纸书

说明：1。双击鼠标左键，出现需填充的多边形，点击相关功能菜单实现区

域填充。

2．进行种子填充，需用鼠标右键，单击多边形内一点，作为开始填充的种子点。

代码见纸书

1.5 字符的生成

字符指数字、字母、汉字等符号。计算机中字符由一个数字编码唯一标识。国际上最流行的字符集是“美国信息交换用标准代码集”简称 ASCII 码。它是用 7 位二进制数进行编码表示 128 个字符，包括字母、标点、运算

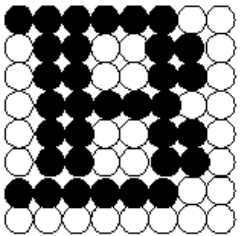
符以及一些特殊符号。我国除采用 ASCII 码外，还另外制定了汉字编码的国家标准字符集 GB2312－80。该字符集分为 94 个区，94 个位，每个符号由一个区码和一个位码共同标识。区码和位码各用一个字节表示。为了能够区分 ASCII 码与汉字编码，采用字节的最高位来标识：最高位为 0 表示 ASCII 码；最高位为 1 表示表示汉字编码。为了在显示器等输出设备上输出字符，系统中必须装备有相应的字库。字库中存储了每个字符的形状信息，字库分为矢量和点阵型两种。

1.5.1 点阵字符

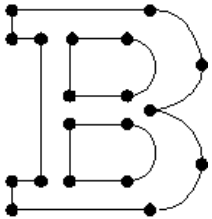
在点阵字符库中，每个字符由一个位图表示。该位为 **1** 表示字符的笔画经过此位，对应于此位的像素应置为字符颜色。该位为 **0** 表示字符的笔画不经过此位，对应于此位的像素应置为背景颜色。在实际应用中，有多种字体（如宋体、楷体等），每种字体又有多种大小型号，因此字库的存储空间是很庞大的。解决这个问题一般采用压缩技术。如： 黑白段压缩；部件压缩；轮廓字形压缩等。其中，轮廓字形法压缩比大，且能保证字符质量，是当今国际上最流行的一种方法。轮廓字形法采用直线或二/三次 **bezier** 曲线的集合来描述一个字符的轮廓线。轮廓线构成一个或若干个封闭的平面区

域。轮廓线定义加上一些指示横宽、竖宽、基点、基线等等控制信息就构成了字符的压缩数据。

点阵字符的显示分为两步。首先从字库中将它的位图检索出来。然后将检索到的位图写到帧缓冲器中。



1	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0



(a)点阵字符　　(b)点阵字库中的位图表示　　(c) 矢量轮廓字符

图 1-19 字符的种类

1.5.2 矢量字符

矢量字符记录字符的笔画信息而不是整个位图，具有存储空间小，美观、变换方便等优点。对于字符的旋转、缩放等变换，点阵字符的变换需要对表示字符位图中的每一像素进行；而矢量字符的变换只要对其笔画端点进行变换就可以了。矢量字符的显示也分为两步。首先从字库中将它的字符信

息。然后取出端点坐标，对其进行适当的几何变换，再根据各端点的标志显示出字符。

1.5.3 字符属性

字符属性一般包括：字体、字高、字宽因子(扩展/压缩)、字倾斜角、对齐方式、字色和写方式等，其中：

宋体：如仿宋体 楷体 黑体 隶书；

字倾斜角：如*倾斜* ；

对齐：如左对齐、中心对齐、右对齐；

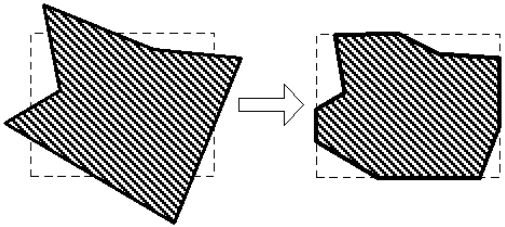
字色：如红、绿、蓝色；

写方式：**替换方式**时，对应字符掩模中空白区被置成背景色。**与方式**时，这部分区域颜色不受影响。

1.6 图形裁剪

在使用计算机处理图形信息时，计算机内部存储的图形往往比较大，而屏幕显示的只是图的一部分。因此需要确定图形中哪些部分落在显示区之内，哪些落在显示区之外，以便只显示落在显示区内的那部分图形。这个选择过程称为**裁剪**。最简单的裁剪方法是把各种图形扫描转换为点之后，再判断各点是否在窗内。但那样太费时，一般不可取。这是因为有些图形组成部分全部在窗口外，可以完全排除，不必进行扫描转换。所以一般采用先裁剪

再扫描转换的方法。



(a)裁剪前　　(b) 裁剪后

图 1-20 多边形裁剪

1.6.1 线裁剪

1．直线和窗口的关系可以分为如下 **3** 类**(**图 **1-21)**：　(1) 整条直线在窗口内。此时，不需剪裁，显示整条直线。　(2) 整条直线在窗口外，此时，不需剪裁，不显示整条直线。　(3) 部分直线在窗口内，部分直线在窗口外。此时，需要求出直线与窗框的交点，并将窗口外的直线部分剪裁掉，显示窗口内的直线部分。　直线剪裁算法有两个主要步骤。首先将不需剪裁的直线挑出，即删去

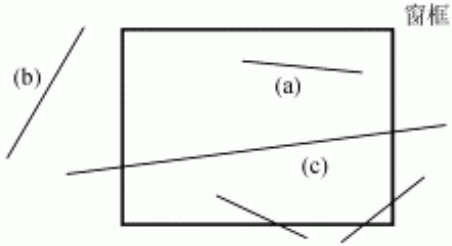
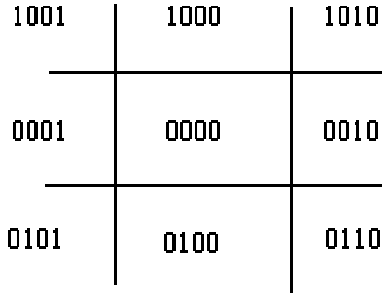


图 1-21　直线与窗口的关系

在窗外的直线。然后，对其余直线，逐条与窗框求交点，并将窗口外的 部分删去。 **2。Cohen-Sutherland 直线剪裁算法**



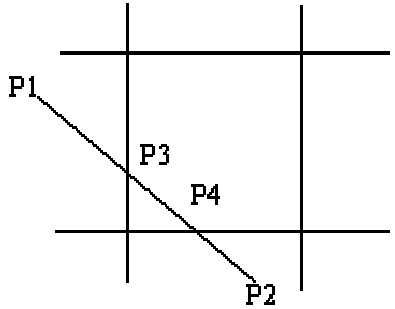


图 1-22　窗口及其邻域的 5 个区域及与直线的关系

以区域编码为基础，将窗口及其周围的 **8** 个方向以 **4 bit** 的二进制数进行编码。如图 **2-22** 所示的编码方法将窗口及其邻域分为 **5** 个区域：⑴ 内域：区域

(0000)。(2) 上域：区域(1001, 1000, 1010)。(3) 下域：区域(0101, 0100, 0110)。(4) 左域：区域(1001, 0001, 0101)。(5) 右域：区域(1010, 0010, 0110)。 当线段的两个端点的编码的逻辑“与”非零时 ，线段为显然不可见的。对某线段的两各端点的区号进行位与运算，可知这两个端点是否同在视区的上、下、左、右。算法的主要思想是，对每条直线，如 P_1P_2 利用以下步骤进行判断：

(1) 对直线两端点 P_1、P_2 编码分别记为 $C_1(P_1)=\{a_1, b_1, c_1, d_1\}$，$C_2(P_2)=\{a_2, b_2, c_2, d_2\}$其中，$a_i$、$b_i$、$c_i$、$d_i$ 取值范围为{1, 0}，$i\in\{1, 2\}$。

(2) 如果 $a_i=b_i=c_i=d_i=0$，则显示整条直线，取出下一条直线，返步骤(1)；否则，进入步骤(3)。

⑶如果$|a_1-a_2|=1$，则求直线与窗上边$(y=y_{W-max})$的交点，并删去交点以上部分。如果$|b_1-b_2|=1$，$|c_1-c_2=|=1$，$|d_1-d_2|=1$，作类似处理。⑷ 返步骤(1)判断下一条直线。

1.6.2 多边形裁剪

多边形剪裁算法的关键在于，通过剪裁，要保持窗口内多边形的边界部分，而且要将窗框的有关部分按一定次序插入多边形的保留边界之间，从而使剪裁后的多边形的边仍然保持封闭状态，以便填色算法得以正确实现

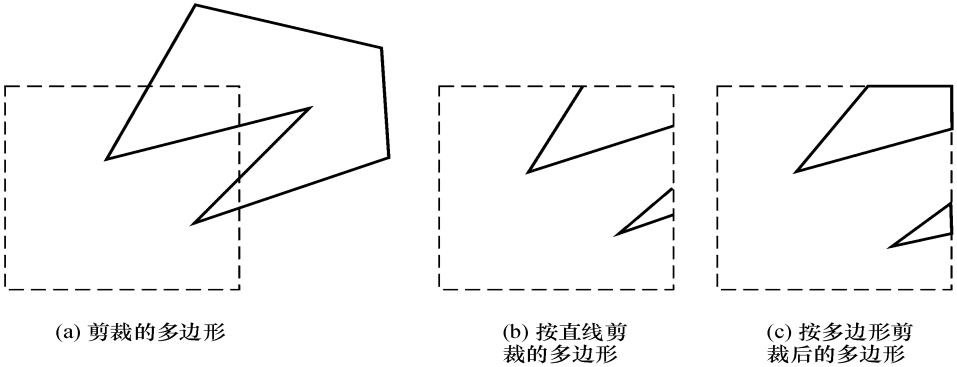


图 1-23 多边形裁剪原理示意图

Sutherland-Hodgman 算法：**思路**：将多边形的各边先相对于窗口的某一条边界进行裁剪，然后将裁剪结果再与另一条边界进行裁剪，如此重复多次，便可得到最终结果。**实现方法**：

①设置二个表

输入顶点表(向量)—用于存放被裁剪多边形的顶点 **p1-pm**。输出顶点表(线性链表)—用于存放裁剪过程中及结果的顶点 **q1-qn**。

②输入顶点表中各顶点要求按一定顺序排列，一般可采用顺时针或逆时针方向。

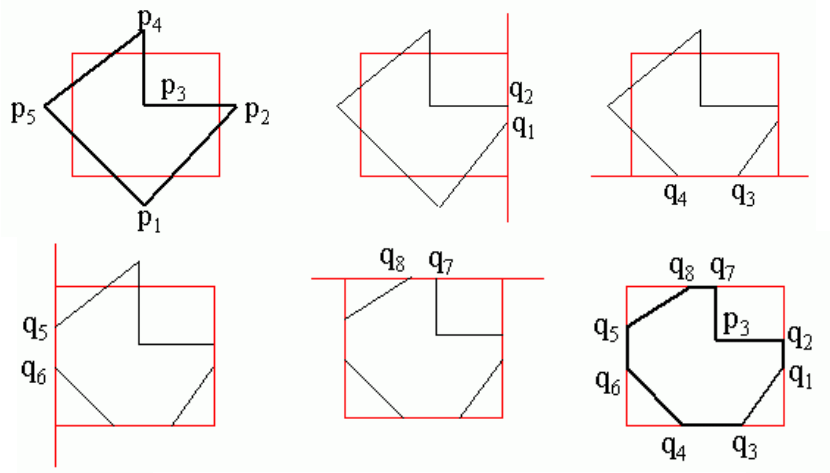


图 1-24 多边形裁剪操作示意图

③相对于裁剪窗口的各条边界，按顶点表中的顺序，逐边进行裁剪。

具体操作：

（1）**Pi** 若位于边界线的可见一侧，则 **Pi** 送输出顶点表

（2）**Pi** 若位于边界线的不可见一侧，则将其舍弃。

（3）除第一个顶点外，还要检查每一个 **Pi** 和前一顶点 **Pi-1** 是否位于窗口边界的同一侧，若不在同一侧，则需计算出交点送输出顶点表。

（4）最后一个顶点 **Pn** 则还要与 **P1** 一起进行同样的检查。

1．6．3 字符裁剪

前面我们介绍了字符和文本的输出。当字符和文本部分在窗口内，部分在窗口外时，就提出了字符裁剪问题。字符串裁剪可按三个精度来进行：串

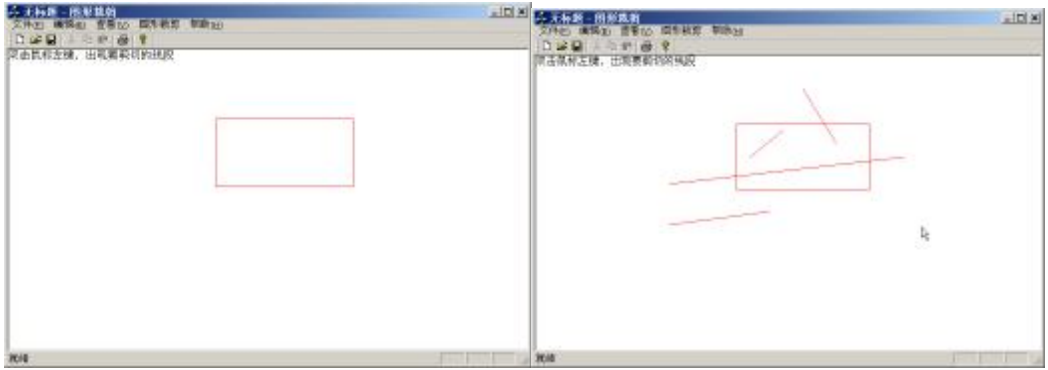
精度、字符精度、以及笔画\象素精度。采用串精度进行裁剪时，将包围字串的外接矩形对窗口作裁剪。当字符串方框整个在窗口内时予以显示，否则不显示。采用字符精度进行裁剪时，将包围字的外接矩形对窗口作裁剪，某个字符方框整个落在窗口内予以显示，否则不显示。采用笔画\象素精度进行裁剪时，将笔划分解成直线段对窗口作裁剪，处理方法同上。

(a)待裁剪字符串　(b)串精度裁剪　(c)字符精度裁剪　(d)象素精度裁剪图 1-25 字符裁剪

1.6.4 图形裁剪编程

一、程序设计功能说明

如图 1-26 所示为图形裁剪的实用程序界面。为本例程序运行时的主界面，首先根据界面提示，在用户区双击鼠标左键，出现所需要裁剪的各种线段，再单击菜单中【图形裁剪】可选择其下拉菜单的各图形裁剪选项完成各种图裁剪（在窗口中红矩形框外的线段或多边形被裁减掉）。



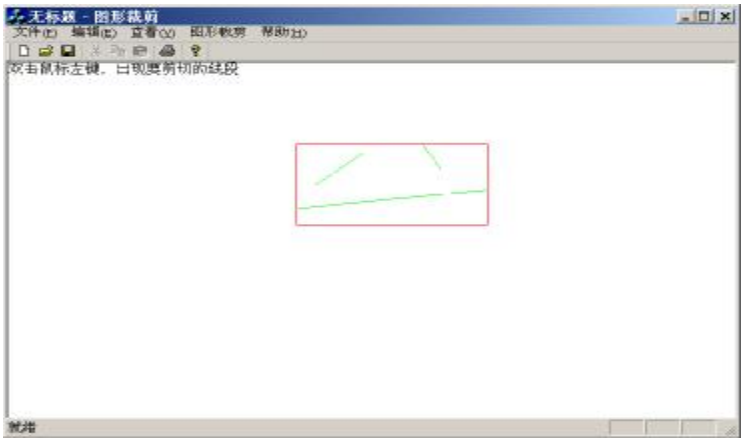


图 1-26

二、程序设计步骤

1. 建工程名称为“图形裁剪”单文档应用程序框架（参看上面单文档应用程序框架的建立）

2. 编辑菜单资源

设计如图 1-26 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 1.9 表中的定义编辑菜单资源。

表 1.9 菜单资源表

菜单标题	菜单项标题	标示符 ID
图形裁剪	线段裁剪	ID_CLIPLINE
	多边形裁剪	ID_CLIPPOLYGON

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 1.10 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 1.10 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
ID_CLIPLINE	CONMMAN	OnIDTRANSLATION
ID_CLIPPOLYGON	CONMMAN	OnIDROTATION

4. 添加代码，在图形裁剪应用程序的相应文件中添加如下黑体字部分代码

1．在“图形裁剪 View.h”文档中的适当位置添加定义存储线段端点的数组：

2.在“图形裁剪 **View.cpp**”文档中的适当位置手工添加以下黑体部分代码：

代码见纸书

说明：为避免上述“图形裁剪”程序执行中，在没有双击左键就直接单击【线段裁剪】或没有双击左键就直接单击【多边形裁剪】的情况发生，设置了消息对话框进行警告。

1.7 Visual C++中基本绘图函数

实际利用 Visual C++中编制图形程序，我们可以利用上述算法自己动手编制基本图形程序，作为图形程序的基类，当然我们还可利用系统中已提供的图形基类。下面简单介绍 Visual C++提供的常用绘制图形函数。

1．点

画点是最基本的绘图操作，在绘图中，画点是通过调用 CDC::SetPixel()或 CDC::SetPixel()函数来实现的，原型如：

(1)COLORREF SetPixel(int x, int y, COLORREF crColor);

(2) COLORREF SetPixel(POINT point, COLORREF crColor);

(3) BOOL SetPixelV(int x, int y, COLORREF crColor);

(2)BOOL SetPixelV(POINT point, COLORREF crColor);

2. 画笔

一般格式：(1)Cpen( )：：Cpen(int nPenStyle, int nWidth, CORLORREF crColor);

各属性意义： nPenStyle 设置画笔的式样，式样有：PS_SOLID（实线），PS_DASH（虚线）、PS_DASHDOT（点划线）、PS_DASHDOTDOT（双点划线）、PS_DOT（点线）、PS_NULL（空笔不画线）；nWidth 设置线的宽度，默认值为 1（1 个像素宽）；crColor 表示颜色，可用 DWOR 表示，

也可用 RGB(r, g, b)表示。

3．画刷，用于指定填充的特征，画刷创建有格式有：

（1）CBrush：：CBrush（创建一个空的画刷对象），可用 GreateSolidBrush()，GreatehatchBrush()，GreatehatchBrushIndrect()，GreatePatternBrush()，GreateDIBPatternBrush()建立画刷；

（2）CBrush::CBrush()建立单一颜色的画刷，用次画刷画出的图形内部将会填充指定颜色；

（3）CBrush::CBrush（int nIndex, COLORREF crColor）；构建名为 hatch 的画刷，特点为画出的多边形内部将填充 nrColor 指定的线条格式，

nrColor 有： HS_BDIAGONAL（45 度左下→右上的斜线）、HS_CROSS（垂直线和水平线）、HS_DIAGCROSS（45 度左上→右下、左上→右下的相交斜线）、HS_HDLAGNAL（45 度左上→右下的斜线）、HS_HORIZONAL（水平线）、HS_VERTICAL（垂直线）

（4）CBrush::CBrush（Cbitmap *pBitmap）中 pBitmap 指向 Cbitmap 对象的指针，这一位图对象包含用作画刷图按的位图，此位图必须为 8×8 大小，否则将对原位图进行裁剪。

创建画刷和画笔后，还要用 CDC 类选中画笔和画刷，用 CPaintDC，CClientDC 或 CWindowDC 来选中、绘图及撤销对象。

CClientDC 对象代表客户程序区域的绘图画面只能在窗口的客户区域中画图。若需处理整个画面（包括客户程序区域和非客户程序区）设备上下文的调用和释放可用 CWindowDC。

4．绘制直线函数

（1）MoveTo()函数用来设置当前的 x,y 的位置，创建有格式有：

CPoint MoveTo(int x, int y)

CPoint MoveTo(POINT point)

其中 x，y 用于定义新位置的坐标，point 指定新位置，可为其传递一个 Point 对象。

功能：是将线的起点从当前位置移到新位置（x,y），并且只移动点不画线。

（2）LineTo()用于绘制起点坐标到始点直线，创建有格式有：

BOOL LineTo(int x, int y)

BOOL LineTo(POINT point)

其中 x，y 用于定义线的终点坐标，point 指定线段端点位置，可为其传递一个 Point 结构或 Point 对象。

功能：是从当前的位置到新位置（x,y）画线（不包括此端点）。

5. 椭圆函数

创建格式有：

BOOL Ellipse(int x1, int y1 , int x2, int y2)

BOOL Ellipse(LPCRECT lpRect)

说明：x1, y1,限定椭圆范围的矩形左上角坐标；x2, y2 限定椭圆范围的矩形右下角坐标。

LpRect 指定椭圆的限定矩形，可为其传递一个 CRect 对象。

6. 函数绘制一段椭圆弧 Arc()

创建格式有：

BOOL Arc(int x1, int y1 , int x2, int y2, int x3, int y3 , int x4, int y4)

BOOL Ellipse(LPCRECT lpRect)

x1, y1,限定椭圆弧范围的矩形左上角坐标；x2, y2 限定椭圆弧范围的矩形右下角坐标。X3, y3 起点坐标；x2, y2 终点坐标。

7. 矩形函数

创建格式有：

BOOL Rectangle(int x1, int y1 , int x2, int y2)

x1, y1 矩形左上角坐标；x2, y2 矩形右下角坐标。

功能使用当前画笔画一矩形。

8. 连续画线函数

创建格式有：

（1）BOOL PolyLine(LPPOINT lpPoints , int nCount);

说明：lpPoints 指向 POINT 结构数组，数组中每一个结构标识一个点的坐标；

nCount:：定义数组中的点数，使用当前画笔右第一个点开始经后续点连续画线直到最后一个点。

(2) BOOL PolyLineTo(LPPOINT *lpPoints , int nCount);

说明：lpPoints 指向 POINT 结构数组指针，画一条或多条直线的指针，数组中村方直线顶点的坐标；

nCount:：定义数组中的点数。

(3) BOOL PolyBezier(LPPOINT *lpPoints);

说明：lpPoints 指向 POINT 结构数组指针，画一条或多条直线的指针，数组中包括曲线的重点和控制点；

nCount:：定义数组中的点数。

绘制三次贝塞尔曲线需要 2 个控制点和一个终点和一个起点，共 4 个点决定一条贝塞尔曲线。

（4）BOOL PolyBezierTo(LPPOINT *lpPoints);画一条或多条贝塞尔曲线。

(5)BOOL PolyBezierTo(LPPOINT *lpPoints);画一条或多条贝塞尔曲线

lpPoints 指向 POINT 结构数组指针，画一条或多条直线的指针，数组中包括曲线的重点和控制点；

nCount:：定义数组中的点数。

9. 多边形绘制函数

创建格式有：

（1）BOOL Polygon(counst POINT lpPoints, int nCount);

说明：lpPoint 指定多边形顶点数组中每一点是一个 POINT 结构或一个 CPoint 对象，nCount 指定数组中顶点数。

（2）BOOL PolyPolygon(LPPOINT lpPoints,lpint lpPolyCounts, int lpPoints);

说明：lpPoint 指向一个 POINT 结构或 CPoint 对象数组，每个数组定义一个多边形的顶点；lpPolyCounts 指向一个整数数组，每个整数说明 lpPoints 数组中一个多边形的顶点数，nCount:：为 LpPolyCount 数组中的项数，即指定要画的多边形数，最多为 2。

10. 填充函数

创建格式有：

（1）BOOL FillSolidRect(LPCRECT lpRect, COLORREF crColor);

（2）BOOL FillSolidRect(int x, int y ,int cx, int cy, COLORREF Clr);

说明：LpRect 指定矩形可传递一个指向 RECT 结构的指针或 CRect 对象。

Clr 为填充颜色；x，y 矩形左下角，cx 为矩形宽，cy 为矩形高。

（3）BOOL ExtFloodFill(int x, int y , COLORREF crColor,UINT nFillType);

说明：X，y 为开始填充处坐标；crColor 为填充颜色；FloodFillBorder

指填充区域由 crColor 参数所指定颜色包围部分；FloodFillSurface 表示填充区域是由 nColor 指定颜色来定义矩形填充

（5）BOOL FloodFill(int x, int y ,COLORREF Clr);

说明：x, y 为填充处逻辑坐标或边界颜色，crColor 指定填充颜色。

以上我们简单介绍了系统中提供的基本绘图函数，更多内容请参见其它参考书。

1．8 课后练习

1．为什么说直线生成算法是二维图形生成技术的基础?

2．根据 DDA 算法编制绘制从(10, 10)到（300，400）直线的程序

3．将中点画线算法推广以便能画出任意斜率的直线。

4．使用中点分割算法实现对直线段进行裁剪的程序。

5．利用本章所建程序框架，在菜单项【圆】的子菜单中添加【直角坐标画圆】子菜单，并添加相应命令和代码实现直角坐标画圆程序，加深对画圆算法的理解并与其它算法程序相比较。

6．利用 **Visual C++**以定义函数实现上述直线、圆、椭圆等图形的绘制，并进行区域填充。