



数字图像处理：使用MATLAB分析与实现

第3章 图像基本运算

主讲教师 张涛

电子信息与电气工程学院



主要内容

- 3. 1 图像几何变换
- 3. 2 图像代数运算
- 3. 3 邻域及模板运算
- 3. 4 综合实例



3.1 图像几何变换

问题的提出:

- 在图像处理过程中，为了观测需要，常常需要对图像进行几何变换，如几何失真图像的校正、图像配准、电影、电视和媒体广告等的影像特技处理等，是图像变形以及校正变形的基础。
- 图像几何变换将图像中任一像素映射到一个新位置，是一种空间变换，关键在于确定图像中点与点之间的映射关系。



3.1 图像几何变换

3.1.1 图像几何变换基础

3.1.2 图像的位置变换

3.1.3 图像的形状变换



3.1.1 图像几何变换基础

图像几何变换

(1) 齐次坐标

- 用 $n+1$ 维向量表示 n 维向量的方法称为齐次坐标表示法。
- 在齐次坐标中，原图像用点集 $[x \ y \ 1]^T$ 表示



3.1.1 图像几何变换基础

图像几何变换

(2) 图像几何变换

- 通过齐次坐标中，原图像进行平移、缩放、旋转等几何变换，可以用一个变换矩阵表示。

- a, b, c, d 用于图形的比例、对称、错切、旋转等基本变换； k, m 用于图形的平移变换； p, q 用于投影变换； s 用于全比例变换。

$$T = \begin{bmatrix} a & b & k \\ c & d & m \\ p & q & s \end{bmatrix}$$

- 实现2D图像几何变换的基本变换的一般过程是：

变换矩阵 $T \times$ 变换前的点集矩阵 = 变换后的点集矩阵



3.1.1 图像几何变换基础

图像几何变换

(3) 图像插值运算

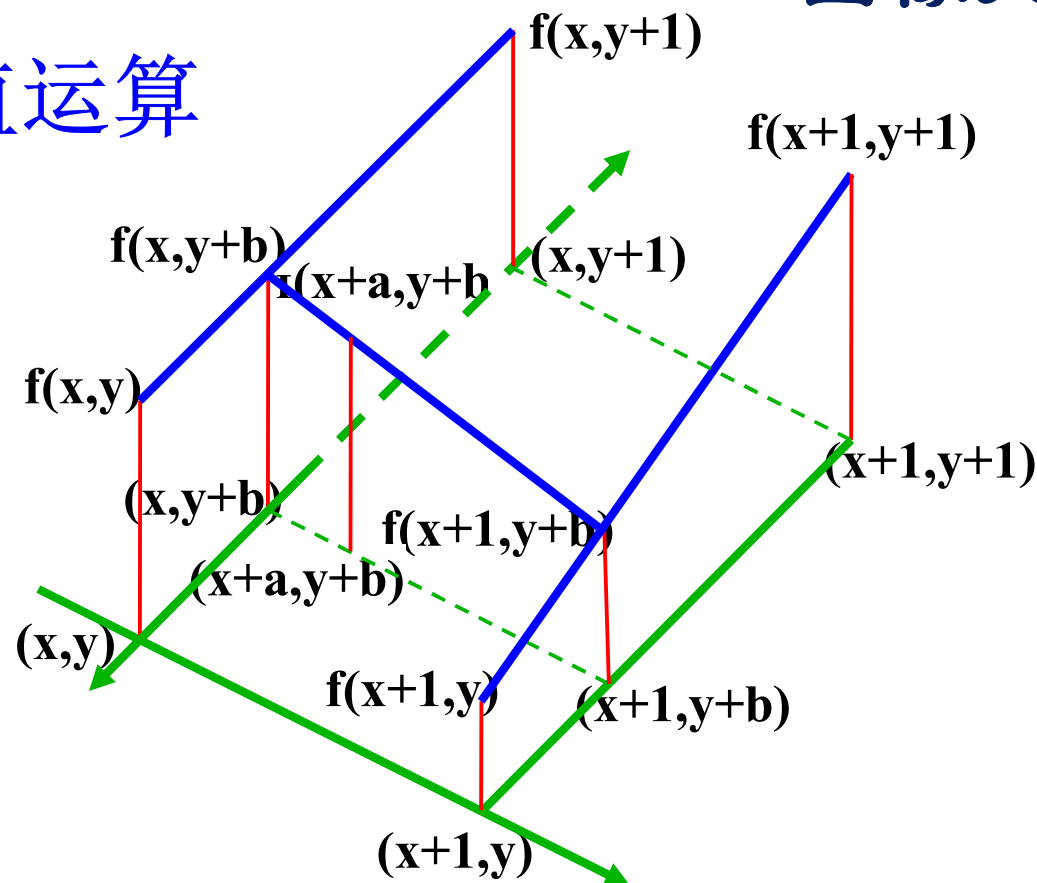
- 指利用已知邻近像素点的灰度值来产生未知像素点的灰度值。
- **最近邻插值**：非整数像素灰度值就等于距离最近的像素的灰度值。
- **双线性插值**：利用非整数像素点周围的四个像素点的相关性，通过双线性算法计算得出的。
- **双三次插值**：利用非整数像素点周围的16个像素点进行计算



3.1.1 图像几何变换基础

图像几何变换

(3) 图像插值运算



双线性插值

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

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

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



3.1.1 图像几何变换基础

图像几何变换

(4) 图像几何变换步骤：后向映射法

- 根据不同的几何变换公式**计算新图像的尺寸**；
- 根据几何变换的**逆变换**，对新图像中的每一点确定其在原图像中的对应点；
- 按对应关系给新图像中各个像素**赋值**：
 - 若新图像中像素点在原图像中的对应点坐标存在，**直接赋值**；
 - 若新图像中像素点在原图像中的对应点坐标超出图像宽高范围，**直接赋背景色**；
 - 若新图像中像素点在原图像中的对应点坐标在图像宽高范围内，**采用插值的方法计算**



3.1.2 图像的位置变换

图像几何变换

- 图像的位置变换是指图像的大小和形状不发生变化，只是图像像素点的位置发生变化，含平移、镜像、旋转。
- 图像的位置变换主要是用于目标识别中的目标配准。

3.1.2 图像的位置变换

图像几何变换

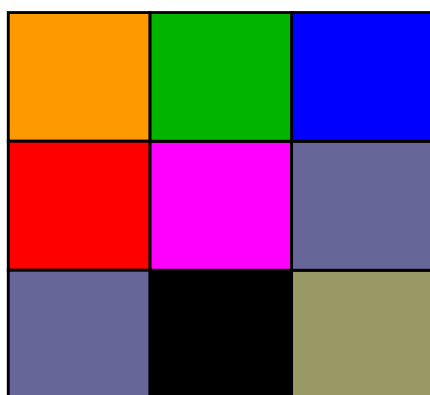
(1) 平移


■ 原理

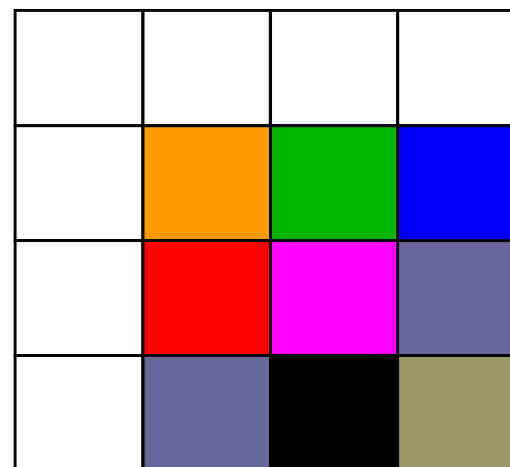
$$\begin{cases} x' = x + \Delta x \\ y' = y + \Delta y \end{cases}$$

矩阵表示

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



$\Delta x = 1$

 $\Delta y = 1$



若平移后不丢失信息，需扩大“画布”



3.1.2 图像的位置变换

图像几何变换

(1) 平移

■ 例程

□ 函数

T = maketform(TRANSFORMTYPE,...)

**B = imtransform(A,TFORM,INTERP,
PARAM1,VAL1,PARAM2,VAL2,...)**



3.1.2 图像的位置变换

图像几何变换

(1) 平移

程序

```
Image=imread('lotus.jpg'); deltax=20; deltay=20;  
T=maketform('affine',[1 0 0;0 1 0;deltax deltay 1]);  
□ NewImage1=imtransform(Image,T,  
                           'XData',[1 size(Image,2)],  
                           'YData',[1,size(Image,1)],  
                           'FillValue',255);  
NewImage2=imtransform(Image,T,  
                       'XData',[1 size(Image,2)+deltax],  
                       'YData',[1,size(Image,1)+deltay],  
                       'FillValue',255);  
subplot(131),imshow(Image),title('原图');  
subplot(132),imshow(NewImage1),title('画布尺寸不变平移');  
subplot(133),imshow(NewImage2),title('画布尺寸扩大平移');
```

3.1.2 图像的位置变换

图像几何变换

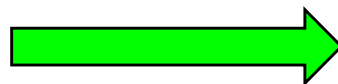
(1) 平移

□ 效果



画布不变

$$\Delta x = 20$$



$$\Delta y = 20$$

画布增大



3.1.2 图像的位置变换

图像几何变换

(2) 镜像

■ 原理

□ 水平镜像

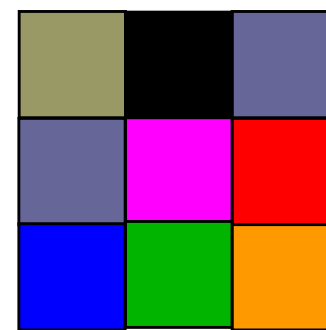
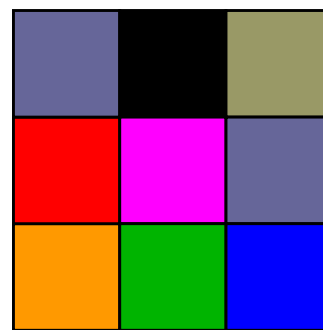
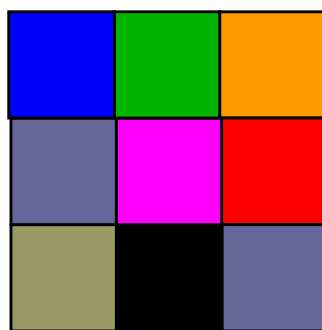
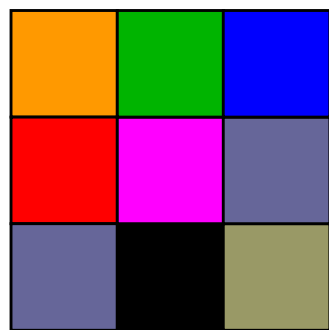
$$\begin{cases} x' = M - 1 - x \\ y' = y \end{cases}$$

□ 垂直镜像

$$\begin{cases} x' = x \\ y' = N - 1 - y \end{cases}$$

□ 对角镜像

$$\begin{cases} x' = M - 1 - x \\ y' = N - 1 - y \end{cases}$$



水平镜像

垂直镜像

对角镜像



3.1.2 图像的位置变换

图像几何变换

(2) 镜像

■ 例程

□ 函数

fliplr(X) flipud(X) flipdim(X,DIM)

□ 程序

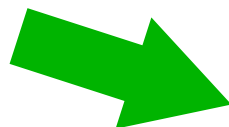
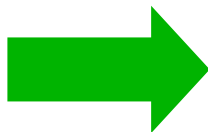
```
Image2=imread('lotus.jpg');  
subplot(221),imshow(Image2);  
HImage=flipdim(Image2,2);  
subplot(222),imshow(HImage);  
VImage=flipdim(Image2,1);  
subplot(223),imshow(VImage);  
CImage=flipdim(HImage,1);  
subplot(224),imshow(CImage);
```


3.1.2 图像的位置变换

图像几何变换

(2) 镜像

□ 效果





3.1.2 图像的位置变换

图像几何变换

(3) 旋转

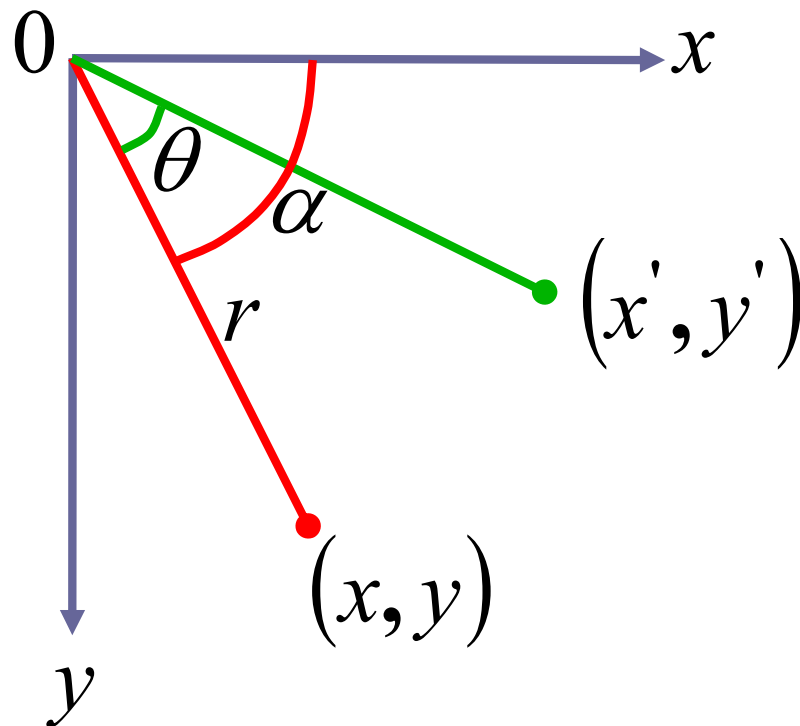
■ 原理

绕图像原点逆时针旋转

$$\begin{cases} x = r \cos \alpha \\ y = r \sin \alpha \end{cases}$$

$$\begin{cases} x' = r \cos(\alpha - \theta) \\ y' = r \sin(\alpha - \theta) \end{cases} \Rightarrow$$

$$\begin{cases} x' = r \cos \alpha \cdot \cos \theta + r \sin \alpha \cdot \sin \theta = x \cos \theta + y \sin \theta \\ y' = r \sin \alpha \cdot \cos \theta - r \cos \alpha \cdot \sin \theta = -x \sin \theta + y \cos \theta \end{cases}$$





3.1.2 图像的位置变换

图像几何变换

(3) 旋转

绕原点旋转计算公式

$$\begin{cases} x' = x \cos \theta + y \sin \theta \\ y' = -x \sin \theta + y \cos \theta \end{cases}$$

旋转逆变换公式

$$\begin{cases} x = x' \cos \theta - y' \sin \theta \\ y = x' \sin \theta + y' \cos \theta \end{cases}$$



3.1.2 图像的位置变换

图像几何变换

(3) 旋转

■ 旋转变换过程

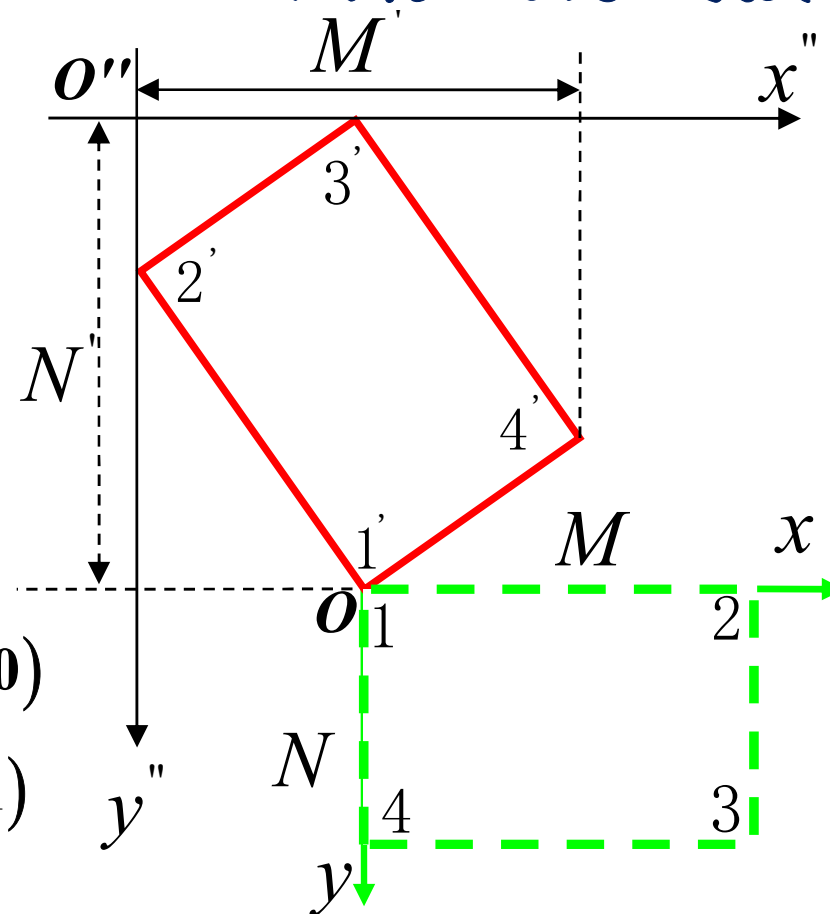
□ 确定旋转后新图像尺寸

◆ 计算原图像四个角在旋转后的坐标

$$(x_1, y_1) = (0, 0) \quad (x_2, y_2) = (M-1, 0)$$

$$(x_3, y_3) = (M-1, N-1) \quad (x_4, y_4) = (0, N-1)$$

$$\begin{cases} (x'_1, y'_1) = (0, 0) \\ (x'_2, y'_2) = ((M-1)\cos\theta, -(M-1)\sin\theta) \\ (x'_3, y'_3) = ((M-1)\cos\theta + (N-1)\sin\theta, -(M-1)\sin\theta + (N-1)\cos\theta) \\ (x'_4, y'_4) = ((N-1)\sin\theta, (N-1)\cos\theta) \end{cases}$$





3.1.2 图像的位置变换

图像几何变换

(3) 旋转

◆ 求 x', y' 方向的最大最小值

◆ 确定新图像的分辨率

$$\begin{cases} M' = \max x' - \min x' + 1 \\ N' = \max y' - \min y' + 1 \end{cases}$$

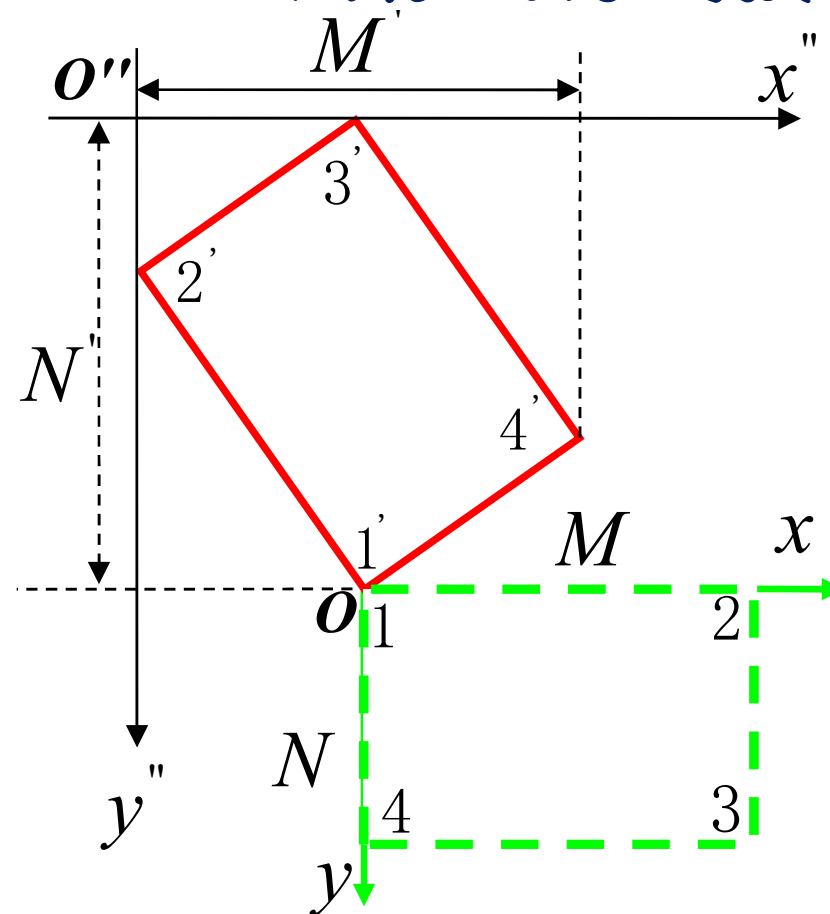
□ 坐标变换

$$x'' \in [0, M' - 1] \quad y'' \in [0, N' - 1]$$

$$\begin{cases} x' = x'' + \min x' \\ y' = y'' + \min y' \end{cases}$$

□ 旋转逆变换

□ 给新图像赋值





3.1.2 图像的位置变换

图像几何变换

(3) 旋转

绕中心点旋转先要将坐标系平移到中心点，再按绕原点旋转进行变换，然后平移回原坐标原点。绕任意点旋转与此相同，仅仅是平移量的不同。



3.1.2 图像的位置变换

图像几何变换

(3) 旋转

■ 实例：有一幅图像 $f(x, y) = \begin{bmatrix} 59 & 60 & 58 \\ 61 & 59 & 57 \\ 62 & 56 & 55 \end{bmatrix}$

以图像原点为坐标原点，将其逆时针旋转 30° 。

□ 确定旋转后新图像尺寸

$$\begin{cases} (x'_1, y'_1) = (0, 0) \\ (x'_2, y'_2) = (2 \cos 30^\circ, -2 \sin 30^\circ) = (1.732, -1) \\ (x'_3, y'_3) = (2 \cos 30^\circ + 2 \sin 30^\circ, -2 \sin 30^\circ + 2 \cos 30^\circ) = (2.732, 0.732) \\ (x'_4, y'_4) = (2 \sin 30^\circ, 2 \cos 30^\circ) = (1, 1.732) \end{cases}$$

$$\max x' = 2.732 \quad \min x' = 0 \quad \max y' = 1.732 \quad \min y' = -1$$



3.1.2 图像的位置变换

图像几何变换

(3) 旋转

$$\begin{cases} M' = \max x' - \min x' + 1 = 3.732 \approx 4 & x'' \in [0 \quad 3] \\ N' = \max y' - \min y' + 1 = 3.732 \approx 4 & y'' \in [0 \quad 3] \end{cases}$$

□ 坐标变换、逆变换及赋值

(x'', y'')	(x', y')	(x, y)	最邻近点
(0,0)	(0,-1)	(0.5,-0.866)	(1,-1)
(0,1)	(0,0)	(0,0)	(0,0)
(0,2)	(0,1)	(-0.5,0.866)	(-1,1)
(0,3)	(0,2)	(-1,1.732)	(-1,2)
(2,0)	(2,-1)	(2.232,0.134)	(2,0)
(2,1)	(2,0)	(1.732,1)	(2,1)
(2,2)	(2,1)	(1.232,1.866)	(1,2)
(2,3)	(2,2)	(0.732,2.732)	(1,3)



3.1.2 图像的位置变换

图像几何变换

(3) 旋转

□ 新图像为： $g(x,y) = \begin{bmatrix} 255 & 60 & 58 & 255 \\ 59 & 59 & 57 & 255 \\ 255 & 61 & 56 & 255 \\ 255 & 62 & 255 & 255 \end{bmatrix}$

在运算过程中，原图中的对应点超出图像范围，新图像中的点在原图像中没有对应点，直接赋背景色255（或赋0）；对于未超出图像范围但不是整数像素的对应点按最邻近插值。



3.1.2 图像的位置变换

图像几何变换

(3) 旋转

□ 为提高效果，可采用双线性插值

(x'', y'')	(x', y')	(x, y)
(1,1)	(1,0)	(0.866,0.5)

对应点(0.866,0.5)点位于 (0,0)、(0,1)、(1,0)、(1,1)四点之间，按双线性插值方法计算(0.866,0.5)的值，并赋给新图像中(1,1)点。

$$f(0,0.5) = f(0,0) + 0.5[f(0,1) - f(0,0)] = 59 + 0.5(61 - 59) = 60$$

$$f(1,0.5) = f(1,0) + 0.5[f(1,1) - f(1,0)] = 60 + 0.5(59 - 60) = 59.5$$

$$f(0.866,0.5) = f(0,0.5) + 0.866[f(1,0.5) - f(0,0.5)] = 59.567 \approx 60$$



3.1.2 图像的位置变换

图像几何变换

(3) 旋转

■ 例程

□ 函数: **B = imrotate(A,ANGLE,METHOD,BBOX)**

□ 程序

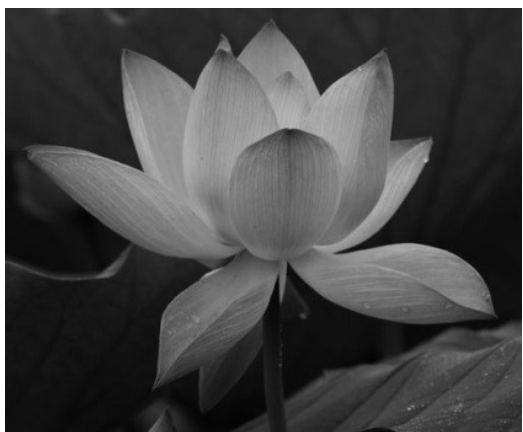
```
Image=im2double(imread('lotus.jpg'));  
NewImage1=imrotate(Image,15);  
NewImage2=imrotate(Image,15,'bilinear');  
subplot(1,2,1),imshow(NewImage1);  
subplot(1,2,2),imshow(NewImage2);
```

3.1.2 图像的位置变换

图像几何变换

(3) 旋转

□ 效果



最近邻
插值



双线性
插值





3.1.2 图像的位置变换

图像几何变换

(3) 旋转

□ 效果



最近邻插值



双线性插值



3.1.3 图像的形状变换

图像几何变换

(1) 缩放

■ 原理

$$\begin{cases} x' = k_x x \\ y' = k_y y \end{cases} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$M \times N$ 的图像缩小为: $k_x M \times k_y N$

$$k_x = k_y$$

按比例缩放

$$k_x \neq k_y$$

不按比例缩放





3.1.3 图像的形状变换

图像几何变换

(1) 缩放

■ 实例

有一幅图像 $f(x,y)=$

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

设缩放比例为：

$$k_x = 0.75 \quad k_y = 0.6$$

试对其进行缩放变换。

□ 确定新图像尺寸

$$M' = k_x M = 0.75 \times 6 = 4.5 \approx 5 \quad N' = k_y N = 0.6 \times 6 = 3.6 \approx 4$$

$$x' \in [0 \quad 4], y' \in [0 \quad 3]$$



3.1.3 图像的形状变换

图像几何变换

(1) 缩放

- 对于新图像中的点，利用缩放变换的逆变换，在原图像中找对应点，并赋值

$$x = \frac{x'}{k_x} = \frac{x'}{0.75} \quad y = \frac{y'}{k_y} = \frac{y'}{0.6}$$

- 产生新图像为：

(x', y')	对应点 (x, y)	最邻近点
(0,0)	(0,0)	(0,0)
(1,0)	(1.33,0)	(1,0)
(2,0)	(2.67,0)	(3,0)
(3,0)	(4,0)	(4,0)
(4,0)	(5.33,0)	(5,0)

$$g(x, y) = \begin{pmatrix} 1 & 2 & 4 & 5 & 6 \\ 13 & 14 & 16 & 17 & 18 \\ 19 & 20 & 22 & 23 & 24 \\ 31 & 32 & 34 & 35 & 36 \end{pmatrix}$$

为提高效果，可采用双线性插值



3.1.3 图像的形状变换

图像几何变换

(1) 缩放

■ 例程

□ 函数

B=imresize(A,SCALE,METHOD))

B=imresize(A,[NUMROWS NUMCOLS], METHOD))

[Y, NEWMAP] = imresize(X, MAP, SCALE, METHOD))

□ 程序

```
Image=im2double(imread('lotus.jpg'));  
NewImage1=imresize(Image,1.5,'nearest');  
NewImage2=imresize(Image,1.5,'bilinear');  
subplot(1,2,1),imshow(NewImage1);  
subplot(1,2,2),imshow(NewImage2);
```

3.1.3 图像的形状变换

图像几何变换

(1) 缩放

□ 效果



最近邻插值



双线性插值



3.1.3 图像的形状变换

图像几何变换

(1) 缩放

□ 效果



最近邻插值



双线性插值



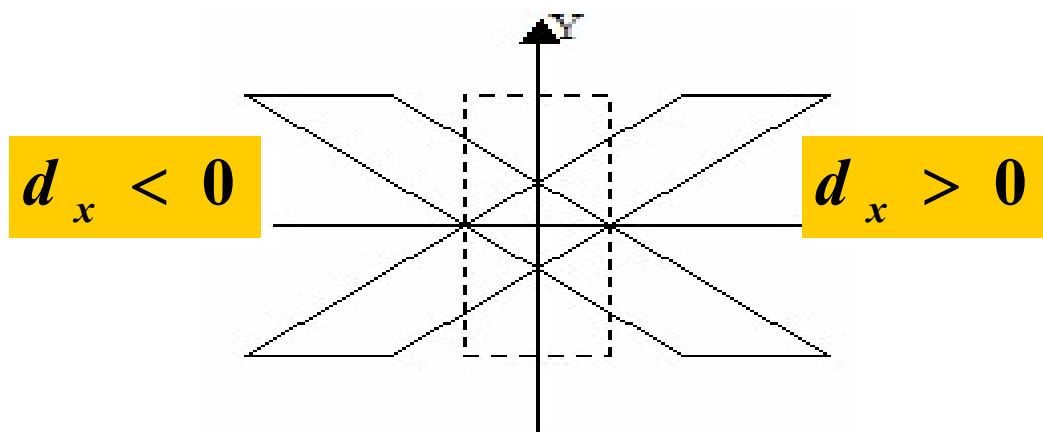
3.1.3 图像的形状变换

图像几何变换

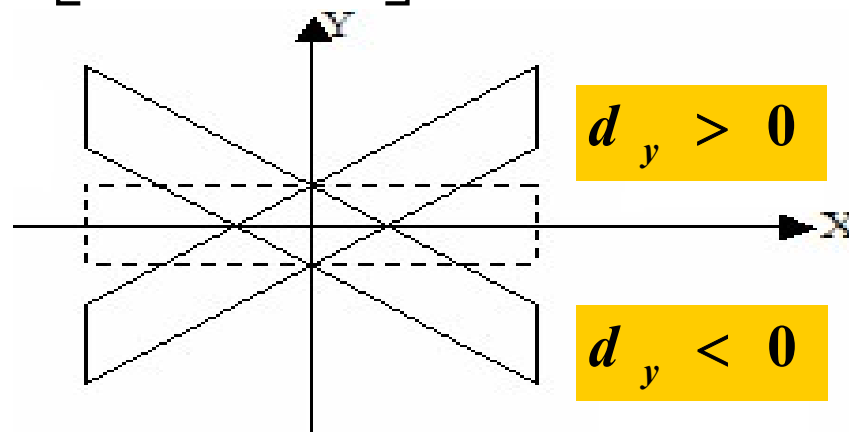
(2) 错切

- 原理 平面景物在投影平面上的非垂直投影，使图像中的图形产生扭变。

$$\begin{cases} x' = x + d_x y \\ y' = d_y x + y \end{cases} \quad T = \begin{bmatrix} 1 & d_x & 0 \\ d_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



X方向错切



Y方向错切



3.1.3 图像的形狀变换

图像几何变换

(2) 错切

■ 例程

□ 函数 **T = maketform(TRANSFORMTYPE,...)**
B = imtransform(A,TFORM,INTERP,
PARAM1,VAL1,PARAM2,VAL2,...)

□ 程序

```
Image=im2double(imread(bird.jpg));  
tform1=maketform('affine',[1 0 0;0.5 1 0; 0 0 1]);  
tform2=maketform('affine',[1 0.5 0;0 1 0; 0 0 1]);  
NewImage1=imtransform(Image,tform1);  
NewImage2=imtransform(Image,tform2);  
subplot(1,2,1),imshow(NewImage1);  
subplot(1,2,2),imshow(NewImage2);
```


3.1.3 图像的形状变换

图像几何变换

(2) 错切

□ 效果

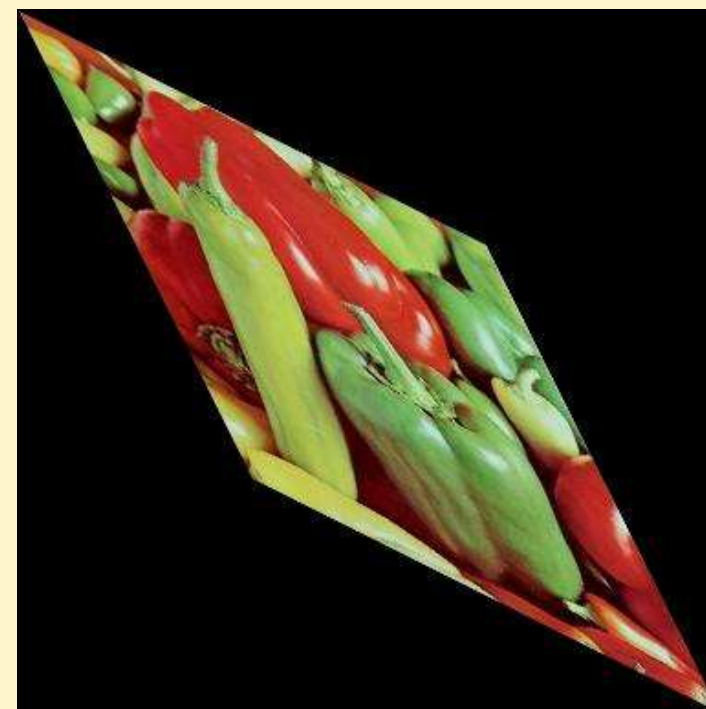
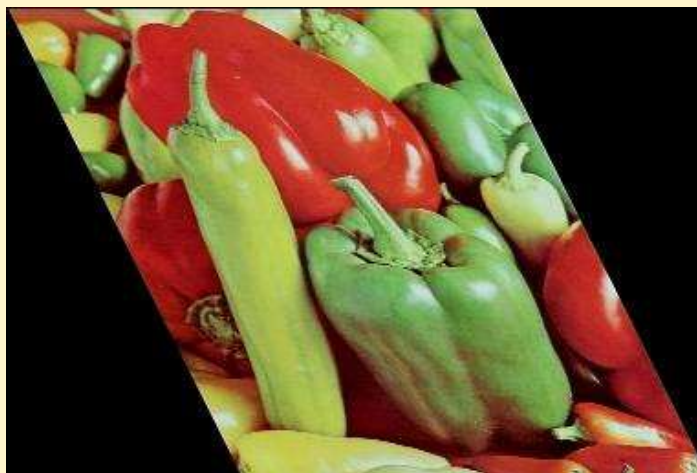


3.1.3 图像的形状变换

图像几何变换

(2) 错切

□ 效果





3.2 图像代数运算

3. 2. 1 图像算术运算

3. 2. 2 图像逻辑运算



3.2.1 图像算术运算

图像代数运算

(1) 加法运算

$$g(x, y) = f_1(x, y) + f_2(x, y)$$

■ 和值处理

$$\square \text{ 截断处理 } \begin{cases} g(x, y) = 255 & g(x, y) > 255 \\ g(x, y) = g(x, y) & \text{其他} \end{cases}$$

□ 加权求和

$$g(x, y) = \alpha f_1(x, y) + (1 - \alpha) f_2(x, y) \quad \alpha \in [0, 1]$$



3.2.1 图像算术运算

图像代数运算

(1) 加法运算

■ 主要应用

- 多幅图像相加求平均去除叠加性噪声
- 将一幅图像的内容经配准后叠加到另一幅图像上去，以改善图像的视觉效果。
- 在多光谱图像中，通过加法运算加宽波段，如绿色波段和红色波段图像相加可以得到近似全色图像。
- 用于图像合成和图像拼接。



3.2.1 图像算术运算

图像代数运算

(1) 加法运算

■ 例程

□ 函数 $Z = \text{imadd}(X, Y)$

□ 程序

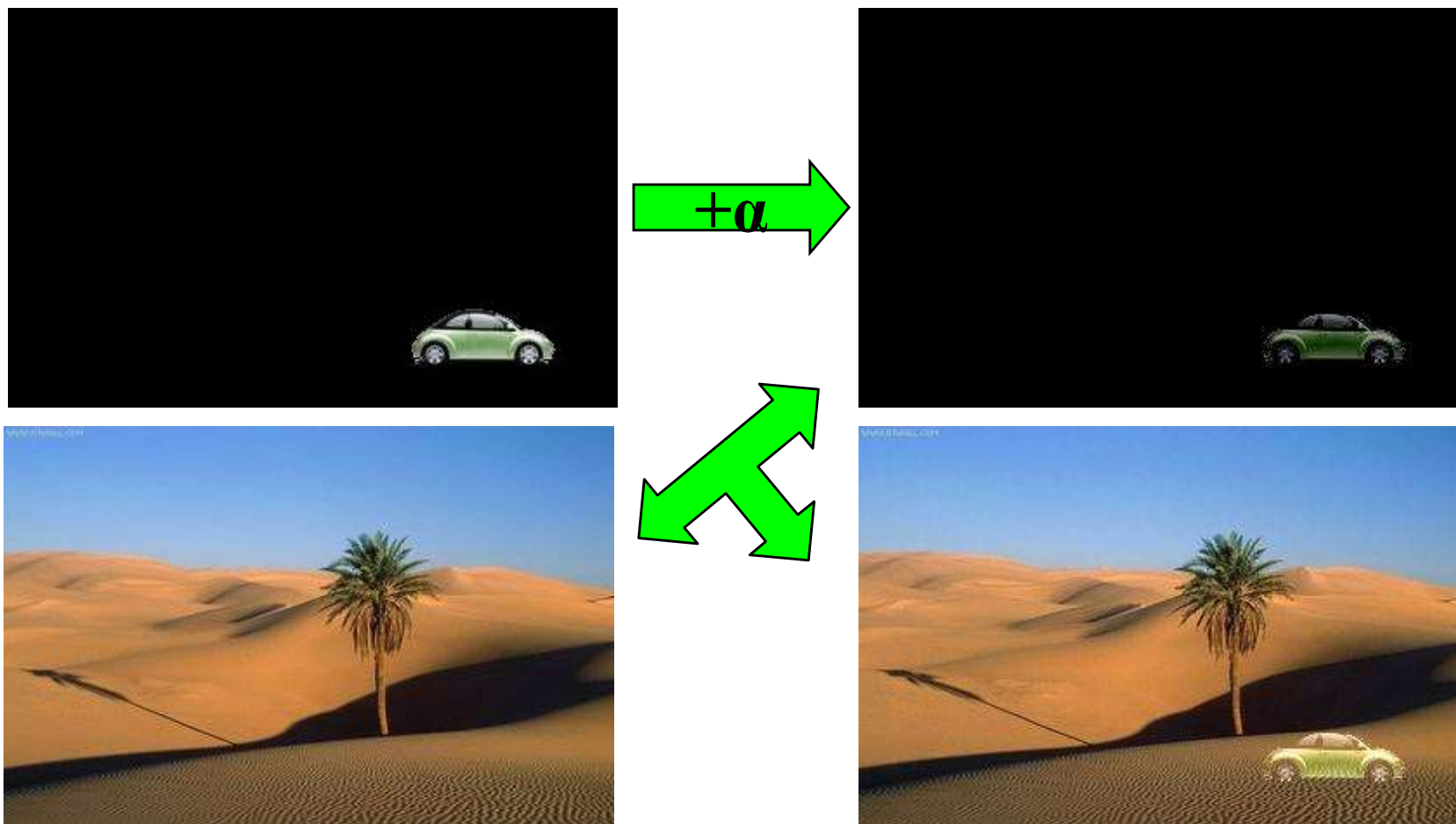
```
Back=imread('desert.jpg');  
Foreground=imread('car.jpg');  
result1=imadd(Foreground,-100);  
result2=imadd(Back,Foreground);  
result3=imadd(Back,result1);  
subplot(221),imshow(Foreground),title('原目标图');  
subplot(222),imshow(result1),title('原目标图加标量');  
subplot(223),imshow(result2),title('原目标图加背景');  
subplot(224),imshow(result3),title('加标量图叠加背景');
```

3.2.1 图像算术运算

图像代数运算

(1) 加法运算

□ 效果





3.2.1 图像算术运算

图像代数运算

(2) 减法运算

$$g(x, y) = f_1(x, y) - f_2(x, y)$$

■ 差值处理

$$\square \text{ 截断处理 } \begin{cases} g(x, y) = 0 & g(x, y) < 0 \\ g(x, y) = g(x, y) & \text{其他} \end{cases}$$

$$\square \text{ 取绝对值 } g(x, y) = |f_1(x, y) - f_2(x, y)|$$



3.2.1 图像算术运算

图像代数运算

(1) 减法运算

■ 主要应用

- 显示两幅图像的差异，检测同一场景两幅图像之间的变化。
- 去除不需要的叠加性图案，加性图案可能是缓慢变化的背景阴影或周期性的噪声，或在图像上每一个像素处均已知的附加污染等。
- 图像分割：如分割运动的车辆，减法去掉静止部分，剩余的是运动元素和噪声。
- 生成合成图像。



3.2.1 图像算术运算

图像代数运算

(2) 减法运算

■ 例程

□ 函数

`Z = imsubtract(X,Y)`

`Z = imabsdiff(X,Y)`

□ 程序

`Back=imread('hallback.bmp');`

`Foreground=imread('hallforeground.bmp');`

`result1=imabsdiff(Back,Foreground);`

`subplot(131),imshow(Back),title('背景图');`

`subplot(132),imshow(Foreground),title('前景图');`

`subplot(133),imshow(result1),title('图像相减');`

3.2.1 图像算术运算

图像代数运算

(2) 减法运算

□ 效果

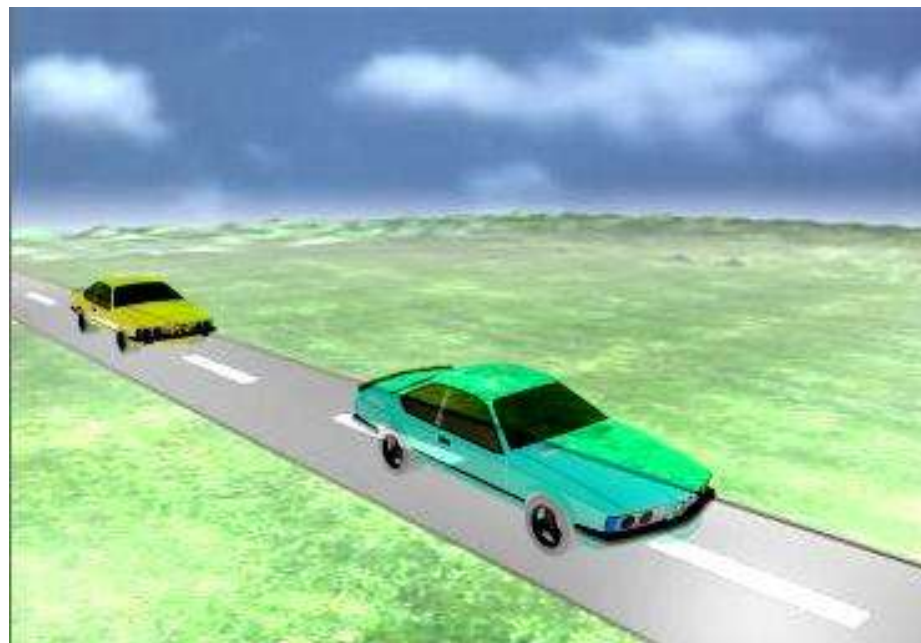
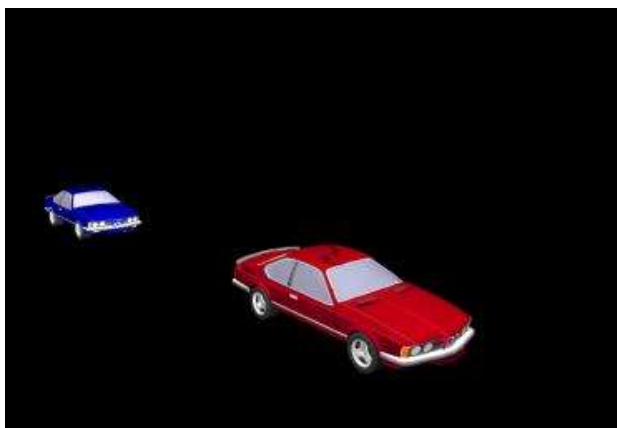


3.2.1 图像算术运算

图像代数运算

(2) 减法运算

□ 效果





3.2.1 图像算术运算

图像代数运算

(3) 乘法运算

$$g(x, y) = f_1(x, y) \times f_2(x, y)$$

■ 主要应用

- 图像的局部显示和提取：用二值模板图像与原图像做乘法来实现。
- 生成合成图像



3.2.1 图像算术运算

图像代数运算

(3) 乘法运算

■ 例程

□ 函数 $Z = \text{immultiply}(X, Y)$

□ 程序

```
Back=im2double(imread('bird.jpg'));  
Templet=im2double(imread('birdtemplet.bmp'));  
result1=immultiply(Templet,Back);  
subplot(131),imshow(Back),title('背景');  
subplot(132),imshow(Templet),title('模板');  
subplot(133),imshow(result1),title('图像相乘');
```

3.2.1 图像算术运算

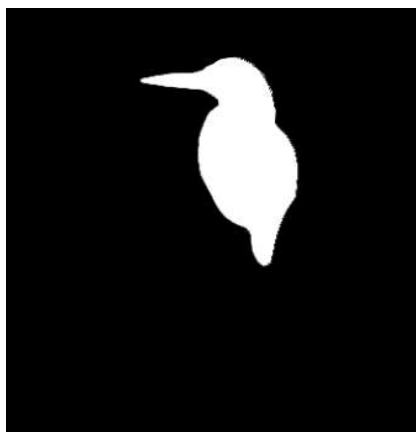
图像代数运算

(3) 乘法运算

□ 效果



×



=





3.2.1 图像算术运算

图像代数运算

(4) 除法运算

$$g(x, y) = f_1(x, y) \div f_2(x, y)$$

- 主要应用

- 用于消除空间可变的量化敏感函数、归一化、产生比率图像等

- 函数 $Z = \text{imdivide}(X, Y)$



3.2.2 图像逻辑运算

图像代数运算

■ 原理

非: $g(x, y) = 255 - f(x, y)$

与: $g(x, y) = f_1(x, y) \& f_2(x, y)$

或: $g(x, y) = f_1(x, y) | f_2(x, y)$

■ 例程

□ 函数

- ◆ $C = \text{bitcmp}(A)$: 按位求补
- ◆ $C = \text{bitand}(A, B)$: 按位求与
- ◆ $C = \text{bitor}(A, B)$: 按位求或
- ◆ $C = \text{bitxor}(A, B)$: 按位求异或
- ◆ $\&$ 、 $|$ 、 \sim : 按运算数据的值, 不按位



3.2.2 图像逻辑运算

图像代数运算

□ 程序

```
Back=imread('bird.jpg');  
Templet=imread('birdtemplet.bmp');  
result1=bitcmp(Back);  
result2=bitand(Templet,Back);  
result3=bitor(Templet,Back);  
result4=bitxor(Templet,Back);  
subplot(221),imshow(result1),title('求反');  
subplot(222),imshow(result2),title('相与');  
subplot(223),imshow(result3),title('相或');  
subplot(224),imshow(result4),title('异或');
```

3.2.2 图像逻辑运算

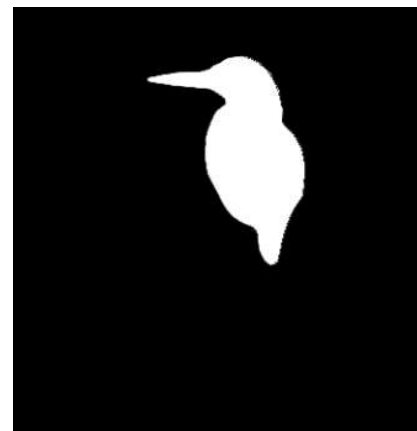
图像代数运算

□ 效果

原图



模板



求反



相与



相或



异或

3.2 图像代数运算



3.2 图像代数运算





3.3 邻域及模板运算

3.3.1 邻点及邻域

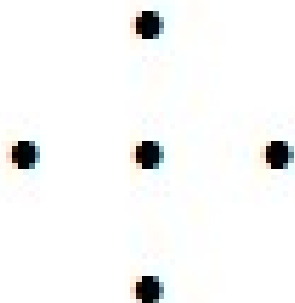
3.3.2 邻域处理与模板运算



3.3.1 邻点及邻域

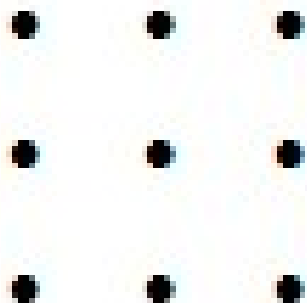
邻域及模板运算

图像中相邻的像素构成邻域，互为邻点



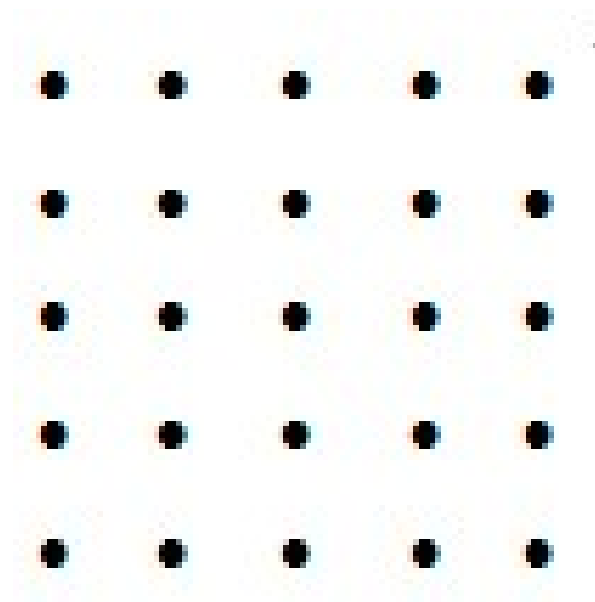
4邻点

(3×3 邻域)



8邻点

(3×3 邻域)

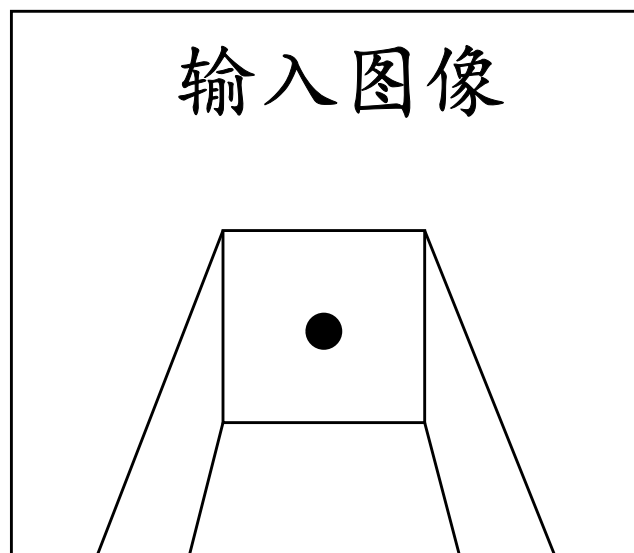


24邻点

(5×5 邻域)

3.3.2 邻域处理与模板运算

邻域及模板运算



$$I = \begin{bmatrix} 0 & 150 & 200 \\ 120 & 50 & 180 \\ 250 & 220 & 100 \end{bmatrix}$$

3×3邻域

$$\begin{bmatrix} h1 & h2 & h3 \\ h4 & h5 & h6 \\ h7 & h8 & h9 \end{bmatrix}$$

3×3模板

加权和计算，得
 $p5$ 的新值：

$$\begin{aligned} & h1 \times p1 + h2 \times p2 + \\ & h3 \times p3 + h4 \times p4 + \\ = & h5 \times p5 + h6 \times p6 + \\ & h7 \times p7 + h8 \times p8 + \\ & h9 \times p9 \end{aligned}$$



3.3.2邻域处理与模板运算

邻域及模板运算

- 模板运算中边缘像素处理
 - 保留该区域中原始像素灰度值不变。
 - 在图像边缘以外补充行列，取值设为零或复制边缘像素灰度值。补充的行列在新图像中要去掉。



3.4综合实例

有一幅蝴蝶、一幅风景图片，基于几何、代数和色彩通道运算，编程实现漫天蝴蝶飞舞的合成图像。

(1) 设计思路

对蝴蝶图片进行随机变换后叠加到风景图片上，依次进行的随机变换为：**三种几何变换、交换两个色彩通道、叠加**到风景图片随机位置上。

3.4综合实例

(2) 设计效果

