视频

第3章

图像几何运算

几何运算(几何操作、几何变换)在实际中广泛应用,特别是在现代图像用户界面和视频游戏中[1]。事实上,没有不使用变焦功能以突出图像中小细节的图形应用。图 3.1 给出了平移、旋转、放缩、倾斜等几何运算的例子。

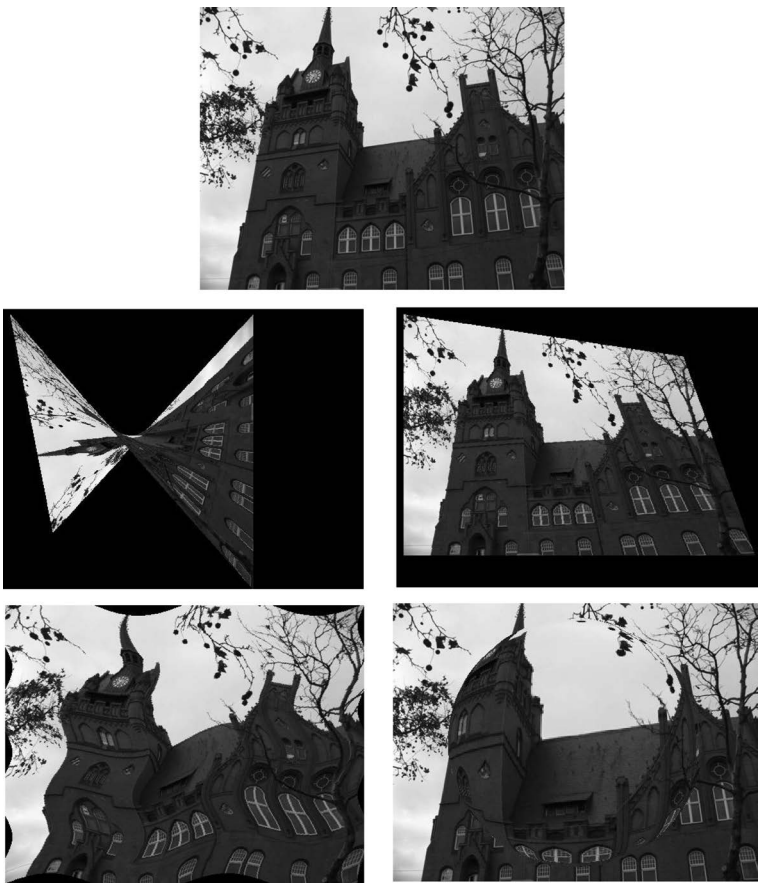


图 3.1　将在本章讨论的几何变换例子

在计算机图形领域，几何运算的话题也很重要，如纹理表达、3-D 环境或对环境的简单实时表达[2]。不过，尽管这类运算看起来普通，但为得到好的效果还是需要消耗相当的计算时间。

从根本上说，应用于原始图像的几何运算对目标图像产生下列变换：

$$I(x,y) \rightarrow I'(x',y') \tag{3.1}$$

其中，不仅像素的值会改变而且它在新图像中的位置也会改变。为此，需要先进行一个几何变换形式的坐标变换，如下：

$$T:\mathbf{R}^2 \rightarrow \mathbf{R}^2 \tag{3.2}$$

这表示必须匹配图像 $I(x,y)$ 的每个坐标 $\boldsymbol{x}=(x,y)$ 与在目标图像 $I'(x',y')$ 的新位置 $\boldsymbol{x}'=(x',y')$。即

$$\boldsymbol{x} \rightarrow \boldsymbol{x}' = T(\boldsymbol{x}) \tag{3.3}$$

如前所述，原始图像和经过变换计算出来图像的坐标都可看作实数 $\mathbf{R}\times\mathbf{R}$ 平面上的点，是连续类型的。但几何变换的主要问题是图像的坐标实际上对应 $\mathbf{Z}\times\mathbf{Z}$ 类型的离散数组，因此计算出来的从 $\boldsymbol{x}$ 到 $\boldsymbol{x}'$ 的变换并没有精确地对应这个数组，即它的坐标值对应两个像素之间的坐标值，其值是多变的。

要解决这个问题，需要通过插值获得坐标变换后的中间值，这是各种几何运算的重要部分。

3.1　坐标变换

式(3.3)是执行变换的一个基本方程，它可以分解为两个独立的方程，即

$$x' = T_x(x,y) \quad 和 \quad y' = T_y(x,y) \tag{3.4}$$

3.1.1　简单变换

简单变换包括平移、放缩、倾斜和旋转[3]。

1. 平移

也称为平动，它可以根据平移矢量 (d_x,d_y) 移动一幅图像，即

$$T_x: x' = x + d_x, \quad T_y: y' = y + d_y, \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{3.5}$$

2. 放缩

它允许扩大或收缩图像矩形数组在 $x(s_x)$ 和/或 $y(s_y)$ 方向上所占据的空间，即

$$T_x: x' = x \cdot s_x, \quad T_y: y' = y \cdot s_y, \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.6}$$

3. 倾斜

它允许倾斜图像矩形数组在 $x(b_x)$ 或 $y(b_y)$ 方向上所占据的空间(在倾斜时只考虑一个方向而在另一个方向保持不变)，即

$$T_x: x' = x + b_x \cdot y, \quad T_y: y' = y + b_y \cdot x, \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & b_x \\ b_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.7}$$

4. 旋转

它允许图像矩形以图像中心作为旋转中心进行一定角度 α 的旋转，即

$$T_x: x' = x \cdot \cos(\alpha) + y \cdot \sin(\alpha), \quad T_y: y' = -x \cdot \sin(\alpha) + y \cdot \cos(\alpha),$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.8}$$

图 3.2 给出了前面讨论的简单变换的图示。

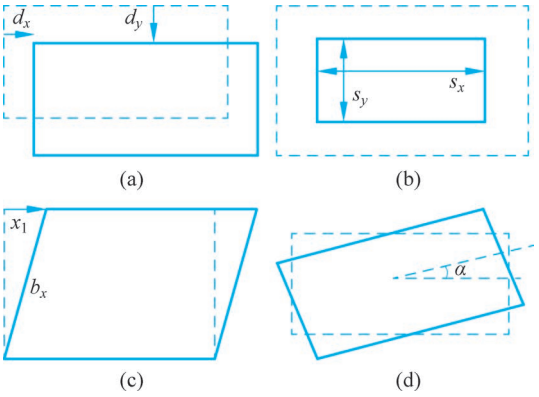


图 3.2 简单几何变换

（a）平移；（b）放缩；（c）倾斜；（d）旋转

3.1.2 齐次坐标

式(3.5)～式(3.8)定义的运算表达了一组重要的变换，称为仿射变换[4]。为将这些运算串接起来，需要用一般的矩阵形式描述它们。一个简练的方法是使用齐次坐标。

在齐次坐标中，每个矢量增加了一个分量(h)，即

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \hat{\boldsymbol{x}} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ h \end{bmatrix} = \begin{bmatrix} hx \\ hy \\ h \end{bmatrix} \tag{3.9}$$

借助这个定义，每个笛卡儿坐标 $\boldsymbol{x}=(x,y)$ 用一个称为齐次坐标的 3-D 矢量 $\hat{\boldsymbol{x}} = [\hat{x} \quad \hat{y} \quad h]^{\mathrm{T}}$ 来表达。如果分量 h 不为 0，则可获得如下实际坐标：

$$x = \frac{\hat{x}}{h} \quad \text{和} \quad y = \frac{\hat{y}}{h} \tag{3.10}$$

考虑到上述情况，将有无穷可能性（取不同的 h 值）来表达齐次坐标格式中的 2-D 点。例如，齐次坐标的矢量 $\hat{\boldsymbol{x}}_1 = [2 \quad 1 \quad 1]^{\mathrm{T}}$、$\hat{\boldsymbol{x}}_1 = [4 \quad 2 \quad 2]^{\mathrm{T}}$ 和 $\hat{\boldsymbol{x}}_1 = [20 \quad 10 \quad 10]^{\mathrm{T}}$ 表示相同的笛卡儿点(2,1)。

3.1.3 仿射变换（三角变换）

借助齐次坐标，对坐标的平移、放缩和旋转变换可以表示成如下形式：

$$\begin{bmatrix} \hat{x}' \\ \hat{y}' \\ \hat{h}' \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.11}$$

这个变换定义称为具有 6 个自由度 $a_{11},a_{12},\cdots,a_{23}$ 的仿射变换，其中，a_{13} 和 a_{23} 定义平移，a_{11}、a_{12}、a_{21} 和 a_{22} 定义放缩、倾斜和旋转。使用仿射变换，则线变换为线、三角形变换为三角形、矩形变换为平行四边形（见图 3.3）。通过这种变换，直线上的点保持了它们在距离方面的联系。

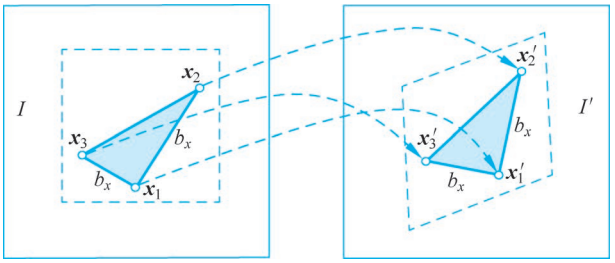


图 3.3　仿射变换（通过定义 3 个点就可以完全刻画仿射变换；通过这种变换，交换图像中直线上的点保持了它们在距离方面的联系）

1. 确定变换参数

定义在式(3.11)中的变换参数完全由 3 对点$(\boldsymbol{x}_1,\boldsymbol{x}'_1)$、$(\boldsymbol{x}_2,\boldsymbol{x}'_2)$和$(\boldsymbol{x}_3,\boldsymbol{x}'_3)$所决定，其中，$\boldsymbol{x}_i=(x_i,y_i)$对应原始图像的点，$\boldsymbol{x}'_i=(x'_i,y'_i)$对应变换图像的点。变换参数可通过解下列方程组获得：

$$\begin{aligned}
x'_1&=a_{11}\cdot x_1+a_{12}\cdot y_1+a_{13}, \quad y'_1=a_{21}\cdot x_1+a_{22}\cdot y_1+a_{23}\\
x'_2&=a_{11}\cdot x_2+a_{12}\cdot y_2+a_{13}, \quad y'_2=a_{21}\cdot x_2+a_{22}\cdot y_2+a_{23}\\
x'_3&=a_{11}\cdot x_3+a_{12}\cdot y_3+a_{13}, \quad y'_3=a_{21}\cdot x_3+a_{22}\cdot y_3+a_{23}
\end{aligned} \tag{3.12}$$

这个方程组有解的条件是 3 对点$(\boldsymbol{x}_1,\boldsymbol{x}'_1)$、$(\boldsymbol{x}_2,\boldsymbol{x}'_2)$和$(\boldsymbol{x}_3,\boldsymbol{x}'_3)$必须线性独立，这意味着它们一定不能落在同一条线上。解这个方程组，可得

$$\begin{aligned}
a_{11}&=\frac{1}{F}\cdot[y_1(x'_2-x'_3)+y_2(x'_3-x'_1)+y_3(x'_1-x'_2)]\\
a_{12}&=\frac{1}{F}\cdot[x_1(x'_3-x'_2)+x_2(x'_1-x'_3)+x_3(x'_2-x'_1)]\\
a_{21}&=\frac{1}{F}\cdot[y_1(y'_2-y'_3)+y_2(y'_3-y'_1)+y_3(y'_1-y'_2)]\\
a_{22}&=\frac{1}{F}\cdot[x_1(y'_3-y'_2)+x_2(y'_1-y'_3)+x_3(y'_2-y'_1)]\\
a_{13}&=\frac{1}{F}\cdot[x_1(y_3x'_2-y_2x'_3)+x_2(y_1x'_3-y_3x'_1)+x_3(y_2x'_1-y_1x'_2)]\\
a_{23}&=\frac{1}{F}\cdot[x_1(y_3y'_2-y_2y'_3)+x_2(y_1y'_3-y_3y'_1)+x_3(y_2y'_1-y_1y'_2)]\\
F&=x_1(y_3-y_2)+x_2(y_1-y_3)+x_3(y_2-y_1)
\end{aligned} \tag{3.13}$$

2. 仿射变换的逆

仿射变换广泛用于确定对图像几何变换之间的对应性，其逆变换 T^{-1} 可通过对式(3.11)求逆得到，即

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} & a_{12}a_{23} - a_{13}a_{22} \\ -a_{21} & a_{11} & a_{12}a_{21} - a_{11}a_{23} \\ 0 & 0 & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{3.14}$$

类似地，参数 $a_{11}, a_{12}, \cdots, a_{23}$ 可通过在原始图像和变换图像之间定义 3 对点并根据式(3.13)计算获得。图 3.4 给出一个利用仿射变换得到的几何变换图像。这里，考虑了如下各个点：$\boldsymbol{x}_1=(400,300)$、$\boldsymbol{x}_1'=(200,280)$、$\boldsymbol{x}_2=(250,20)$、$\boldsymbol{x}_2'=(255,18)$、$\boldsymbol{x}_3=(100,100)$、$\boldsymbol{x}_3'=(120,112)$。

(a) (b)

图 3.4 仿射变换

(a) 原始图像；(b) 对图像仿射变换的效果，使用了如下变换点，$x_1=(400,300)$、$x_1'=(200,280)$、$x_2=(250,20)$、$x_2'=(255,18)$、$x_3=(100,100)$、$x_3'=(120,112)$

3. MATLAB 中的仿射变换

为展示如何实现本章的几何变换，下面将介绍一个测试程序(这里是仿射变换)的代码和方法。程序基于式(3.13)以确定参数 $a_{11}, a_{12}, \cdots, a_{23}$，基于式(3.3)以确定原始图像点 $\boldsymbol{x}_i$ 对应变换图像点 $\boldsymbol{x}_i'$ 的值。实现策略从发现变换值与原始值的对应性开始。沿这个方向执行变换，各个变换图像的元素将具有与原始图像对应的值，但沿其他方向就不一样。不过，尽管采取这个策略，也可能出现由于使用变换中定义的点而使变换图像的值对应原始图像中未定义的点。这里未定义是指它们不在图像空间内，即指标是负的或值大于图像维数。如果一个点在变换图像中没有对应原始图像的对应坐标，那么将设变换图像中的像素为零。图 3.4 清晰地显示了这个变换图像和原始图像之间无对应性的问题。程序 3.1 给出了用来实现图像仿射变换所考虑的变化点 $\boldsymbol{x}_1=(400,300)$、$\boldsymbol{x}_1'=(200,280)$、$\boldsymbol{x}_2=(250,20)$、$\boldsymbol{x}_2'=(255,18)$、$\boldsymbol{x}_3=(100,100)$、$\boldsymbol{x}_3'=(120,112)$。

程序 3.1 使用 MATLAB 实现仿射变换

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program that implements the affine transform from
% a set of transformation points
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Transformation points cnl are defined
% where c indicates the x or y coordinate
% n the point number 1,2 or 3.
% 1 if it corresponds to the original or
% 1 if corresponds to the transform d
x1o = 400;
```

```
y1o = 300;
x1d = 200;
y1d = 280;
x2o = 250;
y2o = 18;
x2d = 255;
y2d = 20;
x3o = 100;
y3o = 100;
x3d = 120;
y3d = 112;

 % Determination of the parameters of Equation 3.13
F = x1o * (y3o - y2o) + x2o * (y1o - y3o) + x3o * (y2o - y1o);

a11 = (1/F) * (y1o * (x2d - x3d) + y2o * (x3d - x1d) + y3o * ...
(x1d - x2d));
a12 = (1/F) * (x1o * (x3d - x2d) + x2o * (x1d - x3d) + x3o * ...
(x2d - x1d));
a21 = (1/F) * (y1o * (y2d - y3d) + y2o * (y3d - y1d) + y3o * ...
(y1d - y2d));
a22 = (1/F) * (x1o * (y3d - y2d) + x2o * (y1d - y3d) + x3o * ...
(y2d - y1d));

a13 = (1/F) * (x1o * (y3o * x2d - y2o * x3d) + x2o * ...
(y1o * x3d - y3o * x1d)  + x3o * (y2o * x1d - y1o * x2d));
a23 = (1/F) * (x1o * (y3o * y2d - y2o * y3d)  + x2o * ...
(y1o * y3d - y3o * y1d)  + x3o * (y2o * y1d - y1o * y2d));

Den = 1/(a11 * a22 - a12 * a21);
Im = imread("fotos/paisaje.jpg");
Im = rgb2gray(Im);
imshow(Im)
figure
 % The indices of the image are obtained
[m n] = size(Im);
 % The transformed image of the same size is created
 % as the original
I1 = zeros(size(Im));
 % The values of the transformed image are calculated
 % by their correspondences with the original image
for re = 1:m
    for co = 1:n
       % Calculation of correspondences according to the
       % Ec. 3.14
         xf = round((a22 * co - a12 * re + ...
                     (a12 * a23 - a13 * a22)) * Den);
yf = round(( - a21 * co + a11 * re + ...
                     (a13 * a21 - a11 * a23)) * Den);
         % If there is no correspondence the pixel = 0
```

```
            if ((xf > n)||(xf < 1)||(yf > m)||(yf < 1))
            I1 (re,co) = 0;
            else
            I1 (re,co) = Im(yf,xf);
            end
        end
    end
    imshow(uint8(I1))
```

3.1.4 投影变换

尽管仿射变换特别适合三角变换，但有时需要考虑变形为矩形。为执行这种 4 个点的矩形几何变换，可建立 8 个自由度，比仿射变换多 2 个。这类变换称为投影变换，可定义如下：

$$\begin{bmatrix}\hat{x}'\\ \hat{y}'\\ \hat{h}'\end{bmatrix}=\begin{bmatrix}h'x'\\ h'y'\\ h'\end{bmatrix}=\begin{bmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ 1\end{bmatrix} \tag{3.15}$$

这个运算对应下面的、对图像坐标的非线性变换：

$$\begin{aligned}x'&=\frac{1}{h'}\cdot(a_{11}x+a_{12}y+a_{13})=\frac{a_{11}x+a_{12}y+a_{13}}{a_{31}x+a_{32}y+1}\\ y'&=\frac{1}{h'}\cdot(a_{21}x+a_{22}y+a_{23})=\frac{a_{21}x+a_{22}y+a_{13}}{a_{31}x+a_{32}y+1}\end{aligned} \tag{3.16}$$

尽管是非线性变换，直线仍保持直线，不过有一个投影效果。这个变换一般将平行线变换为非平行线，正方形变换为多边形，阶为 π 的代数曲线变换为阶为 π 的代数曲线。例如，圆或椭圆通过这个变换变成一条二阶曲线。与仿射变换相反，平行线在结果图像中不再是平行线；直线上点之间的关系或距离不再保持。图 3.5 是投影变换的图示。

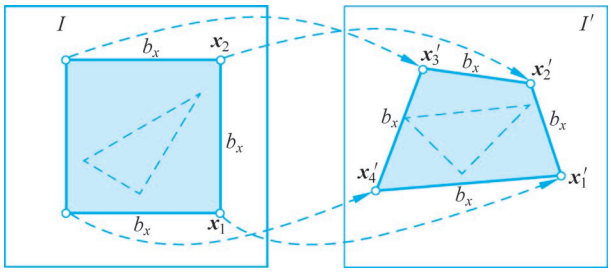


图 3.5 投影变换（在这个变换中，直线变换为直线，正方形变换为多边形，平行线变换为非平行线，直线上点之间的距离关系丢失了）

1. 确定变换参数

投影变换的参数完全由 4 对坐标点 $(\boldsymbol{x}_1,\boldsymbol{x}'_1)$、$(\boldsymbol{x}_2,\boldsymbol{x}'_2)$、$(\boldsymbol{x}_3,\boldsymbol{x}'_3)$ 和 $(\boldsymbol{x}_4,\boldsymbol{x}'_4)$ 所决定，其中，$\boldsymbol{x}_i=(x_i,y_i)$ 对应原始图像的点，$\boldsymbol{x}'_i=(x'_i,y'_i)$ 对应变换图像的点。这 8 个投影参数可通过解下列方程组得到：

$$\begin{aligned}x'_i&=a_{11}x_i+a_{12}y_i+a_{13}-a_{31}x_ix'_i-a_{32}y_ix'_i\\ y'_i&=a_{21}x_1+a_{22}y_1+a_{23}-a_{31}x_iy'_i-a_{32}y_iy'_i\end{aligned} \tag{3.17}$$

其中，$i=1,2,3,4$。将式(3.17)定义的方程组写成针对变换各个参数的矩阵，可以得到：

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix} \tag{3.18}$$

将其写成更紧凑的形式：

$$\boldsymbol{x}' = \boldsymbol{M} \cdot \boldsymbol{a} \tag{3.19}$$

参数 $\boldsymbol{a}=(a_{11},a_{12},\cdots,a_{32})$ 的值可通过用标准的数值方法（如高斯算法）解式(3.18)的方程组得到。

2. 投影变换的逆

如 $\boldsymbol{x}'=\boldsymbol{A}\cdot\boldsymbol{x}$ 形式的线性变换一般可以间接地通过对矩阵 $\boldsymbol{A}$ 求逆来换一种形式求解，即 $\boldsymbol{x}=\boldsymbol{A}^{-1}\cdot\boldsymbol{x}'$。一个基本的要求是 $\boldsymbol{A}$ 为非奇异($\mathrm{Det}(\boldsymbol{A})\neq 0$)。一个 3×3 矩阵可按相对简单的方法计算逆，利用如下关系：

$$\boldsymbol{A}^{-1} = \frac{1}{\mathrm{Det}(\boldsymbol{A})}\boldsymbol{A}_{\mathrm{adj}} \tag{3.20}$$

其中，

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$\mathrm{Det}(\boldsymbol{A}) = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} - a_{13}a_{22}a_{31} \tag{3.21}$$

$$\boldsymbol{A}_{\mathrm{adj}} = \begin{bmatrix} a_{22}a_{33}-a_{23}a_{32} & a_{13}a_{32}-a_{12}a_{33} & a_{12}a_{23}-a_{13}a_{22} \\ a_{23}a_{31}-a_{21}a_{33} & a_{11}a_{33}-a_{13}a_{31} & a_{13}a_{21}-a_{11}a_{23} \\ a_{21}a_{32}-a_{22}a_{31} & a_{12}a_{31}-a_{11}a_{32} & a_{11}a_{22}-a_{12}a_{21} \end{bmatrix}$$

在投影变换中，参数 $a_{33}=1$ 可简化前面方程的计算。因为在齐次坐标中，与标量相乘构成等价点（见 3.1.2 小节），并不需要确定 $\boldsymbol{A}$ 的行列式。因此，只需要计算投影变换的逆、点的齐次坐标以及邻接矩阵 $\boldsymbol{A}_{\mathrm{adj}}$。图 3.6 给出了一幅使用投影变换而发生几何变换的图像。

(a)　(b)

图 3.6　投影变换

（a）原始图像；(b) 对图像投影变换的效果

3. 对单位正方形的投影变换

另一种替代数值方法以求解式(3.18)中8个未知参数的方法是对单位正方体 C_1 进行变换。如图3.7所示，在这个变换中，一个单位正方体 C_1 转换为一个4个点的具有失真特性的多边形 P_1。这个转换考虑了如下点的变换：

$$\begin{cases}(0,0)\rightarrow \boldsymbol{x}'_1, & (1,1)\rightarrow \boldsymbol{x}'_3\\ (1,0)\rightarrow \boldsymbol{x}'_2, & (0,1)\rightarrow \boldsymbol{x}'_4\end{cases} \tag{3.22}$$

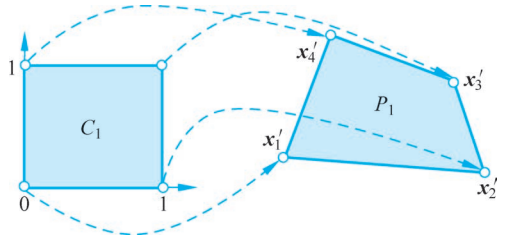


图 3.7　将单位正方形 C_1 投影变换为多边形 P_1

根据变换点之间的联系，式(3.18)定义的方程组简化为

$$\begin{aligned}
x'_1&=a_{13}\\
y'_1&=a_{23}\\
x'_2&=a_{11}+a_{13}-a_{31}\cdot x'_2\\
y'_2&=a_{21}+a_{23}-a_{31}\cdot y'_2\\
x'_3&=a_{11}+a_{12}+a_{13}-a_{31}\cdot x'_3-a_{32}\cdot x'_3\\
y'_3&=a_{21}+a_{22}+a_{23}-a_{31}\cdot y'_3-a_{32}\cdot y'_3\\
x'_4&=a_{12}+a_{13}-a_{32}\cdot x'_4\\
y'_4&=a_{22}+a_{23}-a_{33}\cdot y'_4
\end{aligned} \tag{3.23}$$

通过下列关系可得到对参数 $a_{11},a_{12},\cdots,a_{32}$ 的解：

$$\begin{aligned}
a_{31}&=\frac{(x'_1-x'_2+x'_3-x'_4)\cdot(y'_4-y'_3)-(y'_1-y'_2+y'_3-y'_4)\cdot(x'_4-x'_3)}{(x'_2-x'_3)\cdot(y'_4-y'_3)-(x'_4-x'_3)\cdot(y'_2-y'_3)}\\
a_{32}&=\frac{(y'_1-y'_2+y'_3-y'_4)\cdot(x'_2-x'_3)-(x'_1-x'_2+x'_3-x'_4)\cdot(y'_2-y'_3)}{(x'_2-x'_3)\cdot(y'_4-y'_3)-(x'_4-x'_3)\cdot(y'_2-y'_3)}\\
a_{11}&=x'_2-x'_1+a_{31}\cdot x'_2\\
a_{21}&=y'_2-y'_1+a_{31}\cdot y'_2\\
a_{12}&=x'_4-x'_1+a_{32}\cdot x'_4\\
a_{22}&=y'_4-y'_1+a_{32}\cdot y'_4\\
a_{13}&=x'_1\\
a_{23}&=y'_1
\end{aligned} \tag{3.24}$$

如本节已提到的，通过逆变换矩阵 $\boldsymbol{A}^{-1}$，可以计算逆变换 $\boldsymbol{T}^{-1}$，这里就是从任何4个点的多边形得到单位正方体。

如图3.8所示，从任何4点多边形到另一个4点多边形的变换可以通过在单位框架上执行一个包括两个步骤的方法来实现。这可表示成：

$$P_1 \xrightarrow{\boldsymbol{T}_1^{-1}} C_1 \xrightarrow{\boldsymbol{T}_2} P_2 \tag{3.25}$$

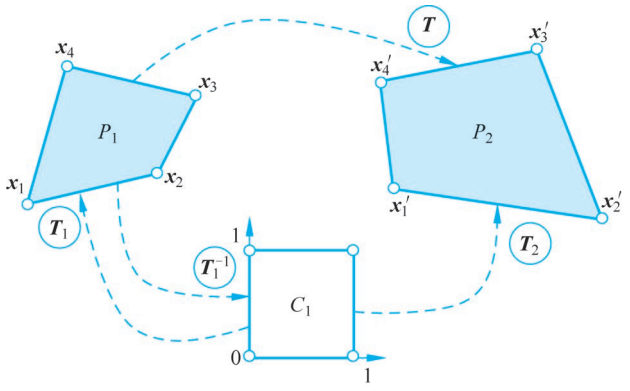


图 3.8　两个 4 点多边形之间的投影变换(这类变换使用包含两个步骤的方法进行；第 1 个包含从多边形 P_1 到单位正方体 C_1 的逆变换 T_1^{-1}；第 2 个包含从单位正方体 C_1 到多边形 P_2 的正变换 T_2，则整个变换可写为 $T=T_2 \cdot T_1^{-1}$)

用于从正方形到各个 4 点多边形的变换 $\boldsymbol{T}_1$ 和 $\boldsymbol{T}_2$ 可从定义 $\boldsymbol{x}_i$ 和 $\boldsymbol{x}_i'$ 的式(3.24)得到，而逆变换 $\boldsymbol{T}_1^{-1}$ 可对矩阵 $\boldsymbol{T}_1$ 计算得到。完整的变换 $\boldsymbol{T}$ 最终是通过如下变换 $\boldsymbol{T}_1^{-1}$ 和 $\boldsymbol{T}_2$ 的结合得到：

$$\boldsymbol{x}' = \boldsymbol{T}(\boldsymbol{x}) = \boldsymbol{T}_2[\boldsymbol{T}_1^{-1}(\boldsymbol{x})] \tag{3.26}$$

或写成矩阵形式：

$$\boldsymbol{x}' = \boldsymbol{T} \cdot \boldsymbol{x} = \boldsymbol{T}_2 \cdot \boldsymbol{T}_1^{-1} \cdot \boldsymbol{x} \tag{3.27}$$

对使用由多边形 P_1 和 P_2 确定的变换，变换矩阵 $\boldsymbol{T}=\boldsymbol{T}_2 \cdot \boldsymbol{T}_1^{-1}$ 只计算一次。

下面给出对两个多边形投影变换的计算示例。

设要对两个多边形 P_1 和 P_2 进行几何变换，其中定义多边形的坐标为

$$\begin{aligned} P_1&: \boldsymbol{x}_1=(2,5), \boldsymbol{x}_2=(4,6), \boldsymbol{x}_3=(7,9), \boldsymbol{x}_4=(5,9) \\ P_2&: \boldsymbol{x}_1'=(4,3), \boldsymbol{x}_2'=(5,2), \boldsymbol{x}_3'=(9,3), \boldsymbol{x}_4'=(7,5) \end{aligned} \tag{3.28}$$

那么，相对于单位正方形的投影变换矩阵为 $\boldsymbol{T}_1$：$C_1 \rightarrow P_1$ 和 $\boldsymbol{T}_2$：$C_1 \rightarrow P_2$：

$$\boldsymbol{T}_1 = \begin{bmatrix} 3.33 & 0.50 & 2 \\ 3.00 & -0.50 & 5 \\ 0.33 & -0.5 & 1 \end{bmatrix}, \quad \boldsymbol{T}_2 = \begin{bmatrix} 1 & -0.50 & 4 \\ -1 & -0.50 & 3 \\ 0 & -0.5 & 1 \end{bmatrix} \tag{3.29}$$

通过结合变换矩阵 $\boldsymbol{T}_2$ 和 $\boldsymbol{T}_1^{-1}$，可得到总的变换矩阵，即 $\boldsymbol{T}=\boldsymbol{T}_2 \cdot \boldsymbol{T}_1^{-1}$，其中，

$$\boldsymbol{T}_1^{-1} = \begin{bmatrix} 0.6 & -0.45 & 1.05 \\ -0.40 & 0.8 & -3.2 \\ -0.4 & 0.55 & -0.95 \end{bmatrix}, \quad \boldsymbol{T} = \begin{bmatrix} -0.8 & 1.35 & -1.12 \\ 1.6 & 1.7 & -2.3 \\ -0.2 & 0.15 & 0.65 \end{bmatrix} \tag{3.30}$$

3.1.5　双线性变换

双线性变换定义如下：

$$\begin{aligned} T_x&: x' = a_1 x + a_2 y + a_3 xy + a_4 \\ T_y&: y' = b_1 x + b_2 y + b_3 xy + b_4 \end{aligned} \tag{3.31}$$

从这个定义可知，完全刻画变换需 8 个参数($a_1,a_2,a_3,a_4,b_1,b_1,b_2,b_3,b_4$)，为计算它们至少需要 4 对点。根据式(3.31)，有些项包含两个变量 xy，因此这种变换是对结果图像进行一个非线性几何变换[5]。在这类变换中(与投影变换不同)直线转换成二阶曲线。

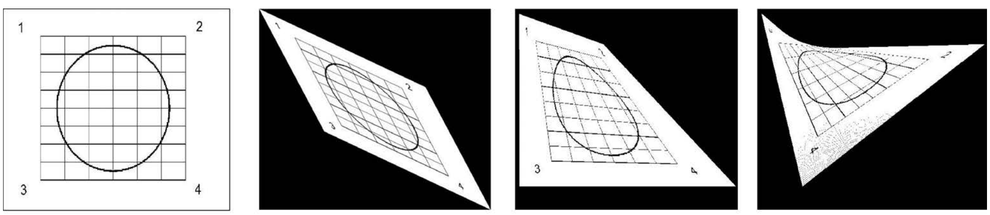
一个双线性变换完全由 4 对点($\boldsymbol{x}_1,\boldsymbol{x}'_1$)，($\boldsymbol{x}_2,\boldsymbol{x}'_2$)，($\boldsymbol{x}_3,\boldsymbol{x}'_3$)，($\boldsymbol{x}_4,\boldsymbol{x}'_4$)所确定。通常情况下，可以定义 4 对点，组成下列方程组来构建变换：

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \\ x'_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & x_1y_1 & 1 \\ x_2 & y_2 & x_2y_2 & 1 \\ x_3 & y_3 & x_3y_3 & 1 \\ x_4 & y_4 & x_4y_4 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$
$$\begin{bmatrix} y'_1 \\ y'_2 \\ y'_3 \\ y'_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & x_1y_1 & 1 \\ x_2 & y_2 & x_2y_2 & 1 \\ x_3 & y_3 & x_3y_3 & 1 \\ x_4 & y_4 & x_4y_4 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \qquad (3.32)$$

对从单位正方形 C_1 到任何多边形 P_1 的双线性变换这种特殊情况，参数($a_1,\cdots,a_4,b_1,\cdots,b_4$)的计算可以简化为

$$\begin{aligned} &a_1 = x'_2 - x'_1, \quad a_2 = x'_4 - x'_1, \quad a_3 = x'_1 - x'_2 + x'_3 - x'_4, \quad a_4 = x'_2 \\ &b_1 = y'_2 - y'_1, \quad b_2 = y'_4 - y'_1, \quad b_3 = y'_1 - y'_2 + y'_3 - y'_4, \quad b_4 = y'_1 \end{aligned} \qquad (3.33)$$

图 3.9 给出一组代表对相同图像执行不同几何变换得到的不同效果。



(a) (b) (c) (d)

图 3.9 几何变换的比较

(a) 原始图像；(b) 仿射变换；(c) 投影变换；(d) 双线性变换

以下介绍在 MATLAB 中如何实现双线性变换。与用于仿射变换的程序 3.1 不同，在双线性变换的情况下，采取发现一个逆变换并直接计算变换图像像素值的策略不易实现。尽管这种计算图像像素值的方式能给出较好的结果，但有时它不允许发现原始图像中经非线性变换没有对应像素的像素值。这个效果在图像中有孔(黑点)处很明显，如图 3.9(d)所示。程序 3.2 给出了实现双线性变换的代码。在该程序中，考虑了从单位正方形到 4 点多边形的变换以简化计算，因此变换点定义如下：

$$0,0) \rightarrow \boldsymbol{x}'_1, (1,1) \rightarrow \boldsymbol{x}'_3, \quad (1,0) \rightarrow \boldsymbol{x}'_2, (0,1) \rightarrow \boldsymbol{x}'_4 \qquad (3.34)$$

程序 3.2 使用 MATLAB 实现双线性变换

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program that implemente the bilinear transformation
% from a set of transformation points
% considering the unit square
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Definition of transformation points
% The unit square method is considered to be used.
x1 = 100;
y1 = 80;
x2 = 500;
y2 = 70;
x3 = 100;
y3 = 350;
x4 = 10;
y4 = 10;
% Determination of parameters a and b.
a1 = x2 - x1;
a2 = x4 - x1;
a 3 = x1 - x2 + x3 - x4;
a4 = x1;
b1 = y2 - y1;
b2 = y4 - y1;
b3 = y1 - y2 + y3 - y4;
b4 = y1;
% Get the size of the image
Im = imread("fotos/paisaje.jpg");
Im = rgb2gray(Im);
[m n] = size (Im);
% Each of the points of the result image is set to black,
% this is because there will be values of the result image
%  that do not have a corresponding value in the original image.
I1 = zeros(size(Im));
% All pixels of the transformed image are traversed
for re = 1:m
    for co = 1 : n
% The coordinates of the original image
% correspond to points on the unit square so
% they are divided between m and n
        re1 = re/m;
        co1 = co/n;
% Get the values of the transformed image
        x = round(a1 * co1 + a2 * re1 + a3 * re1 * co1 + a4);
        y = round(b1 * co1 + b2 * re1 + b3 * re1 * co1 + b4);
% It is protected for pixel values that due to
% the transformation do not have a corresponding
        if ((x >= 1)||(x <= n)||(y >= 1)||(y <= m))
            I1(y,x) = Im(re,co);
        end
    end
end
% Convert the image to a data type
I1 = uint8(I1);
% The image is displayed
imshow(I1)
```

3.1.6　其他非线性几何变换

双线性变换只是非线性变换(不能用简单的矩阵相乘表达)的一个例子。但是,还有许多允许增加图像有用效果和失真的非线性变换。下面 3 个例子扩展了变换的反向公式:

$$\boldsymbol{x} = \boldsymbol{T}^{-1}(\boldsymbol{x}') \tag{3.35}$$

依赖于所考虑的变换种类，反向公式并不简单，尽管实际应用中有很多场合（使用原始变换技术）反转并不是必需的。

1. 扭转变换

扭转变换产生图像绕点 $\boldsymbol{x}_c=(x_c,y_c)$ 的 α 扭转，该扭转随着图像像素与扭转点的距离增加而减少。几何变换在图像上的效果仅限于一定的失真半径 $r_{\max}$，在此之外图像保持不变[6]。变换的反向公式定义如下：

$$
\begin{aligned}
T_x^{-1}:\ x&=\begin{cases} x_c+r\cdot\cos(\beta), & r\leqslant r_{\max}\\ x', & r>r_{\max}\end{cases}\\
T_y^{-1}:\ y&=\begin{cases} y_c+r\cdot\sin(\beta), & r\leqslant r_{\max}\\ y', & r>r_{\max}\end{cases}
\end{aligned}
\tag{3.36}
$$

其中，

$$
\begin{aligned}
&d_x=x'-x_c,\quad d_y=y'-y_c,\quad r=\sqrt{d_x^2+d_y^2}\\
&\beta=\alpha\tan^2(d_y,d_x)+\alpha\cdot\left(\frac{r_{\max}-r}{r_{\max}}\right)
\end{aligned}
\tag{3.37}
$$

图 3.10(a)和(b)给出了扭转变换的两个示例，其中图像的中心为扭转点 $\boldsymbol{x}_c$，$r_{\max}$ 为图像对角线的一半，扭转角度 $\alpha=28°$。

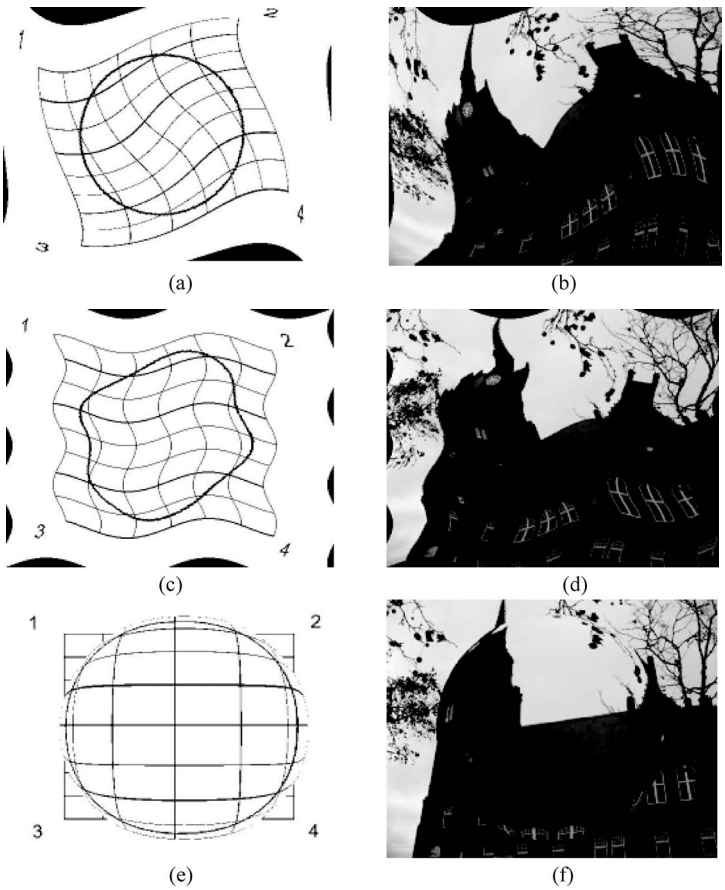


图 3.10　不同的非线性变换

（a）和（b）扭转变换；（c）和（d）波纹变换；（e）和（f）球形失真

2. MATLAB 中的扭转变换

以下介绍在 MATLAB 中如何实现扭转变换。实现的策略是使用对变换的反向公式，即它开始先发现变换值与原始值之间的对应性。这很方便，因为根据式(3.36)变换自身是按反向公式表达的。实现代码见程序 3.3，其中以图像中心为扭转点 x_c，r_{max} 为图像对角线的一半，扭转角度 $\alpha=28°$。

程序 3.3　使用 MATLAB 实现扭转变换

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program that implements the Twirl transformation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Im = imread("fotos\paisaje.jpg")
Im = rgb2gray(Im);
imshow (Im)
figure
% Get the size of the image
[m n] = size(Im);
% The center of rotation is defined
% as the center of the image
xc = n/2;
yc = m/2;
% The angle of rotation is defined approx. 28 degrees
% 1 rad
alfa = 1;
% rmax is defined
rmax = sqrt (xc * xc + yc * yc);
% Convert the image to double to avoid numerical problems
Imd = double(Im);
% The resulting image is filled with zeros in such a way
that
%  where there are no geometric correspondences,
%  the value of the pixels will be zero.
I1 = zeros(size(Im));
% All pixels of the transformed image are traversed
for re = 1:m
    for co = 1:n
% The values defined in 3.34 are obtained
        dx = co - xc;
        dy = re - yc;
        r = sqrt (dx * dx + dy * dy);
% The transformations of
% Equations 3.33 - 3.343 are calculated
        if (r <= rmax)
            Beta = atan2 (dy,dx) + alfa * ...
                    ((rmax - r)/rmax);
            xf = round (xc + r * cos(Beta));
            yf = round (yc + r * sin(Beta));
        else
            xf = co;
            yf = re;
        end
% It is protected for pixel values that due to
% the transformation do not have a corresponding
```

```
            if ((xf >= 1)&&(xf <= n)&&(yf >= 1)&&(yf <= m))
                I1(re,co) = Imd(yf,xf);
            end
        end
    end
        I1 = uint8 (I1);
        imshow(I1)
```

3. 波纹变换

波纹变换产生一个在 x 或 y 方向的局部波形失真。该变换的参数为两个方向上的周期 τ_x 和 τ_y，以及两个方向上的平移强度 a_x 和 a_y[6]。该变换的反向公式定义如下：

$$\begin{aligned} T_x^{-1}: x &= x' + a_x \cdot \sin\left(\frac{2\pi y'}{\tau_x}\right) \\ T_y^{-1}: y &= y' + a_y \cdot \sin\left(\frac{2\pi x'}{\tau_y}\right) \end{aligned} \tag{3.38}$$

图 3.10(c)和(d)给出了波纹变换的两个示例，其中 $\tau_x=120$，$\tau_y=250$，$a_x=10$，$a_y=12$。

4. MATLAB 中的波纹变换

以下介绍在 MATLAB 中如何实现波纹变换。实现的策略是使用对变换的反向公式，即它开始先发现变换值与原始值之间的对应性。这很方便，因为变换自身根据式(3.38)是按反向公式表达的。实现代码见程序 3.4，其中，$\tau_x=120$，$\tau_y=250$，$a_x=10$，$a_y=12$。

程序 3.4　使用 MATLAB 实现波纹变换

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program that implements the Ripple transformation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Im = imread("fotos\paisaje.jpg")
Im = rgb2gray(Im);
imshow(Im)
figure
% Get the size of the image
[m n] = size(Im);
% The periods of the wavelength are defined
tx = 120;
ty = 250;
% Displacements of each of the directions
ax = 10;
ay = 12;
% Convert the image to double to avoid
% numerical problems
Imd = double(Im);
% The resulting image is filled with zeros in such a way
%  that where there are no geometric correspondences,
%  the value of the pixels will be zero.
I1 = zeros(size(Im));
% All pixels of the transformed image are traversed
for re = 1:m
    for co = 1:n
% The transformations of Eq. 3.36 are calculated
            Angulo1 = sin((2 * pi * re)/tx);
```

```
                Angulo2 = sin((2 * pi * co)/ty);
                xf = round(co + ax * Angulo1);
                yf = round(re + ay * Angulo2);
            % It is protected for pixel values that due to the
            % transformation do not have a corresponding
            if ((xf >= 1)&&(xf <= n)&&(yf >= 1)&&(yf <= m))
                I1(re,co) = Imd(yf,xf);
            end
        end
    end
        I1 = uint8(I1);
        imshow(I1);
```

5. 球形失真

球形失真产生的效果与球面透镜类似。为执行这个变换的参数是镜头中心 $\boldsymbol{x}_c=(x_c, y_c)$、最大失真半径 $r_{\max}$ 和镜头折射率 ρ[6]。该变换的反向公式定义如下：

$$\begin{aligned} T_x^{-1}: x &= x' - \begin{cases} z \cdot \tan(\beta_x), & r \leqslant r_{\max} \\ 0, & r > r_{\max} \end{cases} \\ T_y^{-1}: y &= y' - \begin{cases} z \cdot \tan(\beta_y), & r \leqslant r_{\max} \\ 0, & r > r_{\max} \end{cases} \end{aligned} \tag{3.39}$$

其中，

$$\begin{aligned} & d_x = x' - x, \quad d_y = y' - y, \quad r = \sqrt{d_x^2 + d_y^2}, \quad z = \sqrt{r_{\max}^2 - r^2} \\ & \beta_x = \left(1 - \frac{1}{\rho}\right) \arcsin\left(\frac{d_x}{\sqrt{d_x^2 + z^2}}\right) \\ & \beta_y = \left(1 - \frac{1}{\rho}\right) \arcsin\left(\frac{d_y}{\sqrt{d_y^2 + z^2}}\right) \end{aligned} \tag{3.40}$$

图 3.10(e)和(f)给出球形失真的两个示例，其中以图像中心为旋转点 $\boldsymbol{x}_c$，$r_{\max}$ 为图像对角线的一半，镜头折射率 $\rho=1.8$。

6. MATLAB 中的球形失真

以下介绍在 MATLAB 中如何实现球形失真。实现的策略是使用对变换的反向公式，即它开始先发现变换值与原始值之间的对应性。这很方便，因为变换自身根据式(3.39)是按反向公式表达的。所实现的代码见程序 3.5，其中以图像的中心为旋转点 $\boldsymbol{x}_c$，$r_{\max}$ 为图像对角线的一半，镜头折射率 $\rho=1.8$。

程序 3.5　使用 MATLAB 实现球形失真

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Program that implements spherical distortion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Im = imread("fotos\paisaje.jpg")
Im = rgb2gray(Im);
imshow(Im)
figure
% Get the size of the image
[m n] = size(Im);
% The center of the spherical lens is defined as
```

```
% the center of the image
xc = n/2;
yc = m/2;
% Define the lens index
ro = 1.8;
% rmax is defined
rmax = sqrt (xc * xc + yc * yc);
% Convert the image to double to avoid numerical problems
Imd = double(Im);
% The resulting image is filled with zeros
% in such a way that where there are
% no geometric correspondences,
%  the value of the pixels will be zero.
I1 = zeros(size(Im));
% All pixels of the transformed image are traversed
for re = 1:m
    for co = 1:n
% The values defined in 3.37 are obtained
        dx = co - xc;
        dy = re - yc;
        r = sqrt (dx * dx + dy * dy);
        if (r <= rmax)
% The transformations of Eq. 3.37 are calculated
            z = sqrt (rmax * rmax - r * r);
            R1 = dx/(sqrt(dx * dx + z * z));
            R2 = dy/(sqrt(dy * dy + z * z));
            Bx = (1 - (1/ro)) * asin(R1);
            By = (1 - (1/ro)) * asin(R2);
            xf = round(co - z * tan(Bx));
            yf = round(re - z * tan(By));
        else
            xf = co;
            yf = re;
        end
% It is protected for pixel values that due
% to the transformation do not have a corresponding
        If((xf >= 1)&&(xf <= n)&&(yf >= 1)&&(yf <= m))
            I1(re,co) = Imd(yf,xf);
        end
    end
end
      I1 = uint8(I1);
      imshow(I1)
```

3.2 坐标重赋值

到目前为止，在实现几何运算时都假设图像的坐标是连续的，即实数值。但一幅图像的元素值是离散的，仅使用整数值。考虑到这一点，几何变换中一个并不简单的问题是发现原始图像和变换图像之间的坐标对应性（不会由于重新分配或四舍五入而产生信息损失）。

考虑到一个几何变换 $T(x,y)$ 作用于源图像 $I(x,y)$ 而生成目标图像 $I'(x',y')$，所有坐标都是离散的，$x,y\in\mathbf{Z}$ 和 $x',y'\in\mathbf{Z}$，可以采取两个不同的方法，它们的差别在于执行变换的方向不同。这些方法分别称为源-目标映射和目标-源映射。

3.2.1　源-目标映射

在第 1 种方法中，计算原始(源)图像 $I(x,y)$ 中的各个像素在变换(目标)图像 $I'(x',y')$ 中的对应位置。计算出来的坐标 (x',y') 一般不对应整数或离散值(见图 3.11)，因此需要决定原始图像 $I(x,y)$ 的哪个整数值将对应变换图像 $I'(x',y')$ 的哪个整数值。求解这个问题并不简单，需要一个允许从中间值(插值)确定正确值的方法。

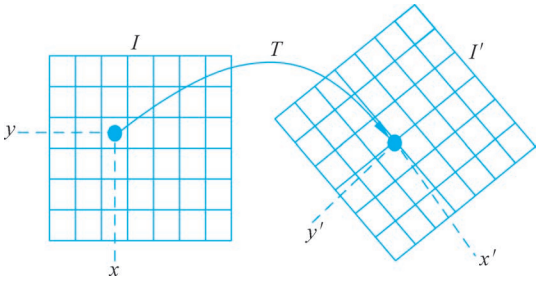


图 3.11　源-目标映射(对源图像 $I(x,y)$ 的各个离散位置，使用变换算子 $T(x,y)$ 计算它在变换图像 $I'(x',y')$ 中的对应位置)

该方法的主要问题是，根据几何变换 $T(x,y)$，变换图像 $I'(x',y')$ 中会有一些元素在原始图像 $I(x,y)$ 中没有对应元素。例如，在放大原始图像的情况下，变换图像的强度函数将有需要填充的间隔。另外，在缩小原始图像的情况下，一些原始图像的像素将不再存在于变换图像中，很明显信息丢失了。

3.2.2　目标-源映射

这个方法与源-目标映射相反，计算变换图像 $I'(x',y')$ 中的各个像素在原始图像 $I(x,y)$ 中的对应位置。为此，需要用它的反向公式来表达变化，即

$$(x,y)=T^{-1}(x',y') \tag{3.41}$$

在这个方法中，与源-目标映射类似，由于逆几何变换 $T^{-1}(x,y)$，将存在原始图像 $I(x,y)$ 中的元素在变换图像 $I'(x',y')$ 中没有对应元素。解决这个问题需要一个允许从中间值(插值)确定正确值的方法。图 3.12 给出了使用目标-源映射方法的图示。

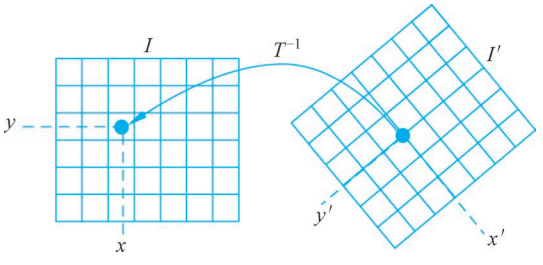


图 3.12　目标-源映射(对目标图像 $I'(x',y')$ 的各个离散位置，使用逆变换算子 $T^{-1}(x,y)$ 计算它在源图像 $I(x,y)$ 中的对应位置)

目标-源映射的优点是对新图像的所有像素和它们在源图像的对应像素的计算是有保证的。据此，在新图像中由于缺少对应性而产生间隙的可能性被消除了。该方法的一个缺点是需要变换的反向公式，这可能在很多情况下不易得到。尽管如此，该方法的特性和计算

优点使其成为最广泛用于几何变换的方法。算法 3.1 给出了使用该方法计算一般几何变换的过程。

算法 3.1　几何变换目标-源映射

作为输入，我们有原始图像和根据其逆变换公式定义的几何变换。
1. 几何变换目标-源映射(I，(x，y)，T)
2. $I(x, y)$→源图像
3. T→几何坐标变换
4. 生成零值的目标图像
5. **for** 所有图像坐标(x', y') **do**

$$(x, y) \leftarrow T^{-1}(x', y')$$

$I(x, y)$←插值(I，x，y)
return (I'，x'，y')

3.3　插值

插值指在没有严格联系的函数值和所表达的点之间计算函数值的方法[7]。在对图像进行几何变换时，需要在变换 $T(x, y)$中使用插值处理。通过变换 $T(x, y)$(或 $T^{-1}(x, y)$)计算变换图像的值，有可能没有与原始图像严格的对应性，因此就需要插值方法。

3.3.1　简单插值方法

为简便说明插值方法，先对 1-D 情况进行分析。考虑有一个离散信号 $g(u)$，如图 3.13(a)所示。为在离散函数的任意中间值位置 $x \in \mathbf{R}$ 进行插值，有许多不同的方法。最简单的近似是将 u 的最近邻 u_0 的值以离散方式赋给 x，即

$$\hat{g}(x) = g(u_0) \tag{3.42}$$

其中，

$$u_0 = \text{Round}(x) = (x + 0.5) \tag{3.43}$$

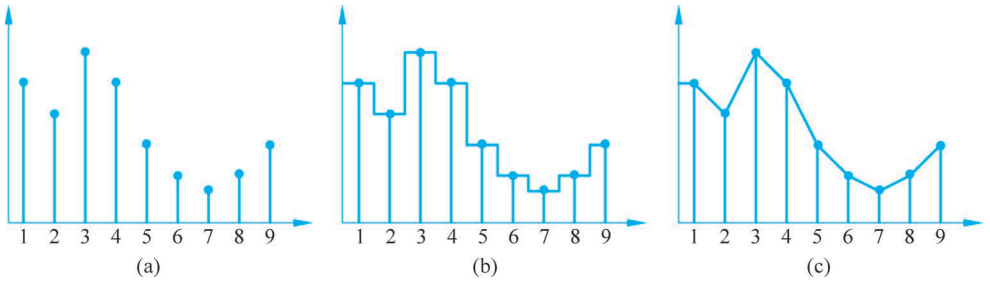


图 3.13　离散信号的插值

(a) 离散信号；(b) 最近邻插值；(c) 线性插值

这种称为最近邻插值的方法如图 3.13(b)所示，是对图 3.13(a)的信号进行插值的结果。这个方法可用于前面讨论的所有几何变换算法中。

另一种简单的插值方法是线性插值，其中计算出来的变量中间值用点间的实线表示，图 3.13(c)给出了对图 3.13(a)的信号进行线性插值的结果。

3.3.2　理想插值

很明显，前述各种插值方法没有给出函数的很好近似，因此需要建立更好的描述离散信号中间值的方法。

本小节的目标是分析与前述简单插值方法相对的方法，即提升最好可能的建模方式以对离散函数进行插值。

一个离散函数具有有限的带宽，其最大值由获得信号的采样频率的一半 $\omega_s/2$ 来刻画。如果用一个宽度为 $\omega_s/2$ 或 $\pm\pi$ 的矩形函数 $H(\omega)$ 乘以频谱（是周期性的），则可以取到频谱 $F(\omega)$ 的一个周期。图 3.14 给出了获得频谱过程的图示。

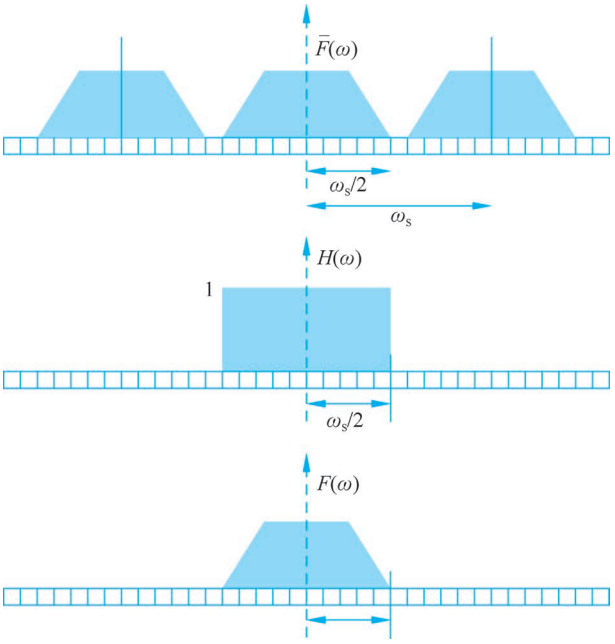


图 3.14　通过乘以矩形函数来分离一个频谱

频域的相乘对应空域的卷积。因此，在频域与一个矩形函数相乘意味着对矩形函数变换的卷积，这里变换就是如下定义的 $\mathrm{sinc}(x)$：

$$\mathrm{sinc}(x) = \left[\frac{\sin(\pi x)}{\pi x}\right] \tag{3.44}$$

理论上，$\mathrm{sinc}(x)$ 函数是重建连续函数的理想插值函数。为重建函数 $g(u)$ 在任意位置 x_0 的值，先将 $\mathrm{sinc}(x)$ 函数的原点移动到 x_0，然后进行 $g(u)$ 和 $\mathrm{sinc}(x)$ 的点对点卷积。考虑到这些操作步骤，插值函数可表示如下：

$$\hat{g}(x_0) = \sum_{u=-\infty}^{\infty} \mathrm{sinc}(x_0 - u) \cdot g(u) \tag{3.45}$$

3.3.3　立方插值

由于 $\mathrm{sinc}(x)$ 函数具有无穷尺寸的插值核，因此该方法在实际中无法实现。因为这个原因，为产生理想插值 $\mathrm{sinc}(x)$ 函数的效果，使用接近其效果的紧凑核。一个广泛使用的实

现紧凑核的方法是立方插值，这里使用了三次多项式进行近似[8]，即

$$w_{\text{cubic}}(x,a)=\begin{cases}(a+2)\cdot|x|^3-(a+3)\cdot|x|^2+1, & 0\leqslant|x|<1\\ a\cdot|x|^3-5a\cdot|x|^2+8a\cdot|x|-4a, & 1\leqslant|x|<2\\ 0, & |x|\geqslant 2\end{cases}\tag{3.46}$$

其中，a 是定义立方衰减的控制参数。图 3.15 给出了使用不同参数 a 的立方插值的不同核。对标准值 $a=-1$，式(3.46)可简化为

$$w_{\text{cubic}}(x,a)=\begin{cases}|x|^3-2\cdot|x|^2+1, & 0\leqslant|x|<1\\ -|x|^3+5\cdot|x|^2-8\cdot|x|+4, & 1\leqslant|x|<2\\ 0, & |x|\geqslant 2\end{cases}\tag{3.47}$$

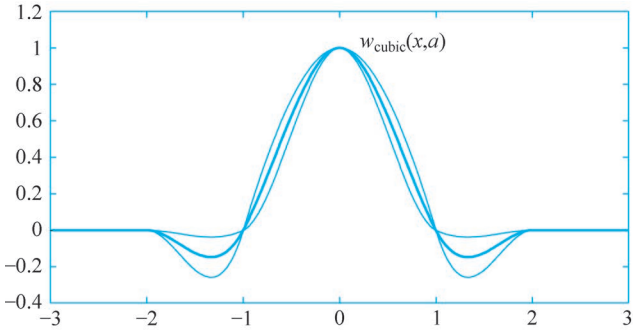


图 3.15　用于插值的不同立方核（该图考虑了控制参数 a 的 3 个不同值，$a=-0.25$、$a=-1$ 和 $a=-1.75$）

图 3.16 给出了函数 sinc(x)和立方核 $w_{\text{cubic}}(x)$的比较，从中可看出两者的差别在$|x|\leqslant 1$时可以忽略，当$|x|>1$时变得明显。当$|x|>2$时情况最严重，此时立方核只有 0 值。

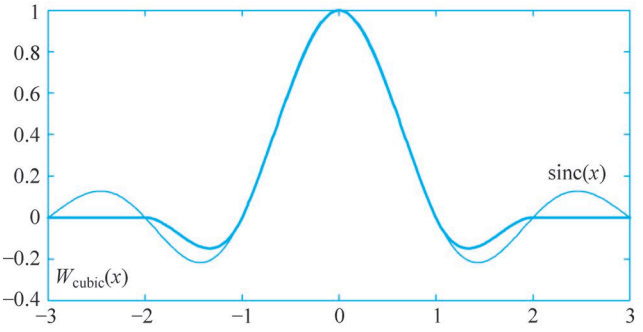


图 3.16　函数 sinc(x)和 $w_{\text{cubic}}(x)$的比较

函数 $g(u)$的立方插值在给定点 x_0 可用下式计算：

$$\hat{g}(x_0)=\sum_{x=x_0-1}^{x_0+2}w_{\text{cubic}}(x_0-u)\cdot g(u)\tag{3.48}$$

图 3.17 给出了 2-D 函数 sinc(x,y)。

与 1-D 情况相同，理想插值所需的核在实际中不能计算，因此需要考虑其他的方案。其他的方案包括在 1-D 情况时讨论的插值方法，还包括下列方法：最近邻插值、双线性插值和双立方插值[8]。

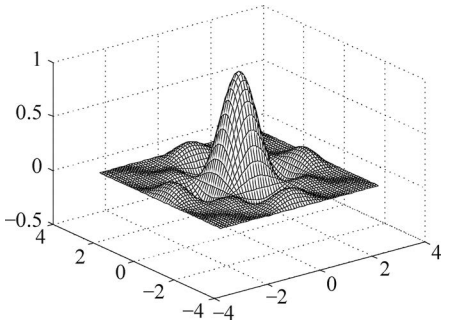


图 3.17　2-D 函数 sinc(x,y)的表达

1. 最近邻插值

为对给定点(x_0,y_0)插值,将最接近它的像素坐标(u_0,v_0)赋给它。在很多情况下,为执行这个方法,只需对几何变换得到的坐标四舍五入:

$$I(x_0,y_0)=I(u_0,v_0) \tag{3.49}$$

其中,

$$u_0=\text{Round}(x_0),\quad v_0=\text{Round}(y_0) \tag{3.50}$$

2. 双线性插值

线性插值代表 1-D 情况,而双线性插值代表 2-D 或图像情况。可使用图 3.18 来说明这种类型插值的操作过程。其中,(x_0,y_0)的邻域像素定义如下:

$$\begin{aligned} A&=I(u_0,v_0), \qquad B=I(u_0+1,v_0)\\ C&=I(u_0,v_0+1), \quad D=I(u_0+1,v_0+1) \end{aligned} \tag{3.51}$$

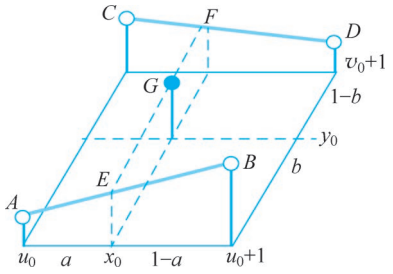


图 3.18　双线性插值。对在位置(x_0,y_0)的点 G 的插值通过考虑邻域像素 A、B、C 和 D 用两个步骤得到,先通过线性插值,考虑被插值点和相邻像素 $a=(x_0-u_0)$之间的距离得到点 E 和 F;再进行垂直插值,利用距离 $b=(y_0-v_0)$得到 E 和 F 之间的点

为得到(x_0,y_0)的值,需要进行两个不同的线性插值,这两个插值是在点 E 和 F 上根据点与最近邻之间的距离 $a=(x_0-u_0)$建立的。这些值根据线性插值获得:

$$\begin{aligned} E&=A+(x_0-u_0)\cdot(B-A)=A+a\cdot(B-A)\\ F&=C+(x_0-u_0)\cdot(D-C)=C+a\cdot(D-C) \end{aligned} \tag{3.52}$$

根据前述值,对点 G 的插值可通过考虑垂直距离 $b=(y_0-v_0)$得到:

$$\begin{aligned} \hat{I}(x_0,y_0)=G&=E+(y_0-v_0)\cdot(F-E)=E+b\cdot(F-E)\\ &=(a-1)(b-1)A+a(b-1)B+(1-a)bC+abD \end{aligned} \tag{3.53}$$

如同构建线性插值核,2-D 双线性插值核 $W_{\text{bilinear}}(x,y)$也可以定义。它是两个 1-D 核

$w_{\text{bilinear}}(x)$和$w_{\text{bilinear}}(y)$的乘积。执行这个乘法就得到：

$$W_{\text{bilinear}}(x,y)=w_{\text{bilinear}}(x)\cdot w_{\text{bilinear}}(y)$$

$$W_{\text{bilinear}}(x,y)=\begin{cases}1-x-y-xy, & 0\leqslant|x|,|y|<1\\ 0, & \text{其他}\end{cases} \tag{3.54}$$

3. 双立方插值

如前所述，双立方插值核$W_{\text{bicubic}}(x,y)$可借助1-D核$w_{\text{bicubic}}(x)$和$w_{\text{bicubic}}(y)$的相乘获得，即

$$W_{\text{bicubic}}(x,y)=w_{\text{bicubic}}(x)\cdot w_{\text{bicubic}}(y) \tag{3.55}$$

式(3.47)定义的1-D立方核可用于式(3.55)的相乘。式(3.55)的双立方核显示在图3.19(a)中，而图3.19(b)给出了双立方核与sinc(x,y)函数的算术差。

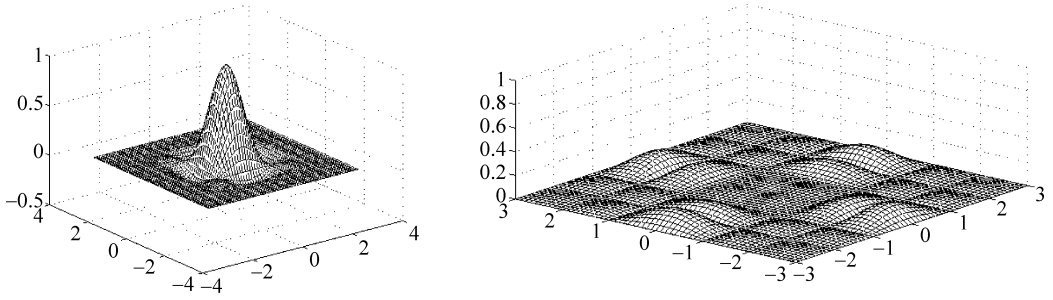


图 3.19

(a) 立方插值的核；(b) 双立方核与sinc(x)函数的差$|\text{sinc}(x,y)-W_{\text{bicubic}}(x,y)|$

3.4 混叠

如本章所讨论的，一个几何变换本质上包括3个步骤：

(1) 对变换图像$I'(x',y')$中的每个坐标，均借助逆变换$T^{-1}(x,y)$获得其在原始图像$I(x,y)$中的对应坐标。

(2) 从变换图像$I'(x',y')$出发，使用前面介绍的核进行插值，以重建图像。

(3) 当几何变换要求生成或消除像素时，就需要有一个准则以处理这些像素来消除伪影。

当一幅图像要减小尺寸时，很明显一些原始像素将由于几何变换而丢失。这个问题会导致混叠。混叠效果可看作一个由于图像缩小而导致的问题，它的后果就是对变换图像添加了原始图像中没有的伪影。

有一些方法可以减少混叠效果。根据速度效率比，较好的方法之一就是在图像处理过程中先加一个低通滤波器。滤波器的效果将减弱伪影问题而不会导致过多的计算负荷。这对执行实时处理的人工视觉系统尤为重要。

3.5 MATLAB中的几何变换函数

本节描述包含在MATLAB图像处理工具箱中实现几何变换的函数。一个几何变换通过将输入图像中像素坐标映射为结果图像中新像素坐标的算子来调整图像中像素之间的

关系。

为改变图像的尺寸，MATLAB 提供了函数 imresize。该函数的通用语法是

```
B = imresize(A,[mre ncol],method);
```

这个函数从原始图像 A 计算一幅新图像 B，其尺寸由矢量[mre ncol]指定。所用插值方法由变量 method 配置，参见表 3.1。

表 3.1　大多数实现几何变换的 MATLAB 函数所用的插值方法

语　法	方　法
'nearest'	使用最近邻插值(这是默认选项)。见 3.3.3 小节
'bilinear'	使用双线性插值。见 3.3.3 小节
'bicubic'	使用双立方插值。见 3.3.3 小节

为旋转图像，使用函数 imrotate。该函数的通用语法是

```
B = imrotate(A,angle,method);
```

这个函数将图像 A 的中心作为旋转轴对其进行旋转。旋转角度由变量 angle 定义。所用插值方法由变量 method 配置，参见表 3.1。

MATLAB 允许实现空间变换，如仿射变换或投影变换。它们的效果总比 imresize 和 imrotate 更有趣和精细。考虑该因素，以及在 MATLAB 中执行这类变换较为复杂，为此开发了下列流程：

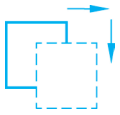
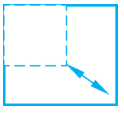
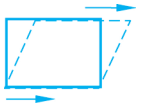
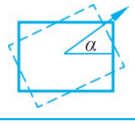
(1) 定义几何变换的参数，这应该包括一系列由算子 $T(x,y)$ 执行映射而定义的参数。

(2) 构建一个称为 TFORM 的变换结构，可看作对所定义参数矩阵形式的结合。

(3) 使用函数 imtrasform 执行空间操作。

作为第 1 步，必须要定义需执行的变换。在很多类型的几何变换中，可用一个尺寸为 3×3 的变换矩阵来刻画参数。表 3.2 给出了一些简单变换及对它们建模的变换矩阵之间的联系。

表 3.2　几何变换及对它们建模的变换矩阵

变换	示　例	变 换 矩 阵	
平移		$\begin{bmatrix}1 & 0 & t_x\\ 0 & 1 & t_y\\ 0 & 0 & 1\end{bmatrix}$	t_x 指定沿 x 轴的平移量 t_y 指定沿 y 轴的平移量
放缩		$\begin{bmatrix}s_x & 0 & 0\\ 0 & s_y & 0\\ 0 & 0 & 1\end{bmatrix}$	s_x 指定沿 x 轴的放缩量 s_y 指定沿 y 轴的放缩量
倾斜		$\begin{bmatrix}1 & i_x & 0\\ i_y & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$	i_x 指定 x 轴方向的斜率 i_y 指定 y 轴方向的斜率
旋转		$\begin{bmatrix}\cos(\alpha) & \sin(\alpha) & 0\\ -\sin(\alpha) & \cos(\alpha) & 0\\ 0 & 0 & 1\end{bmatrix}$	α 指定将图像中心作为旋转轴而旋转的角度

为构建包含变换重要特性的 TFORM 结构，使用函数 maketform。该函数的通用语法是

```
TFORM = maketform(type,MT);
```

这个函数根据 MT 变换矩阵和要执行的几何变换类别的定义构建 TFORM 结构。几何变换的类别可以是'affine'或'projective'。

'affine'变换包括旋转、平移、放缩和倾斜。如在本章中看到的，在这类几何变换中直线保持为直线，并具有平行线保持不变的特殊性质。考虑到这个性质，矩形变成平行四边形。

在'projective'变换中，直线也保持为直线，区别是如果它们平行，则在结果图像中它们不能保持不变。该变换之所以得名，正是因为直线被变换后产生了一种透视效果，即直线被投影到无穷远的点上。

最后，使用函数 imtransform 执行几何运算。该函数的通用语法是

```
B = imtransform(A,TFORM,method);
```

该函数将通过图像 A 上结构 TFORM 建模的几何变换返回到 B。这个插值方法用于配置变量 method，参见表 3.1。

作为示例，将对一幅图像进行沿 x 轴的双重倾斜几何变换。首先，根据表 3.2 构建变换矩阵如下：

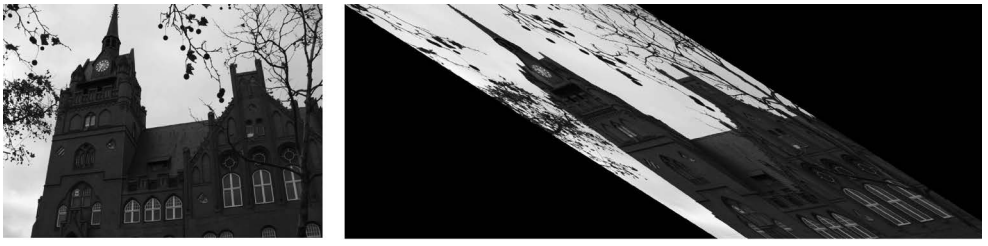
$$\mathbf{MT}=\begin{bmatrix}1 & 0 & 0\\ 2 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \tag{3.56}$$

接下来，生成结构 TFORM，它可以用如下命令行构建：

```
MT = [1 0 0; 2 1 0; 0 0 1];
TFORM = maketform('affine',MT);
```

设将图 3.20(a)的图像存在图像 A 中，使用函数 imtransform 执行几何运算并写出命令行：

```
B = imtransform(A,TFORM,'bilinear');
imshow(B)
```



(a)　　(b)

图 3.20　倾斜几何运算的结果

（a）原始图像；（b）得到的结果

如在图 3.20 中看到的，原始图像有较低的维数（600×800），而结果图像有较高的维数（600×1600）。为使结果图像保留原始图像的尺寸，可以使用函数 imresize，即

```
C = imresize(B,[600 800]);
imshow(C);
```

参考文献

[1] González-Campos J S, Arnedo-Moreno J, Sánchez-Navarro J. GTCards: A video game for learning geometric transformations: A cards-based video game for learning geometric transformations in higher education. In *Ninth international conference on technological ecosystems for enhancing multiculturality (TEEM' 21)*, 2021: 205-209.

[2] Freeman W T, Anderson D B, Beardsley P, et al. Computer vision for interactive computer graphics. *IEEE Computer Graphics and Applications*, 1998, 18(3): 42-53.

[3] Solomon C, Breckon T. *Fundamentals of digital image processing: A practical approach with examples in MATLAB*. Wiley, 2010.

[4] Bebis G, Georgiopoulos M, da Vitoria Lobo N, et al. Learning affine transformations. *Pattern Recognition*, 1999, 32(10): 1783-1799.

[5] Jain A K. *Fundamentals of digital image processing*. Prentice-Hall, Inc, 1989.

[6] Petrou M M, Petrou C. *Image processing: The fundamentals*. John Wiley & Sons, 2010.

[7] Annadurai S. *Fundamentals of digital image processing*. Pearson Education India, 2007.

[8] Han D. Comparison of commonly used image interpolation methods. In *Conference of the 2nd international conference on computer science and electronics engineering (ICCSEE 2013)*. Atlantis Press, 2013: 1556-1559.