第二章　图像的几何变换

图像几何变换是指用数学建模的方法来描述图像位置、大小、形状等变化的方法。在实际场景拍摄到的一幅图像，如果画面过大或过小，都需要进行缩小或放大。如果拍摄时景物与摄像头不成相互平行关系的时候，会发生一些几何畸变，例如会把一个正方形拍摄成一个梯形等。这就需要进行一定的畸变校正。在进行目标物的匹配时，需要对图像进行旋转、平移等处理。在进行三维景物显示时，需要进行三维到二维平面的投影建模。因此，图像几何变换是图像处理及分析的基础。

本章在讲授图像几何变换的基础上，对图像的位置变换（平移、镜像、旋转）、形状变换（比例缩放、错切）和复合变换进行阐述。

2.1 几何变换基础

2.1.1 齐次坐标

数字图像是把连续图像在坐标空间和性质空间离散化了的图像。例如，一幅二维数字图像可以用一组二维（2D）数组，$f(x,y)$来表示，其中x和y表示2D空间xy中一个坐标点的位置，$f(x,y)$代表图像在点(x,y)的某种性质的数值。如果所处理的是一幅灰度图，这时$f(x,y)$表示灰度值，此时$f(x,y)$，x，y都在整数集合中取值。因此，除了插值运算外，常见的图像几何变换可以通过与之对应的矩阵线性变换来实现。

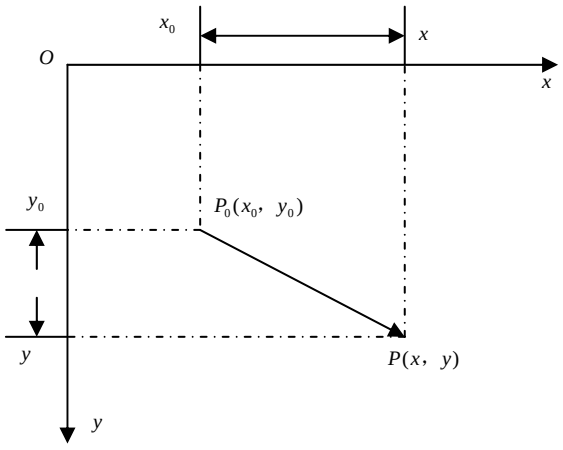


图 2.1.1 图像的平移变换示意图

现设点$\mathbf{P_0}(x_0,y_0)$进行平移后，移到$\mathbf{P}(x,y)$，其中x方向的平移量为Δx，y方向的平移量为Δy。如图 2.1.1 所示，那么，点$\mathbf{P}(x,y)$的坐标为：

$$\begin{cases} x = x_0 + \Delta x \\ y = y_0 + \Delta y \end{cases} \tag{2.1.1}$$

这个变换用矩阵的形式可以表示为：

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{2.1.2}$$

对式（2.1.2）进行简单变换可得：

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{2.1.3}$$

对式（2.1.3）进行进一步变换可得：

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{2.1.4}$$

式（2.1.4）等号右侧左面的矩阵的第 1、2 列构成单位矩阵，第 3 列元素为平移常量。该矩阵是点$\mathbf{P_0}(x_0, y_0)$平移到$\mathbf{P}(x, y)$的平移矩阵，即为变换矩阵。该变换矩阵是 2×3 阶的矩阵，为了符合矩阵相乘时要求前者列数与后者行数相等的规则，需要在点的坐标列矩阵$\begin{bmatrix} x_0 & y_0 \end{bmatrix}^{\mathrm{T}}$中引入第 3 个元素，增加一个附加坐标，扩展为 3×1 的列矩阵$\begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix}^{\mathrm{T}}$。这样式（2.1.3）同式（2.1.4）表述的意义完全相同。为了使式（2.1.4）左侧表示成矩阵$\begin{bmatrix} x & y & 1 \end{bmatrix}^{\mathrm{T}}$的形式，可用三维空间点$(x，y，1)$表示二维空间点$(x，y)$，即采用一种特殊的坐标，可以实现平移变换，变换结果如下：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{2.1.5}$$

现对式（2.1.5）中的各个矩阵进行定义：

$\mathbf{T} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix}$为变换矩阵；

$\mathbf{P} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$为变换后的坐标矩阵；

$\mathbf{P}_0 = \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$ 为变换前的坐标矩阵；

则有：

$$\mathbf{P} = \mathbf{T} \bullet \mathbf{P_0} \qquad (2.1.6)$$

从式（2.1.6）可以看出，引入附加坐标后，扩充了矩阵的第 3 行，但并没有使变换结果受到影响。这种用 $n+1$ 维向量表示 n 维向量的方法称为齐次坐标表示法。

2.1.2 齐次坐标的一般表现形式及意义

1.齐次坐标的一般表现形式

式（2.1.6）给出了图像经过平移后齐次坐标的特殊形式，齐次坐标的一般形式可表示为：

$$\mathbf{P} = \begin{bmatrix} Hx \\ Hy \\ H \end{bmatrix} \qquad (2.1.7)$$

式中，H 为非零的实数。当 $H=1$ 时，$\mathbf{P} = \begin{bmatrix} x & y & 1 \end{bmatrix}^{\mathrm{T}}$ 称规范化齐次坐标。

由点的齐次坐标 $(Hx，\ \mathrm{H}y，\ H)$，求点的规范化齐次坐标 $(x，\ y，\ 1)$。可按如下公式进行：

$$x = \frac{Hx}{H} \qquad y = \frac{Hy}{H} \qquad (2.1.8)$$

2.齐次坐标的意义

齐次坐标的几何意义相当于点 $(x，\ y)$ 落在 3D 空间 $H=1$ 的平面上，如图 2.1.2 所示。如果将 xy 平面内的三角形 abc 的各顶点表示成齐次坐标 $(x_i，\ y_i，\ 1)\ (i=1，\ 2，\ 3)$ 的形式，就变成 $H=1$ 平面内的三角形 $a_1\ b_1\ c_1$ 的各顶点。

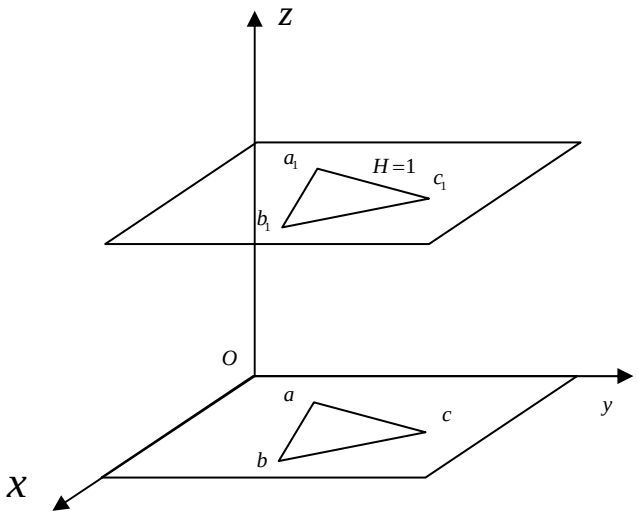


图 2.1.2 齐次坐标的几何意义

2.1.3 二维图像几何变换的矩阵

为了将式（2.1.6）写成一般形式，对包括平移在内的所有几何变换都适用，对 P，T，$\mathbf{P_0}$ 进行重新定义，齐次坐标为：

$$\mathbf{P}=\begin{bmatrix} Hx_1 & Hx_2 & \cdots & Hx_n \\ Hy_1 & Hy_2 & \cdots & Hy_n \\ H & H & \cdots & H \end{bmatrix} \tag{2.1.9}$$

当 $H=1$ 时，规范化的齐次坐标为：

$$\mathbf{P}=\begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{2.1.10}$$

变换矩阵 T 为：

$$\mathbf{T}=\begin{bmatrix} a & b & p \\ c & d & q \\ l & m & s \end{bmatrix} \tag{2.1.11}$$

$$\mathbf{P}_0=\begin{bmatrix} x_{01} & x_{02} & \cdots & x_{0n} \\ y_{01} & y_{02} & \cdots & y_{0n} \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{2.1.12}$$

则上述变换可以用公式表示为：

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \\ 1 & 1 & \cdots & 1 \end{bmatrix}=\mathbf{T}\cdot\begin{bmatrix} x_{01} & x_{02} & \cdots & x_{0n} \\ y_{01} & y_{02} & \cdots & y_{0n} \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{2.1.13}$$

引入齐次坐标后，表示 2D 图像几何变换的 3×3 矩阵的功能就完善了，可以用它完成 2D 图像的各种几何变换。下面讨论 3×3 阶变换矩阵中各元素在变换中的功能。几何变换的 3×3 矩阵的一般形式如式（2.1.11）所示。

3×3 阶矩阵 T 可以分成 4 个子矩阵：$\begin{bmatrix} a & b \\ c & d \end{bmatrix}_{2\times2}$ 子矩阵可使图像实现恒等、比例、镜像、错切和旋转变换；$\begin{bmatrix} p & q \end{bmatrix}^{\mathrm{T}}$ 列矩阵可以使图像实现平移变换；$\begin{bmatrix} l & m \end{bmatrix}$行矩阵可以使图像实现透视变换，但当$l=0$，$m=0$时它无透视作用；s 这一元素可以使图像实现全比例变换。例如将图像进行全比例变换，即：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix}\begin{bmatrix} x_{0i} \\ y_{0i} \\ 1 \end{bmatrix}=\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{2.1.14}$$

由式（2.1.14）可知，该等号在 s≠1 时，等式两端不等，若想使等式成立，将式（2.1.14）变为：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix}\begin{bmatrix} x_{0i} \\ y_{0i} \\ 1 \end{bmatrix}=\begin{bmatrix} sx_i \\ sy_i \\ s \end{bmatrix} \tag{2.1.15}$$

齐次坐标规范化后得：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix}\begin{bmatrix} \frac{x_{0i}}{s} \\ \frac{y_{0i}}{s} \\ \frac{1}{s} \end{bmatrix}=\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{2.1.16}$$

由式（2.1.16）可见，当$s>1$时，图像按比例缩小，如$s=2$，$x=\frac{x_0}{2}$，$y=\frac{y_0}{2}$，图像缩小到原来的 1 / 2；当$0<s<1$时，整个图像按比例放大，如$s=1/2$，$x=2x_0$，$y=2y_0$，图像放大到原来的 2 倍；当$s=1$时，$x=x_0$，$y=y_0$，图像大小不变。

2.2 图像的位置变换

图像的位置变换主要包括图像平移变换、图像镜像变换和图像旋转变换等。

2.2.1 图像平移变换

1 图像平移变换

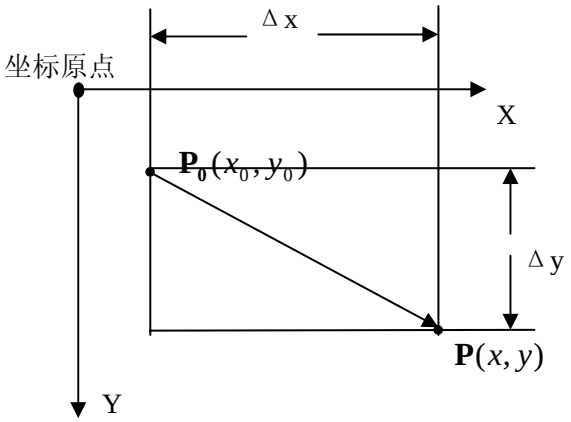


图 2.1.1 像素平移示意图

平移（Translation）变换是几何变换中最简单的一种变换，是将一幅图像上的所有点都按照给定的偏移量在水平方向沿 x 轴、在垂直方向沿 y 轴移动，如图 2.1.1 所示。设点 $\mathbf{P_0}(x_0, y_0)$ 进行平移后移到 $\mathbf{P}(x, y)$，其中 x 方向的平移量为 Δx， y 方向的平移量为 Δy。那么，点 $\mathbf{P}(x, y)$ 的坐标为：

$$\begin{cases} x = x_0 + \Delta x \\ y = y_0 + \Delta y \end{cases} \tag{2.2.1}$$

利用齐次坐标，变换前后图像上的点 $\mathbf{P_0}(x_0, y_0)$ 和 $\mathbf{P}(x, y)$ 之间的关系可以用如下的矩阵变换表示：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{2.2.2}$$

对变换矩阵求逆，可以得到式（2.2.2）的逆变换：

$$\begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta x \\ 0 & 1 & -\Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.2.3}$$

$$\begin{cases} x_0 = x - \Delta x \\ y_0 = y - \Delta y \end{cases} \tag{2.2.4}$$

2 平移变换的几点说明

①平移后图像上的每一点都可以在原图像中找到对应的点。例如，对于新图中的（0，0）像素，代入式（2.2.4）所示的方程组，可以求出对应原图中的像素 $(-\Delta x \quad -\Delta y)$。如果 Δx 或 Δy 大于 0，则点 $(-\Delta x \quad -\Delta y)$ 不在原图像中。对于不在原图像中的点，可以直接将它的

像素值统一设置为 0 或者 255（对于灰度图就是黑色或白色）。

设某一图像矩阵 $\mathbf{F}$ 如式（2.2.5）所示。图像平移后，一方面可以将对于不在原图像中点的像素值统一设置为 0，如式（2.2.6）所对应的矩阵 $\mathbf{G}$；另一方面，也可以将对于不在原图像中的点，将它的像素值统一设置为 255，如式（2.2.7）所对应的矩阵 $\mathbf{H}$。图像平移结果如图 2.2.1 所示。

$$\mathbf{F}=\begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1n-1} & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n-1} & f_{2n} \\ \vdots & \vdots & & \vdots & \vdots \\ f_{n1} & f_{n2} & \cdots & f_{nn-1} & f_{nn} \end{bmatrix} \tag{2.2.5}$$

$$\mathbf{G}=\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & f_{11} & f_{12} & \cdots & f_{1n-1} \\ 0 & f_{21} & f_{22} & \cdots & f_{2n-11} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & f_{n1} & f_{n2} & \cdots & f_{nn-1} \end{bmatrix} \tag{2.2.6}$$

$$\mathbf{H}=\begin{bmatrix} 255 & 255 & 255 & \cdots & 255 \\ 255 & f_{11} & f_{12} & \cdots & f_{1n-1} \\ 255 & f_{21} & f_{22} & \cdots & f_{2n-11} \\ \vdots & \vdots & \vdots & & \vdots \\ 255 & f_{n1} & f_{n2} & \cdots & f_{nn-1} \end{bmatrix} \tag{2.2.7}$$

②若图像平移后并没被放大，说明移出的部分被截断，原图像中有点被移出显示区域。式（2.2.6）所对应的矩阵 $\mathbf{G}$ 和式（2.2.7）所对应的矩阵 $\mathbf{H}$，是式（2.2.5）所对应的矩阵 $\mathbf{F}$ 平移的结果。由于平移后图像没有被放大，使得移出的部分丢失。图像平移结果如图 2.2.2 所示。

③若不想丢失被移出的部分图像，并将新生成的图像扩大，如图 2.2.3 所示，则：

$$\mathbf{G}=\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & f_{11} & f_{12} & \cdots & f_{1n-1} & f_{1n} \\ 0 & f_{21} & f_{22} & \cdots & f_{2n-11} & f_{2n} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & f_{n1} & f_{n2} & \cdots & f_{nn-1} & f_{nn} \end{bmatrix} \tag{2.2.8}$$

图 2.2.1 图像的平移

图 2.2.2 移动后图像大小不变

图 2.2.3 移动后图像被放大

3 程序实现

（1）具体步骤

（1） 取得原图的数据区指针。

（2） 通过对话框输入偏移量 $\Delta x, \Delta y$。

（3） 开辟一个同样大小的缓冲区。

（4） 对原图依次循环每个像素，每读入一个像素点 $\mathbf{P_0}(x_0, y_0)$，根据它的坐标找到目标图像的位置 $\begin{cases} x = x_0 + \Delta x \\ y = y_0 + \Delta y \end{cases}$，将像素 (x_0, y_0) 处的颜色值赋给新图中的 (x, y)。

（2）程序代码

```
/*************************************************************
*函数名称：PingYi()
*函数类型：void
*************************************************************/
void JiHeBianHuanDib::PingYi(int m_Xmove, int m_Ymove)
{
    // 指向源图像的指针
    LPBYTE    lpSrc;
    LPBYTE    p_data;
    // 指向要复制区域的指针
    LPBYTE lpDst;
    // 指向复制图像的指针
    LPBYTE temp;
    // 循环变量
    LONG    i;
    //图像的高和宽
    LONG wide,height;
    // 图像每行的字节数
    LONG lLineBytes;
    p_data=GetData();
    wide=GetWidth();
    height=GetHeight();
    // 计算图像每行的字节数
    lLineBytes = (((wide * 8)+31)/32*4);
    // 暂时分配内存，以保存新图像
```

```
    temp=new BYTE [lLineBytes*height];
    // 初始化新分配的内存，设定初始值为 255
    lpDst = (LPBYTE)temp;
    memset(lpDst, (BYTE)255, lLineBytes * height);
    // 每行
    for( i = 0; i < wide; i++)
    {
        // 每列
        for(int j = 0; j < height; j++)
        {
            // 计算该像素在源 DIB 中的坐标
            lpSrc = (LPBYTE)p_data + lLineBytes * (height - 1 - j) + i;
            int i0,j0;
            // 计算该像素在新 DIB 中的坐标
            i0 = i+m_Xmove;
            j0 = j + m_Ymove;
            // 判断是否在新图范围内
            if( (i0 >= 0) && (i0 < wide) && (j0 >= 0) && (j0 < height))
            {
                lpDst = (LPBYTE)temp + lLineBytes *(height - 1 - j0) + i0;
                // 复制像素
                *lpDst = *lpSrc;
            }
            else
            {
                // 对于源图中没有的像素，直接赋值为 255
                * ((unsigned char*)lpDst) = 255;
            }
        }
    }
    // 复制平移后的图像
    memcpy(p_data, temp, lLineBytes * height);
    // 释放内存
    delete temp;
}
```

2.2.2 图像镜像变换

图像的镜像变换不改变图像的形状。图像的镜像（Mirror）变换分为 3 种：水平镜像、垂直镜像和对角镜像。

1 图像的水平镜像

图像的水平镜像操作是将图像左半部分和右半部分以图像垂直中轴线为中心进行镜像对换。设图像的大小为 $M \times N$（M 行 N 列），水平镜像可按式（2.2.9）计算：

$$\begin{cases} i' = i \\ j' = N - j + 1 \end{cases} \tag{2.2.9}$$

式中：$(i，j)$ 为原图像 $\mathbf{F}(i，j)$ 中像素点的坐标；$(i'，j')$ 为对应像素点 $(i，j)$ 水平镜像变换后图像 $\mathbf{H}(i'，j')$ 中的坐标。设原图像的矩阵为：

$$\mathbf{F} = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & f_{15} \\ f_{21} & f_{22} & f_{23} & f_{24} & f_{25} \\ f_{31} & f_{32} & f_{33} & f_{34} & f_{35} \\ f_{41} & f_{42} & f_{43} & f_{44} & f_{45} \\ f_{51} & f_{52} & f_{53} & f_{54} & f_{55} \end{bmatrix} \tag{2.2.10}$$

经过水平镜像的图像，行的排列顺序保持不变，将原来的列排列. $j=1$，2，3，4，5 转换成 5，4，3，2，1 即：

$$\mathbf{F} = \begin{bmatrix} f_{15} & f_{14} & f_{13} & f_{12} & f_{11} \\ f_{25} & f_{24} & f_{23} & f_{22} & f_{21} \\ f_{35} & f_{34} & f_{33} & f_{32} & f_{31} \\ f_{45} & f_{44} & f_{43} & f_{42} & f_{41} \\ f_{55} & f_{54} & f_{53} & f_{52} & f_{51} \end{bmatrix} \tag{2.2.11}$$

其水平镜像变换的图像如图 2.2.4 所示。



(a)原图



(b)水平镜像处理效果图

图 2.2.4 图像的水平镜像

2 图像的垂直镜像

图像的垂直镜像操作是将图像上半部分和下半部分以图像水平中轴线为中心进行镜像对换。设图像的大小为 $M\times N$（M 行 N 列），垂直镜像可如下计算：

$$\begin{cases} i' = M - i = 1 \\ j' = j \end{cases} \tag{2.2.12}$$

式中：$(i，j)$ 为原图像 $\mathbf{F}(i，j)$ 中像素点的坐标；$(i'，j')$ 为对应像素点 $(i，j)$ 垂直镜像变

换后图像 $\mathbf{H}(i'，\ j')$ 中的坐标。

设原图像的矩阵如式（2.2.10）所示，经过垂直镜像的图像，列的排列顺序保持不变，将原来的行排列 $i=1$，2，3，4，5 转换成 5，4，3，2，l，即：

$$\mathbf{H}=\begin{bmatrix} f_{51} & f_{52} & f_{53} & f_{54} & f_{55} \\ f_{41} & f_{42} & f_{43} & f_{44} & f_{45} \\ f_{31} & f_{32} & f_{33} & f_{34} & f_{35} \\ f_{21} & f_{22} & f_{23} & f_{24} & f_{25} \\ f_{11} & f_{12} & f_{13} & f_{14} & f_{15} \end{bmatrix} \qquad (2.2.13)$$

其垂直镜像变换的图像如图 2.2.5 所示。



(a)原图



(b)垂直镜像处理效果图

图 2.2.5 图像的垂直镜像

附 1 实现步骤

（1） 取得原图的数据区指针。

（2） 开辟一个同样大小的缓冲区。

（3） 每个像素依次循环。在水平镜像中，将原图中的像素点的水平坐标变成镜像后的坐标（用图像的宽度减去坐标值）在显示到图像上。垂直镜像中，则对垂直坐标做相应的处理。

附 2 程序代码

```
/*************************************************************
*函数名称：JingXiang(bool fangshi)
*函数类型：void
*参数说明：fangshi --判断标志为 true 水平镜像，为 false 垂直镜像
*功能：对图像进行镜像显示。
*************************************************************/
void JiHeBianHuanDib::JingXiang(bool fangshi)
{
    // 指向源图像的指针
    LPBYTE lpSrc;
```

```
LPBYTE    p_data;
// 指向要复制区域的指针
LPBYTE lpDst;
// 指向复制图像的指针
LPBYTE temp;
// 循环变量
LONG     i;
LONG     j;
//图像的高与宽
long height=this->GetHeight();
long wide=this->GetWidth();
p_data=this->GetData();
// 暂时分配内存，以保存一行图像
temp = new BYTE[wide*height];
// 判断镜像方式
if (fangshi)
{
      // 水平镜像
      // 针对图像每行进行操作
      for(j = 0; j < height; j++)
      {
      for(i = 0; i < wide;i++)
      {
            lpSrc = (LPBYTE)p_data +wide *j + i;
            lpDst = (LPBYTE)temp + wide* (j+ 1) - i;
            *lpDst = *lpSrc;
      }
}
memcpy(p_data,temp, wide*height);
}
else
{
      // 垂直镜像
      // 针对上半图像进行操作
      for(i = 0; i <    wide;i++)
      for(j = 0; j < height; j++)
      {
            lpSrc = (LPBYTE)p_data + wide * j+i;
            lpDst = (LPBYTE)temp + wide * (height - j - 1)+i;
            *lpDst = *lpSrc;
      }
            memcpy(p_data,temp, wide*height);
}
// 释放内存
```

```
    delete temp;
}
```

3 图像的对角镜像

图像的对角镜像操作是将图像以图像水平中轴线和垂直中轴线的交点为中心进行镜像对换。相当于将图像先后进行水平镜像和垂直镜像。设图像的大小为 $M\times N$，对角镜像的计算如下：

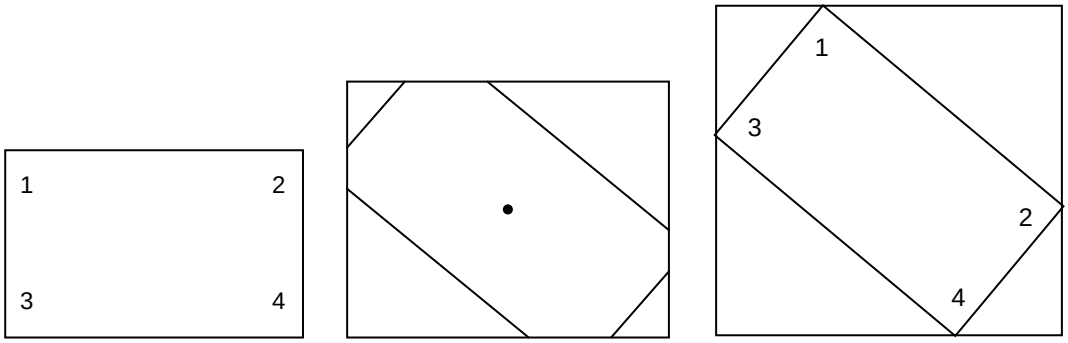
$$\begin{cases} i' = M - i + 1 \\ j' = N - j + 1 \end{cases} \tag{2.2.14}$$

式中：$(i，\ j)$ 为原图像 $\mathbf{F}(i，\ j)$ 中像素点的坐标；$(i'，\ j')$ 为对应像素点 $(i，\ j)$ 对角变换后图像 $\mathbf{H}(i'，\ j')$ 中的坐标。设原图像的矩阵如式（2.2.10）所示，经过对角镜像的图像，将原来的行排列 $i=1$，2，3，4，5 转换成 5，4，3，2，l，将原来的列排列 $j=1$，2，3，4，5 转换成 5，4，3，2，1 即：

$$\mathbf{H} = \begin{bmatrix} f_{55} & f_{54} & f_{53} & f_{52} & f_{51} \\ f_{45} & f_{44} & f_{43} & f_{42} & f_{41} \\ f_{35} & f_{34} & f_{33} & f_{32} & f_{31} \\ f_{25} & f_{24} & f_{23} & f_{22} & f_{21} \\ f_{15} & f_{14} & f_{13} & f_{12} & f_{11} \end{bmatrix} \tag{2.2.15}$$

2.2.3 图像旋转变换

旋转（Rotation）有一个绕着什么转的问题。通常的做法是，以图像的中心为圆心旋转，将图像上的所有像素都旋转一个相同的角度。图像的旋转变换是图像的位置变换，但旋转后，图像的大小一般会改变。和图像平移一样，在图像旋转变换中，可以把转出显示区域的图像截去，旋转后也可以扩大图像范围以显示所有的图像。图 2.2.9 是将图 2.2.8 旋转 30°（顺时针方向）后保持原图大小，转出的部分被裁掉的情况。图 2.2.10 是不裁掉转出部分，旋转后图像变大的情况。



2.2.8 旋转前的图　　2.2.9 保持原图大小的旋转　　2.2.10 图像变大的旋转

采用图 2.2.10 是不裁掉转出部分，旋转后图像变大的做法，首先给出变换矩阵。在我们熟悉的坐标系中，如图 2.2.11 所示，将一个点顺时针旋转 α 角，r 为该点到原点的距离，β 为 r 与 x 轴之间的夹角。在旋转过程中，r 保持不变。

设旋转前 x_0，y_0 的坐标分别为 $x_0 = r\cos\beta$；$y_0 = r\sin\beta$ 当旋转 α 角度后，坐标 x_1，y_1 的值分别为：

$$\begin{aligned} x_1 &= r\cos(\beta - a) = r\cos\beta\cos a + r\sin\beta\sin a = x_0\cos a + y_0\sin a \\ y_1 &= r\sin(\beta - a) = r\sin\beta\cos a - r\cos\beta\sin a = -x_0\sin a + y_0\cos a \end{aligned} \quad (2.2.16)$$

以矩阵的形式表示为：

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} \begin{bmatrix} \cos a & -\sin a & 0 \\ \sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.2.17)$$

式（2.2.17）中，坐标系 xy 是以图像的中心为原点，向右为 x 轴正方向，向上为 y 轴正方向。它与以图像左上角点为原点 O'，向右为 x' 轴正方向，向下为 y' 轴正方向的坐标系 $x'y'$ 之间的转换关系如图 2.2.12 所示。

设图像的宽为 w，高为 h，容易得到：

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} x' & y' & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -0.5w & 0.5h & 1 \end{bmatrix} \quad (2.2.18)$$

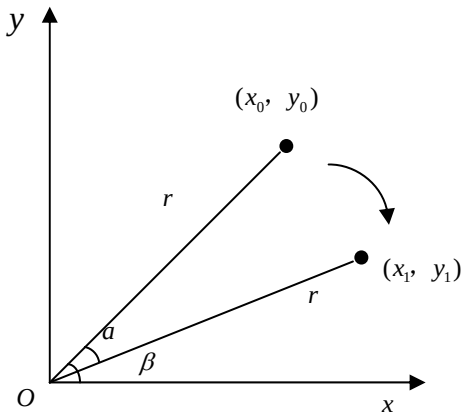


图 2.2.11 旋转示意图

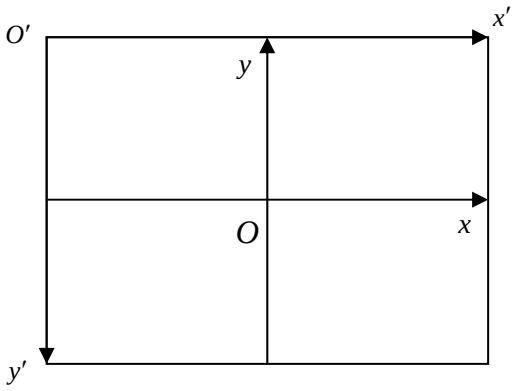


图 2.2.12 旋转中心平移示意图

逆变换为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5w & 0.5h & 1 \end{bmatrix} \tag{2.2.19}$$

有了式（2.2.17）、式（2.2.18）和式（2.2.19），可以将变换分成 3 步来完成：

①将坐标系O'变成O；

②将该点顺时针旋转α角；

③将坐标系O变回O'，这样，就得到了如下的变换矩阵（是上面三个矩阵的级联）：

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -0.5w_{old} & 0.5h_{old} & 1 \end{bmatrix} \begin{bmatrix} \cos a & -\sin a & 0 \\ \sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5w_{new} & 0.5h_{new} & 1 \end{bmatrix} \tag{2.2.20}$$

$$= \begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} \begin{bmatrix} \cos a & \sin a & 0 \\ -\sin a & \cos a & 0 \\ -0.5w_{old}\cos a + & -0.5w_{old}\sin a - & \\ 0.5h_{old}\sin a + 0.5w_{new} & 0.5h_{old}\cos a + 0.5h_{new} & 1 \end{bmatrix}$$

注意，因为新图变大，所以上面公式中出现了w_{old}，h_{old}，w_{new}，h_{new}它们分别表示原图（old）和新图（new）的宽和高。式（2.2.20）的逆变换为：

$$\begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -0.5w_{new} & 0.5h_{new} & 1 \end{bmatrix} \begin{bmatrix} \cos a & \sin a & 0 \\ -\sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5w_{old} & 0.5h_{old} & 1 \end{bmatrix} \tag{2.2.21}$$

$$= \begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} \begin{bmatrix} \cos a & -\sin a & 0 \\ \sin a & \cos a & 0 \\ -0.5w_{new}\cos a + & -0.5w_{new}\sin a - & \\ 0.5h_{new}\sin a + 0.5w_{old} & 0.5h_{new}\cos a + 0.5h_{old} & 1 \end{bmatrix}$$

这样，对于新图中的每一点，就可以根据式（2.2.20）求出对应原图中的点，并得到它的灰度。如果超出原图范围，则填成白色。要注意的是，由于有浮点运算，计算出来的点的坐标可能不是整数，需采用取整处理，即找最接近的点，这样会带来一些误差（图像可能会出现锯齿）。更精确的方法是采用插值，这将在 2.3 节中进行介绍。其逆时针旋转 45° 的图像如图 2.2.7 所示。



（a）原图



（b）旋转 45°的效果图

图 2.2.7 图像的旋转变换

程序代码

```
/*************************************************************
*函数名称：Xuanzhuan(int RotateAngle)
*函数类型：void
*参数说明：RotateAngle --旋转角度
*功能：对图像进行旋转处理。
*************************************************************/
void JiHeBianHuanDib::Xuanzhuan(double RotateAngle)
{
    DWORD           OffBits,SrcBufSize,DstBufSize,DstLineBytes;
    LPBYTE          lpTempPtr,lpPtr,lpSrc,lpTemp;
    double          SrcX1,SrcY1,SrcX2,SrcY2;
    double          SrcX3,SrcY3,SrcX4,SrcY4;
    double          DstX1,DstY1,DstX2,DstY2;
    double          DstX3,DstY3,DstX4,DstY4;
    DWORD           Wold,Hold,Wnew,Hnew;
    DWORD           x0,y0,x1,y1;
    double          cosa,sina; //cos(a),sin(a);
    double          num1,num2;
    int             LineBytes;
    //角度到弧度的转化

    RotateAngle=(double)RADIAN(RotateAngle);
    cosa=(double)cos((double)RotateAngle);
    sina=(double)sin((double)RotateAngle);
    CString s;
    s.Format("%lf",RotateAngle);
    //AfxMessageBox(s);

    //原图的宽度和高度
    lpSrc=this->GetData();
    Wold=this->GetWidth();
    Hold=this->GetHeight();
```

```
//原图的四个角的坐标

SrcX1=(double)(-0.5*Wold);
SrcY1=(double)(0.5*Hold);
SrcX2=(double)(0.5*Wold);
SrcY2=(double)(0.5*Hold);
SrcX3=(double)(-0.5*Wold);
SrcY3=(double)(-0.5*Hold);
SrcX4=(double)(0.5*Wold);
SrcY4=(double)(-0.5*Hold);

//新图四个角的坐标

DstX1=cosa*SrcX1+sina*SrcY1;
DstY1=-sina*SrcX1+cosa*SrcY1;
DstX2=cosa*SrcX2+sina*SrcY2;
DstY2=-sina*SrcX2+cosa*SrcY2;
DstX3=cosa*SrcX3+sina*SrcY3;
DstY3=-sina*SrcX3+cosa*SrcY3;
DstX4=cosa*SrcX4+sina*SrcY4;
DstY4=-sina*SrcX4+cosa*SrcY4;

//计算新图的宽度，高度
Wnew = (DWORD)(max(fabs(DstX4-DstX1), fabs(DstX3-DstX2))+0.5);
Hnew = (DWORD)(max(fabs(DstY4-DstY1), fabs(DstY3-DstY2))+0.5);

//计算矩阵(2.9)中的两个常数，这样不用以后每次都计算了
num1=(double)( -0.5*Wnew*cosa-0.5*Hnew*sina+0.5*Wold);
num2=(double)(0.5*Wnew*sina-0.5*Hnew*cosa+0.5*Hold);

//将新的缓冲区中的每个字节都填成 255，这样以后未处理的像素就是白色
DstBufSize=Wnew*Hnew;
lpTempPtr=new BYTE[Wnew*Hnew];
memset(lpTempPtr,(BYTE)255,Wnew*Hnew);
lpTemp=lpTempPtr;
for(y1=0;y1<Hnew;y1++)
{
    for(x1=0;x1<Wnew;x1++)
    {
        //x0,y0 为对应的原图上的坐标
        x0= (DWORD)(x1*cosa+y1*sina+num1);
        y0= (DWORD)(-1.0f*x1*sina+y1*cosa+num2);
        if( (x0>=0) && (x0<Wold) && (y0>=0) && (y0<Hold))     //在原图范围内
        {
```

```
                    lpPtr=lpSrc+y0*Wold+x0;
                    lpTempPtr=lpTemp+y1*Wnew+x1;
                    *lpTempPtr=*lpPtr; //进行像素的复制
                     lpTempPtr=lpTemp;
                }
            }
        }
        this->SetWidth(Wnew);
        this->SetHeight(Hnew);
        this->m_pData=lpTemp;
}
```

2.3 图像的形状变换

图像的形状变换是指用数学建模的方法对图像形状发生的变化进行描述。最基本的形状变换主要包括图像的缩放及错切等变换。下面就缩放、错切变换进行介绍。

2.3.1 图像比例缩放变换

图像比例缩放是指将给定的图像在 x 轴方向按比例缩放 f_x 倍，在 y 轴方向按比例缩放 f_y 倍，从而获得一幅新的图像。如果 $f_x = f_y$ 即在 x 轴方向和 y 轴方向缩放的比率相同，称这样的比例缩放为图像的全比例缩放。如果 $f_x \neq f_y$，图像的比例缩放会改变原始图像像素间的相对位置，产生几何畸变，如图 2.3.1 所示。设原图像中的点 $\mathbf{P_0}(x_0, y_0)$ 比例缩放后，在新图像中的对应点为 $\mathbf{P}(x, y)$，则 $\mathbf{P_0}(x_0, y_0)$ 和 $\mathbf{P}(x, y)$ 之间的对应关系如图 2.3.2 所示。



（a）原图像



（b）非全比例缩小



（c）全比例缩小

图 2.3.1 图像的缩放