实验 5　直线段裁剪

5.1　实验目的

掌握 Cohen-Sutherland 直线段裁剪算法。

5.2　实验要求

(1) 定义二维坐标系原点位于屏幕中心，x 轴水平向右为正，y 轴垂直向上为正。

(2) 在客户区中央固定绘制颜色为 RGB(128,0,0)的 3 像素宽的矩形代表裁剪窗口。裁剪窗口的左上角点为(－300,100)，右下角点为(300,－100)。

(3) 使用鼠标在屏幕上动态绘制任意直线段。

(4) 选择裁剪按钮根据直线段和窗口的相对位置，对直线段进行裁剪，得到位于窗口内的直线段，删除窗口外的直线段。

(5) 直线段绘制之前，裁剪按钮无效；直线段绘制之后，裁剪按钮有效。

5.3　效果图

直线段裁剪前效果如图 5-1 所示，直线段裁剪后效果如图 5-2 所示。

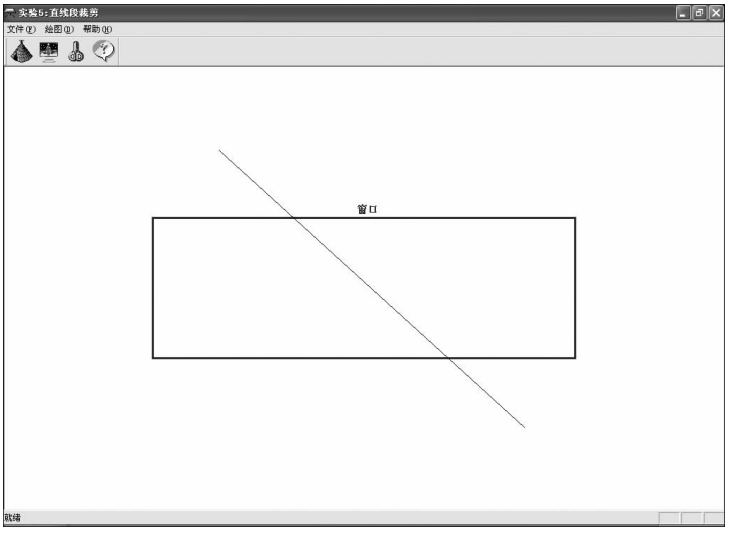

图 5-1　直线段裁剪前效果图

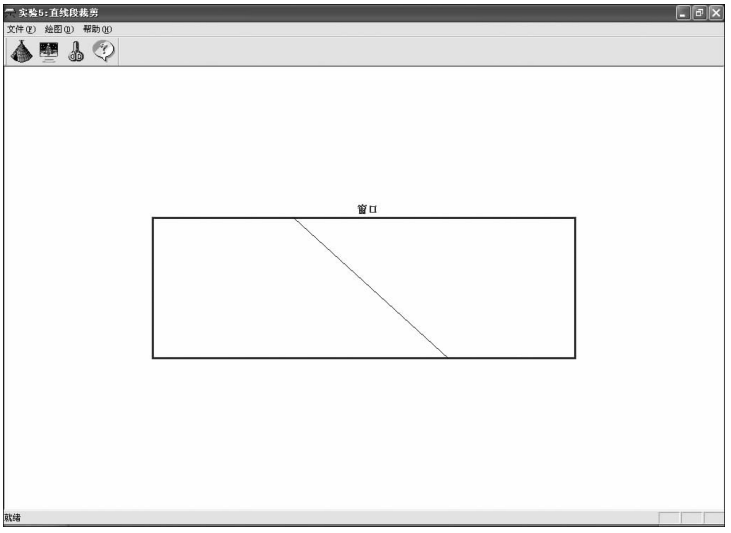

图 5-2　直线段裁剪后效果图

5.4　实验准备

(1) 在学习完主教材 5.5 节后,进行本实验。

(2) 熟悉掌握区域编码规则。

(3) 熟悉“简取”“简弃”和“求交”的判断条件。

(4) 熟悉窗口边界(或其延长线)与直线段交点坐标的计算公式。

5.5　实验步骤

本实验首先修改 CP2 类,绑定直线段顶点及其编码,然后使用双缓冲绘制动态直线,调用 Dan Cohen 和 Ivan Sutherland 所提出的 Cohen-Sutherland 直线段裁剪算法,根据窗口和直线段的相对位置对直线段进行裁剪。

5.5.1　定义 CP2 类

为了将直线端点坐标和编码绑定在一起,修改了 CP2 类定义,增加了直线段端点编码。

```
class CP2
{
public:
    CP2();
    virtual ~CP2();
    double x;                    //直线段端点 x 坐标
    double y;                    //直线段端点 y 坐标
    UINT  rc;                    //直线段端点编码
};
```

x 和 y 为直线段端点坐标，rc 为直线段端点编码。

5.5.2 OnDrw()函数

在 TestView.cpp 文件的 OnDraw()函数中使用双缓冲绘制矩形裁剪窗口和直线段。同时定义二维坐标系原点位于屏幕中心，x 轴水平向右，y 轴垂直向上。裁剪窗口使用 RGB(128,0,0)的 3 像素宽直线段绘制，被裁剪的直线段使用 1 像素宽的黑色直线段绘制。

```
void CTestView::OnDraw(CDC* pDC)
{
    CTestDoc* pDoc=GetDocument();
    ASSERT_VALID(pDoc);
    //TODO: add draw code for native data here
    CRect rect;                                          //定义客户区
    GetClientRect(&rect);                                //获得客户区的大小
    pDC->SetMapMode(MM_ANISOTROPIC);                     //pDC 自定义坐标系
    pDC->SetWindowExt(rect.Width(),rect.Height());       //设置窗口范围
    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
    CDC memDC;                                           //内存 DC
    CBitmap NewBitmap,*pOldBitmap;                       //内存中承载图像的临时位图
    memDC.CreateCompatibleDC(pDC);                       //建立与屏幕 pDC 兼容的 memDC
    NewBitmap.CreateCompatibleBitmap(pDC,rect.Width(),rect.Height());
                                                         //创建兼容位图
    pOldBitmap=memDC.SelectObject(&NewBitmap);           //将兼容位图选入 memDC
    memDC.FillSolidRect(rect,pDC->GetBkColor());         //按原来背景填充客户区,否则是黑色
    memDC.SetMapMode(MM_ANISOTROPIC);                    //memDC 自定义坐标系
    memDC.SetWindowExt(rect.Width(),rect.Height());
    memDC.SetViewportExt(rect.Width(),-rect.Height());
    memDC.SetViewportOrg(rect.Width()/2,rect.Height()/2);
    DrawWindowRect(&memDC);                              //绘制窗口
    if(PtCount>=1)
    {
        memDC.MoveTo(ROUND(P[0].x),ROUND(P[0].y));
        memDC.LineTo(ROUND(P[1].x),ROUND(P[1].y));
    }
    pDC->BitBlt(-rect.Width()/2,-rect.Height()/2,rect.Width(),rect.Height(),
                &memDC,-rect.Width()/2,-rect.Height()/2,SRCCOPY);
                                                         //将内存位图复制到屏幕
    memDC.SelectObject(pOldBitmap);                      //恢复位图
    NewBitmap.DeleteObject();                            //删除位图
    memDC.DeleteDC();                                    //删除 memDC
}
```

5.5.3 绘制裁剪窗口

裁剪窗口的左上角点的 x 坐标为 W_{xl}，y 坐标为 W_{yt}，$W_{xl}=-300$，$W_{yt}=100$，裁剪窗口

的右下角点的 x 坐标为 W_{xr}，y 坐标为 W_{yb}，$W_{xr}=300$，$W_{yb}=-100$。

```
void CTestView::DrawWindowRect(CDC * pDC)                    //绘制裁剪窗口
{
    // TODO: Add your message handler code here and/or call default
    pDC->SetTextColor(RGB(128,0,0));
    pDC->TextOut(-10,Wyt+20,"窗口");
    CPen NewPen3, * pOldPen3;                                //定义 3 个像素宽度的画笔
    NewPen3.CreatePen(PS_SOLID,3,RGB(128,0,0));
    pOldPen3=pDC->SelectObject(&NewPen3);
     pDC->Rectangle(Wxl,Wyt,Wxr,Wyb);
    pDC->SelectObject(pOldPen3);
    NewPen3.DeleteObject();
}
```

5.5.4 鼠标左键按下函数

鼠标左键按下时，确定直线段的起点。PtCount 为直线端点索引的计数器，当 PtCount＝0 时，P[PtCount]为直线段的起点；当 PtCount＝1 时，P[PtCount]为直线段的终点。DrawLine 是 BOOL 型变量，用于判断是否允许画线。由于 OnDraw()函数使用自定义坐标系绘图，而 OnLButtonDown()函数内使用的是物理坐标系，二者需要进行坐标转换。定义了 Convert()函数将设备坐标系内的点转换为自定义坐标系内的点。

```
void CTestView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    if(DrawLine)
    {    if(PtCount<2)
        {
            P[PtCount]=Convert(point);
            PtCount++;
        }
    }
    CView::OnLButtonDown(nFlags, point);
}
```

5.5.5 鼠标移动函数

鼠标移动函数确定直线段的终点。

```
void CTestView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    if(DrawLine)
    {
        if(PtCount<2)
```

```
        {
            P[PtCount]=Convert(point);
            Invalidate(FALSE);
        }
    }
    CView::OnMouseMove(nFlags, point);
}
```

5.5.6 编码函数

根据直线段的任一端点 $P(x,y)$相对于窗口的位置,可以赋予一组 4 位二进制区域码 RC=$C_4C_3C_2C_1$,从右到左依次代表左、右、下、上,如图 5-3 所示。

C_4	C_3	C_2	C_1
上	下	右	左

图 5-3 区域码各位含义

为了保证窗口内直线段端点编码 RC 的 4 位二进制区域码全部为 0,每位二进制区域码编码规则如下:

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 $C_4=0$。

定义一组宏表示二进制区位码的十进制数。

```
#define LEFT     1                                  //代表:0001
#define RIGHT    2                                  //代表:0010
#define BOTTOM   4                                  //代表:0100
#define TOP      8                                  //代表:1000
void CTestView::EnCode(CP2 &pt)                     //端点编码函数
{
    pt.rc=0;
    if(pt.x<Wxl)
    {
        pt.rc=pt.rc | LEFT;
    }
    else if(pt.x>Wxr)
    {
        pt.rc=pt.rc | RIGHT;
    }
    if(pt.y<Wyb)
    {
        pt.rc=pt.rc | BOTTOM;
    }
    else if(pt.y>Wyt)
    {
        pt.rc=pt.rc | TOP;
    }
}
```

5.5.7 裁剪函数

1. 裁剪步骤

(1) 若直线段的两个端点的区域码都为零，即 $RC_0 | RC_1 = 0$(二者按位相或的结果为 0，即 $RC_0 = 0$ 且 $RC_1 = 0$)，说明直线段两端点都在窗口内，应“简取”。

(2) 若直线段的两个端点的区域码都不为 0，即 $RC_0 \& RC_1 \neq 0$。二者按位相与的结果不为 0，即 $RC_0 \neq 0$ 且 $RC_1 \neq 0$，即直线段位于窗口外的同一侧，说明直线段的两个端点都在窗口外，应“简弃”。

(3) 若直线段既不满足“简取”也不满足“简弃”的条件，则需要与窗口进行“求交”判断。这时，直线段必然与窗口边界(及其延长线)相交，分两种情况处理。一种情况是直线段与窗口边界相交，如图 5-4 所示直线段 P_0P_1。按左右下上顺序计算窗口边界与直线段的交点。右边界与 P_0P_1 的交点为 P，P 点的编码为 0000，舍弃 P_0P 直线段，将 P 点的坐标和编码替换为 P_0 点，交换 P_0P_1 点的坐标及其编码，使 P_0 总是位于窗口之外，如图 5-5 所示。下边界与 P_0P_1 的交点为 P，P 点的编码为 0000，舍弃 P_0P 直线段，将 P 点的坐标和编码替换为 P_0 点，如图 5-6 所示。此时，直线段 P_0P_1 被“简取”。另一种情况是直线段与窗口边界的延长线相交，直线段完全位于窗口之外，且不在窗口同一侧，所以 $RC_0 = 0010 \neq 0$，$RC_1 = 0100 \neq 0$，但 $RC_0 \& RC_1 = 0$，如图 5-7 所示的直线段 P_0P_1。按左右下上顺序计算窗口边界延长线与直线段的交点。右边界延长线与 P_0P_1 的交点为 P，P 点的编码为 0110，舍弃 P_0P 直线段，将 P 点的坐标和编码替换为 P_0P_1 点，如图 5-8 所示。此时，直线段 P_0P_1 位于窗口外的同一侧，被“简弃”。

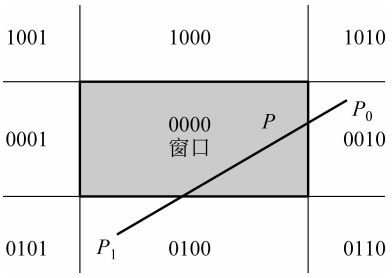

图 5-4 直线段与窗口边界相交

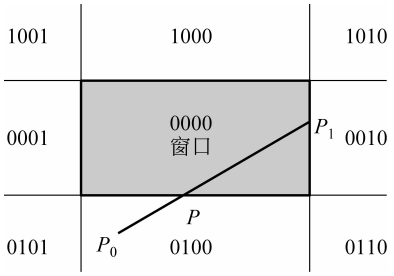

图 5-5 P_0点位于裁剪窗口之外

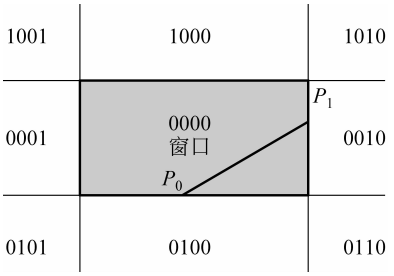

图 5-6 裁剪后的直线段

(4) 实现时，一般按固定顺序左($x = W_{xl}$)，右($x = W_{xr}$)、下($y = W_{yb}$)、上($y = W_{yt}$)计算窗口边界及其延长线与直线段的交点。

2. 交点计算公式

对于端点坐标为 $P_0(x_0, y_0)$ 和 $P_1(x_1, y_1)$ 的直线段，与窗口左边界($x = W_{xl}$)或右边界($x = W_{xr}$)交点的 y 坐标的计算公式为

$$y = k(x - x_0) + y_0，\text{其中 } k = (y_1 - y_0)/(x_1 - x_0)$$

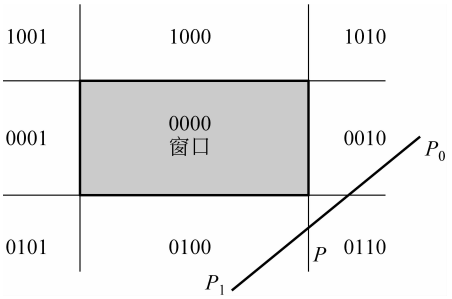

图 5-7　直线段与窗口边界的延长线相交

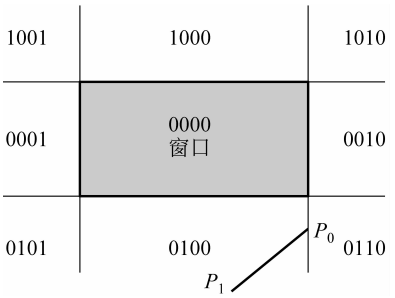

图 5-8　窗口右边界的延长线裁剪后的直线段

与窗口上边界($y=W_{yt}$)或下边界($y=W_{yb}$)交点的 x 坐标的计算公式为

$$x=\frac{y-y_0}{k}+x_0,\quad \text{其中 } k=(y_1-y_0)/(x_1-x_0)$$

```
void CTestView::Cohen()                                    //Cohen－Sutherland 算法
{
    CP2 p;                                                 //交点坐标
    EnCode(P[0]);                                          //起点编码
    EnCode(P[1]);                                          //终点编码
    while(P[0].rc!=0 || P[1].rc!=0)
    {
        if(0!=(P[0].rc & P[1].rc))
        {                                                  //简弃之
            PtCount=0;
            return;
        }
        UINT RC=P[0].rc;
        if(P[0].rc==0) RC=P[1].rc;                         //确保 P0 点位于窗口之外
        //按左、右、下、上的顺序裁剪
        if(RC & LEFT)                                      //P0 点位于窗口的左侧
        {
            p.x=Wxl;                                       //求交点 y
            p.y=P[0].y+(P[1].y-P[0].y) * (p.x-P[0].x)/(P[1].x-P[0].x);
        }
        else if(RC & RIGHT)                                //P0 点位于窗口的右侧
        {
            p.x=Wxr;                                       //求交点 y
            p.y=P[0].y+(P[1].y-P[0].y) * (p.x-P[0].x)/(P[1].x-P[0].x);
        }
        else if(RC & BOTTOM)                               //P0 点位于窗口的下侧
        {
            p.y=Wyb;                                       //求交点 x
            p.x=P[0].x+(P[1].x-P[0].x) * (p.y-P[0].y)/(P[1].y-P[0].y);
        }
        else if(RC & TOP)                                  //P0 点位于窗口的上侧
```

```
        {
            p.y=Wyt;                                    //求交点 x
            p.x=P[0].x+(P[1].x-P[0].x) * (p.y-P[0].y)/(P[1].y-P[0].y);

        }
        if(RC==P[0].rc)
        {
            EnCode(p);
            P[0]=p;
        }
        else
        {
            EnCode(p);
            P[1]=p;
        }
    }
}
```

5.5.8 写出实验报告

结合实验步骤，写出实验报告，同时完整给出 CTestView 类的头文件和源文件。

5.6 思考与练习

1. 实验总结

(1) 在 OnDraw()函数中使用了二维自定义坐标系，原点位于屏幕中心，x 轴水平向右，y 轴垂直向上。而在 OnLButtonDown()和 OnMouseMove()函数中，使用的依然是原点位于屏幕左上角，x 轴水平向右，y 轴垂直向下的二维设备坐标系。自定义坐标系和设备坐标系之间需要进行转换。转换函数为：

```
CP2 CTestView::Convert(CPoint point)                    //坐标系变换
{
    CRect rect;
    GetClientRect(&rect);
    CP2 ptemp;
    ptemp.x=point.x-rect.right/2;
    ptemp.y=rect.bottom/2-point.y;
    return ptemp;
}
```

(2) 裁剪是在直线段绘制后完成的，所以裁剪按钮只有在直线绘制完毕后才能有效。这需要为“裁剪”按钮的 ID 标识符 ID_MCLIP 映射 ON_UPDATE_COMMAND_UI 宏的更新消息处理函数。代码如下：

```
void CTestView::OnUpdateMclip(CCmdUI * pCmdUI)          //剪切图标状态控制
```

```
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable((PtCount>=2)?TRUE:FALSE);
}
```

2. 拓展练习

使用中点分割算法可以避免 Cohen-Sutherland 裁剪算法的求交运算，简单地把直线段等分为两段直线，对每一段直线重复“简取”和“简弃”的处理。对于不能处理的直线段再继续等分下去，直到每一段直线完全能够被“简取”或“简弃”。请编程实现。