

# Lecture 4

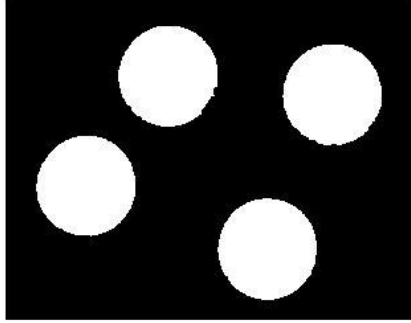
Image Fundamentals III

# Outline

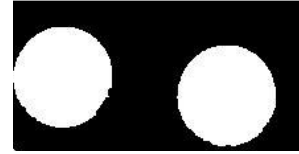
Regions

Distance Measures

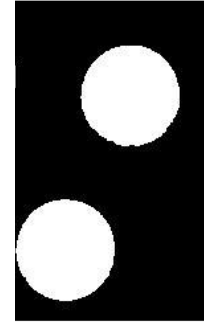
Useful Mathematical Tools



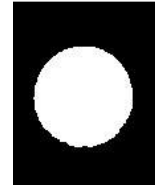
4 Connected components in  
the subset Image (only 1s)  
itself



2 connected components in  
subset S1 (only 1s)



2 connected components in  
subset S2 (only 1s)

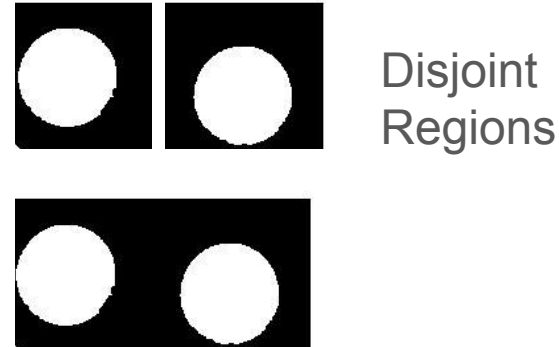


1 connected component in  
subset S3 (only 1s)

If there is only 1 connected component in a subset of 1s, that subset is called a connected set or a region

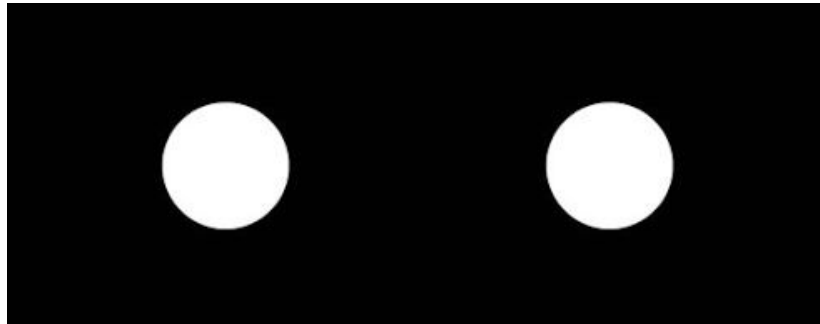
# Adjacent Regions

Two regions,  $R_i$  and  $R_j$  are said to be *adjacent* if their union forms a connected set. Regions that are not adjacent are said to be *disjoint*. We consider 4- and 8-adjacency when referring to regions.



# Foreground & Background

Suppose an image contains  $K$  disjoint regions,  $R_k, k = 1, 2, \dots, K$ , none of which touches the image border.<sup>†</sup> Let  $R_u$  denote the union of all the  $K$  regions, and let  $(R_u)^c$  denote its complement (recall that the *complement* of a set  $A$  is the set of points that are not in  $A$ ). We call all the points in  $R_u$  the *foreground*, and all the points in  $(R_u)^c$  the *background* of the image.



2 disjoint regions  
forming the single  
foreground;  
everything else  
forms the  
background

# Boundary

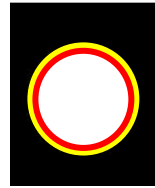
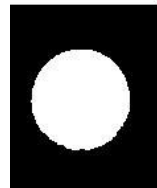
The *boundary* (also called the *border* or *contour*) of a region  $R$  is the set of pixels in  $R$  that are adjacent to pixels in the complement of  $R$ . Stated another way, the border of a region is the set of pixels in the region that have at least one background neighbor. Here again, we must specify the connectivity being used to define adjacency. For example, the point circled in Fig below is not a member of the border of the 1-valued region if 4-connectivity is used between the region and its background, because the only possible connection between that point and the background is diagonal. As a rule, adjacency between points in a region and its background is defined using 8-connectivity to handle situations such as this.



0	0	0	0	0
0	1	1	0	0
0	1	1	0	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

# Inner & Outer Boundary

The previous definition sometimes is referred to as the *inner border* of the region to distinguish it from its *outer border*, which is the corresponding border in the background. This distinction is important in the development of border-following algorithms. Such algorithms usually are formulated to follow the outer boundary in order to guarantee that the result will form a closed path. For instance, the inner border of the 1-valued region in Fig. below is the region itself. This border does not satisfy the definition of a closed path. On the other hand, the outer border of the region does form a closed path around the region.



Inner Boundary  
Outer Boundary

0	0	0
0	1	0
0	1	0
0	1	0
0	1	0
0	0	0



# Boundary (misc.)

If it's the entire image, then its *boundary* (or *border*) is defined as the set of pixels in the first and last rows and columns of the image. This extra definition is required because an image has no neighbors beyond its border. Normally, when we refer to a region, we are referring to a subset of an image, and any pixels in the boundary of the region that happen to coincide with the border of the image are included implicitly as part of the region boundary.

The boundary of a finite region forms a closed path and is thus a “global” concept.



Edges are formed from pixels with derivative values that exceed a preset threshold. Thus, an edge is a “local” concept that is based on a measure of intensity-level discontinuity at a point.

Generally, they are different. However, the one exception in which edges and boundaries correspond is in binary images.



# Distance Measures

For pixels  $p$ ,  $q$ , and  $s$ , with coordinates  $(x, y)$ ,  $(u, v)$ , and  $(w, z)$ , respectively,  $D$  is a *distance function* or *metric* if

**(a)**  $D(p, q) \geq 0$  ( $D(p, q) = 0$  iff  $p = q$ ),

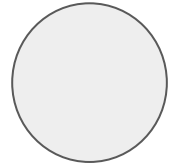
**(b)**  $D(p, q) = D(q, p)$ , and

**(c)**  $D(p, s) \leq D(p, q) + D(q, s)$ .

## Distance Measures (contd..)

The *Euclidean distance* between  $p$  and  $q$  is defined as

$$D_e(p, q) = \left[ (x - u)^2 + (y - v)^2 \right]^{\frac{1}{2}}$$



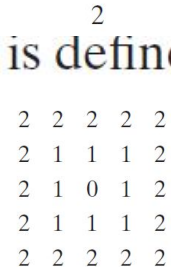
The  $D_4$  distance, (called the *city-block distance*) between  $p$  and  $q$  is defined as

$$D_4(p, q) = |x - u| + |y - v|$$



The  $D_8$  distance (called the *chessboard distance*) between  $p$  and  $q$  is defined as

$$D_8(p, q) = \max(|x - u|, |y - v|)$$



Refer to `dist_demox.m` to see how these shapes can be made

# Elementwise vs Matrix Multiplications

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix}$$

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

Unless specified, consider multiplication means element-wise multiplication

# Linear vs Non-linear Operations

Given two arbitrary constants,  $a$  and  $b$ , and two arbitrary images  $f_1(x, y)$  and  $f_2(x, y)$ .  $\mathcal{H}$  is said to be a *linear operator* if

$$\mathcal{H}[af_1(x, y) + bf_2(x, y)] = a\mathcal{H}[f_1(x, y)] + b\mathcal{H}[f_2(x, y)]$$

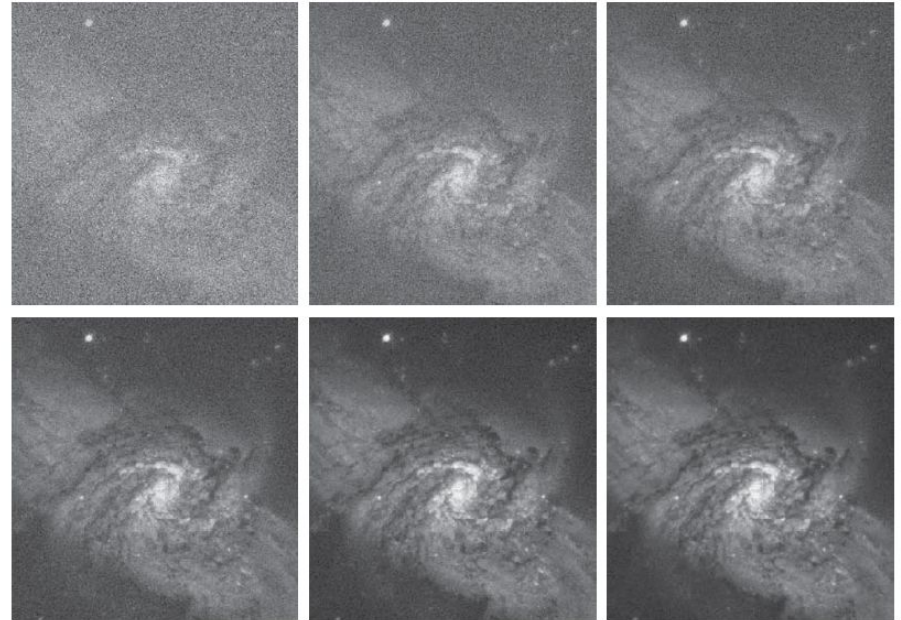
Example of linear operation: `sum()`

Example of non-linear operation: `max()`

## Noise Reduction via Image Averaging

Assumption:

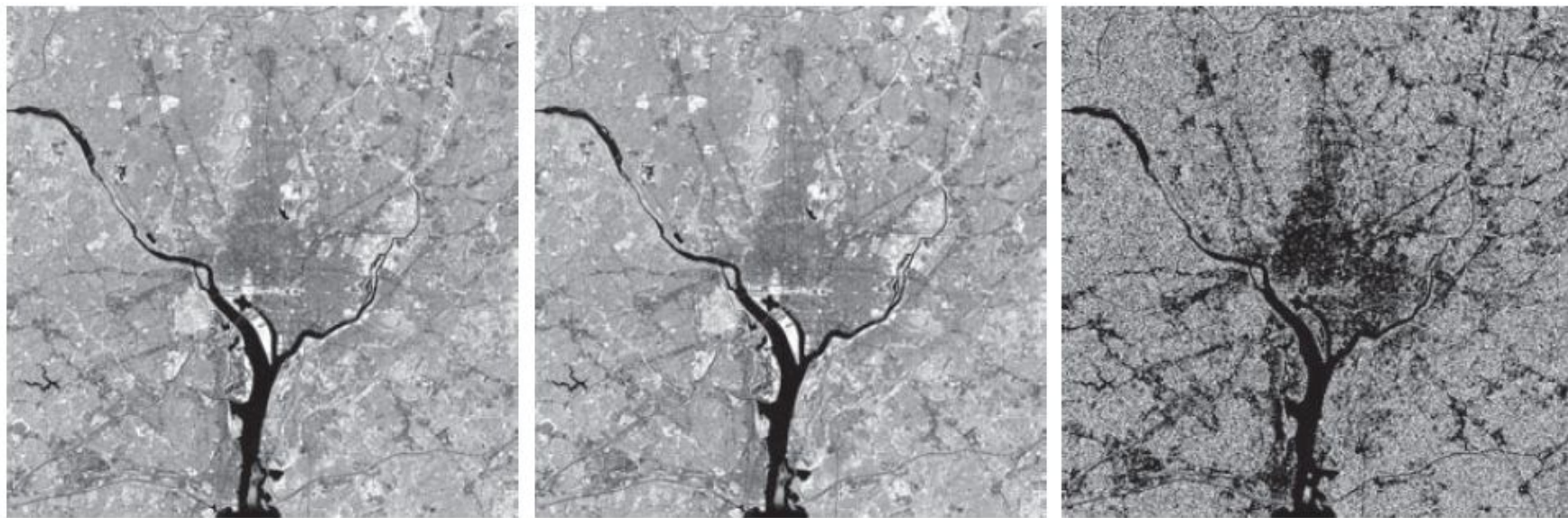
- (i) Additive Gaussian Noise
- (ii) Average of Noise = 0
- (iii) Noise and Image are uncorrelated



a b c  
d e f

**FIGURE 2.29** (a) Image of Galaxy Pair NGC 3314 corrupted by additive Gaussian noise. (b)-(f) Result of averaging 5, 10, 20, 50, and 1,000 noisy images, respectively. All images are of size  $566 \times 598$  pixels, and all were scaled so that their intensities would span the full  $[0, 255]$  intensity scale. (Original image courtesy of NASA.)

# Image Difference



a b c

**FIGURE 2.30** (a) Infrared image of the Washington, D.C. area. (b) Image resulting from setting to zero the least significant bit of every pixel in (a). (c) Difference of the two images, scaled to the range  $[0, 255]$  for clarity. (Original image courtesy of NASA.)



# Image Multiplication



a b c

**FIGURE 2.34** (a) Digital dental X-ray image. (b) ROI mask for isolating teeth with fillings (white corresponds to 1 and black corresponds to 0). (c) Product of (a) and (b).



# Image Division



a b c

**FIGURE 2.33** Shading correction. (a) Shaded test pattern. (b) Estimated shading pattern. (c) Product of (a) by the reciprocal of (b). (See Section 3.5 for a discussion of how (b) was estimated.)

# Background Subtraction

Compute the average frame of the video to calculate the background.

Subtract the foreground and generate a suitable mask

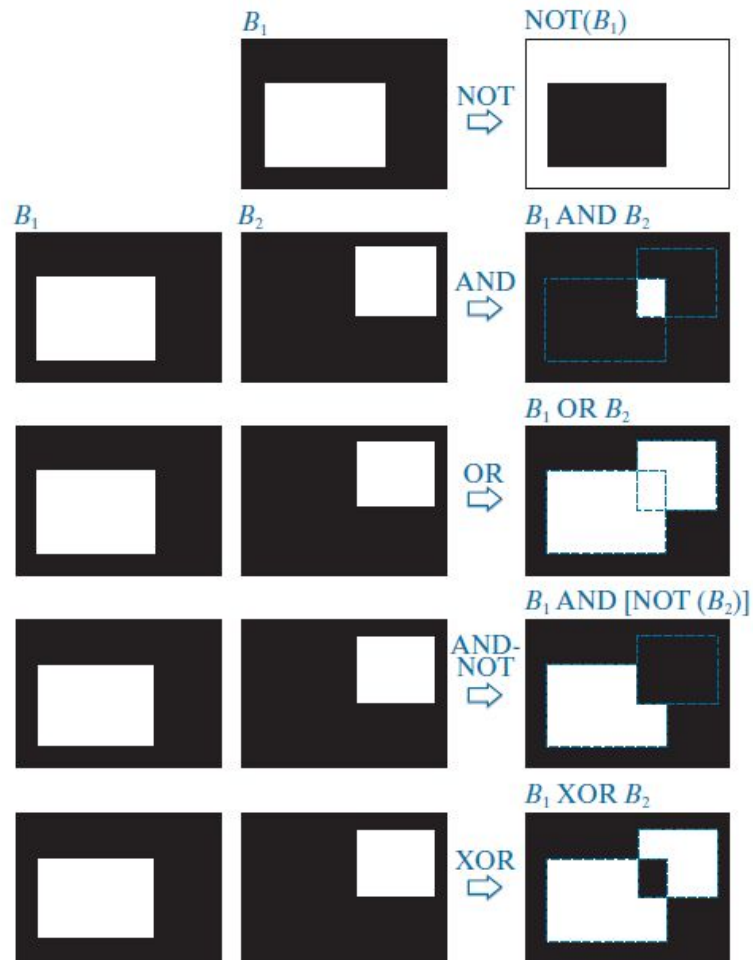
Multiply the mask with the frame

Refer to vidpro.m for  
a demo



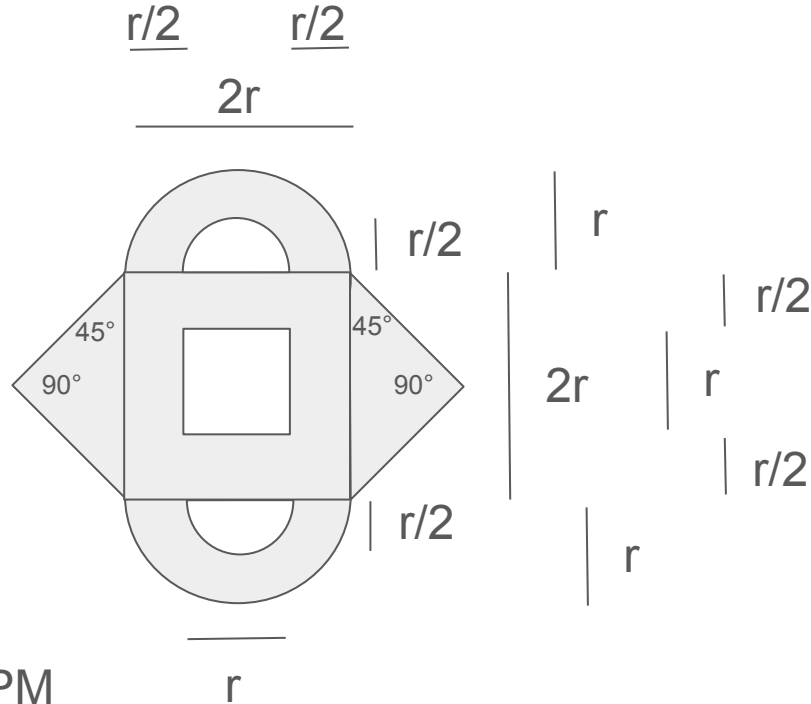
**FIGURE 2.37**

Illustration of logical operations involving foreground (white) pixels. Black represents binary 0's and white binary 1's. The dashed lines are shown for reference only. They are not part of the result.



# Homework 2

Write a program to create a binary image with the following shape as its foreground:



Deadline: Aug 14, 11:59 PM

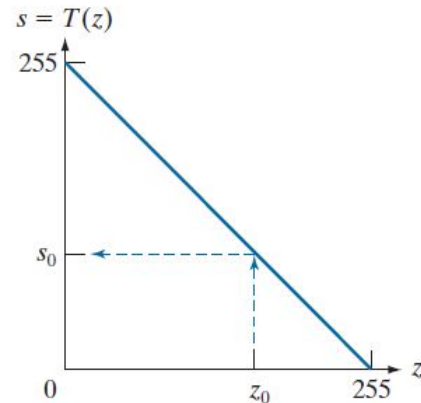
## Single-Pixel Operations

The simplest operation we perform on a digital image is to alter the intensity of its pixels individually using a transformation function,  $T$ , of the form:

$$s = T(z) \quad (2-42)$$

where  $z$  is the intensity of a pixel in the original image and  $s$  is the (mapped) intensity of the corresponding pixel in the processed image. For example, Fig. 2.38 shows the transformation used to obtain the *negative* (sometimes called the *complement*)

**FIGURE 2.38**  
Intensity transformation function used to obtain the digital equivalent of photographic negative of an 8-bit image..



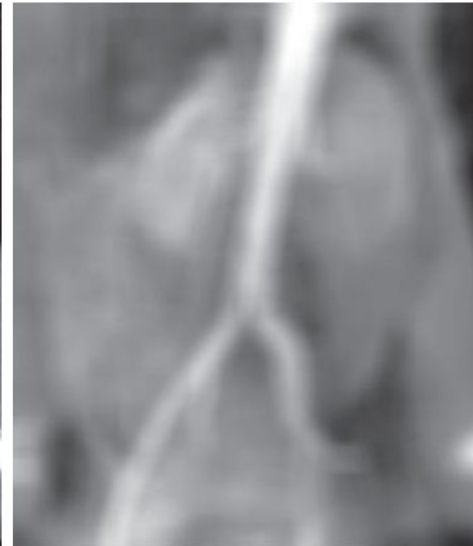
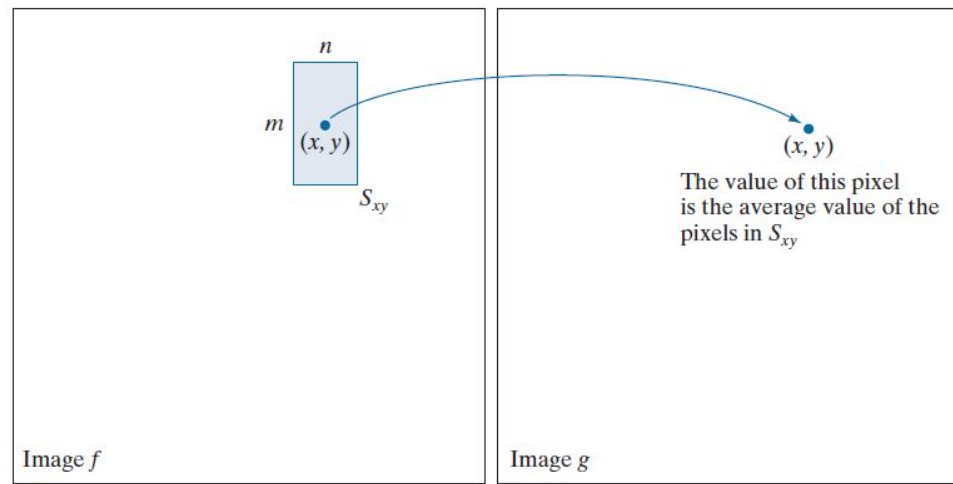
## Neighborhood Operations

Let  $S_{xy}$  denote the set of coordinates of a neighborhood

a rectangular neighborhood of size  $m \times n$  centered on  $(x, y)$ .

We can express this averaging operation as

$$g(x, y) = \frac{1}{mn} \sum_{(r, c) \in S_{xy}} f(r, c)$$



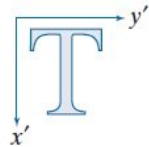
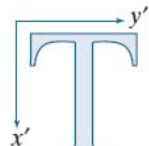
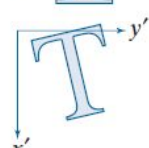
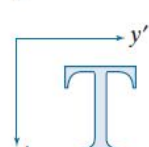
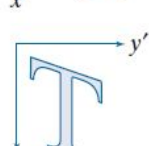
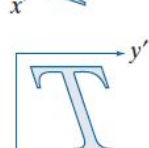


# Affine Transformations

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{A} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2-45)$$

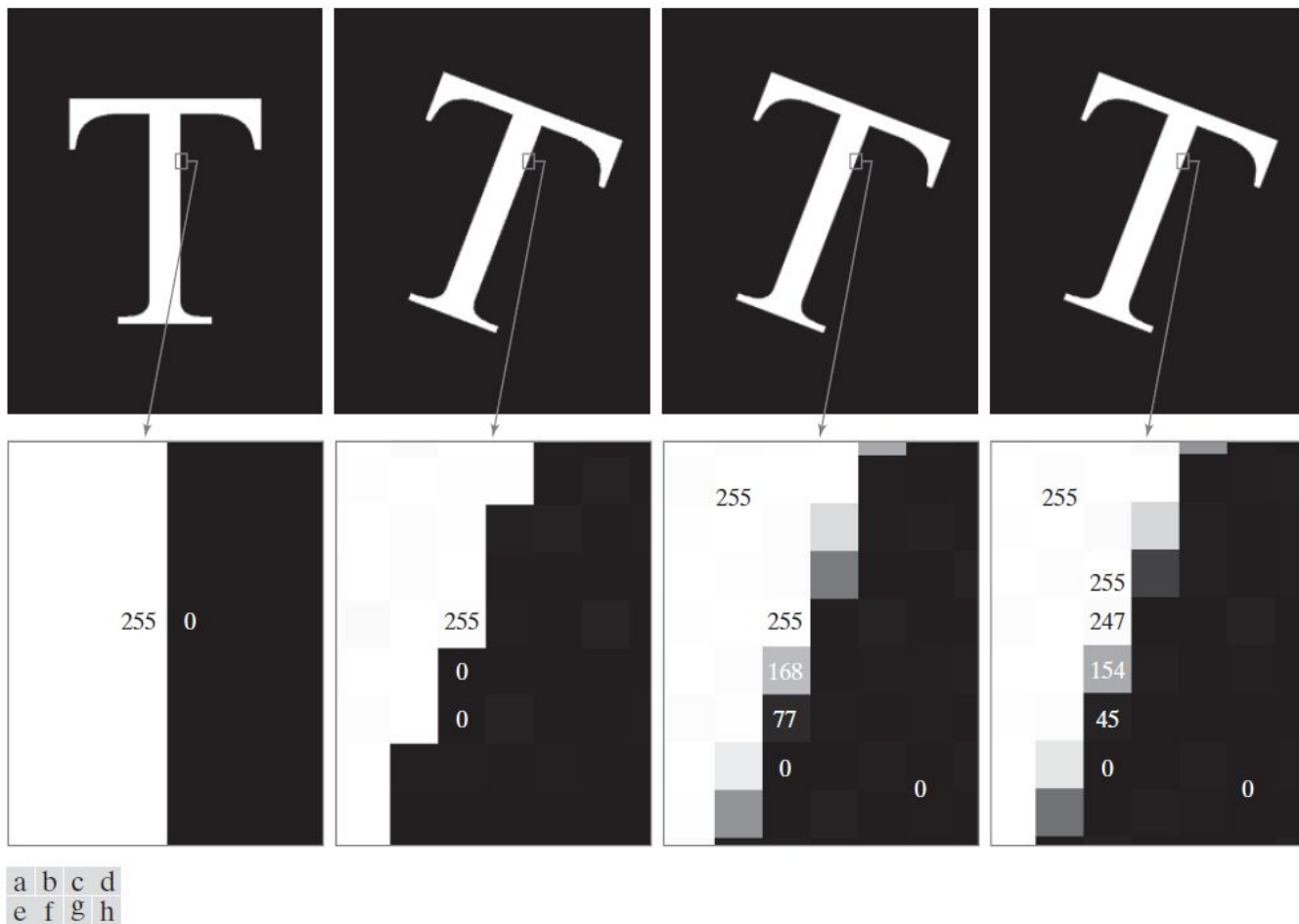
This transformation can *scale*, *rotate*, *translate*, or *sheer* an image, depending on the values chosen for the elements of matrix  $\mathbf{A}$ .

**TABLE 2.3**  
Affine  
transformations  
based on  
Eq. (2-45).

Transformation Name	Affine Matrix, A	Coordinate Equations	Example
Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= x \\ y' &= y \end{aligned}$	
Scaling/Reflection (For reflection, set one scaling factor to -1 and the other to 0)	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= c_x x \\ y' &= c_y y \end{aligned}$	
Rotation (about the origin)	$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= x \cos \theta - y \sin \theta \\ y' &= x \sin \theta + y \cos \theta \end{aligned}$	
Translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned}$	
Shear (vertical)	$\begin{bmatrix} 1 & s_v & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= x + s_v y \\ y' &= y \end{aligned}$	
Shear (horizontal)	$\begin{bmatrix} 1 & 0 & 0 \\ s_h & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{aligned} x' &= x \\ y' &= s_h x + y \end{aligned}$	

Refer to shared trans.m to see how interpolation can be done.

NOTE: We need to always perform reverse operation to go to the original image for getting the nearest neighbors



(a) A  $541 \times 421$  image of the letter T. (b) Image rotated  $-21^\circ$  using nearest-neighbor interpolation for intensity assignments. (c) Image rotated  $-21^\circ$  using bilinear interpolation. (d) Image rotated  $-21^\circ$  using bicubic interpolation. (e)-(h) Zoomed sections (each square is one pixel, and the numbers shown are intensity values).

# Image Registration

In image registration, we have an input image and a reference image. The objective is to transform the input image geometrically to produce an output image that is aligned (registered) with the reference image.

$$x = c_1v + c_2w + c_3vw + c_4$$

$$y = c_5v + c_6w + c_7vw + c_8$$

(v,w) and (x, y) are the coordinates of tie points in the input and reference images, respectively. Get the coefficients by solving 8 equations obtained using 4 tie points

**FIGURE 2.42**  
Image registration.  
(a) Reference image. (b) Input (geometrically distorted image). Corresponding tie points are shown as small white squares near the corners.  
(c) Registered (output) image (note the errors in the border).  
(d) Difference between (a) and (c), showing more registration errors.

