第 5 章　二维变换和裁剪

一、习题解答

1. 如图 5-1 所示，求 $A(4,1)$、$B(7,3)$、$C(7,7)$和 $D(1,4)$构成的四边形绕 $Q(5,4)$逆时针旋转 45°的变换矩阵和变换后图形的顶点坐标。

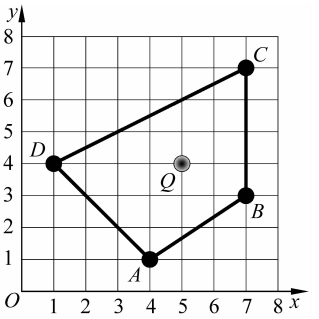

图 5-1　四边形旋转

【解】 这是一个复合变换：首先将参考点 Q 平移到坐标原点，对坐标原点进行逆时针旋转变换，然后再将参考点 Q 平移回原位置。

(1) 平移：将参考点 $Q(5,4)$平移至坐标原点$(0,0)$，变换矩阵为 $\boldsymbol{T}_1$。

$$\boldsymbol{T}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & -4 & 1 \end{bmatrix}$$

(2) 旋转：图形绕坐标原点逆时针旋转 45°，变换矩阵为 $\boldsymbol{T}_2$。

$$\boldsymbol{T}_2 = \begin{bmatrix} \cos\frac{\pi}{4} & \sin\frac{\pi}{4} & 0 \\ -\sin\frac{\pi}{4} & \cos\frac{\pi}{4} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(3) 反平移：将参考点 Q 从坐标原点平移回原处$(5,4)$，变换矩阵为 $\boldsymbol{T}_3$。

$$\boldsymbol{T}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 5 & 4 & 1 \end{bmatrix}$$

(4) 总变换矩阵：$\boldsymbol{T}=\boldsymbol{T}_1 \cdot \boldsymbol{T}_2 \cdot \boldsymbol{T}_3$。

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & -4 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\frac{\pi}{4} & \sin\frac{\pi}{4} & 0 \\ -\sin\frac{\pi}{4} & \cos\frac{\pi}{4} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 5 & 4 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 1 \\ 5-\frac{\sqrt{2}}{2} & 4-\frac{9\sqrt{2}}{2} & 1 \end{bmatrix}$$

(5) 变换前四边形 $ABCD$ 顶点的齐次矩阵为 $\boldsymbol{P}$。

$$\boldsymbol{P} = \begin{bmatrix} 4 & 1 & 1 \\ 7 & 3 & 1 \\ 7 & 7 & 1 \\ 1 & 4 & 1 \end{bmatrix}$$

(6) 变换后四边形顶点 $ABCD$ 的齐次矩阵 $\boldsymbol{P}'$。

$$\boldsymbol{P}'=\boldsymbol{P}\cdot\boldsymbol{T}=\begin{bmatrix}4&1&1\\7&3&1\\7&7&1\\1&4&1\end{bmatrix}\cdot\begin{bmatrix}\frac{\sqrt{2}}{2}&\frac{\sqrt{2}}{2}&0\\-\frac{\sqrt{2}}{2}&\frac{\sqrt{2}}{2}&1\\5-\frac{\sqrt{2}}{2}&4-\frac{9\sqrt{2}}{2}&1\end{bmatrix}=\begin{bmatrix}5+\sqrt{2}&4-2\sqrt{2}&1\\5+\frac{3\sqrt{2}}{2}&4+\frac{\sqrt{2}}{2}&1\\5-\frac{\sqrt{2}}{2}&4+\frac{5\sqrt{2}}{2}&1\\5-2\sqrt{2}&4-2\sqrt{2}&1\end{bmatrix}$$

(7) 变换后的四边形顶点 $ABCD$ 坐标为

$$A'(6.4,1.2),B'(7.1,4.7),C'(4.3,7.5),D'(2.2,1.2)。$$

2. 屏幕客户区 xOy 坐标系的原点位于左上角，x 轴水平向右，最大值为 MaxX，y 轴垂直向下，最大值为 MaxY。建立新坐标系 $x'O'y'$，原点位于屏幕客户区中心(MaxX/2,MaxY/2)，x' 轴水平向右，y' 轴垂直向上，如图 5-2 所示。要求用变换矩阵求解这两个坐标系之间的相互关系。

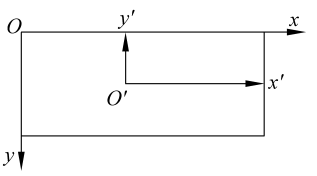

图 5-2　坐标系变换

【解】 在屏幕上绘制图形时，常使用自定义坐标系，原点位于屏幕中心(MaxX/2,MaxY/2)，x' 轴水平向右，y' 轴垂直向上。实现语句为

```
pDC->SetMapMode(MM_ANISOTROPIC);                    //自定义坐标系
pDC->SetWindowExt(Rect.Width(),Rect.Height());
pDC->SetViewportExt(Rect.Width(),-Rect.Height());   //x轴水平向右,y轴垂直向上
pDC->SetViewportOrg(Rect.Width()/2,Rect.Height()/2); //屏幕中心为原点
```

本题给出自定义坐标系和屏幕设备坐标系之间的变换关系。

(1) 将 xOy 坐标系原点平移到(MaxX/2,MaxY/2)处，形成 $x'O'y'$ 坐标系。由于坐标系沿坐标轴的正向移动等同于相应图形沿坐标轴负向移动，变换矩阵 $\boldsymbol{T}_1$ 为

$$\boldsymbol{T}_1=\begin{bmatrix}1&0&0\\0&1&0\\-\frac{\mathrm{MaxX}}{2}&-\frac{\mathrm{MaxY}}{2}&1\end{bmatrix}$$

(2) 将 y 关于 x 轴做反射变换，由于坐标系对 x 轴的反射变换等同于相应图形对 x 轴的反射变换，变换矩阵 $\boldsymbol{T}_2$ 为

$$\boldsymbol{T}_2=\begin{bmatrix}1&0&0\\0&-1&0\\0&0&1\end{bmatrix}$$

(3) 变换矩阵 $\boldsymbol{T}=\boldsymbol{T}_1\cdot\boldsymbol{T}_2$

$$\boldsymbol{T}=\begin{bmatrix}1&0&0\\0&-1&0\\-\frac{\mathrm{MaxX}}{2}&\frac{\mathrm{MaxY}}{2}&1\end{bmatrix}$$

3. 建立坐标系 xOy，原点位于屏幕客户区中心，x 轴水平向右，y 轴垂直向上。在原点

正下方(0,$-b$)点,悬挂一边长为 $a\left(b>\frac{\sqrt{2}a}{2}\right)$的正方形,如图 5-3 所示。要求正方形绕自身中心(0,$-b$)点逆时针旋转。编程实现二维复合旋转变换。

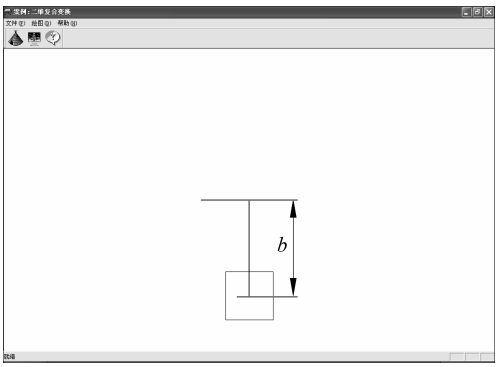

图 5-3　复合旋转变换

【解】 本题要求编程实现二维复合旋转变换。通过映射模式函数可知,屏幕坐标系的原点位于屏幕中心(MaxX/2,MaxY/2),x 轴水平向右,y 轴垂直向上。二维变换矩阵可以使用数组 T[3][3]来实现。二维复合旋转变换分为 3 步完成。平移变换、旋转变换和反平移变换。本题的二维复合旋转实现是通过动画按钮调用 SetTimer 函数和 KillTimer 函数完成的。SetTimer 函数启动定时器,KillTimer 函数关闭定时器。

(1) 读入正方形顶点坐标函数。

```
void CTestView::ReadPoint()
{
    a=100,b=200;
    POld[0][0]=-a/2;POld[0][1]=-(b-a/2);POld[0][2]=1; //正方形顶点齐次坐标
    POld[1][0]=-a/2;POld[1][1]=-(b+a/2);POld[1][2]=1;
    POld[2][0]=a/2;POld[2][1]=-(b+a/2);POld[2][2]=1;
    POld[3][0]=a/2;POld[3][1]=-(b-a/2);POld[3][2]=1;
}
```

(2) 绘制多边形函数。

```
void CTestView::DrawPolygon(CDC * pDC,double p[][3])      //绘制多边形函数
{
    double Tempx,Tempy;
    for(int i=0;i<4;i++)
    {
        if(i==0)
        {
            pDC->MoveTo(ROUND(p[i][0]),ROUND(p[i][1]));
            Tempx=p[i][0];Tempy=p[i][1];
        }
        else
            pDC->LineTo(ROUND(p[i][0]),ROUND(p[i][1]));
```

```
    }
    pDC->LineTo(ROUND(Tempx),ROUND(Tempy));
}
```

(3) 双缓冲函数。

```
void CTestView::DoubleBuffer()                                  //双缓冲函数
{
    double Tx=0,Ty=b;
    //平移变换矩阵
    T1[0][0]=1;T1[0][1]=0;T1[0][2]=0;
    T1[1][0]=0;T1[1][1]=1;T1[1][2]=0;
    T1[2][0]=-Tx;T1[2][1]=Ty;T1[2][2]=1;
    //逆时针自转变换矩阵
    T2[0][0]=cos(beta*PI/180);T2[0][1]=sin(beta*PI/180);T2[0][2]=0;
    T2[1][0]=-sin(beta*PI/180);T2[1][1]=cos(beta*PI/180);T2[1][2]=0;
    T2[2][0]=0;T2[2][1]=0;T2[2][2]=1;
    //反平移变换矩阵
    T3[0][0]=1;T3[0][1]=0;T3[0][2]=0;
    T3[1][0]=0;T3[1][1]=1;T3[1][2]=0;
    T3[2][0]=Tx;T3[2][1]=-Ty;T3[2][2]=1;
    MultiMatrix(POld,T1);
    MultiMatrix(POld,T2);
    MultiMatrix(POld,T3);
    CDC*pDC=GetDC();                                            //客户区 DC
    GetMaxX();GetMaxY();
    pDC->SetMapMode(MM_ANISOTROPIC);                            //自定义坐标系
    pDC->SetWindowExt(MaxX,MaxY);
    pDC->SetViewportExt(MaxX,-MaxY);                            //x轴向右,y轴向上
    pDC->SetViewportOrg(MaxX/2,MaxY/2);                         //屏幕中心为原点
    CDC MemDC,Picture;
    MemDC.CreateCompatibleDC(pDC);
    MemDC.SetMapMode(MM_ANISOTROPIC);
    MemDC.SetWindowExt(MaxX,MaxY);
    MemDC.SetViewportExt(MaxX,-MaxY);
    MemDC.SetViewportOrg(MaxX/2,MaxY/2);
    CBitmap NewBitmap,*OldBitmap;
    NewBitmap.LoadBitmap(IDB_BITMAP2);
    OldBitmap=MemDC.SelectObject(&NewBitmap);
    MemDC.BitBlt(-MaxX/2,-MaxY/2,MaxX,MaxY,&Picture,-MaxX/2,-MaxY/2,SRCCOPY);
    DrawCoor(&MemDC);
    DrawPolygon(&MemDC,POld);
    pDC->BitBlt(-MaxX/2,-MaxY/2,MaxX,MaxY,&MemDC,-MaxX/2,-MaxY/2,SRCCOPY);
    ReleaseDC(pDC);
    MemDC.SelectObject(OldBitmap);
    NewBitmap.DeleteObject();
```

```
}
```

(4) 矩阵相乘函数。

```
void CTestView::MultiMatrix(double P0[][3],double T[3][3])    //矩阵相乘函数
{
    int i,j;
    for(i=0;i<4;i++)
        for(j=0;j<3;j++)
            Pnew[i][j]=P0[i][0]*T[0][j]+P0[i][1]*T[1][j]+P0[i][2]*T[2][j];
    for(i=0;i<4;i++)
    {
        for(j=0;j<3;j++)
            Pold[i][j]=Pnew[i][j];
    }
}
```

(5) 绘制坐标系函数。

```
void CTestView::DrawCoor(CDC*pDC)
{
    Cpen NewPen,*OldPen;
    NewPen.CreatePen(PS_SOLID,3,RGB(128,128,128));
    OldPen=pDC->SelectObject(&NewPen);
    pDC->MoveTo(-100,0);
    pDC->LineTo(100,0);
    pDC->MoveTo(0,0);
    pDC->LineTo(0,-b);
    pDC->SelectObject(OldPen);
    NewPen.DeleteObject();
}
```

(6) 动画菜单映射函数。

```
void CTestView:: OnMENUPlay ()                                    //菜单映射函数
{
    //TODO: Add your command handler code here
    Play=Play?FALSE:TRUE;
    if (Play)                                                     //设置动画的时间步
        SetTimer(1,15,NULL);
    else
        KillTimer(1);
}
```

(7) 定时器消息映射函数。

```
void CTestView::OnTimer(UINT nIDEvent)
{
    //TODO: Add your message handler code here and/or call default
```

```
    beta=10;
    DoubleBuffer();
    CView::OnTimer(nIDEvent);
}
```

(8) 动画按钮状态映射函数。

```
void CTestView::OnUpdateMENUPlay(CCmdUI * pCmdUI)
{
    //TODO: Add your command update UI handler code here
    if(Play)
    {
        pCmdUI->SetCheck(TRUE);
        pCmdUI->SetText("停止动画");
    }
    else
    {
        pCmdUI->SetCheck(FALSE);
        pCmdUI->SetText("播放动画");
    }
}
```

4. 有两个点 $P_1(0,2)$，$P_2(3,3)$用 Cohen-Sutherland 直线段编码裁剪算法裁剪线段 P_1P_2，裁剪窗口为 $w_{xl}=1$，$w_{xr}=6$，$w_{yb}=1$，$w_{yt}=5$，如图 5-4 所示。

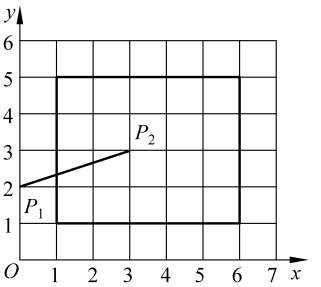

图 5-4　直线段裁剪

要求写出：

(1) 窗口边界划分的 9 个区间的编码原则。

(2) 线段端点的编码。

(3) 裁剪的主要步骤。

(4) 裁剪后窗口内直线段的端点坐标。

【解】 Cohen-Sutherland 直线段编码裁剪算法将窗口及其延长线分成 9 个区域，每段直线的端点都被赋予一组四位二进制代码 $RC=C_4C_3C_2C_1$。若直线段的两个端点的区域编码都为 0，即 $RC_1|RC_2=0$，说明直线段两端点都在窗口内，应“简取”之。若直线段的两个端点的区域编码都不为 0，即 $RC_1\&RC_2\neq 0$，即直线段位于窗外的同一侧，说明直线段的两个端点都在窗口外，应“简弃”之。若直线段既不满足“简取”也不满足“简弃”的条件，直线段必然与窗口相交，需要计算直线段与窗口边界的交点。交点将直线段分为两部分，其中一段完全位于窗口外，可“简弃”之。对另一段直线赋予交点处的区域编码，再次测试，再次求交，直至求出完全位于窗口内的直线段为止。

(1) 窗口边界划分的 9 个区间的编码原则。根据直线段的任一端点相对于窗口的位置，赋予一组 4 位二进制区域码 $RC=C_4C_3C_2C_1$。

第 1 位 C_1：若端点位于窗口之左侧，即 $x<w_{xl}$，则 $C_1=1$，否则 $C_1=0$。

第 2 位 C_2：若端点位于窗口之右侧，即 $x>w_{xr}$，则 $C_2=1$，否则 $C_2=0$。

第 3 位 C_3：若端点位于窗口之下侧，即 $y<w_{yb}$，则 $C_3=1$，否则 $C_3=0$。

第 4 位 C_4：若端点位于窗口之上侧，即 $y>w_{yt}$，则 $C_4=1$，否则 $C_4=0$。

(2) 直线段端点的编码。P_1 点的编码为 0001，P_2 点的编码为 0000。

(3) 裁剪的主要步骤。输入 $P_1(0,2)$，$P_2(3,3)$，$w_{yt}=4$，$w_{yb}=1$，$w_{xr}=4$，$w_{xl}=1$。

$RC_1=0001$，$RC_2=0000$。

因为 $RC_1 | RC_2 \neq 0$，所以不能简取；因为 $RC_1 \& RC_2 = 0$，所以不能简弃。直线段与窗口相交，计算直线段与窗口边界的交点。

求线段 $P_1(0,2)$，$P_2(3,3)$与窗口左界 $w_{xl}=1$ 的交点，把 $x=w_{xl}=1$ 代入直线方程求出 $y=kx+b=2.3$，交点坐标 $P(1,2.3)$替换端点坐标 $P_1(0,2)$，使 P_1 坐标为(1,2.3)，剪掉 P_1P 线段，输出线段 P_1P_2。

(4) 裁剪后窗口内直线段的端点坐标。裁剪后窗口内直线段的端点坐标为 $P_1(1,2.3)$和 $P_2(3,3)$。

5. 窗视变换公式也可以使用窗口和视区的相似原理进行推导，但要求点 $P(x_w,y_w)$在窗口中的相对位置等于点 $P'(x_v,y_v)$在视区中的相对位置，推导以下的窗视变换公式：

$$\begin{cases} x_v = ax_w + b \\ y_v = cy_w + d \end{cases}, \quad \text{变换系数分别为} \begin{cases} a = S_x = \dfrac{v_{xr}-v_{xl}}{w_{xr}-w_{xl}} \\ b = v_{xl} - w_{xl}a \\ c = S_y = \dfrac{v_{yt}-v_{yb}}{w_{yt}-w_{yb}} \\ d = v_{yb} - w_{yb}c \end{cases}$$

【解】 窗视变换公式也可以使用相似原理推导，只是要求点窗口中点的相对位置和视区中点的相对保持对应比例关系。

设观察坐标系下窗口区的左下角坐标为(w_{xl},w_{yb})，右上角坐标为(w_{xr},w_{yt})。设备坐标系中视区的左下角坐标为(v_{xl},v_{yb})，右上角坐标为(v_{xr},v_{yt})。窗口内的点为(x_w,y_w)，视区内的对应点为(x_v,y_v)。

利用相似原理，可以把窗口中的一点 $P(x_w,y_w)$变换为视区中的一点 $P'(x_v,y_v)$，如图 5-5 和图 5-6 所示。要求点 P 在窗口中的相对位置和 P'在视区中的相对位置保持对应比例关系。

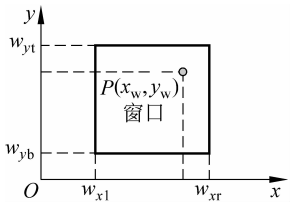

图 5-5　窗口中的点

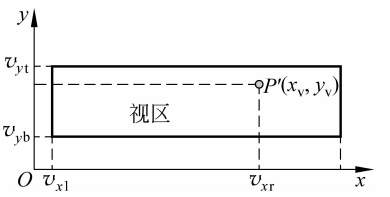

图 5-6　视区中的点

则有

$$\begin{cases} \dfrac{x_w-w_{xl}}{w_{xr}-w_{xl}} = \dfrac{x_v-v_{xl}}{v_{xr}-v_{xl}} \\ \dfrac{y_w-w_{yb}}{w_{yt}-w_{yb}} = \dfrac{y_v-v_{yb}}{v_{yt}-v_{yb}} \end{cases}$$

令

$$\begin{cases} a = \dfrac{v_{xr} - v_{xl}}{w_{xr} - w_{xl}} = S_x \\ b = v_{xl} - w_{xl}a \\ c = \dfrac{v_{yt} - v_{yb}}{w_{yt} - w_{yb}} = S_y \\ d = v_{yb} - w_{yb}c \end{cases}$$

上式简化为$\begin{cases} x_v = a \cdot x_w + b \\ y_v = b \cdot y_w + d \end{cases}$

6. 按照图 5-7 所示，使用对话框输入直线段的起点和终点坐标。在屏幕客户区左侧区域绘制输入直线段和“窗口”，在屏幕客户区右侧区域绘制“视区”并输出裁剪结果，如图 5-8 所示。这里需要用到窗视变换公式。使用 Cohen-Sutherland 算法编程实现。

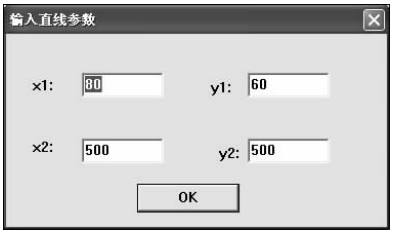

图 5-7　输入对话框

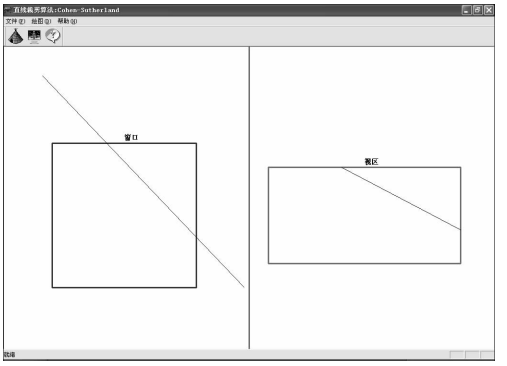

图 5-8　含窗视变换的直线段裁剪效果图

【解】 将屏幕垂直分为两个窗口，左侧代表“窗口”，右侧代表“视区”。在“窗口”中绘制输入的直线，在“视区”中绘制输出结果。

(1) 编码宏定义。

```
#define LEFT 1                                  //编码 0001
#define RIGHT 2                                 //编码 0010
#define BOTTOM 4                                //编码 0100
#define TOP 8                                   //编码 1000
```

(2) 窗口和视区边界定义。

```
CTestView::CTestView()
{
    //TODO: add construction code here
     wxl=100;wxr=400;wyb=200;wyt=500;
     vxl=550;vxr=950;vyb=250;vyt=450;
     a=(vxr-vxl)/(wxr-wxl);b=vxl-wxl*a;
     c=(vyt-vyb)/(wyt-wyb);d=vyb-wyb*c;
}
```

(3) OnDraw 函数。

```
void CTestView::OnDraw(CDC*pDC)
```

```
{
    CTestDoc * pDoc=GetDocument();
    ASSERT_VALID(pDoc);
    //TODO: add draw code for native data here
    CPen Pen1,Pen2,Pen3, * pOldPen;
    Pen1.CreatePen(PS_SOLID,3,RGB(0,0,255));  //定义 3 个像素的画笔
    pOldPen=pDC->SelectObject(&Pen1);
    //绘制窗口
    pDC->MoveTo(ROUND(wxl),ROUND(wyt));pDC->LineTo(ROUND(wxr),ROUND(wyt));
    pDC->LineTo(ROUND(wxr),ROUND(wyb));pDC->LineTo(ROUND(wxl),ROUND(wyb));
    pDC->LineTo(ROUND(wxl),ROUND(wyt));
    pDC->TextOut((ROUND(wxl)+ROUND(wxr))/2,ROUND(wyb)-20,"窗口");
    //窗口标题
    pDC->SelectObject(pOldPen);
    Pen1.DeleteObject();
    Pen2.CreatePen(PS_SOLID,3,RGB(255,0,0));
    pOldPen=pDC->SelectObject(&Pen2);
    //绘制视区
    pDC->MoveTo(ROUND(vxl),ROUND(vyt));pDC->LineTo(ROUND(vxr),ROUND(vyt));
    pDC->LineTo(ROUND(vxr),ROUND(vyb));pDC->LineTo(ROUND(vxl),ROUND(vyb));
    pDC->LineTo(ROUND(vxl),ROUND(vyt));
    pDC->TextOut((ROUND(vxl)+ROUND(vxr))/2,ROUND(vyb)-20,"视区");
    //视区标题
    pDC->SelectObject(pOldPen);
    Pen2.DeleteObject();
    //绘制窗口与视区的分割线
    GetClientRect(&Rect);                    //获得客户区的大小
    Pen3.CreatePen(PS_SOLID,3,RGB(128,128,128));
    pOldPen=pDC->SelectObject(&Pen3);
    pDC->MoveTo(Rect.Width()/2,0);
    pDC->LineTo(Rect.Width()/2,Rect.Height());
    pDC->SelectObject(pOldPen);
    Pen3.DeleteObject();
}
```

(4) 菜单映射函数。

```
void CTestView::OnMENUCohen()              //菜单映射函数
{
    //TODO: Add your command handler code here
    CClientDC dc(this);
    double xw1,yw1,xw2,yw2;
    CInputDlg dlg;
    if(dlg.DoModal()==IDOK)
    {
        xw1=dlg.m_x1;xw2=dlg.m_x2;yw1=dlg.m_y1;yw2=dlg.m_y2;
```

```
    }
    RedrawWindow();                          //刷新屏幕
    //窗口绘制直线
    dc.MoveTo(ROUND(xw1),ROUND(yw1));
    dc.LineTo(ROUND(xw2),ROUND(yw2));
    clip(xw1,yw1,xw2,yw2);
}
```

(5) 裁剪函数。

```
void CTestView::clip(double xw1,double yw1,double xw2,double yw2)
                                             //裁剪函数
{
    double xv1,yv1,xv2,yv2;
    double x,y;
    BOOL Change;
    CClientDC dc(this);
    RC1=EnCode(xw1,yw1);RC2=EnCode(xw2,yw2);
    while(TRUE)
    {
        Change=FALSE;
        if( (RC1|RC2)==0)
        {                                            //取之
            xv1=a*xw1+b;yv1=c*yw1+d;xv2=a*xw2+b;yv2=c*yw2+d;
                                                     //计算视区坐标
            dc.MoveTo(ROUND(xv1),ROUND(yv1));
            dc.LineTo(ROUND(xv2),ROUND(yv2));
            return;
        }
        else if((RC1 & RC2)!=0)
        {                                            //弃之
            return;
        }
        else
        {
            if(RC1==0)              //如果 P1 点在窗口内,交换 P1 和 P2,保证 p1 点在窗口外
            {
                //交换点的坐标值
                double tx,ty;
                tx=xw1;ty=yw1;
                xw1=xw2;yw1=yw2;
                xw2=tx;yw2=ty;
                //交换点的编码值
                unsigned int RC;
                RC=RC1;RC1=RC2;RC2=RC;
                }
            if(RC1 & LEFT )                          //P1 点位于窗口的左侧
```