第3章

数字图像的基本运算

数字图像处理从本质上讲，就是通过对图像施加某种运算，来满足人或机器视觉的应用需求。数字图像的基本运算通常是指对图像的代数运算和几何运算。从运算原理及运算的简单性来说，图像的灰度反转、对数变换和直方图（统计）显然应该属于图像的基本代数运算。

本章首先介绍图像的灰度反转和对数变换；然后介绍图像直方图及其有关运算方法；接着介绍图像的代数运算方法；最后介绍图像的几何运算方法，包括图像的平移变换、图像的旋转变换、图像镜像、图像的转置、图像的缩小与放大等。

3.1 灰度反转

黑白图像的灰度反转（grayscale inversion）就是使灰度值为 1 的像素值变成 0，使灰度值为 0 的像素值变成 1。

对于 256 灰度级图像来说，图像的灰度反转值就是用 255 分别减去原图像 $f(x,y)$ 的各个像素的灰度值。一般地，设图像的灰度级为 L，则图像的灰度反转可表示为

$$g(x,y)=L-1-f(x,y) \tag{3.1}$$

256 灰度级图像的灰度反转原理如图 3.1（a）所示，灰度图像反转示例如图 3.1（b）和图 3.1（c）所示。

(a) 灰度图像反转的关系曲线　(b) 原图像　(c) 灰度反转图像（负片）

图 3.1　灰度图像反转原理及图例

【例 3.1】 灰度反转 MATLAB 程序，程序运行结果如图 3.1 所示。

```
clc; clear all; close all;
img0 = imread('d:\0_matlab 图像课编程\lena.bmp');

result_img = 256 - 1 - img0;                          % 图像阵列中的所有像素的灰度值反转

subplot(1,2,1); imshow(img0); title('原图像');     % 显示原图像
subplot(1,2,2); imshow(result_img); title('灰度反转图像(负片)');
```

3.2 对数变换

对原图像 $f(x,y)$进行对数变换(logarithmic transformation)的解析式可表示为

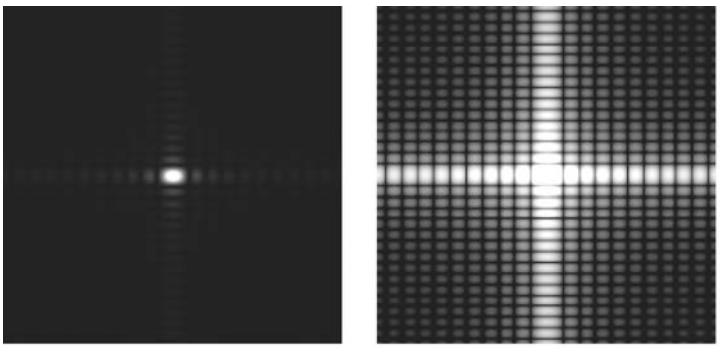
$$g(x,y)=c\cdot\log(1+f(x,y)) \tag{3.2}$$

其中，c 是一个常数。

对数变换的作用是对原图像的灰度值动态范围进行压缩，主要用于调高输入图像的低灰度值。比如对于傅里叶频谱来说，要显示的值的范围往往比较大，而傅里叶频谱要显示的重点是突出最亮的像素(对应于低频成分)，在 8 比特的显示系统中会损失掉频谱图像中的低灰度值部分(对应于高频成分)。在这种情况下，可以依据式(3.2)对频谱进行变换(这时 $c=1$)，调高频谱图像的低灰度值而对高灰度值又尽可能地影响最小。傅里叶频谱一般都是通过这种方式的调整来进行显示。

【例 3.2】 对数图像进行对数变换运算的 MATLAB 程序，程序运行结果如图 3.2 所示。

```
clc; clear all; close all;
img0 = imread('d:\0_matlab 图像课编程\fft_img.jpg')        % 读取灰度图像

img1 = double(img0);
log_img = log(1 + img1);                                   % 对图像进行对数运算

subplot(1,2,1); imshow(img0); title('原傅里叶频谱图像');
subplot(1,2,2); imshow(log_img); title('提升低频成分后的傅里叶频谱图像');
```



(a) 原傅里叶频谱图像　　(b) 提升低频成分后的傅里叶频谱图像

图 3.2　利用对数变换对傅里叶频谱图像进行调整示例

例 3.2 中的相关函数及功能说明如下。

log(x)：MATLAB 中提供的求自然对数函数，对 x 求以 e 为底的对数。当 x 为向量时，对 x 的元素逐个进行以 e 为底的对数运算。

3.3 灰度直方图

在数字图像处理中，灰度图像的直方图(简称灰度直方图，gray histogram)是一种描述一幅灰度图像中灰度级内容的最简单且最有用的工具，也是对灰度图像进行多种处理的基础。

3.3.1　灰度图像直方图与灰度图像的对比度

1. 灰度图像直方图

灰度图像的直方图是关于灰度级分布的函数，描述的是灰度图像中的各灰度级及其出现的频数（该灰度级的像素数目）的统计关系，是一种表示数字图像中灰度级分布函数关系的柱状图表示形式。灰度直方图的横坐标表示灰度级（即亮度级别），取值范围为 0～255；灰度直方图的纵坐标表示图像中各灰度级的像素数目（即统计的各灰度级像素的个数）。灰度图像直方图定义为

$$\begin{cases} h(r_k)=n_k \quad k=0,1,2,\cdots,L-1 \\ H(P)=[h(r_1),h(r_2),\cdots,h(r_{L-1})] \end{cases} \tag{3.3}$$

其中，r_k 表示第 k 级灰度；n_k 表示图像中灰度级为 r_k 的像素的个数；$H(P)$ 是灰度图像的直方图函数。在有些文献中所提及的一维直方图即是本节所讲的灰度图像的直方图。

理解和观看直方图的规则一是“左黑右白”或“左暗右亮”；二是横轴上各（亮度值）点对应的柱状高度就是分布在该亮度的像素个数；三是当柱状接近分布在整个横轴上，且至少有一个峰值时，图像的对比度较好。

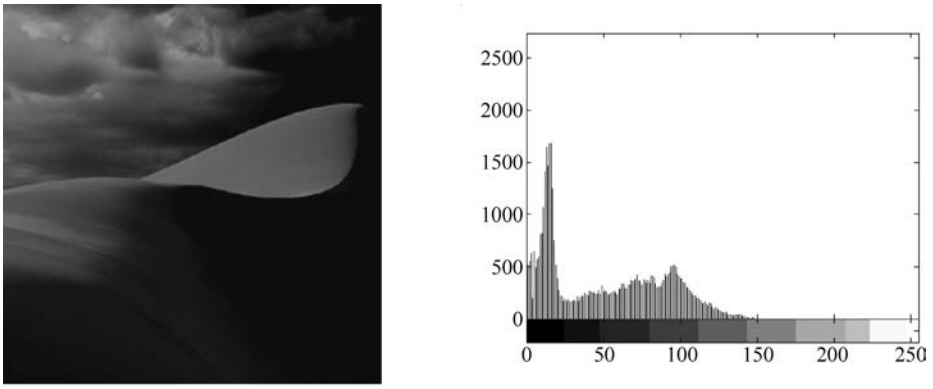
2. 灰度图像的对比度

灰度图像的对比度是对一幅图像中最亮的白和最暗的黑之间亮度层级的测量，一般用图像画面中最大的灰度值（最白亮度）与最小的灰度值（最黑亮度）的比值来表示。因此，白色越亮、黑色越暗，对比度就越高（大）；对比度越大，图像越清晰醒目；对比度越小，图像越不清晰。

对比度直观地反映了图像中目标之内或目标与周围背景之间的亮度差别。如果成像系统在物体成像过程中的对比度选取得比较低，那么该物体所成的像（目标）看起来就比实际物体要暗一些；如果对比度降至 0，则物体将会从图像中消失。

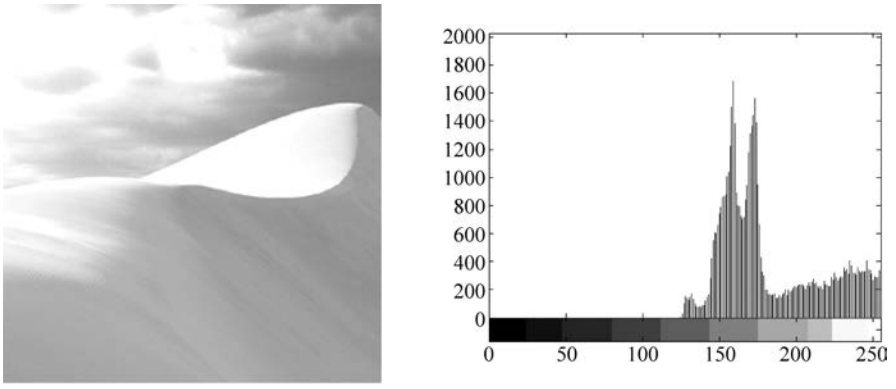
3. 图像直方图的分布特征及其与图像对比度的关系

下面用一组图像及其直方图的示例，说明图像直方图的分布特征及其与图像对比度的关系。图 3.3 是具有 4 种基本图像类型（暗且低对比度、亮且低对比度、低对比度、高对比度）的图像及其灰度直方图。

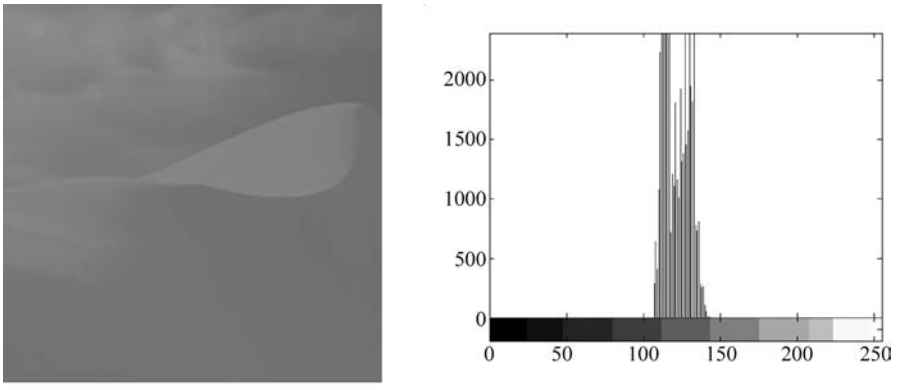


(a) 图像比较暗且对比度较低时的直方图分布特征

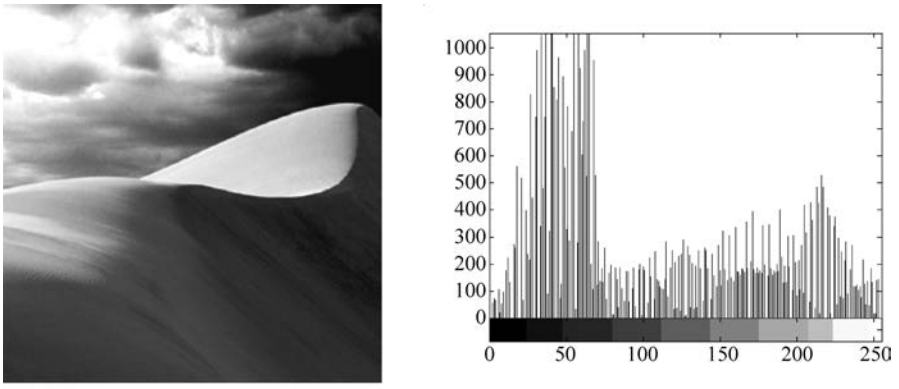
图 3.3　4 种基本图像类型（暗、亮、低对比度、高对比度）图像及其灰度直方图



(b) 图像比较亮且对比度较低时的直方图分布特征



(c) 图像亮度居中且对比度较低时的直方图分布特征



(d) 图像直方图均匀分布且对比度高时的直方图分布特征

图 3.3 （续）

图 3.3 说明了 4 种基本图像类型图像的直方图分布特征：当图像比较暗时，图像中的各像素的灰度值都比较小，所以直方图中的灰度分布主要集中在低像素极一端（左端）；由于图像中最大“白像素”和最小“黑像素”的比值较小，所以该类直方图的图像的对比度也较低。当图像比较亮时，图像中的各像素的灰度值都比较大，所以直方图中的灰度分布主要集中在高像素极一端（右端）；同理由于图像中最大“白像素”和最小“黑像素”的比值较小，所以该类直方图的图像的对比度也较低。当图像的对比度比较差（低）时，说明图像中多数较亮像素的灰度值与图像中多数较暗像素的灰度值的差别比较小，所以图像的直方图中的灰

度分布就比较集中地分布在某个灰度值差较小的范围内，即图像直方图会聚集在某些灰度值范围内。当图像的对比度较好(高)时，图像的直方图中的灰度就会相对比较均匀地分布在整个灰度级范围内。

3.3.2　灰度直方图的特征

灰度图像直方图具有如下一些特征。

(1) 灰度直方图仅能反映灰度图像中具有不同灰度值的像素出现的次数(或频次)，不能反映出灰度图像中各像素所在的(空间)位置信息。

(2) 灰度图像与灰度图像直方图之间存在着多对一的映射关系，即任一特定的灰度图像都有唯一对应的直方图，但不同的灰度图像可能有相同的直方图。

(3) 对于空间分辨率为 $M\times N$，且灰度级范围为$[0,L-1]$的图像，有关系式

$$\sum_{j=0}^{L-1}h(j)=M\times N \tag{3.4}$$

(4) 如果一幅图像由两个不连接的区域组成，则整幅图像的直方图等于两个不连接的区域的直方图之和。

3.3.3　归一化灰度图像直方图

由于式(3.3)所定义的灰度直方图反映的是图像中各灰度的实际出现频数，这样当某个灰度值的频数(计数值)远远大于其他灰度值的频数时，根据图像的某个或某些像素出现的最大频数来确定直方图的纵坐标的最大尺度既不方便也不太现实。也就是说，可能会由于直方图的纵坐标与横坐标尺寸过于悬殊而难于直观显示完整直方图的情况。为了能完整地显示任何形式的图像的直方图，通常采用归一化形式的直方图，即将直方图的横坐标取值和纵坐标取值变换到区间$[0,1]$。通常，如不加特殊说明，人们提到的直方图都是指归一化的直方图。

设 r_k 为图像 $f(x,y)$的第 k 级灰度，n_k 是图像 $f(x,y)$中灰度级为 r_k 的像素个数，$n=\sum_{k=1}^{L-1}n_k$ 是图像 $f(x,y)$的像素总数，则图像 $f(x,y)$的(归一化)灰度直方图定义为

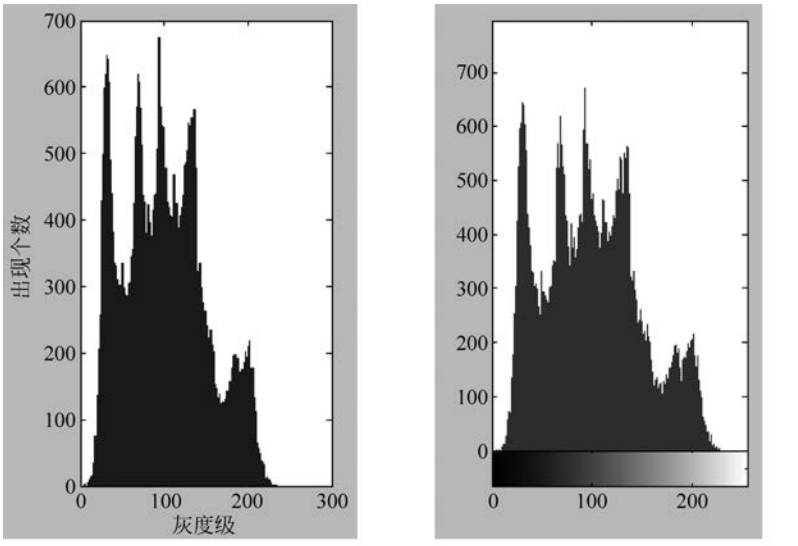
$$\begin{cases}P(r_k)=\dfrac{n_k}{n} \quad 0\leqslant r_k\leqslant 1;\quad k=0,1,\cdots,L-1\\ H(P)=[P(r_1),P(r_2),\cdots,P(r_{L-1})]\end{cases} \tag{3.5}$$

有关直方图的应用，将会在后续相关的章节中介绍。

【例 3.3】 编写 MATLAB 程序生成灰度图像的灰度直方图，并与由 MATLAB 直方图函数得到的灰度直方图进行比较，程序运行结果如图 3.4 所示。

```
clc; clear all; close all;                          % 清除命令窗口和工作空间,关闭 figure 窗口
img0 = imread('d:\0_matlab 图像课编程\lena.jpg') ;% 读入图像文件

[h,w] = size(img0);                                 % 获取图像的大小(高度 h,宽度 w)
gray_num = zeros(1,256);    % 创建一个 1～256 的数组记录各像素值出现的频数
imshow(img0); title('原图像');                      % 显示原图像

for i = 1:h
```

```
    for j = 1:w
        for k = 0:255 % 统计各像素值出现的频数
            if img0(i,j) == k
                gray_num(k + 1) = gray_num(k + 1) + 1; % MATLAB 矩阵的第一个元素下标应是 1
            end
        end
    end
end

figure; subplot(1,2,1);                          % 在另一个 figure 窗口显示直方图
bar(gray_num);                                   % 创建显示窗口,画直方图
title('编程画的图像灰度直方图');
xlabel('灰度级'); ylabel('出现个数');

subplot(1,2,2);
imhist(img0);                                    % 用 MATLAB 的直方图函数显示直方图
title('MATLAB 函数画的图像灰度直方图');
```



(a) 编程画的图像灰度直方图　　(b) MATLAB函数画的图像灰度直方图

图 3.4　编程画图像灰度直方图运行结果

在例 3.3 的程序中，统计各个灰度级出现个数时是按灰度级的范围 0～255 进行的，但在 MATLAB 中，数组的个数是从 1 开始算起的。程序的实现语句如下。

```
for k = 0:255
   if img0(i,j) == k
      gray_num(k + 1) = gray_num(k + 1) + 1;
   end
end
```

例 3.3 中的相关函数及功能说明如下。

(1) bar(Y)：当 Y 为矢量(本例即是)时，将其中的每个元素在窗口中绘制为一个条形。

(2) xlabel(x)：该函数用于在绘图窗口中的横轴(x 轴)方向上显示一个“标签”(即一个字符串)，这个字符串用于表明 x 轴的意义。例如，本例中横轴为灰度值坐标，所以其标签为 xlabel('灰度级')。

(3) ylabel(y)：该函数用于在绘图窗口中的纵轴(y 轴)方向上显示一个“标签”，用于表明 y 轴的意义。例如，本例中纵轴为各灰度级出现的频数，所以其标签为 ylabel('出现个数')。

(4) imhist(I)：MATLAB 中提供的默认显示 256 灰度级图像 I 的直方图的函数。当格式为 imhist(I,n)时，n 用于指定图像 I 的灰度级数目。

3.4　图像的代数运算

图像的代数运算包括两幅图像的相加运算和相减运算。

3.4.1　图像的相加运算

图像相加是通过对两幅大小相同的图像对应位置像素的相加运算，以产生一幅新的含有两幅图像信息的图像的方法。图像相加也称为图像合成。设 $f_1(x,y)$和 $f_2(x,y)$分别表示大小为 $M\times N$ 的两幅输入图像，图像 $f_1(x,y)$和图像 $f_2(x,y)$相加后得到的结果输出图像为 $g(x,y)$，且 $x\in[0,M-1]$，$y\in[0,N-1]$，则两幅图像的相加运算可表示为

$$g(x,y)=f_1(x,y)+f_2(x,y) \tag{3.6}$$

图像相加运算的实现方法主要有以下两种。

1. 两幅图像内容的相加/合成

两幅 256 灰度级图像对应坐标位置像素值的相加，其结果必然会超过其最大的灰度表示范围 256，最常用的处理方法是将两幅图像像素灰度值相加后的平均值作为相加结果，如式(3.7)所示。

$$g(x,y)=\text{IntegerRound}\left(\frac{1}{2}f_1(x,y)+\frac{1}{2}f_2(x,y)\right) \tag{3.7}$$

其中，IntegerRound 为四舍五入取整函数，即将相加运算的结果置为整数值。由于数字图像的灰度值都是整数，所以严格来说有关图像的任何相加、相减、相乘和相除运算，对其运算结果都应用函数 IntegerRound 进行四舍五入的取整处理。但为了简化起见，本书后续有关图像灰度值的运算公式，均略去了 IntegerRound 函数。

图 3.5 给出了一个两幅灰度图像按式(3.7)相加的示例。图 3.5(a)和图 3.5(b)是两幅不同的输入图像，图 3.5(c)是由两幅输入图像进行相加运算后的合成图像(彩色图像详见文前彩插)。

(a) 鹦鹉图像　(b) 湖面图像　(c) 合成结果图像

图 3.5　图像的相加运算示例

式(3.7)可以推广到两幅灰度图像按不同比例灰度值的叠加/合成，如式(3.8)所示。

$$g(x,y)=\alpha f_1(x,y)+\beta f_2(x,y) \tag{3.8}$$

其中，$\alpha+\beta=1$。

两幅灰度图像相加还有另一种典型方式，根据两幅图像所有像素灰度值相加结果的最小值和最大值情况，将其等比例缩小到结果图像灰度值符合 0～255 的灰度值范围。这样做可以使图像的亮度分布到整个 256 灰度区间。

两幅灰度图像及彩色图像的相加/合成运算，也可以推广到多幅图像的相加/合成。

【例 3.4】 图像相加(合成)运算的 MATLAB 程序，程序运行结果如图 3.5 所示。

```
clc; clear all; close all;
f1 = imread('d:\0_matlab 图像课编程\parrot.jpg');
f2 = imread('d:\0_matlab 图像课编程\sea_cloud.jpg');

f3 = 0.5 * f1 + 0.5 * f2;

subplot(1,3,1); imshow(f1); title('鹦鹉图像');
subplot(1,3,2); imshow(f2); title('湖面图像');
subplot(1,3,3); imshow(f3); title('合成结果图像');
```

2. 多幅图像的叠加去加性噪声

利用多幅灰度图像的叠加运算可以去除加性(additive)随机噪声。

图像的加性噪声可以简单地理解为在图像的背景中，与其相邻像素的灰度值相比，那些有明显差异的一些随机的、离散的和孤立的像素点，最容易理解的是黑区域上叠加的白点或白区域上叠加的黑点。

由于噪声产生的随机性，从同一场景获取的多幅图像中的噪声不可能完全相同。因此，可以通过对同一场景的多幅图像的灰度值求平均值，实现消除图像叠加噪声的目的。

当噪声互不相关且均值为零时，利用式(3.9)的 N 幅灰度图像的均值定义式可以降低噪声的影响。

$$g(x,y)=\frac{1}{N}\sum_{i=1}^{N}f_i(x,y) \tag{3.9}$$

图 3.6(a)是由于闪电、雷击、大气中的电暴等而使获取的天文图像有加性噪声，图 3.6(b)是类似于图 3.6(a)的两幅图像的去加性噪声的结果，图 3.6(c)是类似于图 3.6(a)的 4 幅图像的去加性噪声的结果，图 3.6(d)是类似于图 3.6(a)的 16 幅图像的去加性噪声的结果。可见，叠加的图像幅数越多，去除加性噪声的效果越明显。

(a) 有加性噪声图像　(b) 两幅去噪效果　(c) 4幅去噪效果　(d) 16幅去噪效果

图 3.6　利用图像叠加运算去加性噪声效果示例

3.4.2　图像的相减运算

设 $f_1(x,y)$和 $f_2(x,y)$表示大小为 $M\times N$ 的两幅输入图像，从图像 $f_1(x,y)$中的各位置的像素值中减去图像 $f_2(x,y)$的相应位置的像素值后，得到的结果输出图像为 $g(x,y)$，且 $x\in[0,M-1]$，$y\in[0,N-1]$，则两幅图像的相减运算可表示为

$$g(x,y)=f_1(x,y)-f_2(x,y) \tag{3.10}$$

当两幅 256 灰度级图像对应坐标位置像素值相减的结果大于或等于零时，则取其为结果图像中对应位置像素的灰度值；当相减结果小于零时，一般都是取零为结果值。当然，对于某些特殊的应用目的，也可以取其绝对值为结果值。

图像相减运算的典型应用是图像的变化检测。例如，在目前得到广泛应用的图像监控系统中，通过定时地将图像监控系统拍摄的现场图像与该现场初始情况下的图像进行相减运算，就可以判定被监控场景是否有异样情况出现。如图 3.7 所示，图 3.7(a)是监控系统某时刻拍摄的现场监控图像，图 3.7(b)是被监控现场的初始图像，图 3.7(c)是从图 3.7(a)中减去图 3.7(b)后的结果图像。

(a) 某时刻监控图像　(b) 现场初始图像　(c) 相减结果图像

图 3.7　图像的相减运算示例

需要注意的是，图像的灰度级是一个已有约定的有限大小的整数。以 256 的灰度级图像为例，图像代数运算的结果理应要求像素值不能大于 255，也不能为负数。所以在有关的应用中一般都有对运算结果中不符合要求的像素值的处理约定。例如，对运算结果为负的像素值取 0 值或取其绝对值，对运算结果大于 255 的像素值取 255 或取关于 256 的模运算(MOD)结果等。

【例 3.5】 图像相减运算(图像变化检测)的 MATLAB 程序，程序运行结果如图 3.7 所示。

```
clc; clear all; close all;
f1 = imread('d:\0_matlab 图像课编程\nopeson.jpg');
f2 = imread('d:\0_matlab 图像课编程\haspeson.jpg');

f3 = f1 - f2;

subplot(1,3,1); imshow(f1); title('某时刻监控图像');      % 无人的楼道图像
subplot(1,3,2); imshow(f2); title('现场初始图像');        % 有人的楼道图像
subplot(1,3,3); imshow(f3); title('相减结果图像');        % 变化检测结果图像
```

3.5　图像的几何运算

图像的几何运算又称为图像的几何变换，用于使原图像产生大小、形状和位置等变化效

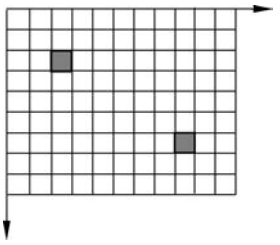


图 3.8　平移原理图

果。图像的几何运算包括图像的平移变换、图像的旋转变换、图像镜像、图像转置和图像的缩放等。

3.5.1　图像平移变换

图像平移(Image Translation)变换，是指将一幅图像或一幅图像的子图像块(以下简称为图像块)中的所有像素点，都按指定的 x 方向和 y 方向的偏移量 Δx 和 Δy 进行移动。

设初始坐标为(x_0，y_0)的像素平移(Δx，Δy)后的坐标为(x_1，y_1)，如图 3.8 所示，则有

$$\begin{cases} x_1 = x_0 + \Delta x \\ y_1 = y_0 + \Delta y \end{cases} \tag{3.11}$$

图像平移变换式(3.11)的矩阵形式为

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3.12}$$

图像的平移一般分为以下两种情况。

(1) 图像块平移。即将一幅图像中的某个子图像块平移到另一处。例如，在图 3.9 中，lena 图像中 lena 眼部的一个子图像块被平移到了该图像的右下角，该子图像块同时覆盖掉了新位置上原来的那些图像内容。

(a) 原图像　　(b) 平移图像子块后的图像

图 3.9　图像(图像子块)平移示例

(2) 整幅图像平移。整幅图像平移后，相对于原来图像(位置)来说，如果完整保持被平移的原图像内容，形成的新结果图像的幅面就被放大了；如果平移后的结果图像仍保持原来图像的幅面大小，被移出的部分就要被截掉。

【例 3.6】 原图像在 x 方向平移 dx(设为 20)个像素位置、在 y 方向平移 dy(设为 40)个像素位置，整幅图像平移后不放大而将移出的部分截断，程序运行结果如图 3.10 所示。

```
clc; clear; close all;
img0 = imread('d:\0_matlab 图像课编程\lena.jpg');

[h,w] = size(img0);                    % 获取图像矩阵行数 h 和列数 w
img1 = zeros(h,w);                     % 建立大小为 h×w 的零值矩阵，暂存平移结果

dx = 20; dy = 40;                      % 在 x 方向和 y 方向的平移距离(像素个数)初值
h1 = h - dx;                           % 行方向下移 dx 个像素位置
```

```
w1 = w - dy;                          % 列方向右移 dy 个像素位置

for i = 1:h1
    for j = 1:w1
        x = i + dx;                   % 计算平移后像素的行坐标
        y = j + dy;                   % 计算平移后像素的列坐标
        img1(x,y) = img0(i,j);        % 将(i,j)处的像素点平移到(x,y)处
    end
end
result_img = uint8(img1);

subplot(1,2,1); imshow(img0); title('平移前原图像');    % 显示原图像
subplot(1,2,2); imshow(result_img); title('平移后结果图像');
```

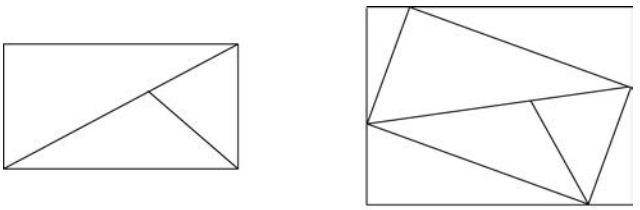
(a) 平移前原图像　　(b) 平移后结果图像

图 3.10　例 3.6 的整幅图像平移截断移出部分结果

3.5.2　图像旋转变换

图像旋转(image rotation)变换，是指以图像的中心为原点，将图像中的所有像素(即整幅图像)旋转一个相同的角度。

与图像平移变换类似，图像旋转变换的结果图像也分为两种情况：一是旋转后的图像幅面被放大(按外接矩形尺寸)，如图 3.11 所示；二是保持图像旋转前后的幅面大小，把旋转后图像被转出原幅面大小的部分截断。



(a) 旋转前的图像　　(b) 旋转后的图像

图 3.11　整幅图像旋转后图像幅面放大示例

1. 旋转后图像幅面放大的图像旋转变换

通常情况下，当一幅图像旋转后且要保留原图像中的所有画面时，按旋转后的外接矩形尺寸度量的该图像一定会变大许多。从分析图像的旋转原理角度，下面的讨论假设原图像旋转后的图像幅面仍小于屏幕大小，即不考虑因图像幅面变大而超出屏幕大小需

要截断的情况。

实现图像旋转变换的实质是以显示屏上的图像中心为原点，将图像中的所有像素（即整幅图像）旋转一个相同的角度。图像旋转变换公式的推导分为两步，第一步是先把待旋转图像的中心看作是图像平面的 xOy 显示坐标系的原点（由于是基于显示屏上的图像旋转，所以按图像显示坐标，横坐标为 y 轴，纵坐标为 x 轴），并基于该图像平面的 xOy 显示坐标系推导出图像上像素点旋转变换公式；第二步再把第一步推导的变换公式映射到原点位于屏幕左上角的图像显示坐标，由此得出在显示坐标下的图像旋转变换公式。

1）基于图像平面 xOy 坐标系原点的旋转变换

在图 3.12 中，待旋转图像的中心是图像平面的 xOy 坐标系的原点，并设位于 (x_0, y_0) 处的点（为了全文表示上的一致，像素点坐标 x 和 y 的顺序不变）到坐标原点的直线 r 与 y 轴的夹角为 α，当线段 r 沿顺时针（如图 3.12(a)所示）或逆时针（如图 3.12(b)所示）旋转 β 角度后，就使位于 (x_0, y_0) 处的点被旋转至 (x_1, y_1) 处。

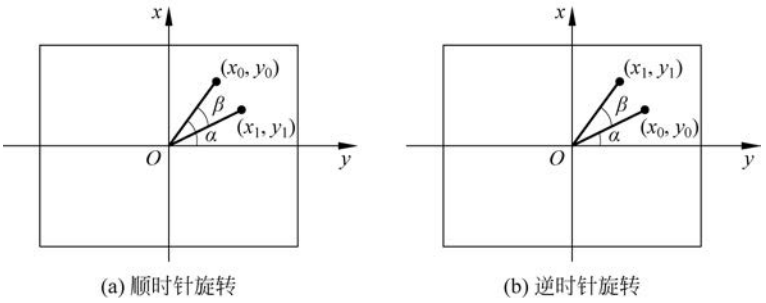


图 3.12　基于图像平面 xOy 坐标系原点的旋转原理示意图

下面以图 3.12(a)的顺时针旋转 β 角度为例，说明旋转变换的原理及公式。

显然，在旋转前

$$\begin{cases} x_0 = r\sin\alpha \\ y_0 = r\cos\alpha \end{cases} \tag{3.13}$$

旋转后

$$\begin{cases} x_1 = r\sin(\alpha-\beta) = r\sin\alpha\cos\beta - r\cos\alpha\sin\beta \\ y_1 = r\cos(\alpha-\beta) = r\cos\alpha\cos\beta + r\sin\alpha\sin\beta \end{cases} \tag{3.14}$$

将式(3.13)代入式(3.14)中，得到

$$\begin{cases} x_1 = x_0\cos\beta - y_0\sin\beta \\ y_1 = x_0\sin\beta + y_0\cos\beta \end{cases} \tag{3.15}$$

由式(3.15)即可得到基于图像平面 xOy 坐标系原点的顺时针旋转变换的矩阵表示形式为

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3.16}$$

同理，依据图 3.12(b)的逆时针旋转示意图可推出基于图像平面 xOy 坐标系原点的逆时针旋转变换的矩阵表示形式为

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3.17}$$

2）基于图像显示坐标的图像像素点的顺时针旋转变换

由于图像像素点的旋转变换是在原点位于屏幕左上角的图像显示坐标实现的，所以还需要将式(3.16)的顺时针变换矩阵表示形式映射到图像的显示坐标中。图 3.13 给出了图像的像素点与图像的显示坐标及图像平面 xOy 坐标关系的图示说明。

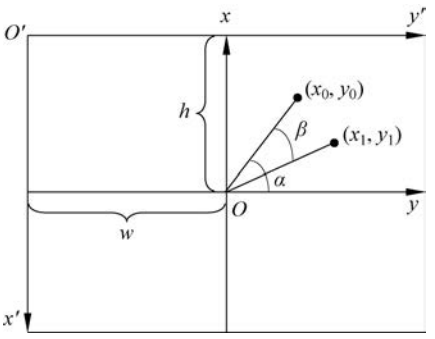


图 3.13　图像的像素点与图像的显示坐标和 xOy 平面坐标的关系示例

假设图像的宽度 width 用 w 表示，图像的高度 high 用 h 表示，则由图 3.13 可知：在原点 O' 位于左上角，x' 轴方向朝下，y' 轴方向朝右的图像显示坐标 $x'O'y'$ 中，如果用 (x_0', y_0') 表示 (x_0, y_0) 在图像显示坐标 $x'O'y'$ 中的位置，用 (x_1', y_1') 表示 (x_1, y_1) 在图像显示坐标 $x'O'y'$ 中的位置，则有

$$\begin{cases} x_0' = 0.5h - x_0 \\ y_0' = 0.5w + y_0 \end{cases} \tag{3.18}$$

$$\begin{cases} x_1' = 0.5h - x_1 \\ y_1' = 0.5w + y_1 \end{cases} \tag{3.19}$$

由式(3.18)可得其逆变换的矩阵表达式为

$$\begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & -0.5h \\ 0 & 1 & 0.5w \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0' \\ y_0' \\ 1 \end{bmatrix} \tag{3.20}$$

由式(3.19)可得其矩阵表示形式为

$$\begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0.5h \\ 0 & 1 & 0.5w \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{3.21}$$

将式(3.16)和式(3.20)依次代入式(3.21)中，即可得到图像像素点的旋转变换公式为

$$\begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0.5h \\ 0 & 1 & 0.5w \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & -0.5h \\ 0 & 1 & 0.5w \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0' \\ y_0' \\ 1 \end{bmatrix}$$

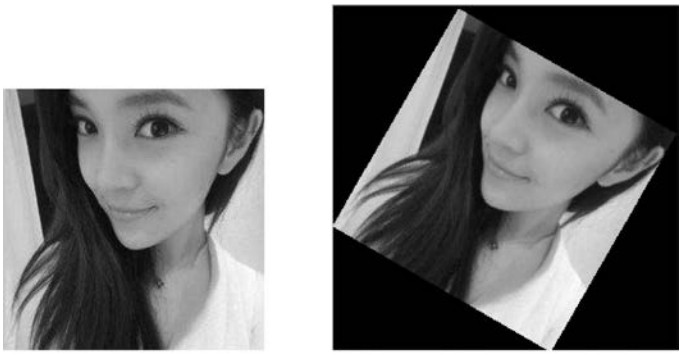
化简计算上式，并用 x_1 和 x_0 分别代替 x_1' 和 x_0'，用 y_1 和 y_0 代替 y_1' 和 y_0'，可得

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & \sin\beta & 0.5h - 0.5h \times \cos\beta - 0.5w \times \sin\beta \\ -\sin\beta & \cos\beta & 0.5w + 0.5h \times \sin\beta + 0.5w \times \cos\beta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3.22}$$

需要注意的是，在利用式(3.22)进行图像旋转变换时，计算得到的坐标值 x_1 和 y_1

一般不会是整数，但数字图像旋转后的坐标值必须是整数，因此应尽可能地取与 x_1 和 y_1 最接近的整数值。也正是因为这种近似，所以图像旋转后会有一定的改变，但这种改变并不明显。

【例 3.7】 图像旋转后，保持原图像幅面大小（不截断），即结果图像变大，程序运行结果如图 3.14 所示。



(a) 原图像　　(b) 旋转后图像

图 3.14　旋转后幅面放大的图像旋转变换结果

```
clc; clear all; close all;
img = imread('d:\0_matlab 图像课编程\girl.jpg');
[h w] = size(img);

jiaodu = 30;                                    % 顺时针旋转角度的初值,本次为 30°
theta = jiaodu/180 * pi;                        % 将旋转角度转换为弧度
rot = [cos(theta) -sin(theta) 0;sin(theta) cos(theta) 0;0 0 1];    % 顺时针旋转
% rot = [cos(theta) sin(theta) 0; -sin(theta) cos(theta) 0;0 0 1]; % 逆时针旋转

pix1 = [1 1 1] * rot;                           % 求变换后的新图像左上角像素(1, 1)的坐标
pix2 = [1 w 1] * rot;                           % 求变换后的新图像右上角像素(1, w)的坐标
pix3 = [h 1 1] * rot;                           % 求变换后的新图像左下角像素(h, 1)的坐标
pix4 = [h w 1] * rot;                           % 求变换后的新图像右下角像素(h, w)的坐标

% 变换后图像的高度
height = round(max([abs(pix1(1) - pix4(1)) + 0.5 abs(pix2(1) - pix3(1)) + 0.5]));
% 变换后图像的宽度
width = round(max([abs(pix1(2) - pix4(2)) + 0.5 abs(pix2(2) - pix3(2)) + 0.5]));
imgn = zeros(height,width);       % 生成变换后图像的 0 值像素阵列,结果图像阵列初值

% 取得 x 方向的负轴超出的偏移量
delta_x = abs(min([pix1(1) pix2(1) pix3(1) pix4(1)]));
% 取得 y 方向的负轴超出的偏移量
delta_y = abs(min([pix1(2) pix2(2) pix3(2) pix4(2)]));

for i = 1 - delta_x:height - delta_x
    for j = 1 - delta_y:width - delta_y
      pix = [i j 1]/rot;                        % 用变换后图像的点的坐标去寻找原图像点的坐标
      % 否则有些变换后的图像的像素点无法完全填充
      float_X = pix(1) - floor(pix(1));
      float_Y = pix(2) - floor(pix(2));
```

```
        if pix(1)>=1 && pix(2)>=1 && pix(1)<= h && pix(2)<= w
          pix_up_left=[floor(pix(1)) floor(pix(2))];          %4个相邻的点
          pix_up_right=[floor(pix(1)) ceil(pix(2))];
          pix_down_left=[ceil(pix(1)) floor(pix(2))];
          pix_down_right=[ceil(pix(1)) ceil(pix(2))];

          value_up_left=(1-float_Y)*(1-float_X);          %计算相邻4个点的权重
          value_up_right=float_Y*(1-float_X);
          value_down_left=(1-float_Y)*float_X;
          value_down_right=float_Y*float_X;

          imgn(i+delta_x,j+delta_y)=value_up_left*img(pix_up_left(1),pix_up_left
(2))+...
          value_up_right*img(pix_up_right(1),pix_up_right(2))+ ...
          value_down_left*img(pix_down_left(1),pix_down_left(2))+ ...
          value_down_right*img(pix_down_right(1),pix_down_right(2));
        end
      end
end

imshow(img); title('原图像');
figure; imshow(uint8(imgn)); title('旋转后图像');
```

对例 3.7 中的相关编程思路和程序代码解译如下。

(1) 程序语句 rot=[cos(theta) −sin(theta) 0; sin(theta) cos(theta) 0; 0 0 1]的含义为

$$\text{rot}=\begin{bmatrix}\cos(\text{theta}) & -\sin(\text{theta}) & 0\\ \sin(\text{theta}) & \cos(\text{theta}) & 0\\ 0 & 0 & 1\end{bmatrix}$$

(2) 语句 pix1=[1 1 1]*rot 的功能是求变换后结果图像左上角像素的坐标,计算结果是一个一维向量,其中 pix1(1)是高度坐标,pix1(2)是宽度坐标值。

$$\begin{aligned}\text{pix1}&=[1\ \ 1\ \ 1]\begin{bmatrix}\cos(\text{theta}) & -\sin(\text{theta}) & 0\\ \sin(\text{theta}) & \cos(\text{theta}) & 0\\ 0 & 0 & 1\end{bmatrix}\\ &=[\cos(\text{theta})+\sin(\text{theta})\quad -\sin(\text{theta})+\cos(\text{theta})\quad 1]\\ &\cdots\end{aligned}$$

语句 pix4=[h w 1]*rot 的功能是求变换后结果图像右下角像素的坐标,计算结果是一个一维向量,其中 pix4(1)是高度坐标,pix4(2)是宽度坐标值。

$$\begin{aligned}\text{pix4}&=[h\quad w\quad 1]\begin{bmatrix}\cos(\text{theta}) & -\sin(\text{theta}) & 0\\ \sin(\text{theta}) & \cos(\text{theta}) & 0\\ 0 & 0 & 1\end{bmatrix}\\ &=[h\cdot\cos(\text{theta})+w\cdot\sin(\text{theta})\quad -h\cdot\sin(\text{theta})+w\cdot\cos(\text{theta})\quad 1]\end{aligned}$$

(3) 语句 height=round(max([abs(pix1(1)−pix4(1))+0.5 abs(pix2(1)−pix3(1))+0.5]))和语句 width=round(max([abs(pix1(2)−pix4(2))+0.5 abs(pix2(2)−pix3(2))+0.5]))含义是:当图像旋转一定角度后,把图像的四个角的像素点 pix1、pix2、pix3、pix4 中互为对角的两个顶点 pix1 和 pix3 的 x(高度)值相减取绝对值再加 0.5,并四舍五入取整作

为旋转后结果图像的高；把互为对角的两个顶点 pix2 和 pix4 的 y(宽度)值相减取绝对值再加 0.5，并四舍五入取整作为旋转后结果图像的宽。

例 3.7 中的相关函数及功能说明如下。

(1) round(x)：四舍五入取整。

(2) y=floor(x)：对 x 向下取整(即取不大于 x 的最大整数)，值 y 为不大于取整后的 x 的最大整数。

(3) ceil(x)：返回不小于 x 的最小整数值，即向正无穷方向取整。

(4) max(x)：①若 x 是一个向量，则返回 x 中元素的最大值；若 x 是一个矩阵，则以行的形式返回每一列元素的最大值。②max(x,[],2)以列的形式返回每一行元素的最大值；max(x,[],1)以行的形式返回每一列元素的最大值。③max(x,y)返回数值 x 与 y 中较大者。

(5) min(x)：求向量 x 中元素的最小值，其他格式类似于 max。

(6) abs(x)：求 x 的绝对值。

2. 旋转前后幅面大小保持不变的图像旋转变换

设原图像被旋转角度 β(弧度制)后，原图像中位于(x_0,y_0)处的像素点被移位至(x_1,y_1)处。在不考虑图像旋转后图像幅面是否超出原幅面大小的情况下，横轴为 y 轴、竖轴为 x 轴的 xoy 平面的旋转变换公式(3.16)即为图像顺时针旋转的图像变换公式。

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3.23}$$

对应地，式(3.17)即为图像逆时针旋转的图像变换公式。

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3.24}$$

在根据式(3.23)或式(3.24)对原图像进行旋转变换时，需要判断变换后的每个像素的坐标(x_1,y_1)是否超出了原图像的尺寸范围。只有那些 x_1 值和 y_1 值都没有超出图像尺寸范围的像素才得以保留，而其他像素由于其坐标超出了原图像尺寸而无法显示，所以要将其丢弃。

另外需要注意的是，当图像旋转的角度值为负值时，利用式(3.23)运算的结果相当于沿逆时针旋转；利用式(3.24)运算的结果相当于沿顺时针旋转。

【例 3.8】 保持原图像尺寸大小不变(超出原图像大小部分被截断)的图像旋转 MATLAB 程序。

```
clc; clear; close all;
img0 = imread('d:\0_matlab 图像课编程\girl.jpg'); % 读入原始图像
[h,w,m] = size(img0);                             % 获取图像大小

img1 = 255 * ones(h,w,m);          % 构造结果矩阵,每个像素点默认初始化为 255(白色)
alfa = 15 * 3.1415926 / 180.0; % 旋转角度赋值(设为 15°),并将其转换为弧度
tras = [cos(alfa) -sin(alfa) 0; sin(alfa) cos(alfa) 0; 0 0 1];      % 顺时针旋转变换矩阵
% tras = [cos(alfa) sin(alfa) 0; -sin(alfa) cos(alfa) 0; 0 0 1];    % 逆时针旋转变换矩阵
```

```
for i = 1:h
    for j = 1:w
        temp = [i; j; 1];              % 数学坐标(x = j,y = i)
        temp = tras * temp;    % 坐标变换,变换后像素坐标向量 = 变换矩阵 × 变换前坐标向量
        x = uint16(temp(1, 1));        % 取出 i 值并转换成 16 位的无符号数据
        y = uint16(temp(2, 1));        % 取出 j 值并转换成 16 位的无符号数据
        % 判断变换后的像素坐标是否越界,只保留原图像幅面大小范围内的像素值
        if (x <= h) & (y <= w) & (x >= 1) & (y >= 1)
            img1(i, j,:) = img0(x, y,:);
        end
    end
end;

subplot(1,2,1); imshow(img0);          % 显示原图像
subplot(1,2,2); imshow(uint8(img1)); % 显示旋转结果图像
```

例 3.8 的程序运行结果如图 3.15(b)所示。当把程序中的第 6 行语句改成注释(在该行前面添加%),而执行程序中的第 7 行语句(去掉该行前面的%)时,运行结果如图 3.15(c)所示。

(a) 原图像　(b) 顺时针旋转15°图像　(c) 逆时针旋转15°图像

图 3.15　旋转后幅面保持不变的图像旋转变换结果

例 3.8 中的相关函数及功能如下。

(1) ones(h,w,m)：建立一个大小为 h×w,元素值为 double 类型的全 1 值矩阵。

(2) uint16(x)：将 x 映射到无符号 16 位范围内。

3.5.3　图像镜像变换

图像镜像(image mirroring)变换分为图像水平镜像变换和图像垂直镜像变换两种。

1. 图像水平镜像

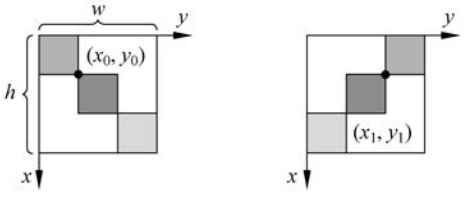
图像水平镜像(horizontal mirroring)是指以原图像为参照,使原图像和水平镜像结果图像与虚拟的垂直轴成对称关系,如图 3.16(a)和图 3.16(b)所示。

设图像的高度 height 为 h,宽度 width 为 w。在显示坐标中,原图像中位于(x_0, y_0)处的像素点,经水平镜像后在水平镜像结果图像上的对应像素点为(x_1, y_1)。根据图 3.17 的映射关系,则图像水平镜像变换可表示为

(a) 原图像　　(b) 水平镜像结果图像

图 3.16　图像水平镜像结果示例

$$\begin{cases} x_1 = x_0 \\ y_1 = w - y_0 + 1 \end{cases} \tag{3.25}$$



(a) 原图像显示坐标　　(b) 图像水平镜像显示坐标

图 3.17　图像水平镜像映射关系

式(3.25)的矩阵表示形式为

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & w+1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3.26}$$

【例 3.9】 图像水平镜像 MATLAB 程序，程序运行结果如图 3.16 所示。

```
% 图像水平镜像 MATLAB 程序(horizontal_mirroring33.m)
clc; clear; close all;
img0 = imread('d:\0_matlab 图像课编程\lena.bmp');

[h,w] = size(img0);                                    % 获取矩阵的行数 h 和列数 w

for x0 = 1:h            % 把原图像最(偏)右边的列移到新图像的最(偏)左边
    for y0 = 1:w
        result_img(x0,w - y0 + 1) = img0(x0,y0);       % 根据式(3.25)进行坐标变换
    end
end

subplot(1,2,1);  imshow(img0);  title('原图像');     % 显示原图像
subplot(1,2,2);  imshow(result_img);  title('水平镜像结果图像');
```

2. 图像垂直镜像

图像垂直镜像(vertical mirroring)是指以原图像为参照，使原图像和垂直镜像结果图像与虚拟的水平轴成对称关系，如图 3.18 所示。

设图像的高度 height 为 h，宽度 width 为 w。在显示坐标中，原图像中位于(x_0, y_0)处

的像素点，经垂直镜像后在垂直镜像结果图像上的对应像素点为(x_1, y_1)。根据图 3.19 的映射关系，则图像垂直镜像变换可表示为

$$\begin{cases} x_1 = h - x_0 + 1 \\ y_1 = y_0 \end{cases} \tag{3.27}$$

式(3.27)的矩阵表示形式为

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & h+1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3.28}$$

图 3.18　图像垂直镜像结果示例

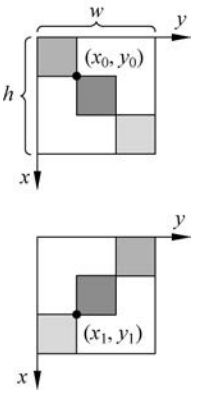


图 3.19　图像垂直镜像映射关系

【例 3.10】 图像垂直镜像 MATLAB 程序，程序运行结果如图 3.18 所示。

```
clc; clear; close all;
img0 = imread('d:\0_matlab 图像课编程\lena.bmp');

[h,w] = size(img0);                                    % 获取图像矩阵的行数 h 和列数 w

for x0 = 1:h                % 把原图像最/偏下边的行移到新图像最/偏上边的行
    for y0 = 1:w
        result_img(h - x0 + 1,y0) = img0(x0,y0);   % 根据式(3.27)进行坐标变换
    end
end

subplot(2,1,1); imshow(img0);                          % 显示原图像
subplot(2,1,2); imshow(result_img);                    % 显示垂直镜像结果图像
```

3.5.4　图像转置变换

图像转置(image transpose)变换是指将图像显示坐标的 x 轴与 y 轴对换，显示效果示例如图 3.20 所示。

设原图像位于(x_0,y_0)处的像素点，经图像转置变换后在转置变换结果图像上的对应像素点为(x_1,y_1)。根据图 3.21 的映射关系，则图像转置变换可表示为

$$\begin{cases} x_1 = y_0 \\ y_1 = x_0 \end{cases} \tag{3.29}$$

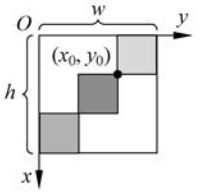
式(3.29)的矩阵表示形式为

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3.30}$$

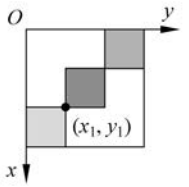
(a) 转置前原图像

(b) 转置后的图像

图 3.20 图像转置变换示例



(a) 原图像显示坐标



(b) 转置图像显示坐标

图 3.21 图像转置映射关系

需要特别注意的是，图像的转置变换与图像的旋转变换是不一样的。原图像不论是顺时针旋转 90°，还是逆时针旋转 90°，都不会得到图像转置后的结果。

【例 3.11】 图像转置 MATLAB 程序，程序运行结果如图 3.20 所示。

```
clc; clear; close all;
img0 = imread('d:\0_matlab 图像课编程\lena.bmp');

[h,w] = size(img0);                                    % 获取图像的行数 h 和列数 w

for i = 1:h
    for j = 1:w
        result_img(j,i) = img0(i,j);                   % 根据式(3.29)进行坐标变换
    end
end

subplot(1,2,1); imshow(img0); title('转置前原图像');       % 显示原图像
subplot(1,2,2); imshow(result_img); title('转置后的图像'); % 显示转置结果图像
```

3.5.5 图像缩小与放大

图像缩放(image scaling)是指对图像进行缩小或放大，即对数字图像的大小进行调整的过程。图像缩放是一种非平凡的过程，需要在处理效率以及结果的平滑度(smoothness)和清晰度(sharpness)上做一个权衡。

设原图像中位于(x_0,y_0)处的像素点，经对原图像的行和列按相同比例 r 缩放后，在缩放后的结果图像上的对应像素点为(x_1,y_1)，则图像缩放前后像素点的坐标可表示为

$$\begin{cases} x_1 = rx_0 \\ y_1 = ry_0 \end{cases} \tag{3.31}$$

图像缩放的矩阵表示形式为

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{3.32}$$

其中，当 $0<r<1$ 时为缩小原图像；当 $r>1$ 时为放大原图像。

1. 缩小图像

缩小图像(shrink image)的目的一般有两个：一是为了使缩小后的图像符合显示区域的大小要求；二是为了生成被缩小图像(原图像)的缩略图。缩小后的图像的平滑度和清晰度相对于原图像都有所增强。

缩小图像也称为下采样(subsampled)或降采样(downsampled)。对于一般的行数和列数都为偶数的图像来说，一种方法是只取原图像偶数行和偶数列交叉处的像素，如图 3.22 所示；另一种方法是只取原图像奇数行和奇数列交叉处的像素，如图 3.23 所示。这种缩小图像方法的结果从整体上看，就是将原图像缩小到原来大小的 1/4。

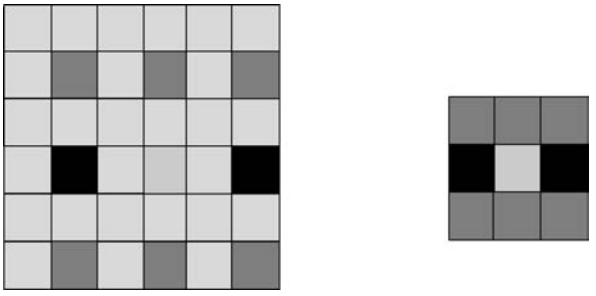


图 3.22　仅取偶数行和偶数列像素缩小原图像

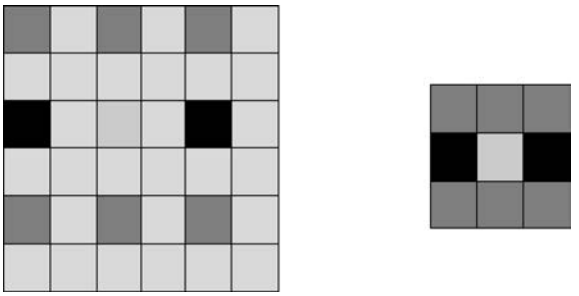


图 3.23　仅取奇数行和奇数列像素缩小原图像

如果要再将缩小后的图像缩小为原来的 1/4，只要在前述缩小的图像的基础上，采用原来方法进行即可。当然还有其他一些不按整数比例缩小图像的方法，限于篇幅，此处不再赘述。

【例 3.12】 仅取奇数行和奇数列像素的图像缩小 MATLAB 程序，程序运行结果如图 3.24 所示。

```
% 按整倍数缩小图像的 MATLAB 程序
clc; clear; close all;
img0 = imread('d:\0_matlab 图像课编程\lena.jpg');
```

```
[h,w] = size(img0);
N = 2;     % 图像缩小倍数(每 N 行/列留 1 行/列)
result_row = round(h/N);  result_col = round(w/N);    % 结果图像行数和列数
temp_img = zeros(result_row,result_col);              % 生成全零值结果图像阵列

x1 = 1;                                               % 保留的行方向的初始行数值
for x0 = 1:N:h                                        % 行数按步长 N = 2 循环,即去掉偶数行
    y1 = 1;                                           % 保留地列方向的初始列数值
    for y0 = 1:N:w                                    % 列数按步长 N = 2 循环,即去掉偶数列
        temp_img(x1,y1) = img0(x0,y0);                % 像素值赋值
        y1 = y1 + 1;
    end
    x1 = x1 + 1;
end
result_img = uint8(temp_img);                         % 转换成 8 位图像

figure; imshow(img0); title('原图像');                % 显示原图像
figure; imshow(result_img); title('缩小 1 倍后的图像');
```

2. 最近邻域插值放大图像方法

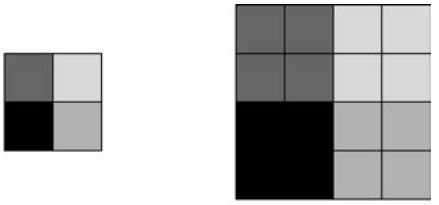
放大图像(enlarge image)的目的一般是为了使放大后的图像更好地显示在更高分辨率的显示设备上。放大图像的方法相对比较多，也相对复杂一些，从了解图像放大原理的角度，下面介绍最基本也是最典型的最近邻域插值图像放大方法。

1) 按整数倍数放大图像的最近邻域插值法

按整数倍数放大图像(比如将图像放大两倍)的最近邻域插值法的基本思想是：将原图像中的每一个原像素原封不动地复制映射到放大后的新图像的 4 个像素中。该方法的原理如图 3.25 所示。

(a) 原图像　(b) 缩小1倍后的图像

图 3.24　例 3.12 程序运行结果



(a) 原图像　(b) 放大后的图像

图 3.25　按最近邻域插值法将图像放大 4 倍示例

2) 按非整数倍数放大图像的最近邻域插值法

设放大前的图像称为原图像，其在显示坐标的 x 方向和 y 方向的坐标值分别用 x_{old} 和 y_{old} 表示，图像高度(x 方向)用 h_{old} 表示，图像宽度(y 方向)用 w_{old} 表示。放大后的图像称为目标(新)图像，其在显示坐标的 x 方向和 y 方向的坐标值分别用 x_{new} 和 y_{new} 表示，图像高度(x 方向)用 h_{new} 表示，图像宽度(y 方向)用 w_{new} 表示。则按非整数倍数放大图像的最近邻域插值法的原图像和目标图像的坐标关系可表示为

$$\begin{cases} x_{\text{old}} = \text{IntegerRound}(x_{\text{new}} \times (h_{\text{old}}/h_{\text{new}})) \\ y_{\text{old}} = \text{IntegerRound}(y_{\text{new}} \times (w_{\text{old}}/w_{\text{new}})) \end{cases} \tag{3.33}$$

当已知图像放大系数 m 时，由 $m=h_{new}/h_{old}$ 和 $x_{old}=x_{new}\times(h_{old}/h_{new})=x_{new}/(h_{new}/h_{old})=x_{new}/m$ 得

$$\begin{cases}x_{old}=\text{IntegerRound}(x_{new}/m)\\ y_{old}=\text{IntegerRound}(y_{new}/m)\end{cases}\tag{3.34}$$

下面用一个例子来说明按非整数倍数放大图像的最近邻域插值法的应用方法。

【例 3.13】 设已知有一个 3×3 的灰度图像，如图 3.26(a)所示。利用按非整数倍数放大图像的最近邻域插值法将该图像放大为 4×4 的图像，如图 3.26(b)所示。

234 38 22	234 38 22 22	234 38 38 22
67 44 12	67 44 12 12	67 44 44 12
89 65 63	89 65 63 63	67 44 44 12
	89 65 63 63	89 65 65 63
(a) 原图像	(b) 放大后的图像1	(c) 放大后的图像2

图 3.26　按非整数倍放大图像的最近邻域插值法的放大结果示例

解：在插值放大后的目标图像中根据式(3.31)逐个地计算从(0,1)至(3,3)的每个像素的值。

对于目标图像中坐标为(0,0)处的像素，因为有

$$x_{old}=x_{new}\times(h_{old}/h_{new})=0\times(3/4)=0$$

$$y_{old}=y_{new}\times(w_{old}/w_{new})=0\times(3/4)=0$$

即目标图像中位于(0,0)处的像素值应是原图像中位于(0,0)处的像素值，该值为 234。

对于目标图像中坐标为(0,1)处的像素，因为有

$$x_{old}=x_{new}\times(h_{old}/h_{new})=0\times(3/4)=0$$

$$y_{old}=y_{new}\times(w_{old}/w_{new})=1\times(3/4)=0.75\approx 1$$

即目标图像中位于(0,1)处的像素值应是原图像中位于(0,1)处的像素值，该值为 38。

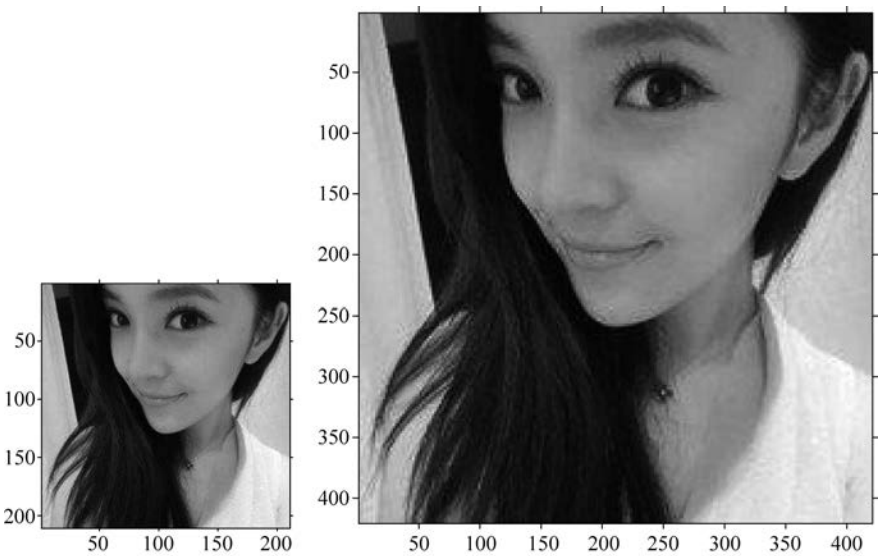
同理，可得目标图像中位于(0,2)处和(0,3)处的像素值都是 22；位于(1,0)处、(1,1)处、(1,2)处和(1,3)处的像素值分别是 67、44、12 和 12；位于(2,0)处、(2,1)处、(2,2)处和(2,3)处的像素值分别是 89、65、63 和 63；位于(3,0)处、(3,1)处、(3,2)处和(3,3)处的像素值分别是 89、65、63 和 63。放大后的结果图像如图 3.19(b)所示。

需要注意的是，由于 MATLAB 的阵列下标是从(1,1)开始算起的，上述的新图像坐标范围就会是(1：4,1：4)而不是(0：3,0：3)，加上下标的四舍五入取整影响，得到的放大后的新图像为图 3.26(c)。

【例 3.14】 以任意倍数 m 放大图像的 MATLAB 程序，程序运行结果如图 3.27 所示。

```
clc; clear all; close all;
img = imread('d:\0_matlab 图像课编程\girl.jpg');

[h_old, w_old] = size(img);
m = input('\n 请输入任意值的图像放大倍数 k = '); % 整数、小数或分数值

h_new = round(m * h_old);                    % 确保非整数的 k 倍放大后，图像大小为整数
w_new = round(m * w_old);
```

```
for i = 1:h_new                                          %逐一为像素点赋值
    for j = 1:w_new
        %变换公式(3.33),x_old = x_new * (h_old/h_new),y_old = y_new * (h_old/h_new)
        x_old = round(i/m);
        if  x_old > h_old
            x_old = h_old;
        else
            x_old(x_old < 1) = 1;
        end

        y_old = round(j/m);
        y_old(y_old > w_old) = w_old;
        y_old(y_old < 1) = 1;

        img1(i,j) = img(x_old,y_old);
    end
end

figure, imshow(img); axis on;
title(['原图像(大小:',num2str(h_old),'×',num2str(w_old),')']);
figure; imshow(img1); axis on;
title(['放大后的图像(大小:',num2str(h_new),'×',num2str(w_new),')']);
```



(a) 原图像(大小: 210×210)　　(b) 放大后的图像(大小: 420×420)

图 3.27　最邻近插值图像放大 2 倍程序运行结果

在例 3.14 中，if-else-end 语句为

```
if  x_old > h_old
    x_old = h_old;
else
    x_old(x_old < 1) = 1;
end
```

如果用以下语句代替，将会使程序显得更简洁(这就是 MATLAB 不同于其他程序语言的独特之处)。

```
x_old(x_old > h_old) = h_old;
x_old(x_old < 1) = 1;
```

例 3.14 中的相关函数及功能说明如下。

(1) axis on：显示坐标轴上的标记、单位和格栅。

最近领域插值法放大的图像可保留图像中所有的原始信息，但是会产生锯齿现象和马赛克现象。为了克服最近领域插值法的不足，人们进一步提出了双线性插值法、三次样条插值和基于边缘的图像插值方法等，下面介绍双线性插值法，其余方法感兴趣的读者可以参考有关文献。

(2) a=input('...')：接收用户输入，输入可以是一个值，也可以是一个矩阵(用一对方括号括住，不同的行用分号分隔)，并把输入值(或矩阵值)保存到变量 a 中。引号里是提示信息，可以根据输入格式加入一定的换行(\n)符号。

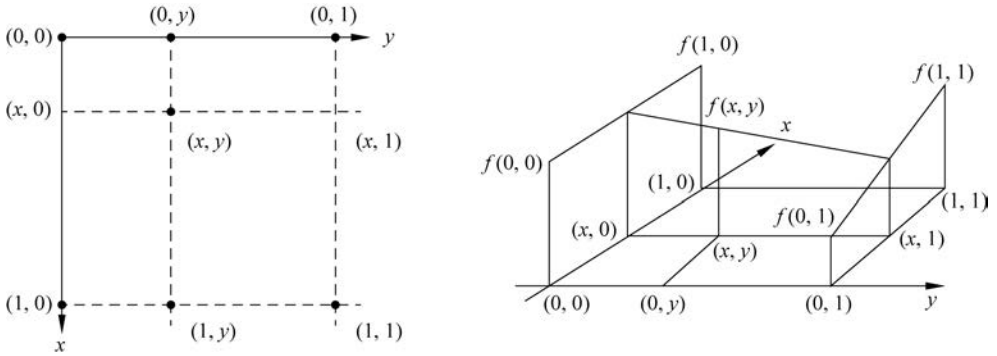
3. 双线性插值放大图像方法

双线性插值法通过输入图像到输出图像的映射，即通过计算输出像素被映射到输入图像中 4 个像素之间的非整数位置的像素点灰度值，并将具有该灰度值的像素点插入该位置，来实现对输入图像的放大。

已知像素点(0,0)、(0,1)、(1,0)和(1,1)如图 3.28(a)所示，要插值的点为(x,y)。双线性插值的基本思路是：首先在 x 方向上线性插值，在(0,0)和(1,0)两个点之间插入点$(x,0)$，在(0,1)和(1,1)两个点之间插入点$(x,1)$；然后在 y 方向线性插值，即通过计算出的点$(x,0)$和$(x,1)$，在 y 方向上插值计算出点(x,y)。在单位正方形顶点的值已知的情况下，正方形内任意点 $f(x,y)$的值可由如下的双线性方程得到。

$$f(x,y)=ax+by+cxy+d \tag{3.35}$$

其中，a、b、c 和 d 是待定常数，其值由正方形 4 个顶点的值确定。



(a) 插值点关系示意图　　(b) 插值运算示意图

图 3.28　双线性插值运算图示

双线性插值运算如图 3.28(b)所示。首先，在 x 方向上作线性插值，对上端的两个点(0,0)和(1,0)进行线性插值可得

$$f(x,0)=f(0,0)+x[f(1,0)-f(0,0)] \tag{3.36}$$

同理，对低端的两个点(0,1)和(1,1)进行线性插值可得

$$f(x,1)=f(0,1)+x[f(1,1)-f(0,1)] \tag{3.37}$$

然后，在 y 方向上作线性插值，有

$$f(x,y)=f(x,0)+y[f(x,1)-f(x,0)] \tag{3.38}$$

将式(3.36)和式(3.37)代入式(3.38)，有

$$f(x,y)=f(0,0)+x[f(1,0)-f(0,0)]+y[f(0,1)+x[f(1,1)-f(0,1)]-f(0,0)-x[f(1,0)-f(0,0)]]$$

整理上式，即可得双线性插值公式为

$$f(x,y)=[f(1,0)-f(0,0)]x+[f(0,1)-f(0,0)]y+[f(1,1)+f(0,0)-f(0,1)-f(1,0)]xy+f(0,0) \tag{3.39}$$

基于上述算法原理，就可以在一幅输入图像上，根据各像素的 4 个相邻点的灰度值，分别在 x 和 y 方向上进行二次线性插值，得到结果图像的过程示意如图 3.29 所示。

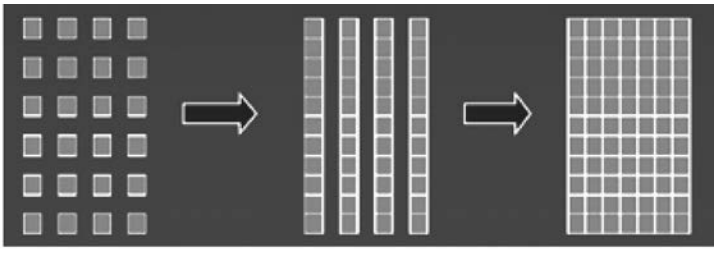
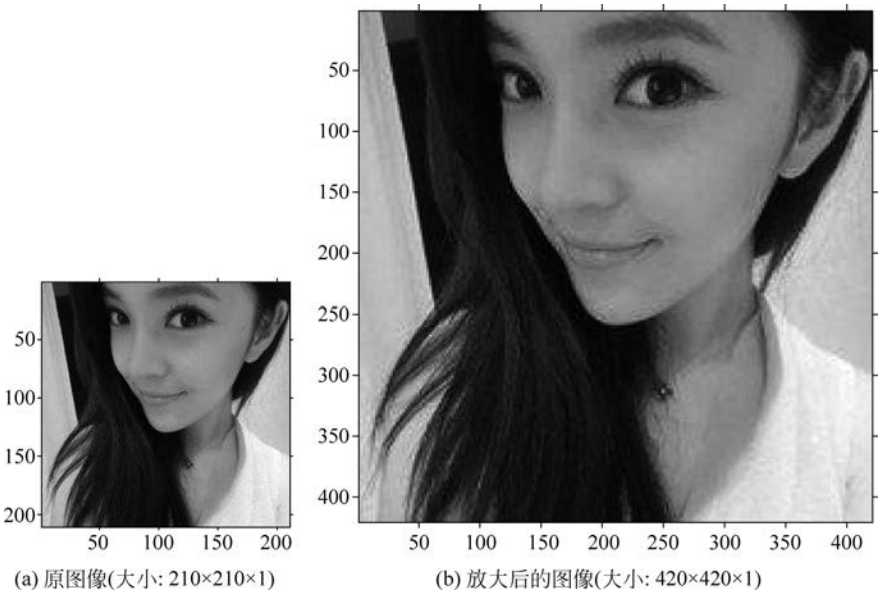


图 3.29　利用双线性插值算法对图像进行插值过程示例

【例 3.15】 双线性插值放大图像的 MATLAB 程序，程序运行结果如图 3.30 所示。

```
clc; clear all; close all;
img = imread('d:\0_matlab 图像课编程\girl.jpg');

zmf = 2; % 放大因子(倍数)赋初值,应该大于 1,这里设为 2
[h,w,c] = size(img);

% 扩展原图像 img 的像素阵列的边缘,得到临时图像 img1
img2 = zeros(h + 2,w + 2,c);                    % 创建全零值的结果图像,结果图像赋初值

img1(1,2:w + 1) = img(1,:);                     % 给扩展图像 img1 的最上一行赋值
img1(1,1) = img(1,1);                           % 给扩展图像 img1 的左上角像素赋值
img1(1,w + 2) = img(1,w);                       % 给扩展图像 img1 的右上角像素赋值
img1(2:h + 1,1) = img(:,1);                     % 给扩展图像 img1 的最左一列赋值
img1(2:h + 1,2:w + 1) = img(1:h,1:w);           % 扩展图像 img1 的中间赋值原图像的像素值
img1(2:h + 1,w + 2) = img(:,w);                 % 给扩展图像 img1 的最右一列赋值
img1(h + 2,2:w + 1) = img(h,:);                 % 给扩展图像 img1 的最下一行赋值
img1(h + 2,1) = img(h,1);                       % 给扩展图像 img1 的左下角像素赋值
img1(h + 2,w + 2) = img(h,w);                   % 给扩展图像 img1 的右下角像素赋值
% figure; imshow(img1);  title('四周扩展后的原图像');

% 根据缩放因子得到新图像的大小,并创建新图像
h1 = round(h * zmf);                            % 计算放大后的图像高度,就近取整
w1 = round(w * zmf);                            % 计算放大后的图像宽度,就近取整
img2 = zeros(h1,w1,c);                          % 创建全零值的结果图像,结果图像赋初值
```

```
%将新图像的某个像素(i,j)映射到原始图像(i0,j0)处,并插值
for j = 1:w1                                    %对新图像按列逐个元素扫描
    for i = 1:h1
        i0 = (i-1)/zmf; j0 = (j-1)/zmf;
        ii = floor(i0); jj = floor(j0);   %向下取整
        u = i0 - ii; v = j0 - jj;
        ii = ii + 1; jj = jj + 1;
        img2(i,j,:) = (1-u)*(1-v)*img1(ii,jj,:) +(1-u)*v*img1(ii,jj+1,:)...
                    + u*(1-v)*img1(ii+1,jj,:) +u*v*img1(ii+1,jj+1,:);
    end
end
img2 = uint8(img2);

%显示原图像与放大后图像,并显示坐标轴上的标记和单位
figure, imshow(img); axis on;
title(['原图像(大小: ',num2str(h),'×',num2str(w),'×',num2str(c),')']);
figure; imshow(img2); axis on;
title(['放大后的图像(大小: ',num2str(h1),'×',num2str(w1),'×',num2str(c)',')']);
```



(a) 原图像(大小: 210×210×1)　　(b) 放大后的图像(大小: 420×420×1)

图 3.30　双线性插值放大 2 倍图像程序运行结果

在例 3.15 中,第 6 行至第 14 行语句的功能是让(h+2)×(w+2)得到 img1 图像,中间是 h * w 的原图像,并把原图像的最外一行和最外一列像素赋值给 img1 的最外一行和最外一列。其中,第 10 行的语句 img1(2:h+1,2:w+1)=img(1:h,1:w)的功能相当于下列双层 for 循环的功能。由此可以进一步体会 MATLAB 强大的语句表达能力。

```
for i = 1:h
    for j = 1:w
        img1(i+1,j+1) = img(i,j);
    end
end
```

习 题 3

1. 解释下列名词概念。

(1) 灰度图像的对比度　　(2) 归一化直方图

(3) 图像镜像　　(4) 图像转置

2. 在有些图像处理系统中，为什么要对输入图像的像素亮度进行对数运算处理？

3. 什么是图像直方图？灰度直方图有哪些性质？

4. 设有一灰度图像的直方图，请回答下列问题：

(1) 当该直方图偏左时，该图像的亮度和对比度有什么特征？

(2) 当该直方图偏右时，该图像的亮度和对比度有什么特征？

(3) 当该直方图挤在某个狭小区域时，该图像的对比度有什么特征？

5. 简述图像变化检测的原理及其应用领域。

6. 假设要把一幅图像放大为原来大小的两倍，请简述其放大原理或实现方法。

7. 已知有如图 3.31 所示的原图像：

1	2	4	6
5	4	2	3
4	3	2	1
5	6	7	8

图 3.31　习题 3.7 图

(1) 请给出该图像的水平镜像结果图像。

(2) 请给出该图像的垂直镜像结果图像。

(3) 请给出该图像转置后的结果图像。

(4) 请给出该图像向左旋转 90°后的结果图像。

8. 编写一个实现图像水平镜像功能的 MATLAB 程序，并显示原图像和镜像结果图像。