第5章

二维变换和二维观察

5.1 图形变换基本知识

为方便读者更好地理解图形变换，本节对图形变换的基本知识做简单介绍。

5.1.1 矢量和矩阵

1. 矢量的相关概念

（1）矢量的定义。

$$\boldsymbol{U}=\begin{bmatrix}u_x\\u_y\\u_z\end{bmatrix}\quad \boldsymbol{V}=\begin{bmatrix}v_x\\v_y\\v_z\end{bmatrix}$$

（2）矢量的和。

$$\boldsymbol{U}+\boldsymbol{V}=\begin{bmatrix}u_x+v_x\\u_y+v_y\\u_z+v_z\end{bmatrix}$$

（3）矢量的数乘。

$$\boldsymbol{k}\cdot\boldsymbol{U}=\begin{bmatrix}ku_x\\ku_y\\ku_z\end{bmatrix}$$

（4）矢量的点积。

$$\boldsymbol{U}\cdot\boldsymbol{V}=u_xv_x+u_yv_y+u_zv_z$$

（5）矢量的长度。

$$\|\boldsymbol{U}\|=\sqrt{\boldsymbol{U}\cdot\boldsymbol{U}}=\sqrt{u_x^2+u_y^2+u_z^2}$$

（6）矢量的性质。

$$\boldsymbol{U}\cdot\boldsymbol{V}=\boldsymbol{V}\cdot\boldsymbol{U}$$
$$\boldsymbol{U}\cdot\boldsymbol{V}=\boldsymbol{0}\Leftrightarrow\boldsymbol{U}\perp\boldsymbol{V}$$
$$\boldsymbol{U}\cdot\boldsymbol{U}=\boldsymbol{0}\Leftrightarrow\boldsymbol{U}=\boldsymbol{0}$$

（7）矢量的夹角。

矢量的夹角如图 5-1 所示。

$$\cos\theta=\frac{\boldsymbol{U}\cdot\boldsymbol{V}}{\|\boldsymbol{U}\|\cdot\|\boldsymbol{V}\|}$$

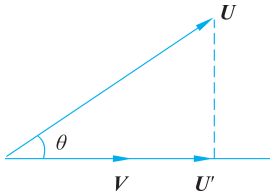


图 5-1 矢量的夹角

或

$$\boldsymbol{U}\cdot\boldsymbol{V}=\|\boldsymbol{U}\|\cdot\|\boldsymbol{V}\|\cos\theta$$

(8) 矢量的叉积。

$$\boldsymbol{U}\times\boldsymbol{V}=\begin{vmatrix} i & j & k \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix}=(u_yv_z-u_zv_y)i+(u_zv_x-u_xv_z)j+(u_xv_y-u_yv_x)k$$

或

$$\boldsymbol{U}\times\boldsymbol{V}=\|\boldsymbol{U}\|\cdot\|\boldsymbol{V}\|\sin\theta\cdot\boldsymbol{I}$$

其中,$\boldsymbol{I}$ 表示 $\boldsymbol{U}\times\boldsymbol{V}$ 的单位向量。

2. 矩阵的相关概念

(1) 矩阵的定义。

矩阵是由 $m\times n$ 个数按照一定位置排列的一个整体,简称 $m\times n$ 矩阵。

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

(2) 矩阵的加法。

设 $\boldsymbol{A}$、$\boldsymbol{B}$ 为两个具有相同行和列元素的矩阵,则:

$$\boldsymbol{A}+\boldsymbol{B}=\begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

(3) 矩阵的数乘。

$$\boldsymbol{k}\cdot\boldsymbol{A}=[ka_{ij}]\quad i=1,\cdots,m,j=1,\cdots,n$$

(4) 矩阵的乘法。

设 $\boldsymbol{A}$ 为 2×3 矩阵,$\boldsymbol{B}$ 为 3×2 矩阵,则

$$\boldsymbol{C}=\boldsymbol{A}\cdot\boldsymbol{B}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}\cdot\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

$$=\begin{bmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} \end{bmatrix}$$

(5) 单位矩阵。

在一个矩阵中,其主对角线上各元素 $a_{ij}=1$,而其余元素都为 0,这种矩阵称为单位矩阵。n 阶单位矩阵通常记作 $\boldsymbol{I}_n$。

(6) 逆矩阵。

若矩阵 $\boldsymbol{A}$ 存在 $\boldsymbol{A}\cdot\boldsymbol{A}^{-1}=\boldsymbol{A}^{-1}\cdot\boldsymbol{A}=\boldsymbol{I}$,则称 $\boldsymbol{A}^{-1}$ 为 $\boldsymbol{A}$ 的逆矩阵。

(7) 矩阵的转置。

把矩阵 $\boldsymbol{A}=(a_{ij})_{m\times n}$ 的行和列互换而得到的 $n\times m$ 矩阵称为 $\boldsymbol{A}$ 的转置矩阵,记为 $\boldsymbol{A}^{\mathrm{T}}$。

$$(\boldsymbol{A}^{\mathrm{T}})^{\mathrm{T}}=\boldsymbol{A},(\boldsymbol{A}+\boldsymbol{B})^{\mathrm{T}}=\boldsymbol{A}^{\mathrm{T}}+\boldsymbol{B}^{\mathrm{T}},(a\boldsymbol{A})^{\mathrm{T}}=a\boldsymbol{A}^{\mathrm{T}},(\boldsymbol{A}\cdot\boldsymbol{B})^{\mathrm{T}}=\boldsymbol{B}^{\mathrm{T}}\cdot\boldsymbol{A}^{\mathrm{T}}$$

若 $\boldsymbol{A}$ 为 n 阶矩阵，且 $\boldsymbol{A}=\boldsymbol{A}^{\mathrm{T}}$，则 $\boldsymbol{A}$ 是对称矩阵。

(8) 矩阵运算的基本性质。

① 矩阵的交换律与结合律。

$$\boldsymbol{A}+\boldsymbol{B}=\boldsymbol{B}+\boldsymbol{A}$$

$$\boldsymbol{A}+(\boldsymbol{B}+\boldsymbol{C})=(\boldsymbol{A}+\boldsymbol{B})+\boldsymbol{C}$$

② 矩阵数乘的分配律及结合律。

$$a(\boldsymbol{A}+\boldsymbol{B})=a\boldsymbol{A}+a\boldsymbol{B}$$

$$a(\boldsymbol{A}\cdot\boldsymbol{B})=(a\boldsymbol{A})\cdot\boldsymbol{B}=\boldsymbol{A}\cdot(a\boldsymbol{B})$$

$$(a+b)\boldsymbol{A}=a\boldsymbol{A}+b\boldsymbol{A}$$

$$a(b\boldsymbol{A})=(ab)\boldsymbol{A}$$

③ 矩阵乘法的结合律及分配律。

$$\boldsymbol{A}(\boldsymbol{B}\cdot\boldsymbol{C})=(\boldsymbol{A}\cdot\boldsymbol{B})\boldsymbol{C}$$

$$(\boldsymbol{A}+\boldsymbol{B})\cdot\boldsymbol{C}=\boldsymbol{A}\cdot\boldsymbol{C}+\boldsymbol{B}\cdot\boldsymbol{C}$$

$$\boldsymbol{C}\cdot(\boldsymbol{A}+\boldsymbol{B})=\boldsymbol{C}\cdot\boldsymbol{A}+\boldsymbol{C}\cdot\boldsymbol{B}$$

注意：矩阵的乘法不适合交换律。

5.1.2　齐次坐标

所谓齐次坐标表示法，就是由 $n+1$ 维向量表示一个 n 维向量。例如将 n 维向量$(p_1,p_2,\cdots,p_n)$表示为$(hp_1,hp_2,\cdots,hp_n,h)$，其中 h 称为哑坐标。普通坐标和齐次坐标有如下关系。

(1) h 可以取不同的值，所以同一点的齐次坐标不是唯一的。例如普通坐标系下的点(2,3) 变换为齐次坐标可以是(1,1.5,0.5)，(4,6,2)，(6,9,3) 等。

(2) 普通坐标与齐次坐标是“一对多”的关系。由普通坐标$\times h$ 可转换为齐次坐标，由齐次坐标$\div h$ 可以得到普通坐标。

(3) 当 $h=1$ 时，产生的齐次坐标称为“规格化坐标”，也称为“规范化齐次坐标”，因此前 n 个坐标就是普通坐标系下的 n 维坐标。

(x,y)点对应的齐次坐标为(x_h,y_h,h)，其中 $x_h=hx$，$y_h=hy$，$h\neq0$。(x,y)点对应的齐次坐标为三维空间的一条直线，即：

$$\begin{cases}x_h=hx\\y_h=hy\\z_h=h\end{cases}\tag{5-1}$$

将 $h=1$ 时的坐标称为规范化齐次坐标，齐次坐标有如下作用。

(1) 将各种变换用阶数统一的矩阵表示。提供了用矩阵运算把二维、三维甚至高维空间上的一个点从一个坐标系变换到另一个坐标系的有效方法。

(2) 便于表示无穷远点。例如：$(x\times h,y\times h,h)$，令 $h=0$。

(3) 采用齐次坐标形式表示的变换矩阵可以将直线变换成直线段，将平面变换成平面，将多边形变换成多边形，将多面体变换成多面体。

(4) 齐次坐标变换具有统一的表示形式，便于变换合成以及硬件实现。

5.2 基本二维变换

图形变换是将图形的几何信息经过几何变换后产生新的图形的过程，是计算机图形学的基础内容之一。图形变换可以把用户坐标系与设备坐标系联系起来，可以由简单图形生成复杂图形，可以用二维图形表示三维形体等。图形变换有两种形式，一种是图形不变，坐标系改变，另一种是图形改变而坐标系不变，本节是针对坐标系的改变而言的，这种变换又称为图形的几何变换。基本二维图形的几何变换（简称基本二维变换）是改变二维图形坐标描述的变换，例如改变二维图形的方向、尺寸和形状等。具体来说，图形变换包括平移、旋转、缩放等，下面分别进行介绍。

5.2.1 平移变换

所谓平移变换是图形对象沿直线运动产生的变换。如图 5-2 所示，点 $P(x,y)$ 平移到 $P'(x',y')$，其平移向量为 (t_x,t_y)，平移公式可以表示为：

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned} \tag{5-2}$$

图 5-3 给出了平移变换的例子，如果用矩阵表示平移变换，过程如下所示。

$$\boldsymbol{P} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad \boldsymbol{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}, \quad \boldsymbol{P'} = \boldsymbol{P} + \boldsymbol{T} \tag{5-3}$$

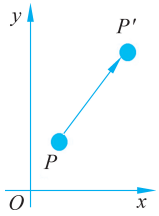


图 5-2 二维平移变换

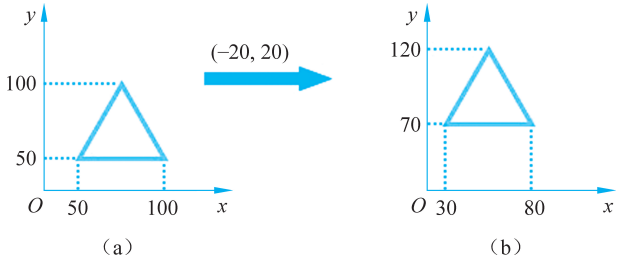


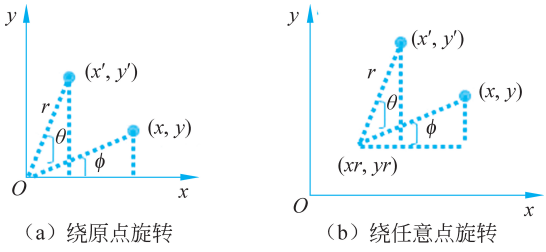
图 5-3 二维平移变换实例

5.2.2 旋转变换

所谓旋转变换是图形对象沿圆弧路径运动产生的变换，如图 5-4 所示。旋转变换有 3 个输入参数：①基准点，即绕哪一点旋转，可以是坐标原点或任意一点；②旋转角度 θ；③旋转方向，本书约定逆时针方向为正方向。

接下来推导旋转变换的变换矩阵。为方便起见，此处讨论绕原点的旋转，如图 5-5 所示。已知二维平面上的 $P(x,y)$ 点绕坐标原点逆时针旋转 θ 角到 $P'(x',y')$，那么 P' 的坐标是多少呢？假设 $P(x,y)$ 点与 x 轴夹角为 ϕ，$P(x,y)$ 点到坐标原点的距离为 r，那么 $P'(x',y')$ 的坐标推导公式如下：

$$\begin{aligned} x' &= r\cos(\theta+\phi) = r(\cos\phi\cos\theta - \sin\phi\sin\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ &= x\cos\theta - y\sin\theta \\ y' &= r\sin(\theta+\phi) = r(\cos\phi\sin\theta + \sin\phi\cos\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \\ &= x\sin\theta + y\cos\theta \end{aligned} \tag{5-4}$$



（a）绕原点旋转　　（b）绕任意点旋转

图 5-4　绕原点和任意点的二维旋转变换

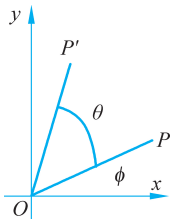


图 5-5　二维旋转变换

绕二维空间任意点的旋转变换坐标应该如何表示呢？可以先将空间任意点平移到坐标原点，再进行旋转变换，最后做反向平移即可。和平移变换一样，旋转变换也是一种不产生变形而移动对象的刚体变换。用矩阵表示绕原点的旋转变换如式(5-5)所示。

$$\boldsymbol{P}=\begin{bmatrix}x\\y\end{bmatrix},\quad \boldsymbol{P}'=\begin{bmatrix}x'\\y'\end{bmatrix},\quad \boldsymbol{R}=\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix},\quad \boldsymbol{P}'=\boldsymbol{R}\cdot\boldsymbol{P} \tag{5-5}$$

5.2.3　缩放变换

缩放变换又称为变比变换，是改变图形对象大小的变换。缩放变换的输入参数包括：①变比因子(s_x，s_y)；②基准点；③变比方向。基准点为坐标原点的缩放变换可表示如下：

$$\begin{aligned}x'&=xs_x\\ y'&=ys_y\end{aligned} \tag{5-6}$$

基准点为固定参考点(x_f，y_f)的缩放变换公式为：

$$\begin{aligned}x'&=x_f+(x-x_f)s_x\\ y'&=y_f+(y-y_f)s_y\end{aligned} \tag{5-7}$$

如果用矩阵形式表示基准点为坐标原点的缩放变换，如式(5-8)所示。

$$\boldsymbol{P}=\begin{bmatrix}x\\y\end{bmatrix},\quad \boldsymbol{P}'=\begin{bmatrix}x'\\y'\end{bmatrix},\quad \boldsymbol{S}=\begin{bmatrix}s_x & 0\\ 0 & s_y\end{bmatrix},\quad \boldsymbol{P}'=\boldsymbol{S}\cdot\boldsymbol{P} \tag{5-8}$$

图 5-6 是以一个基准点为坐标原点的缩放变换的例子，在 x 轴方向上放大 2 倍，在 y 轴方向上不变，变比方向分别沿 x 轴和 y 轴的正方向。

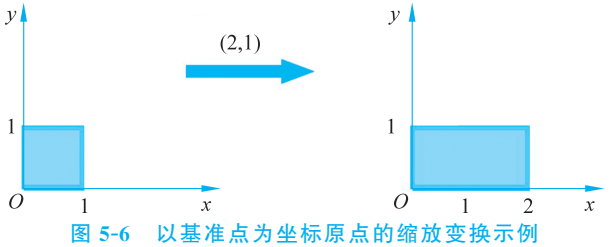


图 5-6　以基准点为坐标原点的缩放变换示例

对于二维缩放变换，有如下结论：

(1) 如果$|s_x|$或$|s_y|$大于 1，则表示图形在 x 轴方向或 y 轴方向放大；

(2) 如果$|s_x|$或$|s_y|$小于 1，则表示图形在 x 轴方向或 y 轴方向缩小；

(3) 如果$|s_x|=|s_y|$，则表示均匀缩放；

(4) 如果$|s_x|\neq|s_y|$，则表示差值缩放；

(5) 如果$|s_x|$或$|s_y|$等于1,则表示图形在x轴方向或y轴方向不变;

(6) 如果s_x或s_y小于零,则表示图形在x轴方向或y轴方向做镜面变换。

5.2.4 基本二维变换的矩阵表示

在图形系统中,矩阵是实现变换的标准方法。在上述讨论中,平移变换、旋转变换和缩放变换的矩阵表示形式分别如下。

(1) 平移变换:$\boldsymbol{P}'=\boldsymbol{P}+\boldsymbol{T}$。

(2) 旋转变换:$\boldsymbol{P}'=\boldsymbol{R}\cdot\boldsymbol{P}$。

(3) 缩放变换:$\boldsymbol{P}'=\boldsymbol{S}\cdot\boldsymbol{P}$。

对于平移变换、旋转变换和缩放变换,可以表示为普通矩阵形式:

$$\boldsymbol{P}'=\boldsymbol{M}_1\cdot\boldsymbol{P}+\boldsymbol{M}_2 \tag{5-9}$$

能否进一步将公式(5-9)表示为$\boldsymbol{P}'=\boldsymbol{M}\cdot\boldsymbol{P}$的形式呢?通过引入规范化齐次坐标可以实现上述矩阵的表示形式,即二维空间的点用齐次坐标表示:

$$\boldsymbol{P}(x,y)\rightarrow(x_h,y_h,h)\rightarrow(x,y,1)^{\mathrm{T}}$$

假设点$\boldsymbol{P}(x,y)$为xOy平面上二维图形变换前的一点坐标,变换后该点的坐标为$\boldsymbol{P}'(x',y')$。引入规范化齐次坐标后,点$\boldsymbol{P}$可以用一个矩阵来表示,这个矩阵既可以是行向量矩阵也可以是列向量矩阵,即

$$\begin{bmatrix}x & y & 1\end{bmatrix}\quad\text{或者}\quad\begin{bmatrix}x\\y\\1\end{bmatrix}$$

引入齐次坐标后,平移变换、旋转变换和缩放变换的矩阵表示形式如下所示。

(1) 平移变换。

$$\boldsymbol{P}'=\boldsymbol{T}(t_x,t_y)\cdot P,\quad\text{即}\quad\begin{bmatrix}x'\\y'\\1\end{bmatrix}=\begin{bmatrix}1&0&t_x\\0&1&t_y\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}x\\y\\1\end{bmatrix} \tag{5-10}$$

(2) 旋转变换。

$$\boldsymbol{P}'=\boldsymbol{R}(\theta)\cdot\boldsymbol{P},\quad\text{即}\quad\begin{bmatrix}x'\\y'\\1\end{bmatrix}=\begin{bmatrix}\cos\theta&-\sin\theta&0\\\sin\theta&\cos\theta&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}x\\y\\1\end{bmatrix} \tag{5-11}$$

(3) 缩放变换(变比变换)。

$$\boldsymbol{P}'=\boldsymbol{S}(s_x,s_y)\cdot\boldsymbol{P},\quad\text{即}\quad\begin{bmatrix}x'\\y'\\1\end{bmatrix}=\begin{bmatrix}s_x&0&0\\0&s_y&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}x\\y\\1\end{bmatrix} \tag{5-12}$$

这3种变换都是针对坐标原点和x或y轴方向的。需要说明的是,上面的矩阵表示采用的是列向量矩阵,也可以采用行向量矩阵,在上述矩阵的等式两边做转置即可。本书后续将不再强调图形变换采用何种矩阵,请读者自行判断。

引入齐次坐标后,二维图形变换都可以统一表示成如下形式:

$$\boldsymbol{P}'_{\text{最终坐标}}=\boldsymbol{M}_{\text{变换矩阵}}\cdot\boldsymbol{P}_{\text{原坐标}} \tag{5-13}$$

其中,$M_{\text{变换矩阵}}$可以表示成3×3的方阵,这不但可以提高矩阵运算的速度,而且便于硬件实

现。下面编程实现基本二维平移变换。

【源程序 5-1】　绘制二维直角坐标系。

```
/*画二维直角坐标系(Canvas 结构体),坐标原点在(300,300) */
void drawCCS(Canvas _canvas)
{
    //创建颜色变量_color(白色)
    RGBA _color;
    _color.m_r = 255;
    _color.m_g = 255;
    _color.m_b = 255;
    _color.m_a = 0;
    //定义坐标点数组 pt
    Point2D pt[] =
            {
                    {200, 300},
                    {500, 300},
                    {495, 295},
                    {495, 305},
                    {300, 500},
                    {300, 20},
                    {305, 25},
                    {295, 25}
            };
    //绘制坐标系的线段,使用 DDA 算法
    drawLineDDA(pt[0], pt[1], _color, _canvas);
    drawLineDDA(pt[2], pt[1], _color, _canvas);
    drawLineDDA(pt[1], pt[3], _color, _canvas);
    drawLineDDA(pt[4], pt[5], _color, _canvas);
    drawLineDDA(pt[5], pt[6], _color, _canvas);
    drawLineDDA(pt[5], pt[7], _color, _canvas);
}
```

绘制二维直角坐标系时用到了 DDA 画线算法,代码如源程序 4-2 所示。

【源程序 5-2】　绘制单个三角形。

```
void drawTriangleSingle(Point2D shape[3], Canvas _canvas)
{
    //创建颜色变量_color
    RGBA _color;
    _color.m_r = 255;
    _color.m_g = 255;
    _color.m_b = 255;
    _color.m_a = 0;
    //创建存储三角形顶点的数组 points,并复制 shape 中的 3 个坐标
    Point2D points[3] = {0};
    for(int i = 0; i < 3; i++)
    {
        points[i] = shape[i];
    }
```

```
    //将三角形的坐标系移动到画布中心并翻转
    for(int i = 0; i < 3; i++)
    {
        points[i].x = 300 + points[i].x;
        points[i].y = 300 - points[i].y;
    }
    //创建表示三角形的四边形 spt1,设 RGB 颜色为_color,用 drawLineDDA 函数绘制直线填充三角形
    Point2D spt1[] =
            {
                    {points[0].x, points[0].y},
                    {points[1].x, points[1].y},
                    {points[2].x, points[2].y},
                    {points[0].x, points[0].y},
            };
    //用 drawLineDDA 函数填充 spt1 数组的两点之间的线段
    drawLineDDA(spt1[0], spt1[1], _color, _canvas);
    drawLineDDA(spt1[1], spt1[2], _color, _canvas);
    drawLineDDA(spt1[2], spt1[0], _color, _canvas);
}
```

【源程序 5-3】 3×3 矩阵相乘函数。

```
/* 3*3 矩阵相乘(数组 1,数组 2,数组 3,参与矩阵计算的点数)*/
void multiplyMatrix2D(float a[3][3], float b[3][3], float c[3][3], int npt)
{
    int i, j, k;
    float sum;
    //循环遍历所有点
    for(i = 0; i < npt; i++)
    {
        //循环遍历第一个矩阵中的元素
        for(k = 0; k < 3; k++)
        {
            sum = 0;
            //循环遍历第二个矩阵中的元素
            for(j = 0; j < 3; j++)
            {
                //对应位置的两个元素相乘并累加到 sum 中
                sum += a[i][j] * b[j][k];
                //将计算结果存储在新矩阵 c 的对应位置中
                c[i][k] = sum;
            }
        }
    }
}
```

【源程序 5-4】 二维平移变换。

```
/* 2D 三角形的平移(三角形各顶点的坐标数组,数组长度,x 轴平移距离,y 轴平移距离,Canvas 结构体) */
void translate2D(Point2D ptArray[], int arrayLength, float xs, float ys, Canvas _canvas)
```

```
{
    //创建一个空间大小为 arrayLength 的新数组 Rshape
    Point2D * Rshape = (Point2D * ) malloc(arrayLength * sizeof(Point2D));
    //定义一个 3 * 3 的单位矩阵
    float trans[3][3] = {
            1, 0, 0,
            0, 1, 0,
            0, 0, 1
    };
    //将平移矩阵中的最后一列修改为(xs, ys, 1)
    trans[2][0] = xs;
    trans[2][1] = ys;
    //创建 1 * 3 的点向量和结果向量,初始化为零
    float pnt[1][3] = {0, 0, 1};
    float result[1][3] = {0};
    for(int i = 0; i < arrayLength; i++)
    {
        //遍历原始顶点数组,在点向量中设置当前元素坐标值
        pnt[0][0] = ptArray[i].x;
        pnt[0][1] = ptArray[i].y;
        //将点向量和转移矩阵相乘,将结果存储在结果向量中
        multiplyMatrix2D(pnt, trans, result, 1);
        //将计算出的新坐标值设置到仿射变换后的顶点数组中
        Rshape[i].x = result[0][0];
        Rshape[i].y = result[0][1];
    }
    //将 Rshape 数组中的值复制到原始二维坐标数组 ptArray 中
    for(int i = 0; i < arrayLength; i++)
    {
        ptArray[i] = Rshape[i];
    }
    //释放占用的内存空间
    free(Rshape);
}
```

二维平移变换中用到了源程序 5-3 对二维数据进行处理。

在 main.c 文件的 Render()指定位置填写源程序 5-5 进行二维平移变换，源程序 5-5 用到了源程序 5-1 绘制直角坐标系，也用到了源程序 5-2 绘制单个三角形，且用到了源程序 5-4 进行二维平移变换，运行界面如图 5-7 所示。

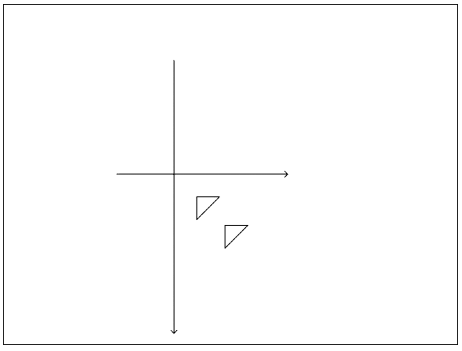


图 5-7　二维平移程序运行界面

【源程序 5-5】 通过确定 x 轴和 y 轴的平移量，实现对三角形的平移。

```
floatPoint2D ptArray2D[] =
        {
                {40, 40},
                {40, 80},
                {80, 40}
        };
drawCCS(_canvas);
drawTriangleSingle(ptArray2D, _canvas);
translate2D(ptArray2D, 3, 50, 50, _canvas);       //分别向 x 轴和 y 轴方向平移 50
drawTriangleSingle(ptArray2D, _canvas);
```

【源程序 5-6】 二维旋转变换。

```
/* 2D 三角形的旋转(三角形各顶点的坐标数组,数组长度,角度,Canvas 结构体) */
void rotate2D(Point2D ptArray[], int arrayLength, float jd, Canvas _canvas)
{
    //用动态内存分配的方式声明两个 Point2D 类型的指针数组: shape 和 Rshape,并使其大小
    //为 arrayLength
    Point2D * shape = (Point2D *) malloc(arrayLength * sizeof(Point2D));
    Point2D * Rshape = (Point2D *) malloc(arrayLength * sizeof(Point2D));
    //将 ptArray 中的点复制到 shape 数组中
    for(int i = 0; i < arrayLength; i++)
    {
        shape[i] = ptArray[i];
    }
    //将角度转换为弧度制
    jd = jd * DEC;
    //创建 3 * 3 的变换矩阵
    float rotate[3][3] =
            {
                    0, 0, 0,
                    0, 0, 0,
                    0, 0, 1
            };
    //计算旋转矩阵中的数值
    rotate[0][0] = rotate[1][1] = cos(jd);
    rotate[0][1] = sin(jd);
    rotate[1][0] = -rotate[0][1];
    //创建 1 * 3 的点向量和结果向量,初始化为零
    float pnt[1][3] = {0, 0, 1};
    float result[1][3] = {0};
    for(int i = 0; i < arrayLength; i++)
    {
        //遍历原始顶点数组,在点向量中设置当前元素坐标值
        pnt[0][0] = shape[i].x;
        pnt[0][1] = shape[i].y;
        //将点向量和转移矩阵相乘,将结果存储在结果向量中
        multiplyMatrix2D(pnt, rotate, result, 1);
```