实验三 二维图形变换&裁剪

实验报告

一、综述

本次实验内容有关二维图形显示处理的原理、流程和实现方法，包括二维图形空间建模、变换、裁剪、视见变换等。通过本次实验，我能够掌握二维图形的绘制处理以及简单的交互控制手段，实现二维图形的基本变换，同时，对基本变换的算法有了更深入的理解。

二、程序框架

实验程序为 MFC 框架，cg20192DTransView.h 为视图层的头文件，负责声明各种成员变量和成员函数：void ScaleMatrix 定义缩放矩阵，void RotateMatrix 定义旋转矩阵，void TranslateMatrix 定义平移矩阵，void TransLine 实现了直线的 2D 变换，void TransPolygon 实现了多边形的 2D 变换。

cg20192DTransView.cpp 为视图层的源文件，负责定义并实现三种变换矩阵：平移，缩放，旋转，并计算变换序列得到复合矩阵。还包括直线的裁剪算法，以及多边形的裁剪算法。

CCgTransControl.h 为窗口面板中的按键和视图窗口定义成员变量及成员函数，CCgTransControl.cpp 实现面板的功能，包括视见变换，以及各种交互功能：void ViewTransLine 实现了直线的视见变换，void ViewTransPolygon 实现了多边形的视见变换 。

三、算法描述

1. 基本变换矩阵

(1).缩放矩阵

$$S=\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
void Ccg20192DTransView::ScaleMatrix(float Sx, float Sy, float m[3][2])
{
    for (int i = 0; i < 3; i++) {
        m[i][0] *= Sx;
        m[i][1] *= Sy;
    }
}
```

(2).旋转矩阵

$$R=\begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
void Ccg20192DTransView::RotateMatrix(float S, float C, float m[3][2])
{
    float temp;
    for (int i = 0; i < 3; i++) {
        temp = m[i][0];
        m[i][0] = temp * C - m[i][1] * S;
        m[i][1] = temp * S + m[i][1] * C;
    }
}
```

(3).平移矩阵

$$T=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 1 \end{bmatrix}$$

```
void Ccg20192DTransView::TranslateMatrix(float Tx, float Ty, float m[3][2])
{
    // 矩阵平移变换的位移量
    m[2][0] += Tx;
    m[2][1] += Ty;
}
```

2. Cohn-Sutherland 直线裁剪算法

Cohn-Sutherland 直线裁剪算法即对直线段 p1（x1，y1）p2（x2，y2）进行裁剪

1.基本思想：对每条直线段 p1(x1,y1)p2(x2,y2)分三种情况处理

(1) 直线段完全可见，“简取”之。

(2) 直线段完全不可见，“简弃”之。

(3) 直线段既不满足“简取”的条件，也不满足“简弃”的条件，需要对直线段按交点进行分段，分段后重复上述处理。

2 编码方法

编码：对于任一端点(x,y)，根据其坐标所在的区域，赋予一个 4 位的二进制码 D3D2D1D0。

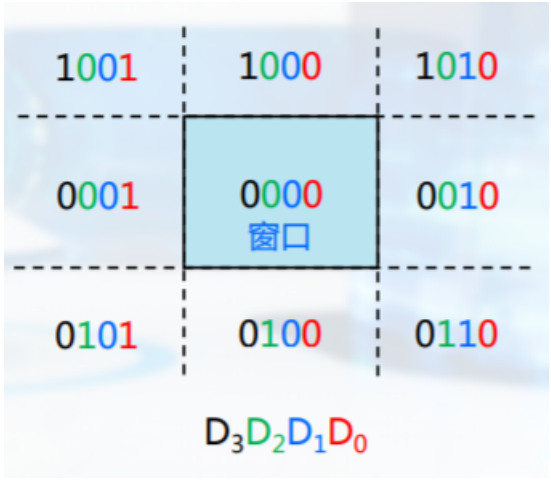
编码规则如下：

若 x<wxl，则 D0=1，否则 D0=0；

若 x>wxr，则 D1=1，否则 D1=0；

若 y<wyb，则 D2=1，否则 D2=0；

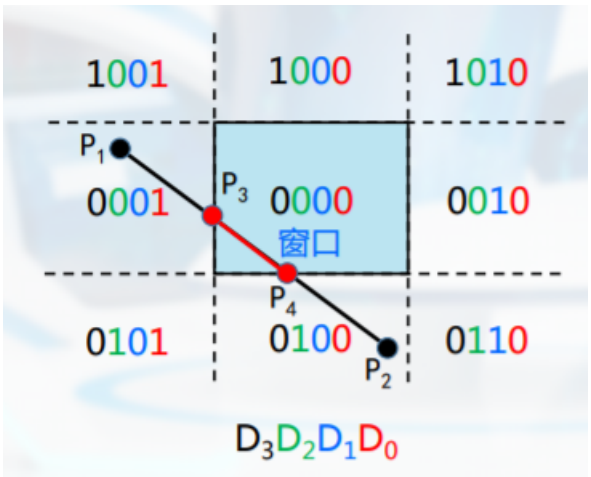
若 y>wyt，则 D3=1，否则 D3=0。



裁剪一条线段时，先求出端点 p1 和 p2 的编码 code1 和 code2，然后：(1)若 code1|code2=0，对直线段应简取之。(2)若 code1&code2≠0，对直线段可简弃之。(3)若上述两条件均丌成立。则需求出直线段不 窗口边界的交点。在交点处把线段一分为二， 其中必有一段完全在窗口外，可以弃之。再对 另一段重复进行上述处理，直到该线段完全被 舍弃或者找到位于窗口内的一段线段为止。

3.具体做法：

按左、下、右、上的顺序求出直线段与窗口边界的交点，分段处理 例：对于直线段 P1P2 ¬求出 P1P2 与左边界有实交点 P3，一分 为二，简弃直线段 P1P3，处理 P2P3 ¬求出 P2P3 与下边界的实交点 P4，一分 为二，简弃 P2P4，剩下的 P3P4 可以简取。



3. Liang-Barsky 直线裁剪算法：

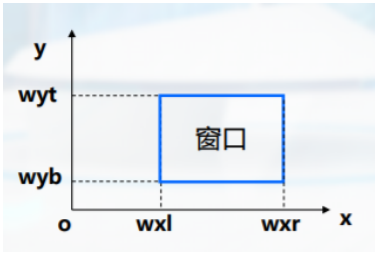
任意直线段 I（X1 ,Y1）J(X2 ,Y2)的参数方程：

$$x = x1 + u \cdot (x2 - x1)$$

$$y = y1 + u \cdot (y2 - y1)$$

$$（0 \le u \le 1）$$

给定裁剪窗口：



如果任一点在窗口内则：

$$wxl \le x1 + u \cdot (x2 - x1) \le wxr$$

$$wyb \le y1 + u \cdot (y2 - y1) \le wyt$$

$$u \cdot (x1 - x2) \le x1 - wxl \text{ 左边界}$$
$$u \cdot (x2 - x1) \le wxr - x1 \text{ 右边界}$$
$$u \cdot (y1 - y2) \le y1 - wyb \text{ 下边界}$$
$$u \cdot (y2 - y1) \le wyt - y1 \text{ 上边界}$$

$$u \cdot pk \le qk，\ k = 1,2,3,4$$

令：

$$p1 = x1 - x2$$
$$p2 = x2 - x1$$
$$p3 = y1 - y2$$
$$p4 = y2 - y1$$

$$q1 = x1 - wxl$$
$$q2 = wxr - x1$$
$$q3 = y1 - wyb$$
$$q4 = wyt - y1$$

u • pk≤qk 其中：k=1, 2, 3, 4

1. 取“=”时求得的 u 对应的是直线不窗口边界的交点
2. 1、2、3、4 分别对应左、右、下、上边界
3. u=0 和 1 时分别对应直线的起点和终点

$$Uone = \max(0, uk|pk < 0, uk|pk < 0)$$
$$Utwo = \min(1, uk|pk > 0, uk|pk > 0)$$

例如：对于 IJ P1、P4 小于 0 Uone 在 0、u1、u4 取大者 P2、P3 大于 0 Utwo 在 1、u2、u3

取小者

限制条件： 如果 Uone≤ Utwo 取可求得两端点

4. Sutherland-Hodgman 多边形裁剪算法

1.基本思想：一次用窗口的一条边来裁剪多边形。

算法的输入是以顶点序列表示的多边形，输出也是一个顶点序列，这些顶点能够构成一个或多个多边形。

处理对象：任意凸多边形。

窗口的任意一条边的所在直线（裁剪线）把窗口所在平面分成两部分：

可见一侧：包含窗口那部分

不可见一侧：不包含窗口那部分

2. 算法说明：

1、已知：多边形顶点数组 src，顶点个数 n，

定义新多边形顶点数组 dest。

2、赋初值：用变量 flag 来标识：

0 表示在内侧，1 表示在外侧。

3、对多边形的 n 条边进行处理，对当前点号的考虑为：0～n-1。

```
for(i=0; i<n; i++)
{
  if(当前第 i 个顶点是否在边界内侧？)
        {
    if(flag!=0) /*前一个点在外侧吗？*/
    {
      flag=0; /*从外到内的情况，将标志置 0,作为下一次循环的前一点标志*/
      (dest + j) =求出交点;   /*将交点 dest 放入新多边形*/
                j++;
            }

    (dest + j)= (src + i);  /*将当前点 srci 放入新多边形*/
            j++;
      }
else
{
    if(flag==0) /*前一个点在内侧吗？*/
    {
      flag=1; /*从内到外的情况，将标志置 1,作为下一次循环的前一点标志*/
      (dest + j) =求出交点;   /*将交点 dest 放入新多边形*/
                j++;
            }
```

```
    }
        s= (src + i);  /*将当前点作为下次循环的前一点*/
  }
```

3.算法特点：

Sutherland－Hodgeman 多边形裁剪算法具有一般性，被裁剪多边形可以是任意凸多边形，裁剪窗口不局限于矩形。

上面的算法是多边形相对窗口的一条边界进行裁剪的实现，对于窗口的每一条边界依次调用该算法程序，并将前一次裁剪的结果多边形作为下一次裁剪时的被裁剪多边形，即可得到完整的多边形裁剪程序。

5. Weiler－Atherton 多边形裁剪算法

一．Weiler－Atherton 任意多边形裁剪算法思想：

假设被裁剪多边形和裁剪窗口的顶点序列都按顺时针方向排列。当两个多边形相交时，交点必然成对出现，其中一个是从被裁剪多边形进入裁剪窗口的交点，称为“入点”，另一个是从被裁剪多边形离开裁剪窗口的交点，称为“出点”。算法从被裁剪多边形的一个入点开始，碰到入点，沿着被裁剪多边形按顺时针方向搜集顶点序列；而当遇到出点时，则沿着裁剪窗口按顺时针方向搜集顶点序列。按上述规则，如此交替地沿着两个多边形的边线行进，直到回到起始点。这时，收集到的全部顶点序列就是裁剪所得的一个多边形。由于可能存在分裂的多边形，因此算法要考虑：将搜集过的入点的入点记号删去，以免重复跟踪。将所有的入点搜集完毕后算法结束。

二．Weiler－Atherton 任意多边形裁剪算法步骤：

1、顺时针输入被裁剪多边形顶点序列Ⅰ放入数组 1 中。

2、顺时针输入裁剪窗口顶点序列Ⅱ放入数组 2 中。

3、求出被裁剪多边形和裁剪窗口相交的所有交点，并给每个交点打上“入”、“出”标记。 然后将交点按顺序插入序列Ⅰ得到新的顶点序列Ⅲ，并放入数组 3 中； 同样也将交点按顺序插入序列Ⅱ得到新的顶点序列Ⅳ，放入数组 4 中；

4、初始化输出数组 Q，令数组 Q 为空。接着从数组 3 中寻找“入”点。如果“入”点没找到，程序结束。

5、如果找到“入”点，则将“入”点放入 S 中暂存。

6、将“入”点录入到输出数组 Q 中。并从数组 3 中将该“入”点的“入”点标记删去。

7、沿数组 3 顺序取顶点：如果顶点不是“出点”，则将顶点录入到输出数组 Q 中，流程转第 7 步。否则，流程转第 8 步。

8、沿数组 4 顺序取顶点：如果顶点不是“入点”，则将顶点录入到输出数组 Q 中，流程转第 8 步。否则，流程转第 9 步。

9、如果顶点不等于起始点 S，流程转第 6 步，继续跟踪数组 3。否则，将数组 Q 输出；流程转第 4 步，寻找可能存在的分裂多边形。算法在第 4 步：满足“入”点没找到的条件时，算法结束。算法的生成过程见下图所示。

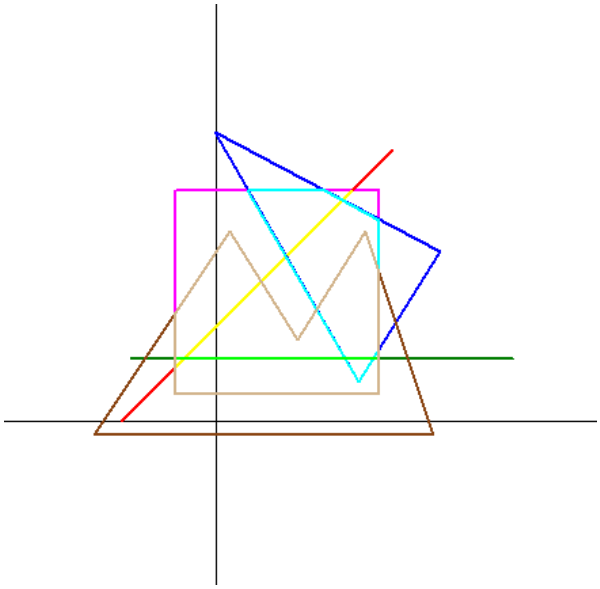
三．Weiler－Atherton 任意多边形裁剪算法特点：

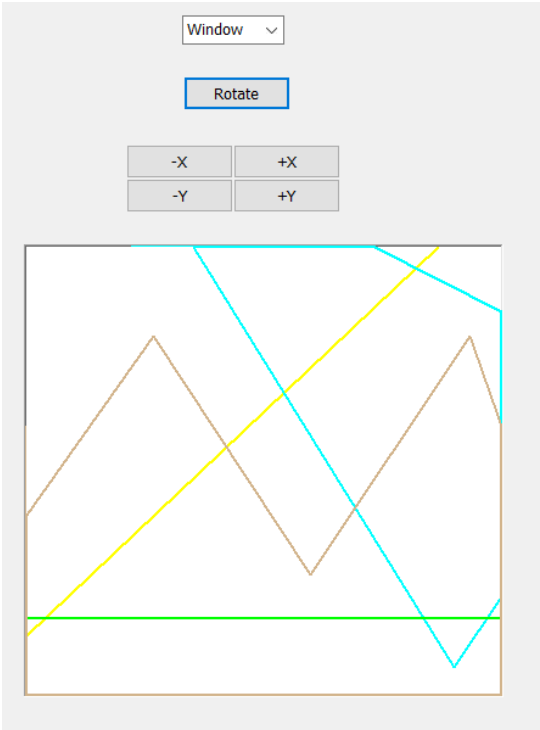
1、裁剪窗口可以是矩形、任意凸多边形、任意凹多边形。

2、可实现被裁剪多边形相对裁剪窗口的内裁或外裁，即保留窗口内的图形或保留窗口外的图形，因此在三维消隐中可以用来处理物体表面间的相互遮挡关系。3、裁剪思想新颖，方法简洁，裁剪一次完成，与裁剪窗口的边数无关。

四、处理流程说明

如图所示，在右侧交互窗口选择需要操作的对象，执行相关操作，进行二维图形的变换处理，并在视图窗口中观察到裁剪后的视见变换结果。





五、源代码

```
//计算变换矩阵
void Ccg20192DTransView::CalculateMatrix(float transMatrix[3][2])
{
    Ccg20192DTransDoc* pDoc = GetDocument();

    switch (pDoc->m_transDir) {
    case 0: // -X
        switch (pDoc->m_transMode) {
        case 0: // Move
            TranslateMatrix(-DELTAX, 0, transMatrix);
            break;
        case 1: // rotate
            RotateMatrix(-sin(DELTATHETA), cos(DELTATHETA), transMatrix);
            break;
        case 2: // Scale
            ScaleMatrix(SSCALEX, 1, transMatrix);
            break;
        }
        break;
    case 1: // +X
        switch (pDoc->m_transMode) {
        case 0: // Move
            TranslateMatrix(DELTAX, 0, transMatrix);
            break;
        case 1: // rotate
            RotateMatrix(sin(DELTATHETA), cos(DELTATHETA), transMatrix);
            break;
        case 2: // Scale
            ScaleMatrix(LSCALEX, 1, transMatrix);
            break;
        }
        break;
    case 2: // -Y
        switch (pDoc->m_transMode) {
        case 0: // Move
            TranslateMatrix(0, -DELTAY, transMatrix);
```

```
            break;
        case 1: // rotate
            RotateMatrix(-sin(DELTATHETA), cos(DELTATHETA), transMatrix);
            break;
        case 2: // Scale
            ScaleMatrix(1, SSCALEY, transMatrix);
            break;
        }
        break;
    case 3: // +Y
        switch (pDoc->m_transMode) {
        case 0: // Move
            TranslateMatrix(0, DELTAY, transMatrix);
            break;
        case 1: // rotate
            RotateMatrix(sin(DELTATHETA), cos(DELTATHETA), transMatrix);
            break;
        case 2: // Scale
            ScaleMatrix(1, LSCALEY, transMatrix);
            break;
        }
        break;
    }
}

//缩放矩阵
void Ccg20192DTransView::ScaleMatrix(float Sx, float Sy, float m[3][2])
{
    for (int i = 0; i < 3; i++) {
        m[i][0] *= Sx;
        m[i][1] *= Sy;
    }
}

//旋转矩阵
void Ccg20192DTransView::RotateMatrix(float S, float C, float m[3][2])
{
    float temp;
    for (int i = 0; i < 3; i++) {
```

```
        temp = m[i][0];
        m[i][0] = temp * C - m[i][1] * S;
        m[i][1] = temp * S + m[i][1] * C;
    }
}

//平移矩阵
void Ccg20192DTransView::TranslateMatrix(float Tx, float Ty, float m[3][2])
{
    // 矩阵平移变换的位移量
    m[2][0] += Tx;
    m[2][1] += Ty;
}



//绘制直线
void Ccg20192DTransView::DisplayLine(CDC* pDC, CPoint p1, CPoint p2, COLORREF
rgbColor)
{
    Ccg20192DTransDoc* pDoc = GetDocument();

    CPen newPen;
    CPen* oldPen;
    CPoint VP1, VP2;

    newPen.CreatePen(PS_SOLID, 2, rgbColor);
    oldPen = (CPen*)pDC->SelectObject(&newPen);

    VP1.x = m_wndWidth / 2 + p1.x;
    VP1.y = m_wndHeight / 2 - p1.y;
    VP2.x = m_wndWidth / 2 + p2.x;
    VP2.y = m_wndHeight / 2 - p2.y;

    pDC->MoveTo(VP1);
    pDC->LineTo(VP2);

    pDC->SelectObject(oldPen);
    newPen.DeleteObject();
}
```

```
//直线 2D 变换
void Ccg20192DTransView::TransLine(CPoint p1, CPoint p2, CPoint* tp1, CPoint* tp2,
float transMatrix[3][2])
{
    // 更改移动后的线段端点坐标 p1, p2,利用矩阵乘法(x,y,1) * transMatrix[3][2]
    tp1->x = p1.x * transMatrix[0][0] +
             p1.y * transMatrix[1][0] +
                    transMatrix[2][0];
    tp1->y = p1.x * transMatrix[0][1] +
             p1.y * transMatrix[1][1] +
                    transMatrix[2][1];
    tp2->x = p2.x * transMatrix[0][0] +
             p2.y * transMatrix[1][0] +
                    transMatrix[2][0];
    tp2->y = p2.x * transMatrix[0][1] +
             p2.y * transMatrix[1][1] +
                    transMatrix[2][1];
}

// Cohn-Sutherland Subdivision Line Clip
int  Ccg20192DTransView::ClipLine(int* x1, int* y1, int* x2, int* y2)
{
    int visible, m_window[4];
    Ccg20192DTransDoc* pDoc = GetDocument();
    //创建四条边
    m_window[0] = pDoc->m_wndLx;    m_window[1] = pDoc->m_wndRx;
    m_window[2] = pDoc->m_wndRy;    m_window[3] = pDoc->m_wndLy;
    for (int i = 0; i < 4; i++) { //沿着窗口四条边分别裁剪线段
        //先判断是否完全可见或完全不可见
        visible = LineVisible(x1, y1, x2, y2);
        if (visible == 1) return 1;         //整体完全可见
        if (visible == 0) return 0;         //整体完全不可见
        //其他情况：部分可见，且第一个点在窗口外
        if (LineCross(*x1, *y1, *x2, *y2, i)) { //判断是否经过该边界
            if (i < 2 && *x2 - *x1) {           //左边界和右边界
                float m = (float)(*y2 - *y1) / (*x2 - *x1); //计算斜率
                float iy = m * (m_window[i] - *x1) + *y1;   //计算出边界与线段
的交点
```

```
    //更新端点，舍弃窗口边界之外的部分
    if (i == 0) {   //左边界
        if (*x1 < *x2) {
            *x1 = m_window[i];
            *y1 = iy;
        }
        else {
            *x2 = m_window[i];
            *y2 = iy;
        }
    }
    else {          //右边界
        if (*x1 > *x2) {
            *x1 = m_window[i];
            *y1 = iy;
        }
        else {
            *x2 = m_window[i];
            *y2 = iy;
        }
    }
}
else if (*y2 - *y1) {           //上边界和下边界
    float m = (float)(*x2 - *x1) / (*y2 - *y1);
    float ix = m * (m_window[i] - *y1) + *x1;
    if (i == 2) {   //上边界
        if (*y1 > *y2) {
            *x1 = ix;
            *y1 = m_window[i];
        }
        else {
            *x2 = ix;
            *y2 = m_window[i];
        }
    }
    else {          //下边界
        if (*y1 < *y2) {
            *x1 = ix;
            *y1 = m_window[i];
```

```
                    }
                    else {
                        *x2 = ix;
                        *y2 = m_window[i];
                    }
                }
            }
        }
    }
    return 1;
}

int Ccg20192DTransView::LineVisible(int* x1, int* y1, int* x2, int* y2)
{
    int pcode1, pcode2;
    // 为0表示位于框内部
    //| 0101 | 0100 | 0110 |
    //| 0001 | 0000 | 0010 |
    //| 1001 | 1000 | 1010 |
    pcode1 = pCode(x1, y1);
    pcode2 = pCode(x2, y2);

    if (!pcode1 && !pcode2)    return 1;     //完全可见
    if ((pcode1 & pcode2) != 0)  return 0;     //完全不可见
    if (pcode1 == 0) {
        float temp;
        temp = *x1;  *x1 = *x2;  *x2 = temp;
        temp = *y1;  *y1 = *y2;  *y2 = temp;
    }
    return 2;
}

int Ccg20192DTransView::pCode(int* x, int* y)
{
    int code = 0;
    Ccg20192DTransDoc* pDoc = GetDocument();

    if (*x <= pDoc->m_wndLx)  code |= 1;//左
    if (*x >= pDoc->m_wndRx)  code |= 2;//右
```

```
    if (*y >= pDoc->m_wndRy)  code |= 4;//上
    if (*y <= pDoc->m_wndLy)  code |= 8;//下

    return code;
}

int Ccg20192DTransView::LineCross(int x1, int y1, int x2, int y2, int i)
{
    int visible1, visible2;

    visible1 = pVisible(x1, y1, i);
    visible2 = pVisible(x2, y2, i);

    if (visible1 != visible2) return 1;
    else                      return 0;

}

int Ccg20192DTransView::pVisible(int x, int y, int i)
{
    int visible = 0;
    Ccg20192DTransDoc* pDoc = GetDocument();

    switch (i) {
    case 0: //左边界
        if (x >= pDoc->m_wndLx)  visible = 1; break;
    case 1: //右边界
        if (x <= pDoc->m_wndRx)  visible = 1; break;
    case 2: //上边界
        if (y <= pDoc->m_wndRy)  visible = 1; break;
    case 3: //下边界
        if (y >= pDoc->m_wndLy)  visible = 1; break;
    }
    return visible;
}

// Liang-Barsky Line Clip
int Ccg20192DTransView::LB_ClipLine(int* x1, int* y1, int* x2, int* y2)
{
```

```
    Ccg20192DTransDoc* pDoc = GetDocument();
    float dx = *x1 - *x2, dy = *y1 - *y2;
    if (dy == 0) {  // 水平
        if (*x1 > * x2) {   //交换
            int temp = *x2; *x2 = *x1;  *x1 = temp;
        }
        if (*y1 >= pDoc->m_wndRy || *y1 <= pDoc->m_wndLy)   //超出范围
            return 0;
        if ((*x1 >= pDoc->m_wndRx && *x2 >= pDoc->m_wndRx) || (*x1 <= pDoc->m_wndLx
&& *x2 <= pDoc->m_wndLx))   //超出范围
            return 0;
        if (*x1 <= pDoc->m_wndLx)
            *x1 = pDoc->m_wndLx;
        if (*x2 >= pDoc->m_wndRx)
            *x2 = pDoc->m_wndRx;
    }
    else if (dx == 0) {   //垂直
        if (*y1 > * y2) {   // 交换
            int temp = *y2; *y2 = *y1;  *y1 = temp;
        }
        if (*x1 >= pDoc->m_wndRx || *x1 <= pDoc->m_wndLx)   //超出范围
            return 0;
        if ((*y1 >= pDoc->m_wndRy && *y2 >= pDoc->m_wndRy) || (*y1 <= pDoc->m_wndLy
&& *y2 <= pDoc->m_wndLy))   //超出范围
            return 0;
        if (*y1 <= pDoc->m_wndLy)
            *y1 = pDoc->m_wndLy;
        if (*y2 >= pDoc->m_wndRy)
            *y2 = pDoc->m_wndRy;
    }
    else {  //一般情况
        float x11, x22, y11, y22;

        int p1 = *x1 - *x2;
        int p2 = *x2 - *x1;
        int p3 = *y1 - *y2;
        int p4 = *y2 - *y1;

        int q1 = *x1 - pDoc->m_wndLx;
```

```
int q2 = pDoc->m_wndRx - *x1;
int q3 = *y1 - pDoc->m_wndLy;
int q4 = pDoc->m_wndRy - *y1;

float u1 = (float)q1 / p1;
float u2 = (float)q2 / p2;
float u3 = (float)q3 / p3;
float u4 = (float)q4 / p4;

int pk1, pk2, pk3, pk4;
float uk1, uk2, uk3, uk4;
if (p1 < 0) {
    pk1 = p1;
    uk1 = u1;
    pk3 = p2;
    uk3 = u2;
}
else {
    pk3 = p1;
    uk3 = u1;
    pk1 = p2;
    uk1 = u2;
}
if (p3 < 0) {
    pk2 = p3;
    uk2 = u3;
    pk4 = p4;
    uk4 = u4;
}
else {
    pk4 = p3;
    uk4 = u3;
    pk2 = p4;
    uk2 = u4;
}

float umax = max(0, max(uk1, uk2));
float umin = min(1, min(uk3, uk4));
```

```
        if (umax > umin)
            return 0;
        else {
            x11 = *x1 + umax * (*x2 - *x1);
            y11 = *y1 + umax * (*y2 - *y1);
            x22 = *x1 + umin * (*x2 - *x1);
            y22 = *y1 + umin * (*y2 - *y1);

            *x1 = (int)(x11 + 0.5);
            *x2 = (int)(x22 + 0.5);
            *y1 = (int)(y11 + 0.5);
            *y2 = (int)(y22 + 0.5);
        }
        return 1;
    }
}

//绘制多边形
void Ccg20192DTransView::DisplayPolygon(CDC* pDC, int pointNumber,
    CPoint transPolygon[N], COLORREF rgbColor)
{
    Ccg20192DTransDoc* pDoc = GetDocument();

    CPen newPen;
    CPen* oldPen;
    CPoint VPolygon[N];

    newPen.CreatePen(PS_SOLID, 2, rgbColor);
    oldPen = (CPen*)pDC->SelectObject(&newPen);

    for (int i = 0; i < pointNumber; i++) {
        VPolygon[i].x = m_wndWidth / 2 + transPolygon[i].x;
        VPolygon[i].y = m_wndHeight / 2 - transPolygon[i].y;
    }

    pDC->MoveTo(VPolygon[0]);
    for (int i = 1; i < pointNumber; i++) pDC->LineTo(VPolygon[i]);

    pDC->SelectObject(oldPen);
```

```
    newPen.DeleteObject();
}


void Ccg20192DTransView::TransPolygon(int pointNumber, CPoint spPolygon[N],
    CPoint transPolygon[N], float transMatrix[3][2])
{
    Ccg20192DTransDoc* pDoc = GetDocument();

    for (int i = 0; i < pointNumber; i++) {
        transPolygon[i].x = spPolygon[i].x * transMatrix[0][0] +
                            spPolygon[i].y * transMatrix[1][0] +
                                             transMatrix[2][0];
        transPolygon[i].y = spPolygon[i].x * transMatrix[0][1] +
                            spPolygon[i].y * transMatrix[1][1] +
                                             transMatrix[2][1];
    }
}


// Sutherland-Hodgman Polygon Clip
int Ccg20192DTransView::ClipPolygon(int n, CPoint* tPoints, int* cn, CPoint*
cPoints)
{
    int Nin, Nout, ix, iy, Sx, Sy;
    Ccg20192DTransDoc* pDoc = GetDocument();

    Nin = n;
    for (int i = 0; i < 4; i++) {  // Along the window border
        *cn = 0;
        for (int j = 0; j < Nin; j++) {  // Scan polygon every point and line.
            if (j > 0) {
                if (LineCross(Sx, Sy, tPoints[j].x, tPoints[j].y, i)) {
                    interSect(Sx, Sy, tPoints[j].x, tPoints[j].y, i, &ix, &iy);
                    outPut(ix, iy, cn, cPoints);
                }
            }
            Sx = tPoints[j].x;
            Sy = tPoints[j].y;
```

```
            if (pVisible(Sx, Sy, i)) outPut(Sx, Sy, cn, cPoints);
        }

        Nin = *cn;
        if (*cn == 0) return 0;
        for (int j = 0; j < Nin; j++) {
            tPoints[j].x = cPoints[j].x;
            tPoints[j].y = cPoints[j].y;
        }

        if (cPoints[0].x != cPoints[Nin - 1].x ||
            cPoints[0].y != cPoints[Nin - 1].y) {

            tPoints[Nin].x = cPoints[Nin].x = cPoints[0].x;
            tPoints[Nin].y = cPoints[Nin].y = cPoints[0].y;

            Nin++;
            *cn = Nin;
        }
    }
    return 1;
}

void Ccg20192DTransView::interSect(int Sx, int  Sy, int Px, int Py,
    int  i, int* ix, int* iy)
{
    Ccg20192DTransDoc* pDoc = GetDocument();

    // Please fill in the right code below lines ...
    switch (i) {
    case 0: // Left
        *ix = pDoc->m_wndLx;
        *iy = (Sy - Py) * (pDoc->m_wndLx - Px) / (Sx - Px) + Py;
        break;
    case 1: // Right
        *ix = pDoc->m_wndRx;
        *iy = (Sy - Py) * (pDoc->m_wndRx - Px) / (Sx - Px) + Py;
        break;
    case 2: // Top
```

```
        *iy = pDoc->m_wndRy;
        *ix = (Sx - Px) * (pDoc->m_wndRy - Py) / (Sy - Py) + Px;
        break;
    case 3: // Bottom
        *iy = pDoc->m_wndLy;
        *ix = (Sx - Px) * (pDoc->m_wndLy - Py) / (Sy - Py) + Px;
        break;
    }
}

void Ccg20192DTransView::outPut(int x, int y, int* cn, CPoint* cPoints)
{
    cPoints[*cn].x = x;
    cPoints[*cn].y = y;
    (*cn)++;
}

void CCgTransControl::ViewTransLine(CDC* pDC, CRect dcRect)
{
    CPen newPen;
    CPen* oldPen;
    Ccg20192DTransDoc* pDoc = (Ccg20192DTransDoc*)GetDocument();

    // Create new red-color pen to Draw Clipping Line
    newPen.CreatePen(PS_SOLID, 2, RGB(255, 255, 0));
    oldPen = (CPen*)pDC->SelectObject(&newPen);

    int vx1, vy1, vx2, vy2;

    int wndWidth = pDoc->m_wndRx - pDoc->m_wndLx;  // Space Window size parameters.
    int wndHeight = pDoc->m_wndRy - pDoc->m_wndLy;

    float Sx = (float)(dcRect.right) / wndWidth;    // View-Space Viewport size
parameters.
    float Sy = (float)(dcRect.bottom) / wndHeight;

    // Please fill the right codes below lines ...
    vx1 = (pDoc->CP1.x - pDoc->m_wndLx) * Sx;
    vy1 = (pDoc->CP1.y - pDoc->m_wndLy) * Sy;
```

```
    vx2 = (pDoc->CP2.x - pDoc->m_wndLx) * Sx;
    vy2 = (pDoc->CP2.y - pDoc->m_wndLy) * Sy;

    pDC->MoveTo(vx1, dcRect.bottom - vy1);
    pDC->LineTo(vx2, dcRect.bottom - vy2);

    // Remember: must delete newPen every time.
    pDC->SelectObject(oldPen);
    newPen.DeleteObject();
}

void CCgTransControl::ViewTransPolygon(CDC* pDC, CRect dcRect)
{
    CPen newPen;
    CPen* oldPen;
    Ccg20192DTransDoc* pDoc = (Ccg20192DTransDoc*)GetDocument();

    // Create new red-color pen to Draw Clipping Line
    newPen.CreatePen(PS_SOLID, 2, RGB(0, 255, 255));
    oldPen = (CPen*)pDC->SelectObject(&newPen);

    CPoint m_VPolygon[N];

    int wndWidth = pDoc->m_wndRx - pDoc->m_wndLx;  // Space Window size parameters.
    int wndHeight = pDoc->m_wndRy - pDoc->m_wndLy;

    float Sx = (float)(dcRect.right) / wndWidth;    // View-Space Viewport size
parameters.
    float Sy = (float)(dcRect.bottom) / wndHeight;

    // Please fill the right codes below lines ...
    for (int i = 0; i < pDoc->m_clipPointNumber; i++) {
        m_VPolygon[i].x = (pDoc->m_clipPolygon[i].x - pDoc->m_wndLx) * Sx;
        m_VPolygon[i].y   =   dcRect.bottom   -   (pDoc->m_clipPolygon[i].y   -
pDoc->m_wndLy) * Sy;
    }

    pDC->MoveTo(m_VPolygon[0]);
    for (int i = 1; i < pDoc->m_clipPointNumber; i++) pDC->LineTo(m_VPolygon[i]);
```

```
    // Remember: must delete newPen every time.
    pDC->SelectObject(oldPen);
    newPen.DeleteObject();
}
```