


# Lecture #3

- MATLAB® image processing (cont.)
  - vectorization
- Histograms
- Mathematics of image processing
- Geometric transforms
- Image Warping

# Pixel Indexing in MATLAB

"IP_Function" is some arbitrary image processing function that you or someone else has written.

"For" loops in Matlab are inefficient, whereas Matlab's native indexing procedures are very fast.

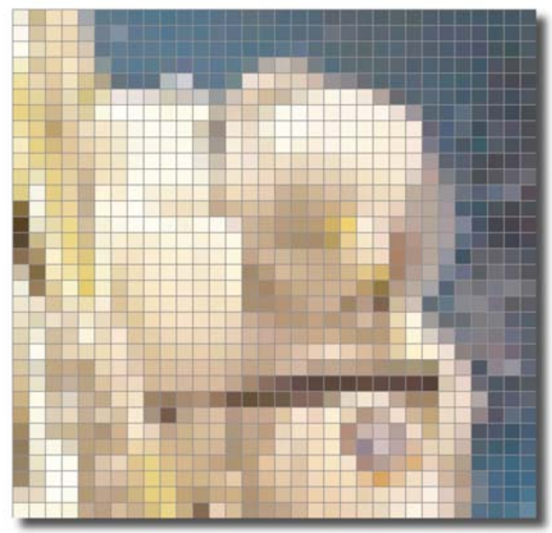
Rather than

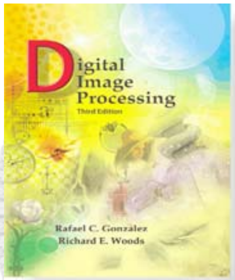
```
for r = 1:R
   for c = 1:C
      J(r,c,:) = IP_Function(I(r,c,:));
   end
end
```

use, if possible

```
J = IP_Function(I);
```

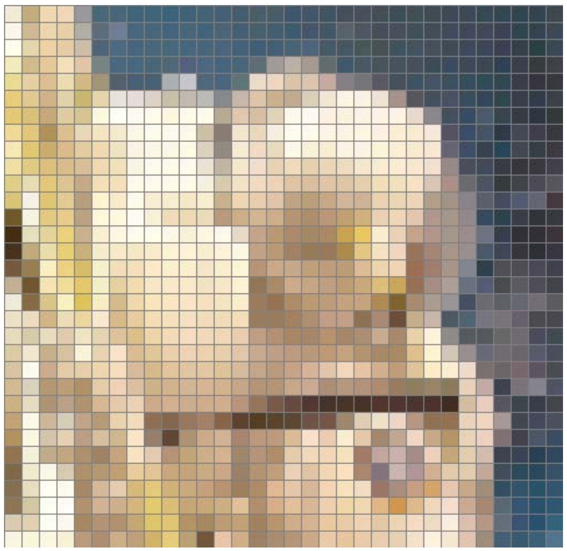
But, sometimes that is not possible.
For example, if the output, *J*, is decimated with respect to the input, *I*, the above will not work (unless, of course, it is done within IP_function).



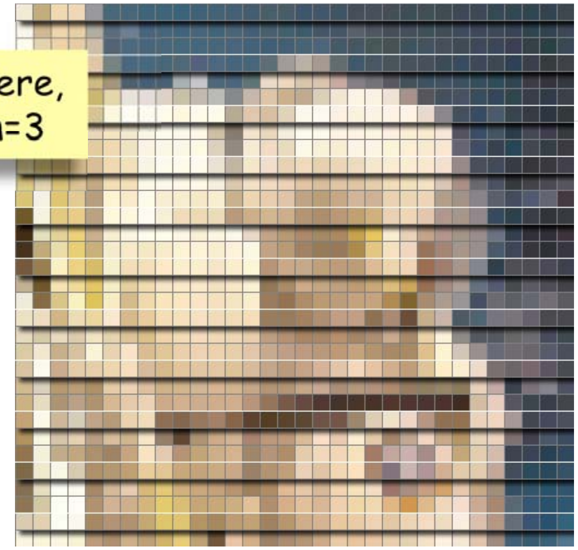



# Pixel Indexing in M

Here, n=3

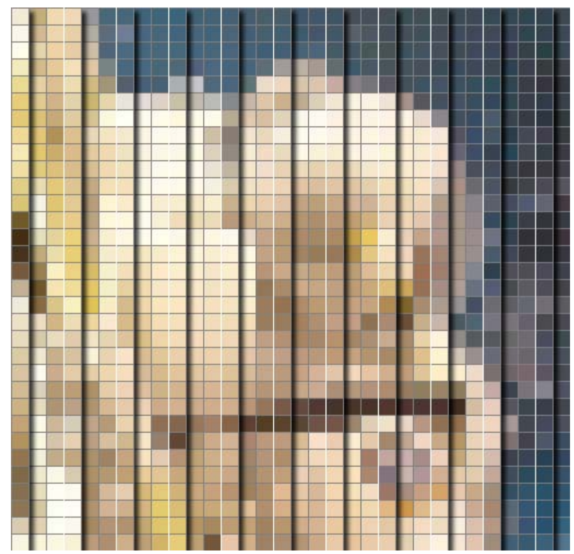


```
r = 1:n:R;
I(r,:,:)
```



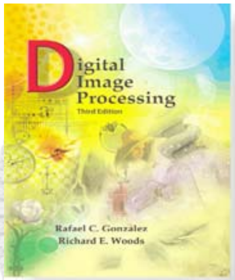
r = [1 4 7 10 13 16 19 22 25 28 31]

```
c = 1:n:C;
I(:,c,:)
```



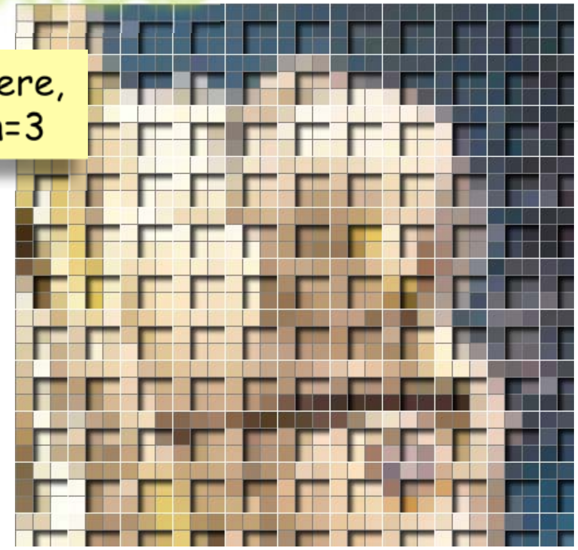
c = [1 4 7 10 13 16 19 22 25 28 31]

To decimate the above image by a factor of *n*, create a vector, **r**, that contains the index of every *n*th row, and a similar vector, **c**.

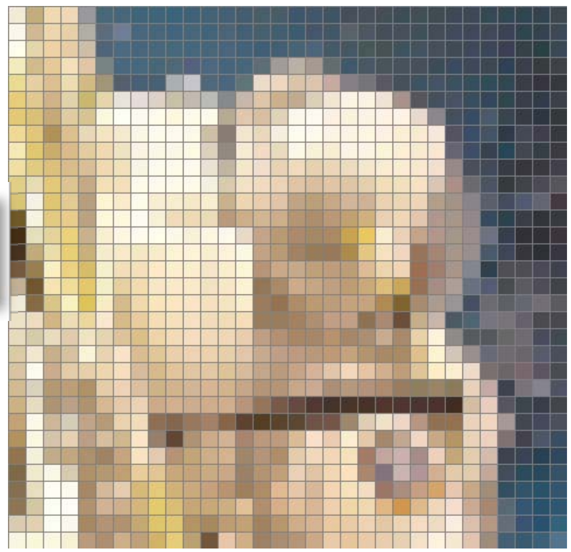


# Pixel Indexing in MATLAB

Here, n=3

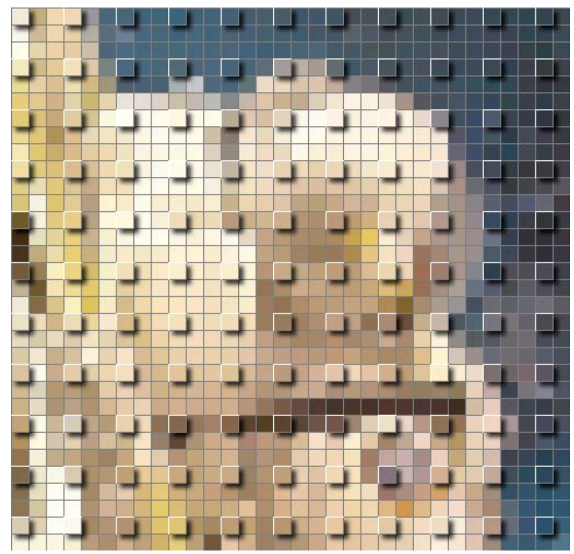


This is called, 'vectorizing'.



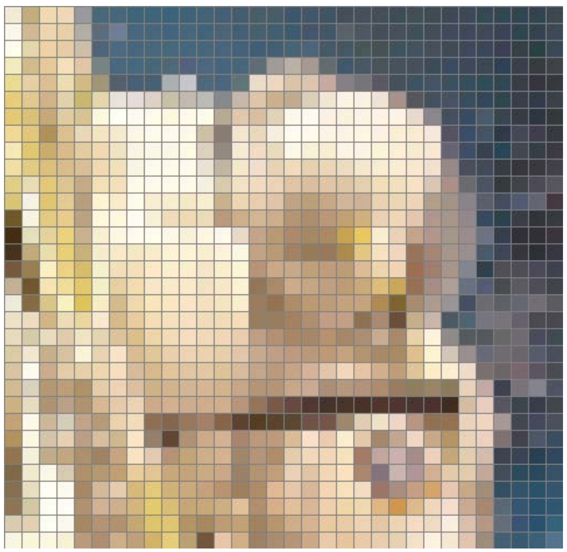
Take the pixels indexed by both $r$ and $c$.

`I(r,c)`



Then, vectors **r** and **c** used as index arguments for image *I* select every *n*th column in every *n*th row.

# Pixel Indexing in MATLAB

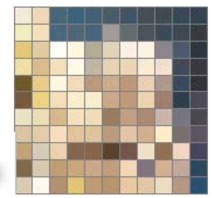


image, `I`

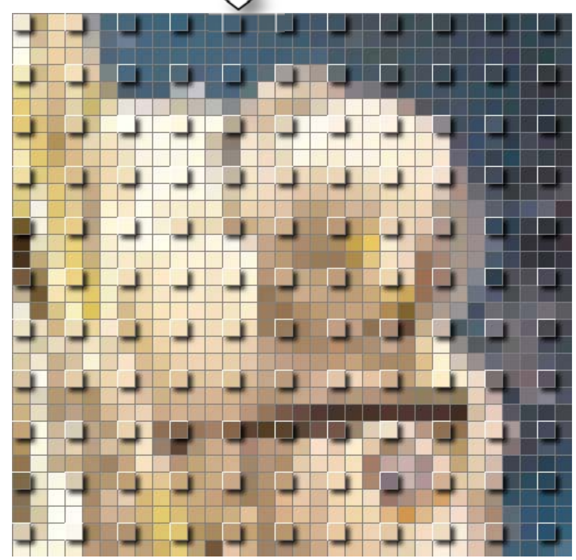
```
r = 1:n:R;
```

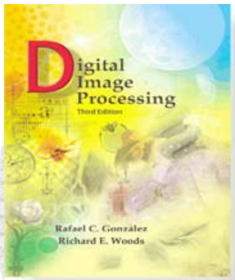
```
c = 1:n:C;
```

Here, n=3



```
J = I(r,c,:);
```






# Pixel Indexing in MATLAB

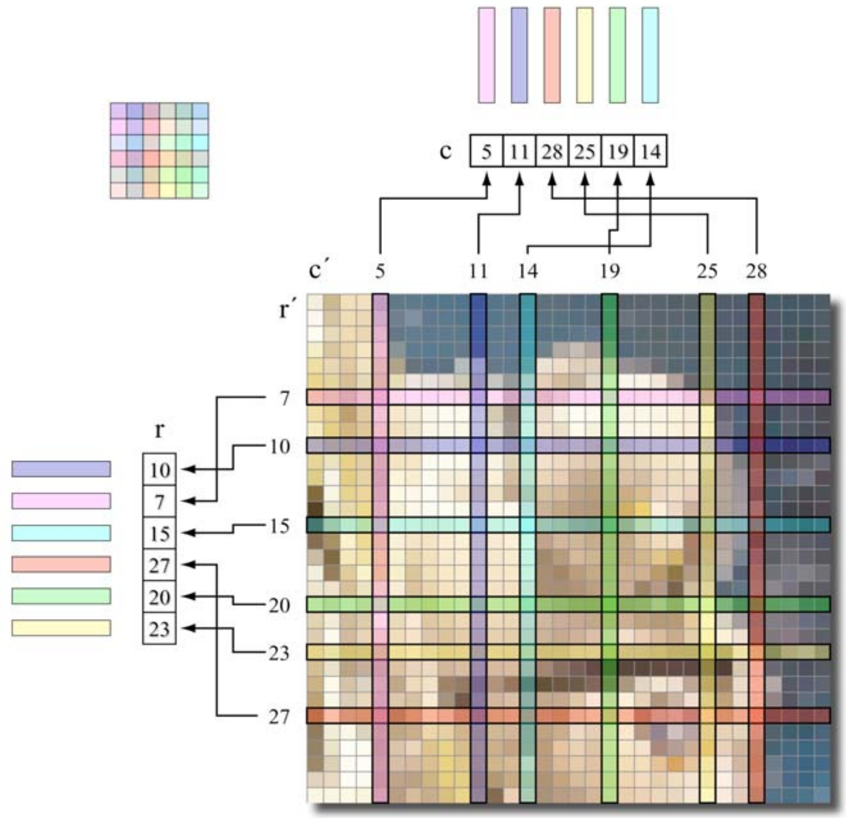
Indexing in Matlab is fully general.

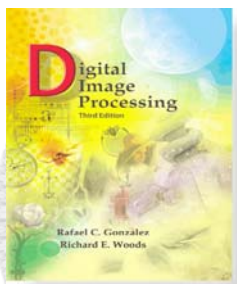
If `I` is $R$ x $C$ x $B$, vectors **r** and **c** can contain any numbers $1 \le r_k \le R$ and $1 \le c_k \le C$.

The numbers can be in any order and can be repeated within **r** and **c.**

The result of `I(r,c)` is an ordinal shuffling of the pixels from I as indexed by **r** and **c**.

Whenever possible, avoid using 'for' loops; vectorize instead.







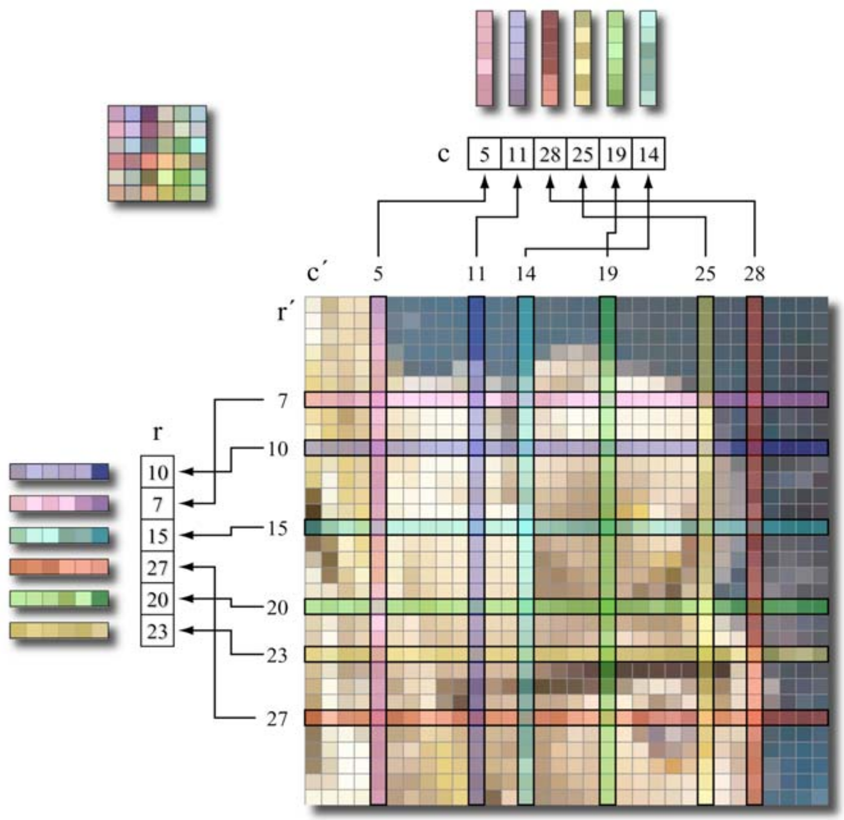

# Pixel Indexing in MATLAB

Indexing in Matlab is fully general.

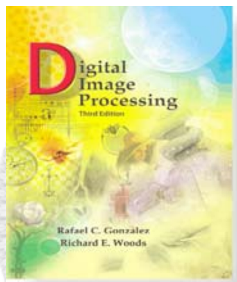
If `I` is $R$ x $C$ x $B$, vectors **r** and **c** can contain any numbers $1 \le r_k \le R$ and $1 \le c_k \le C$.

The numbers can be in any order and can be repeated within **r** and **c.**

The result of `I(r,c)` is an ordinal shuffling of the pixels from I as indexed by **r** and **c**.




Whenever possible, avoid using 'for' loops; vectorize instead.






# Pixel Indexing in MATLAB

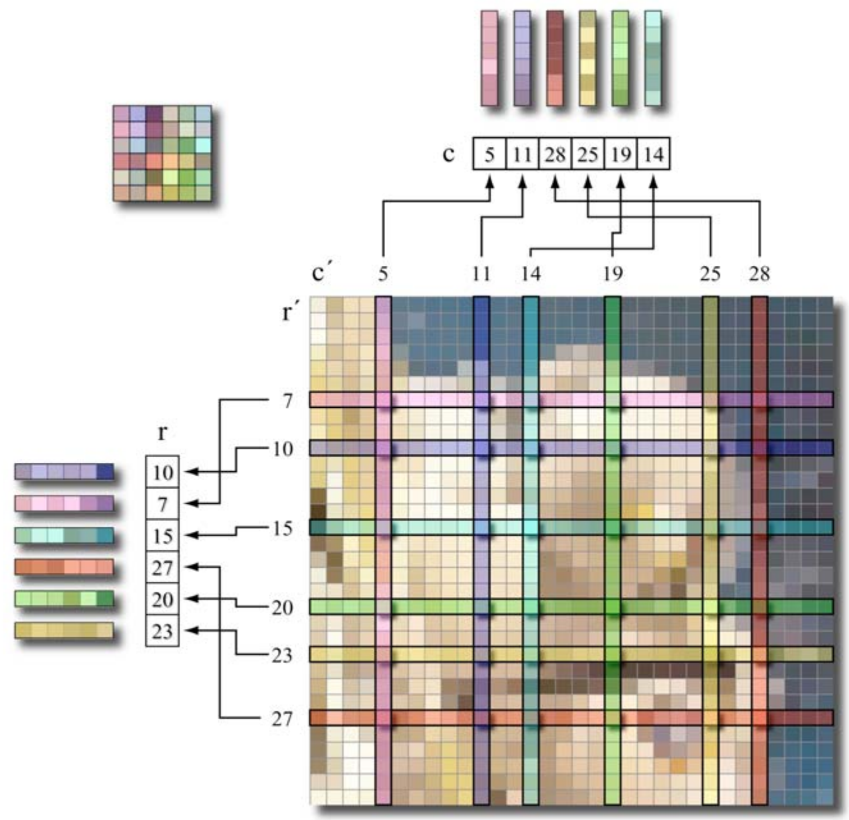
Indexing in Matlab is fully general.

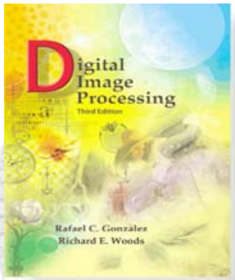
If `I` is $R$ x $C$ x $B$, vectors **r** and **c** can contain any numbers $1 \leq r_k \leq R$ and $1 \leq c_k \leq C$.

The numbers can be in any order and can be repeated within **r** and **c.**

The result of `I(r,c)` is an ordinal shuffling of the pixels from I as indexed by **r** and **c**.

Whenever possible, avoid using 'for' loops; vectorize instead.







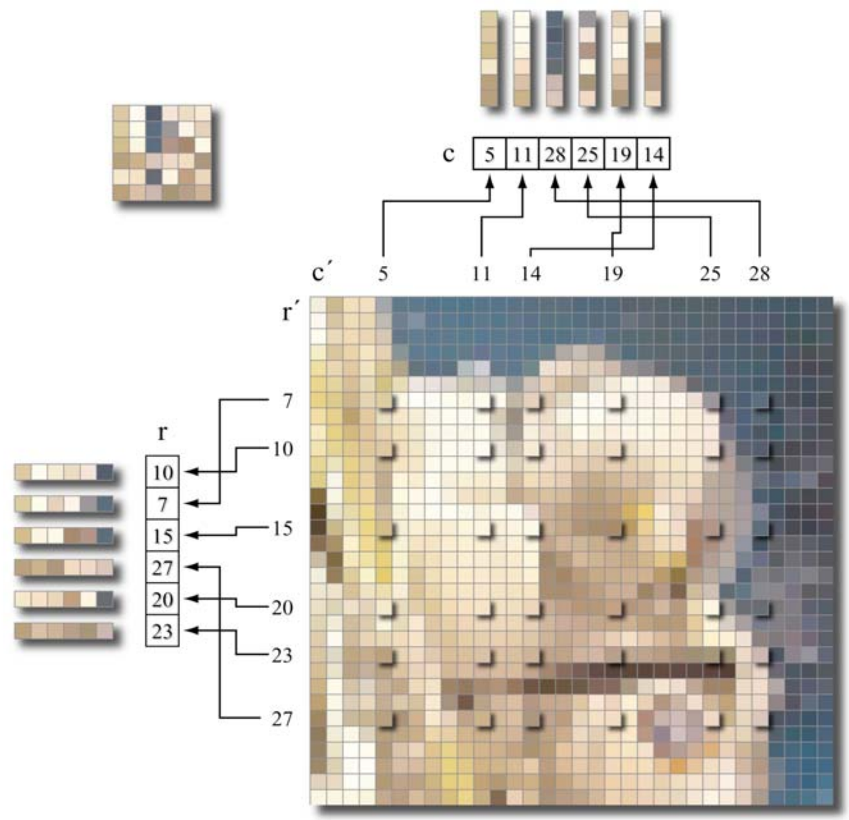

# Pixel Indexing in MATLAB

Indexing in Matlab is fully general.

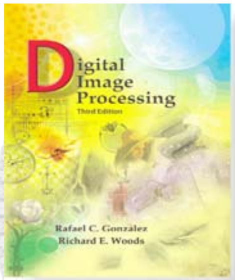
If `I` is $R$ x $C$ x $B$, vectors **r** and **c** can contain any numbers $1 \le r_k \le R$ and $1 \le c_k \le C$.

The numbers can be in any order and can be repeated within **r** and **c.**

The result of `I(r,c)` is an ordinal shuffling of the pixels from I as indexed by **r** and **c**.




Whenever possible, avoid using 'for' loops; vectorize instead.





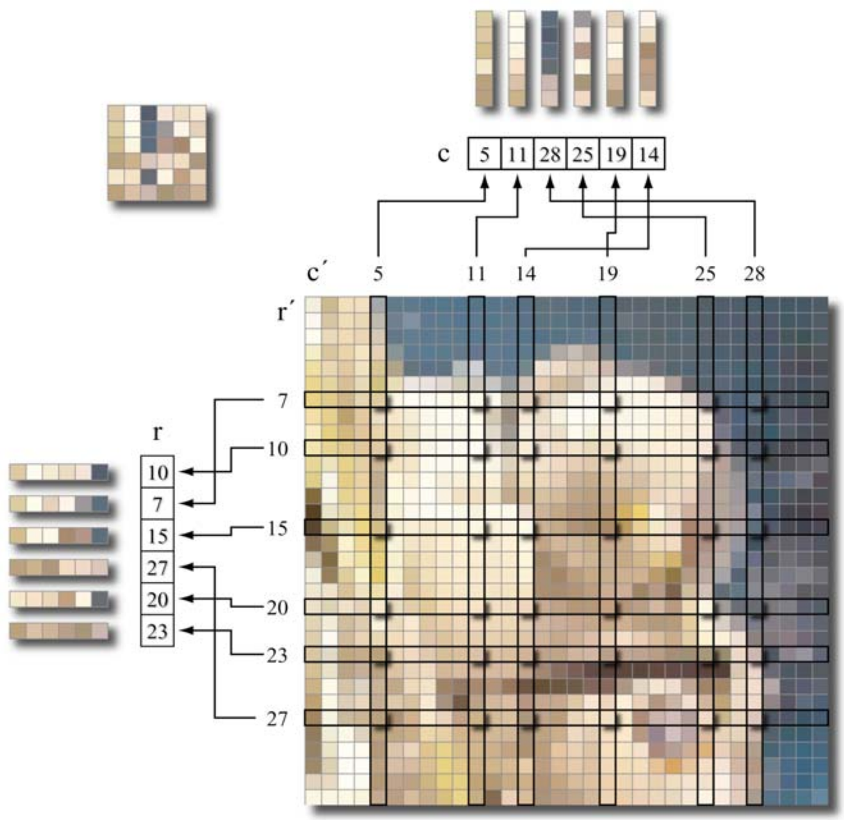

# Pixel Indexing in MATLAB

Indexing in Matlab is fully general.

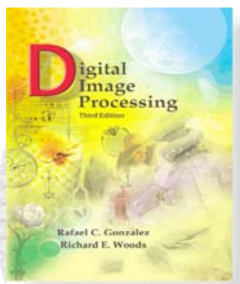
If `I` is $R$ x $C$ x $B$, vectors **r** and **c** can contain any numbers $1 \leq r_k \leq R$ and $1 \leq c_k \leq C$.

The numbers can be in any order and can be repeated within **r** and **c.**

The result of `I(r,c)` is an ordinal shuffling of the pixels from I as indexed by **r** and **c**.




Whenever possible, avoid using 'for' loops; vectorize instead.





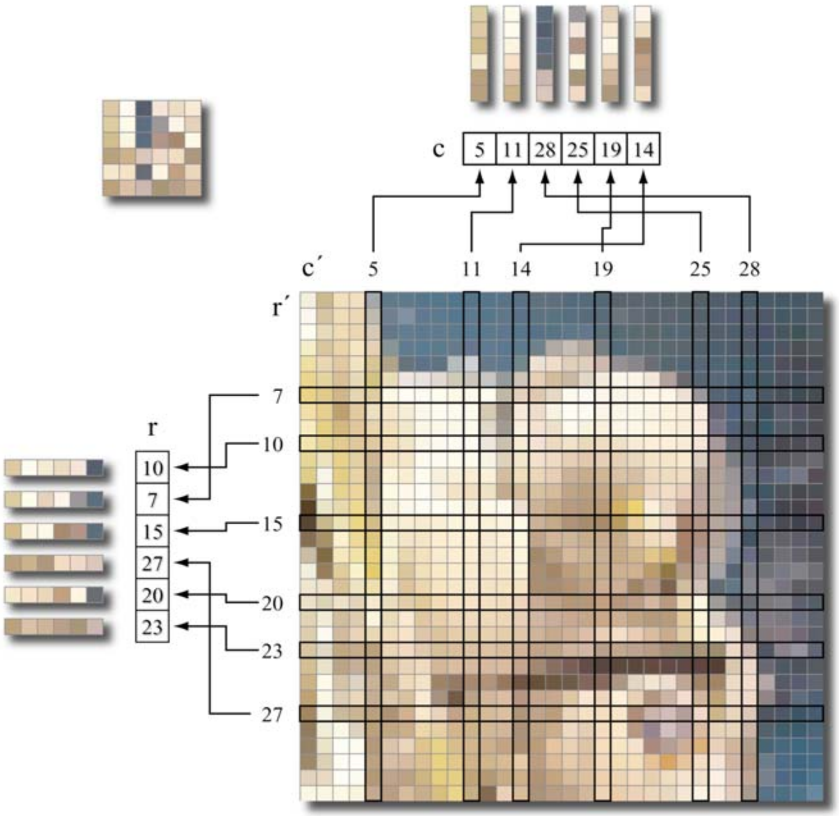
# Pixel Indexing in MATLAB

Indexing in Matlab is fully general.

If `I` is $R$ x $C$ x $B$, vectors **r** and **c** can contain any numbers $1 \leq r_k \leq R$ and $1 \leq c_k \leq C$.

The numbers can be in any order and can be repeated within **r** and **c.**

The result of `I(r,c)` is an ordinal shuffling of the pixels from I as indexed by **r** and **c**.



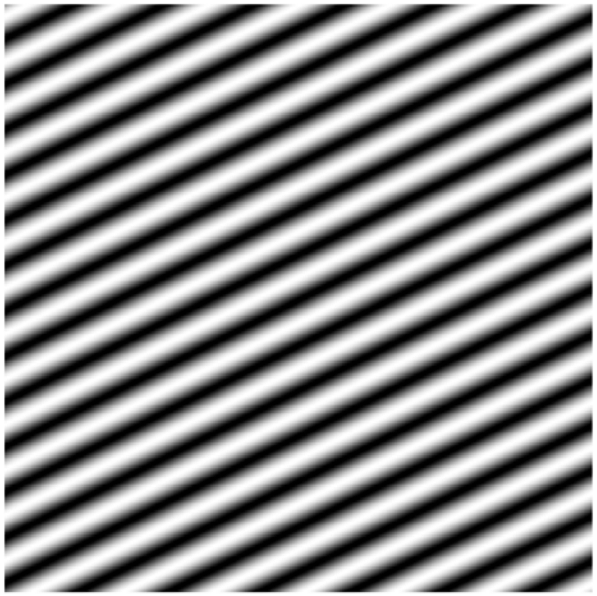

Whenever possible, avoid using 'for' loops; vectorize instead.

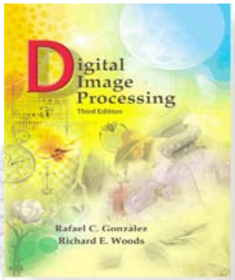
# MATLAB/Image Processing Toolbox

```
function [rt, f, g] = twodsin(A,u0, v0, M, N)
% TWODSIN Compares for loops vs. vectorization.
% The comparison is based on implementing the function
% f(x,y)=Asin(u0x+v0y) for x=0,1,2,…,M-1 and
% y=0,1,2,…,N-1.  The inputs to the function are
% M and N and the constants in the function.

% First implement using for loops
tic %start timing
for r=1:M
   u0x=u0*(r-1);
   for c=1:N
     v0y=v0*(c-1);
     f(r,c)=A*sin(u0x+v0y);
   end
end
t1=toc; % End timing

% Now implement using vectorization
tic %start timing
r=0:M-1;
c=0:N-1;
[C,R]=meshgrid(c,r);
%special MATLAB function for fast 2D function evaluations
% creates all the (x,y) pairs for function evaluation
g=A*sin(u0*R+v0*C);

t2=toc; %End timing

%compute the ratio of the two times
rt=t1/(t2+eps); %use eps in case t2 is close to zero.
```

# MATLAB/Image Processing Toolbox

```
>> [rt,f,g]=twodsin(1, 1/(2*pi), 1/(4*pi), 512, 512);
>> rt
rt =
    34.2520   %according to GWE.  I only got ~19.
>>g=mat2gray(g);
>> imshow(g)  %show in separate window.
```

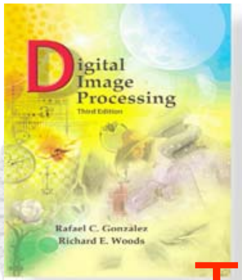






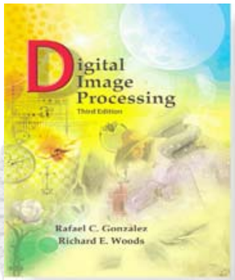

# MATLAB/Image Processing Toolbox

```
imshow (f,G)
%F is an image array
%G is the number of intensity levels.  Default is 256.

imwrite(f, ‘filename’)
% filename MUST contain a recognized file format extension
% .tif or .tiff identify TIFF
% .jpg identifies jpeg
% additional parameters for tiff and jpeg identify compression, etc.
imfiinfo filename
% returns all kind of cool file information such as size

Imread(‘filename’)
% filename MUST contain an appropriate extension
```




# The Histogram of a Grayscale Image

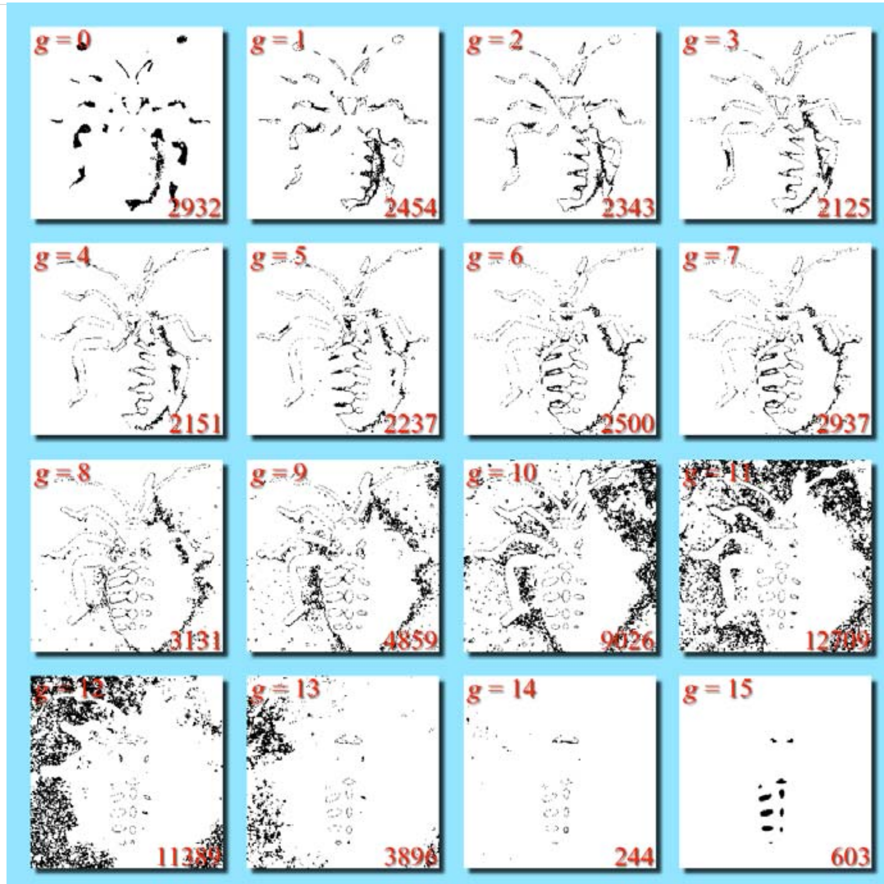
- Let $I$ be a 1-band (grayscale) image.
- $I(r,c)$ is an 8-bit integer between 0 and 255.
- Histogram, $h_I$, of $I$:
  - a 256-element array, $h_I$
  - $h_I(g)$, for $g$ = 1, 2, 3, …, 256, is an integer
  - $h_I(g)$ = number of pixels in $I$ that have value $g$-1.




# The Histogram of a Grayscale Image

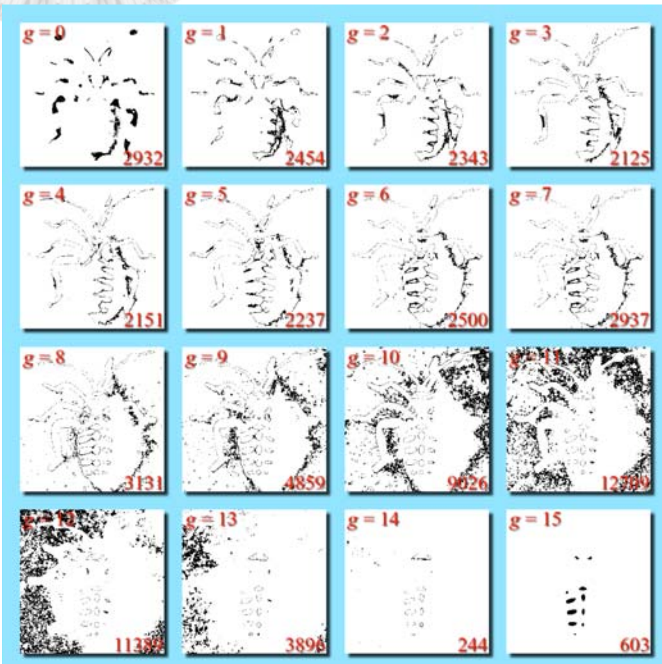
16-level (4-bit) image

lower RHC: number of pixels with intensity *g*

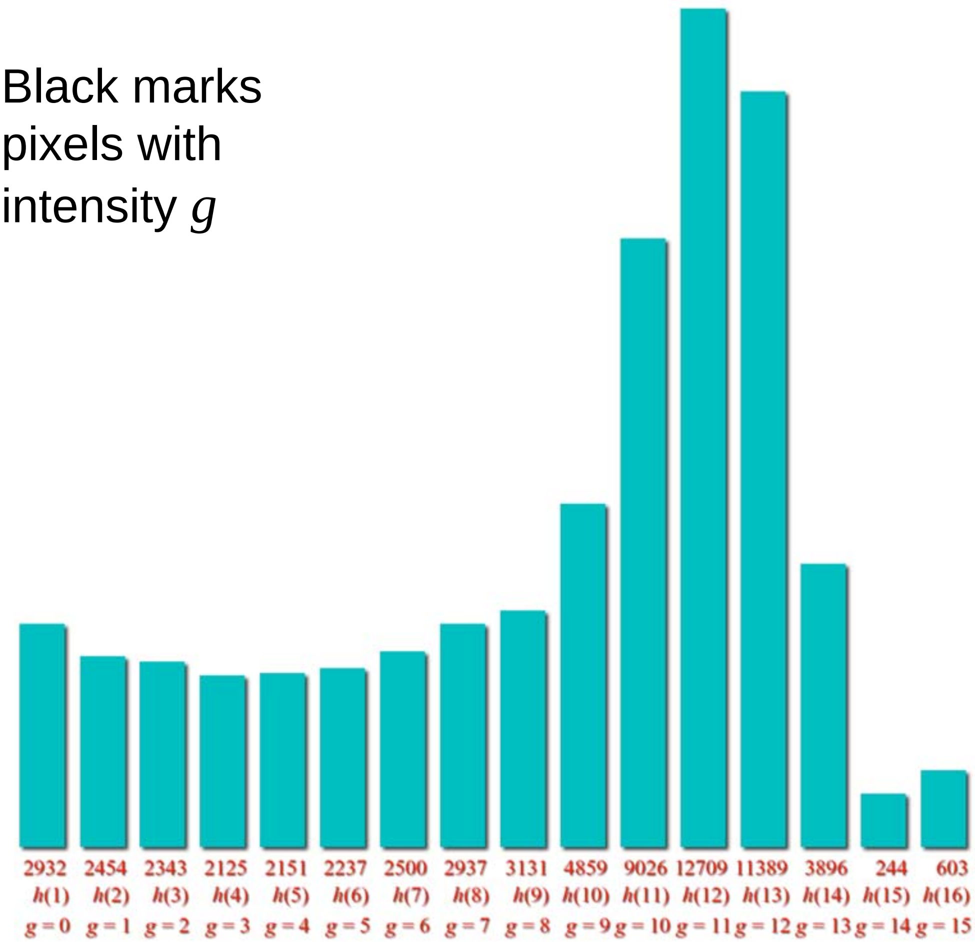



black marks pixels with intensity *g*

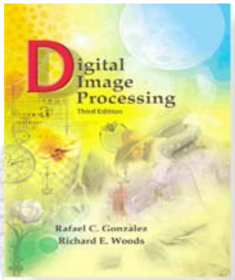
# The Histogram of a Grayscale Image




Black marks pixels with intensity $g$

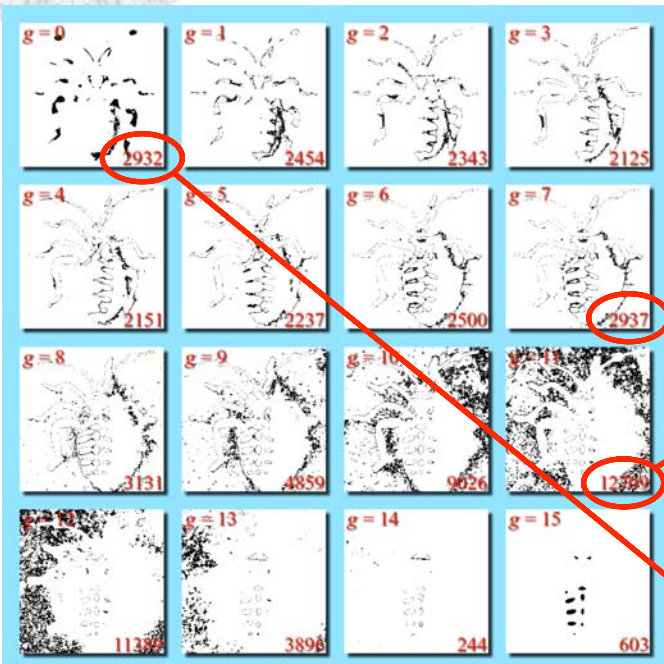



Plot of histogram:
number of pixels with intensity $g$

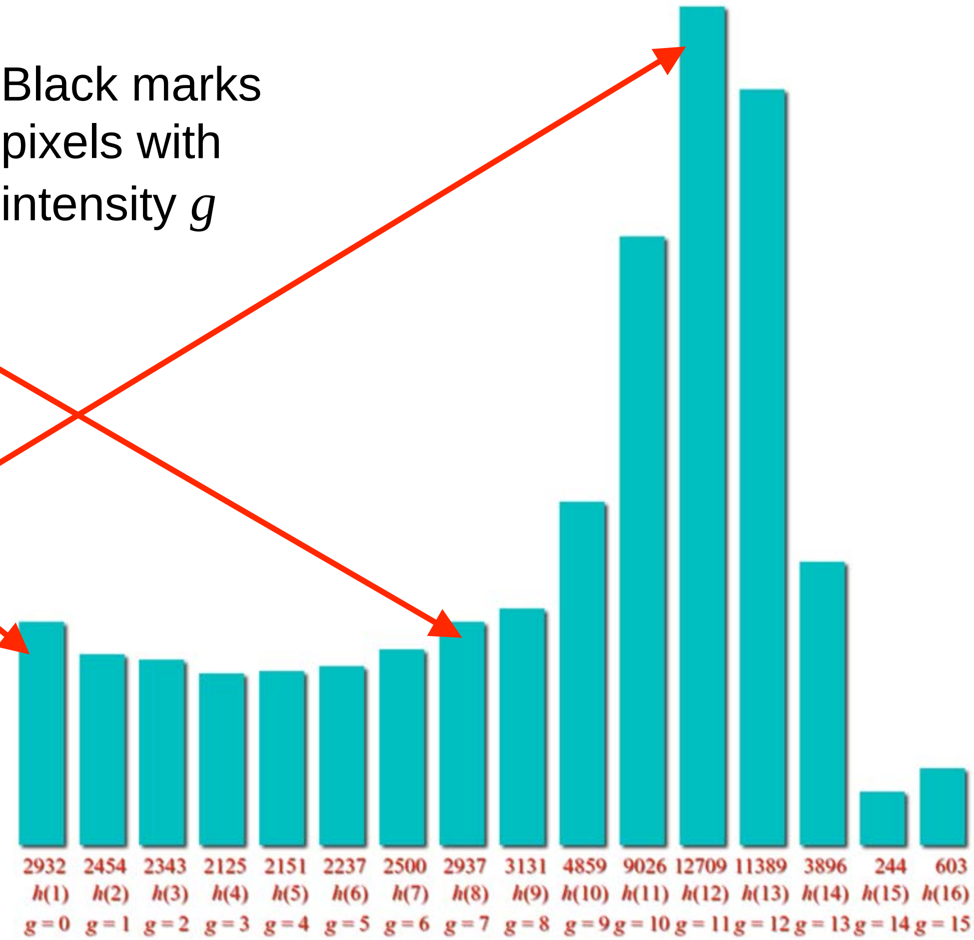





# The Histogram of a Grayscale Image



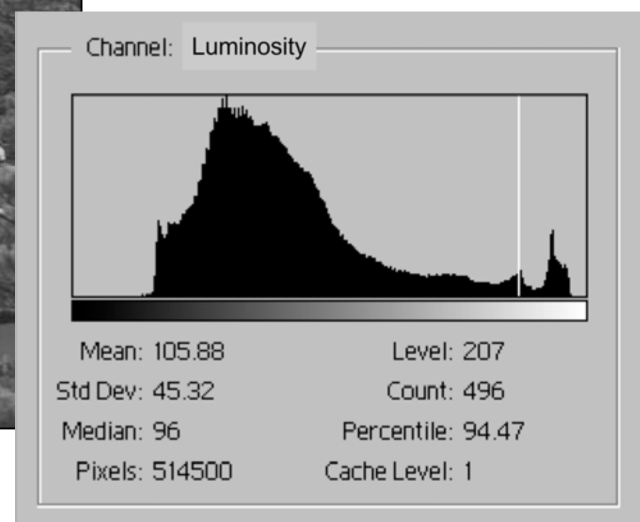

Black marks pixels with intensity $g$




Plot of histogram:
number of pixels with intensity $g$

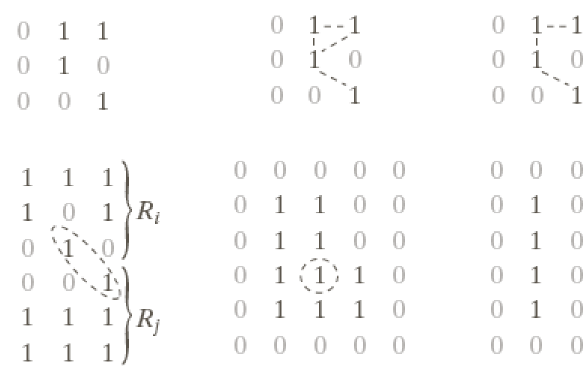
# The Histogram of a Grayscale Image

$h_I(g+1) =$ the number of pixels in $I$ with graylevel $g$.



# MATLAB/Image Processing Toolbox

```
>> h=imhist(f)                          %any previously loaded image
>> h1=h(1:10:256)                       %create bins for horiz axis
>> horz=(1:10:256;                      %
>> bar(horiz, h1)                       %
>> axis([0 255 0 15000])                %expand lower range of y-axis
>> set(gca, ‘xtick’, 0:50:255)          %gca means ‘get current axis’
>> set(gca, ‘ytick’, 0:2000:15000)  %lab h & v ticks
```

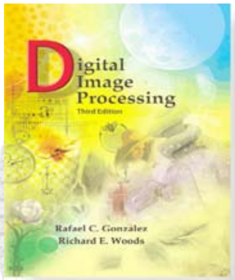


a b c
d e f

**FIGURE 2.25** (a) An arrangement of pixels. (b) Pixels that are 8-adjacent (adjacency is shown by dashed lines; note the ambiguity). (c) $m$-adjacency. (d) Two regions that are adjacent if 8-adjecency is used. (e) The circled point is part of the boundary of the 1-valued pixels only if 8-adjacency between the region and background is used. (f) The inner boundary of the 1-valued region does not form a closed path, but its outer boundary does.

# Connectivity

- 4-adjacency
- 8-adjacency
- m-adjacency




# Distance Measures

- Euclidian - circle

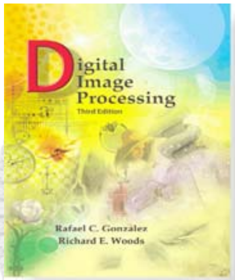
$$D_e(p,q) = \sqrt{(x-s)^2 + (y-t)^2}$$

- D4 (city block) - diamond

$$D_4(p,q) = |x-s| + |y-t|$$

- D8 (chessboard) - square

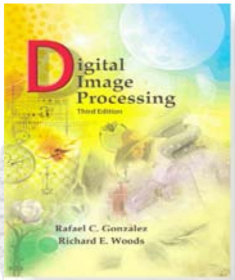
$$D_8(p,q) = \max(|x-s|, |y-t|)$$




# Array v. Matrix operations

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

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



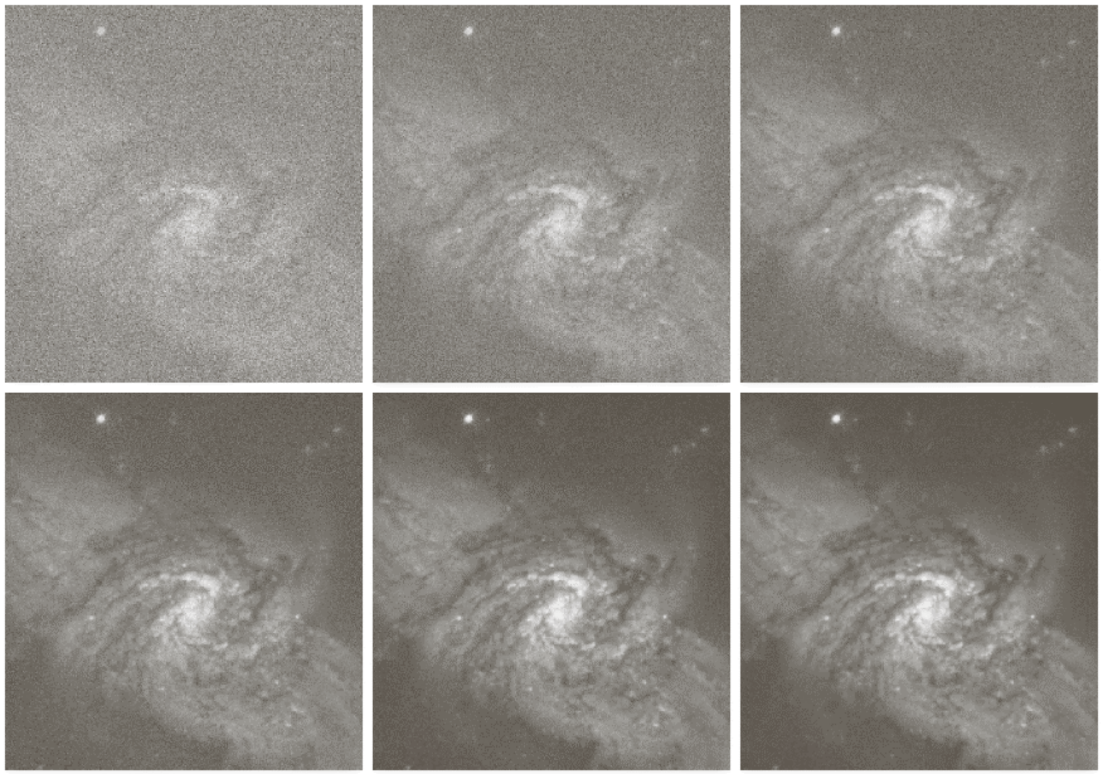

# Arithmetic Operations

$$s(x,y) = f(x,y) + g(x,y)$$
$$d(x,y) = f(x,y) - g(x,y)$$

$$p(x,y) = f(x,y) \times g(x,y)$$

$$v(x,y) = f(x,y) \div g(x,y)$$

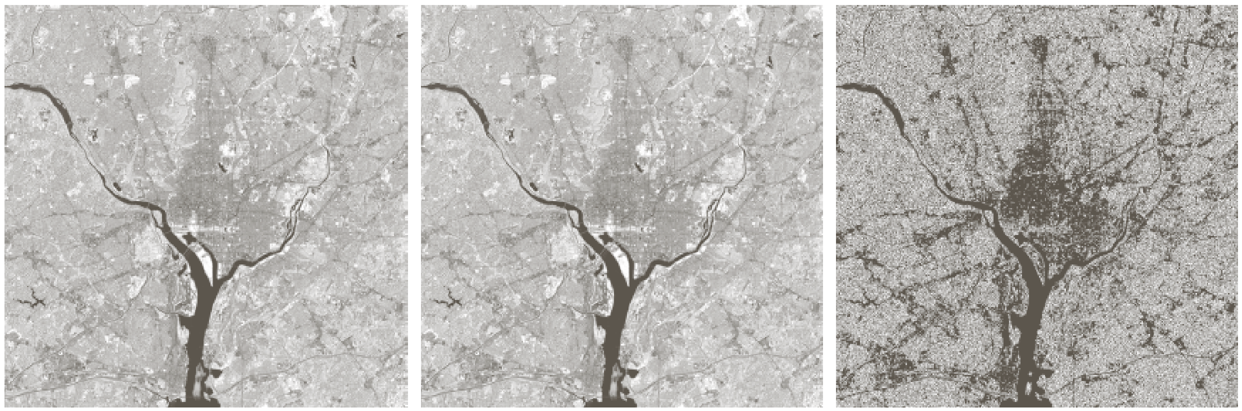
# Averaging



a b c
d e f

**FIGURE 2.26** (a) Image of Galaxy Pair NGC 3314 corrupted by additive Gaussian noise. (b)–(f) Results of averaging 5, 10, 20, 50, and 100 noisy images, respectively. (Original image courtesy of NASA.)

# Differences



a b c

**FIGURE 2.27** (a) Infrared image of the Washington, D.C. area. (b) Image obtained by 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.

# Differences

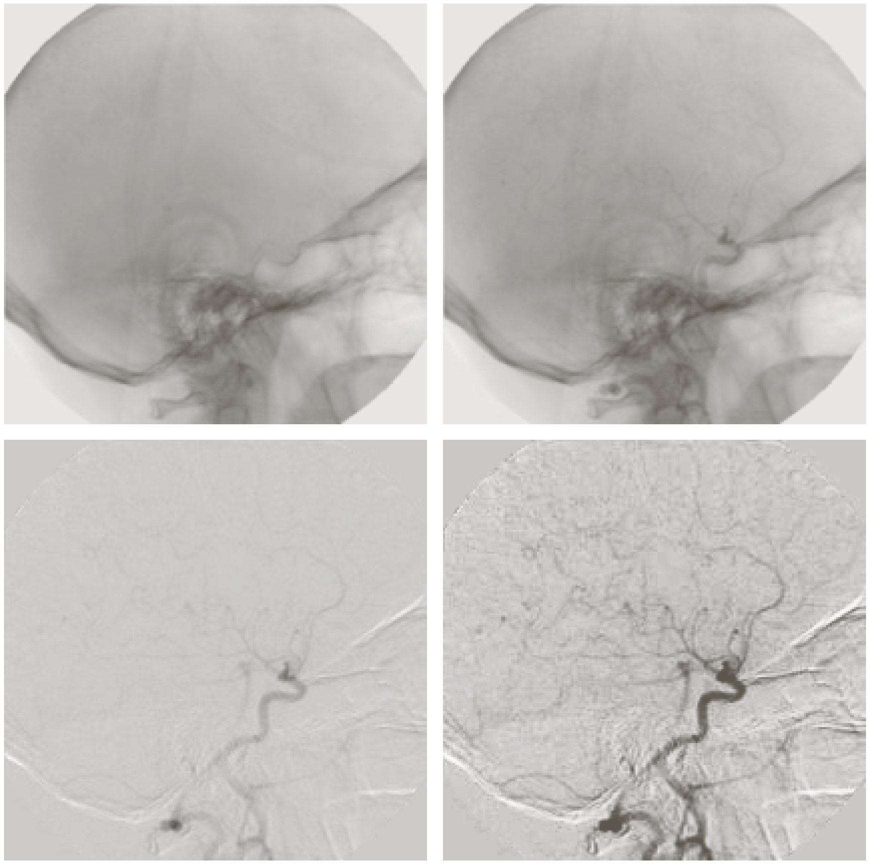



**FIGURE 2.28** Digital subtraction angiography. (a) Mask image. (b) A live image. (c) Difference between (a) and (b). (d) Enhanced difference image. (Figures (a) and (b) courtesy of The Image Sciences Institute, University Medical Center, Utrecht, The Netherlands.)

# Multiplication

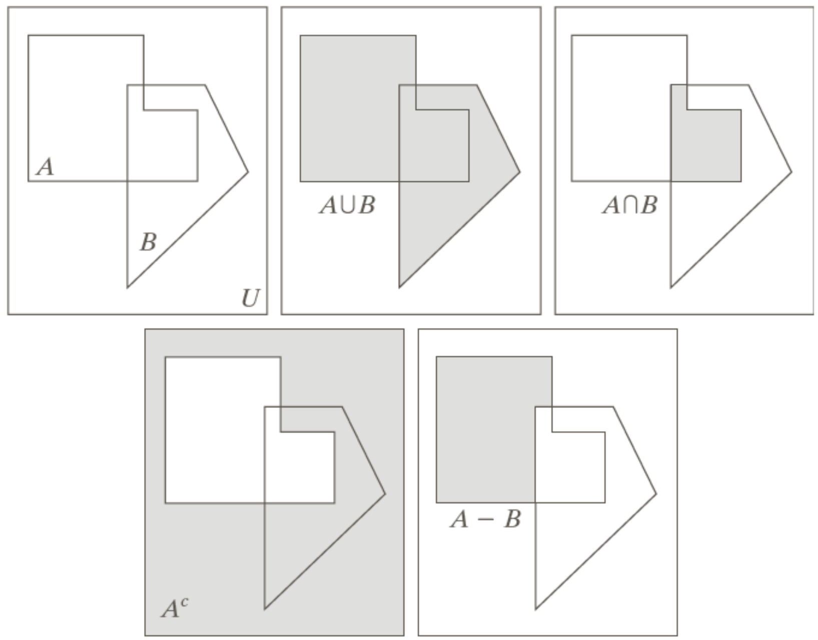
a b c

**FIGURE 2.29** Shading correction. (a) Shaded SEM image of a tungsten filament and support, magnified approximately 130 times. (b) The shading pattern. (c) Product of (a) by the reciprocal of (b). (Original image courtesy of Mr. Michael Shaffer, Department of Geological Sciences, University of Oregon, Eugene.)

# Multiplication

FIGURE 2.30 (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).

# Set Operations



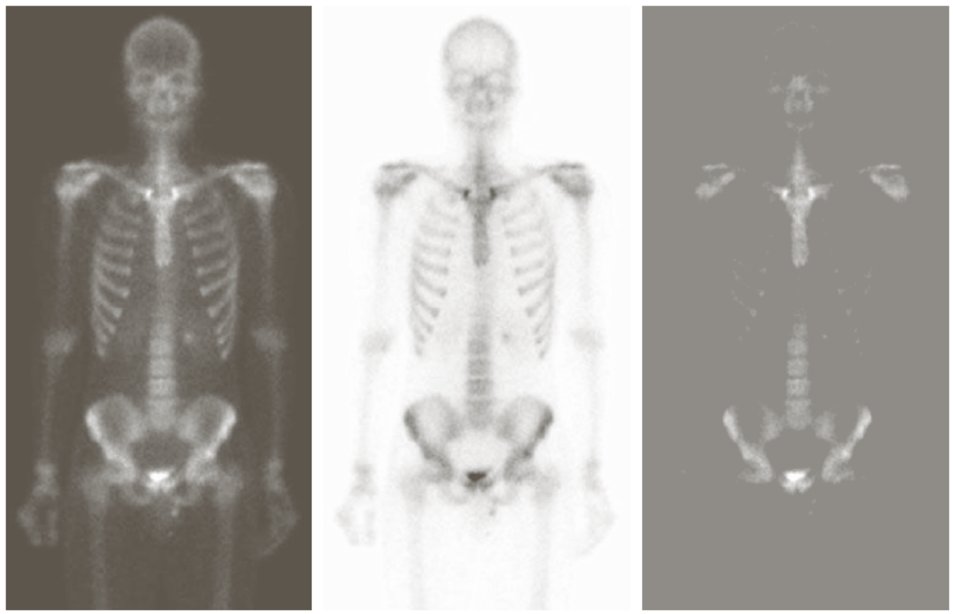

a b c
d e

**FIGURE 2.31** (a) Two sets of coordinates, $A$ and $B$, in 2-D space. (b) The union of $A$ and $B$. (c) The intersection of $A$ and $B$. (d) The complement of $A$. (e) The difference between $A$ and $B$. In (b)–(e) the shaded areas represent the member of the set operation indicated.

$$A - B = A \cap B^c = \{ w \mid w \in A, w \notin B \}$$

# Gray Scale Set Operations

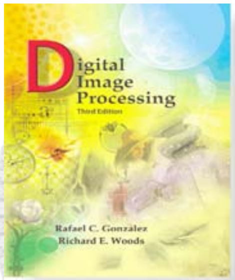
$$A^c = \{(x, y, K - z) \mid (x, y, z) \in A\}$$



a b c

**FIGURE 2.32** Set operations involving gray-scale images. (a) Original image. (b) Image negative obtained using set complementation. (c) The union of (a) and a constant image. (Original image courtesy of G.E. Medical Systems.)

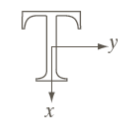
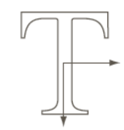
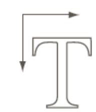
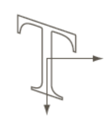
$$A \cup B = \{\max_z (a, b) \mid a \in A, b \in A\}$$



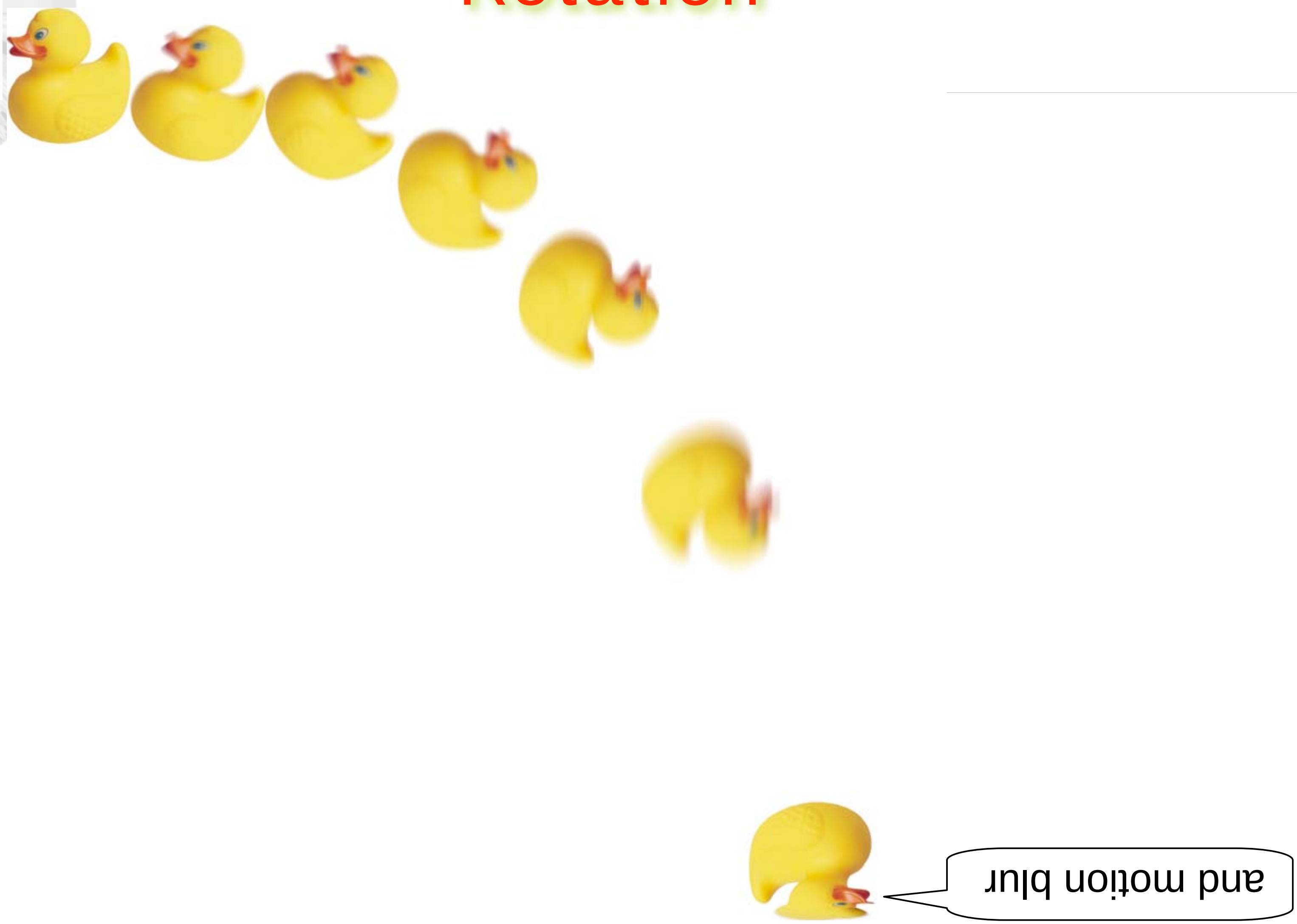

# Geometric Transformations

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} v & w & 1 \end{bmatrix} T = \begin{bmatrix} v & w & 1 \end{bmatrix} \begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

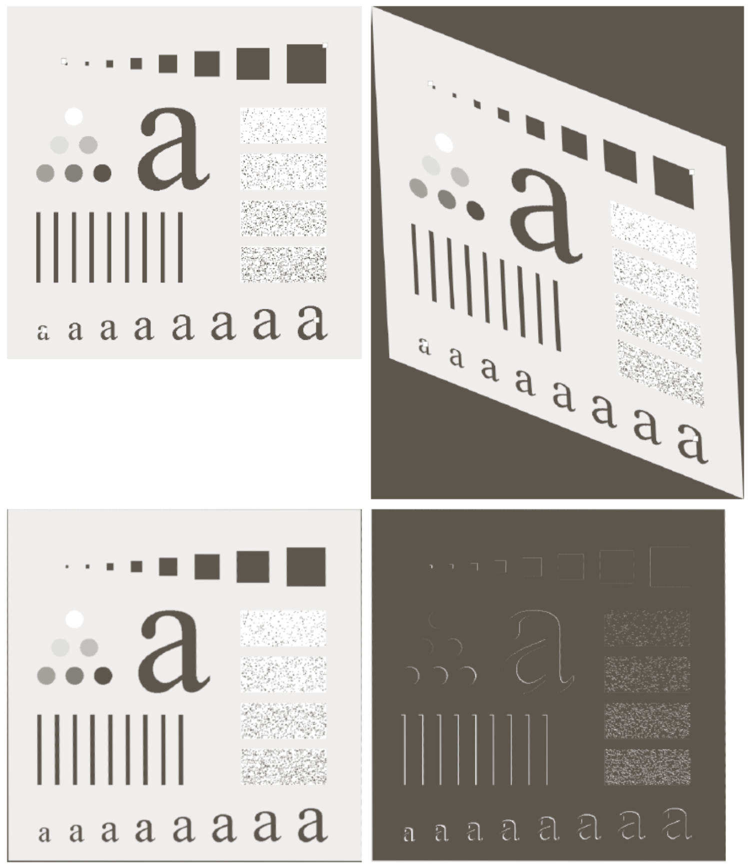
**TABLE 2.2**
Affine transformations based on Eq. (2.6.–23).

| Transformation Name | Affine Matrix, T | Coordinate Equations | Example |
|---|---|---|---|
| Identity | $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $x = v$<br>$y = w$ |   |
| Scaling | $\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $x = c_x v$<br>$y = c_y w$ |  |
| Rotation | $\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $x = v\cos\theta - w\sin\theta$<br>$y = v\cos\theta + w\sin\theta$ |  |
| Translation | $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$ | $x = v + t_x$<br>$y = w + t_y$ |  |
| Shear (vertical) | $\begin{bmatrix} 1 & 0 & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $x = v + s_v w$<br>$y = w$ |  |
| Shear (horizontal) | $\begin{bmatrix} 1 & s_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $x = v$<br>$y = s_h v + w$ |  |

# Rotation



# Image Registration



a b
c d

**FIGURE 2.37** Image registration. (a) Reference image. (b) Input (geometrically distorted image). Corresponding tie points are shown as small white squares near the corners. (c) Registered image (note the errors in the borders). (d) Difference between (a) and (c), showing more registration errors.

# Image Warping

# Geometric Transformations (Warping)

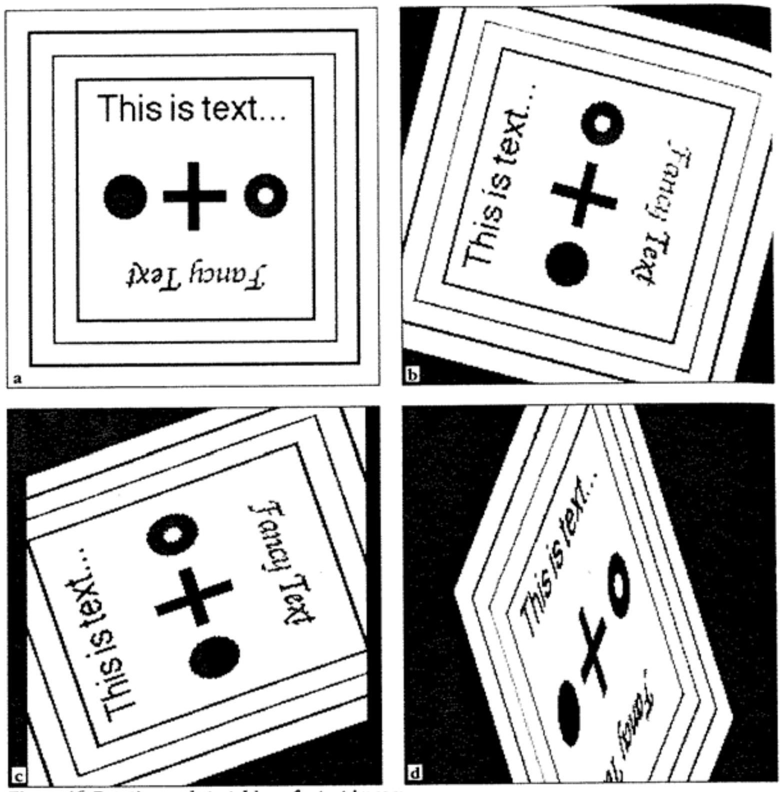
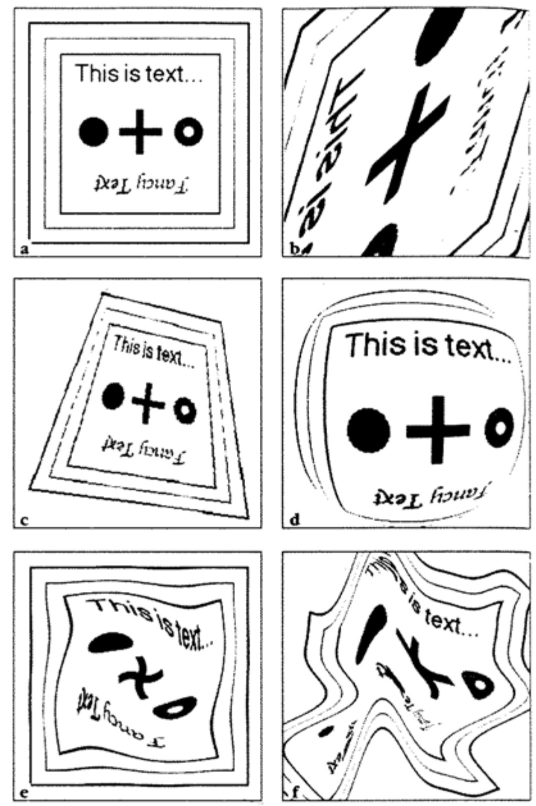
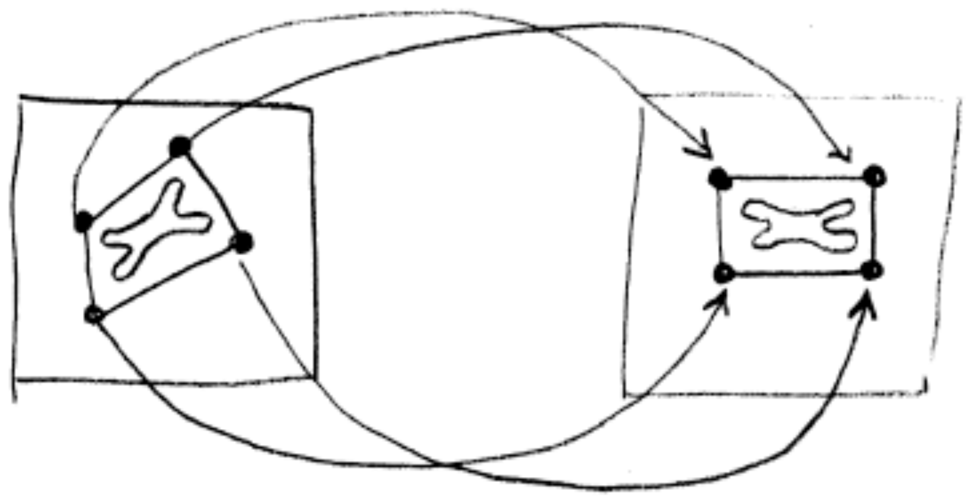


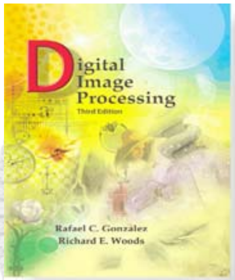

***Figure 46. Rotation and stretching of a test image:***
***a)*** *original;*
***b)*** *rotation only, no change in scale;*
***c)*** *rotation and uniform stretching while maintaining angles;*
***d)*** *general rotation and stretching.*

***Figure 49. Some additional examples of image warping:***
***a)*** *original test image;*
***b)*** *linear warping with reversal;*
***c)*** *quadratic warping showing trapezoidal foreshortening (no interpolation);*
***d)*** *cubic warping in which lines are curved (approximation here is to a spherical surface);*
***e)*** *twisting the center of the field while holding the edges fixed (also cubic warping);*
***f)*** *arbitrary warping in which higher order and trigonometric terms are required.*



# Spatial Warping





$$G(x,y) = F(x',y') = F(\underline{ax+by+cxy+d},\ \underline{ex+fy+gxy+h})$$

basically do a curve fit: four x's map to four x′'s.

i.e.

$$x_1' = ax_1 + by_1 + cx_1y_1 + d$$
$$x_2' = ax_2 + by_2 + cx_2y_2 + d$$
$$x_3' = ax_3 + by_3 + cx_3y_3 + d$$
$$x_4' = ax_4 + by_4 + cx_4y_4 + d$$

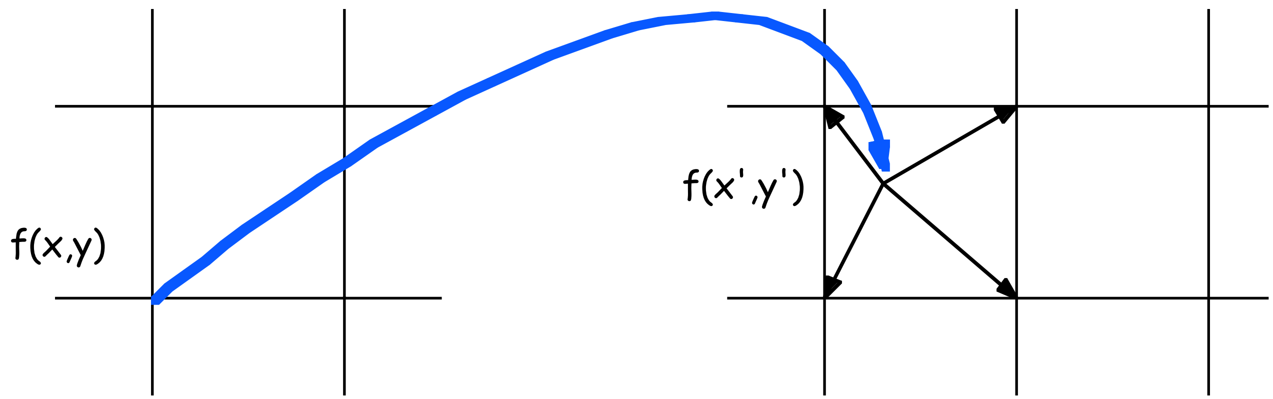
$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & x_1y_1 & 1 \\ x_2 & y_2 & x_2y_2 & 1 \\ x_3 & y_3 & x_3y_3 & 1 \\ x_4 & y_4 & x_4y_4 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

and solve for $\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$

a b c d

**FIGURE 2.36** (a) A 300 dpi image of the letter T. (b) Image rotated 21° clockwise using nearest neighbor interpolation to assign intensity values to the spatially transformed pixels. (c) Image rotated 21° using bilinear interpolation. (d) Image rotated 21° using bicubic interpolation. The enlarged sections show edge detail for the three interpolation approaches.

# Gray Level Interpolation forward mapping




# Gray Level Interpolation backward mapping (pixel filling)