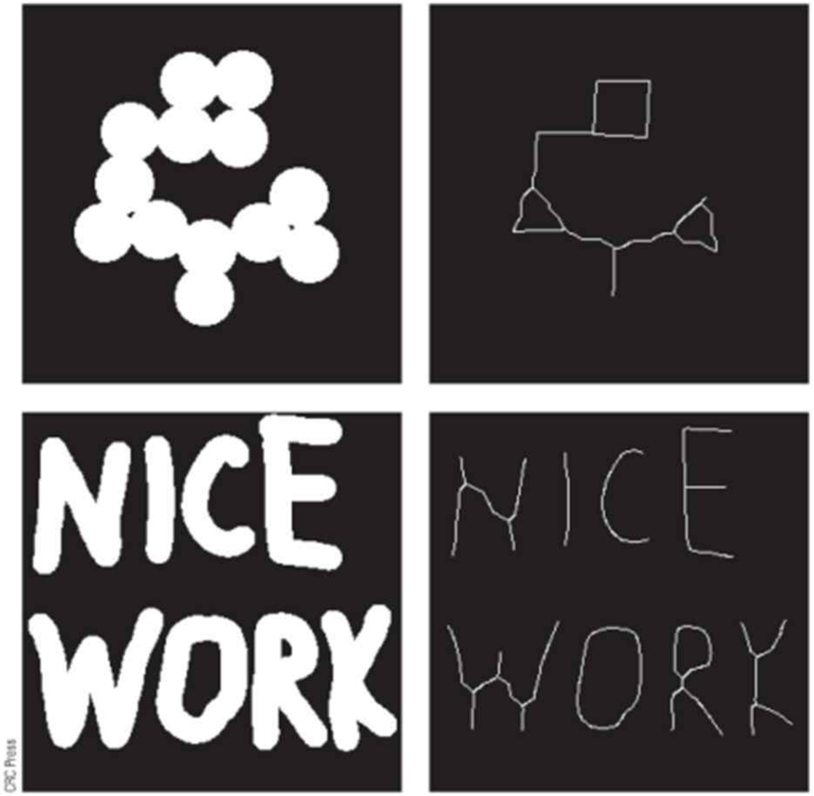
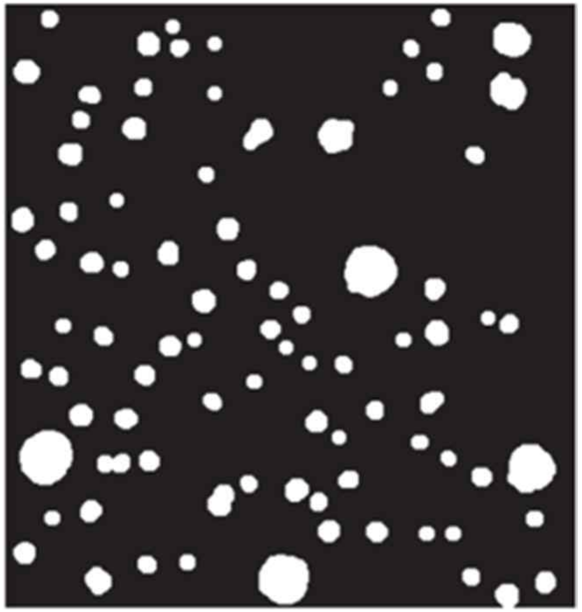


Chapter 11:

Image Topology

Introduction

- We are often interested in only the very basic aspects of an image:
  - ✓ The number of occurrences of a particular object
  - ✓ Whether there are holes or not, and so on
- The investigation of these fundamental properties of an image is called **digital topology** or **image topology.**



```
>> n=imread('nodules1.tif');
>> nt=~im2bw(n,0.5);
>> n2=imopen(nt,strel('disk',5));
```

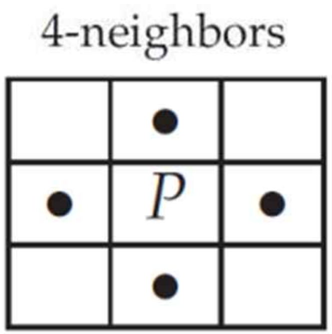
A Topology Question:
How many blobs are in this image ?

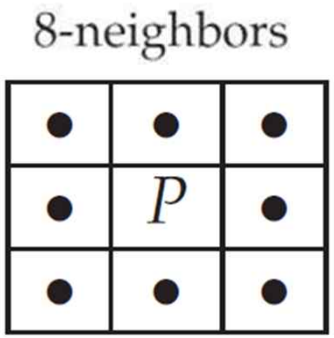
Neighbor and Adjacency

- A first task is to define the concept of adjacency:

  "Under what conditions a pixel may be regarded as being **next to** another pixel ?"

- For this chapter, the concern will be with **binary images** only, and thus we will be dealing only with **positions** of pixels





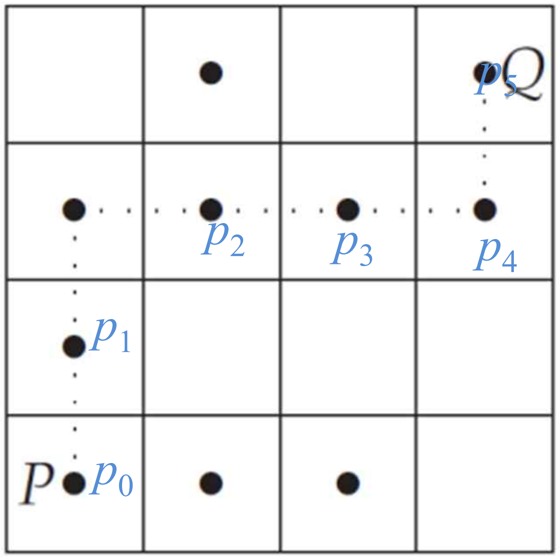
- P and Q are 4-adjacent if they are one of 4-neighbors each other and 8-adjacent if they are 8-neighbors.

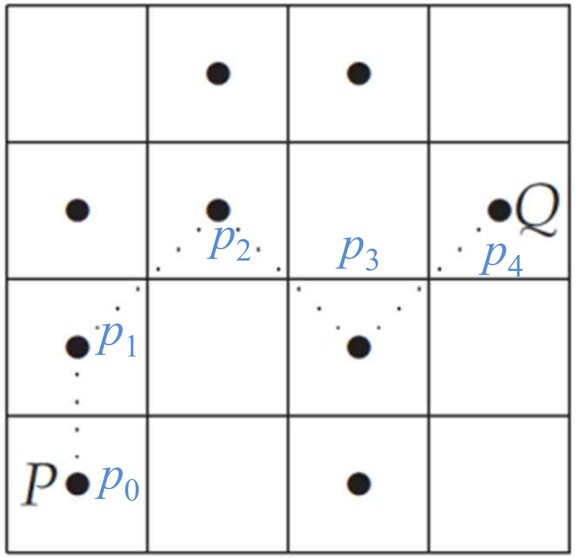
Paths and Components

- P and Q are any two (not necessarily adjacent) pixels

  ✓ If the path contains only 4-adjacent pixels, as the path does in the below diagram, then P and Q are "4-connected".

  ✓ If the path contains 8-adjacent pixels, as the path does in the below diagram, then P and Q are "8-connected".

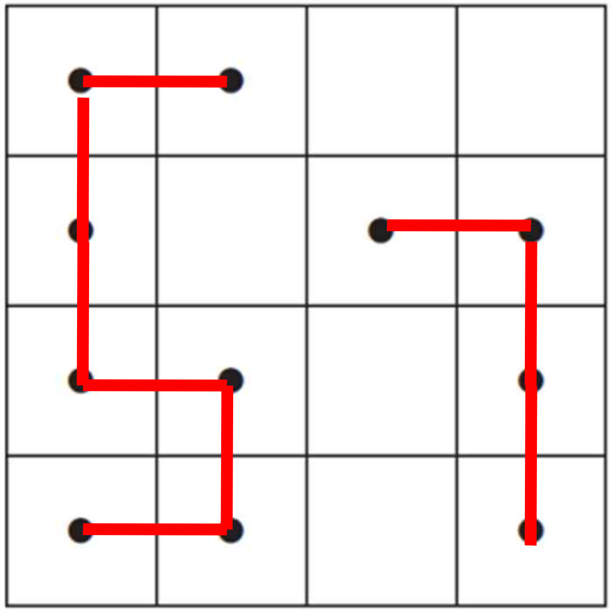




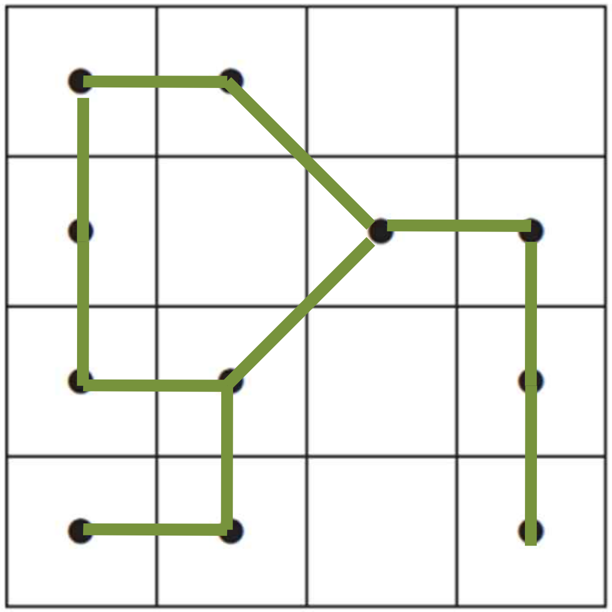
$$P = p_0, p_1, p_2, \ldots, p_n = Q$$

Paths and Components

- A set of pixels, all of which are 4-connected to each other, is called a **4-component**.
- If all the pixels are 8-connected, the set is an **8-component**.



Two 4-components



One 8-component

Equivalence Relations

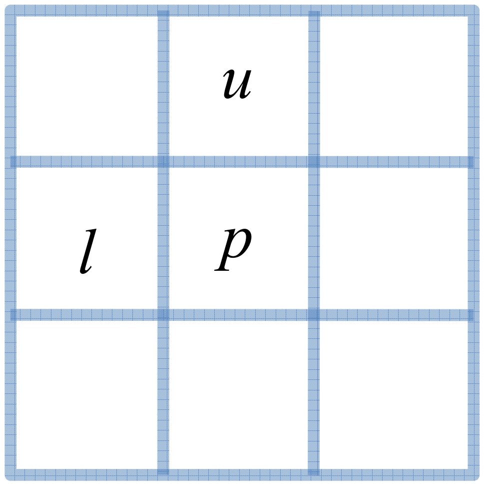
- A relation $x \sim y$ between two objects x and y is an equivalence relation if the relation is
  1. **Reflexive**, $x \sim x$ for all x,
  2. **Symmetric**, $x \sim y \Longleftrightarrow y \sim x$ for all x and y,
  3. **Transitive**, if $x \sim y$ and $y \sim z$, then $x \sim z$ for all x, y, and z

- Example
  - For numerical equility: x and y are equivalent($x \sim y$) if x = y.
  - For divisors: x$\sim$y if x and y have the same remainder then it is divided by the same number.
  - For connected component, P$\sim$Q if P and Q are connected pixels.

- An equivalence class is the set of all objects equivalent to each other.

Component Labeling (Connected Component)

- Foreground pixel(fg): a pixel in the image (dots)
- background pixel(bg): a pixel not in the image



Step 1. Check the state of p.

If (p is a fg pixel),

If (u and l are bg pixels), assign a new label to p.

elseif (one of u or l is a fg pixel), assign its label to p.

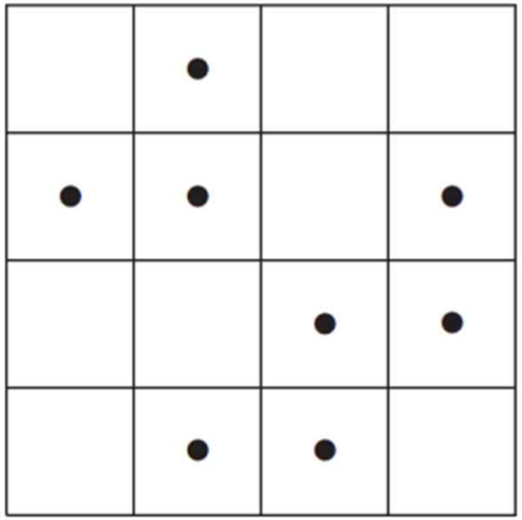
elseif (u and l are fg pixels and have the same label),
assign that label to p.

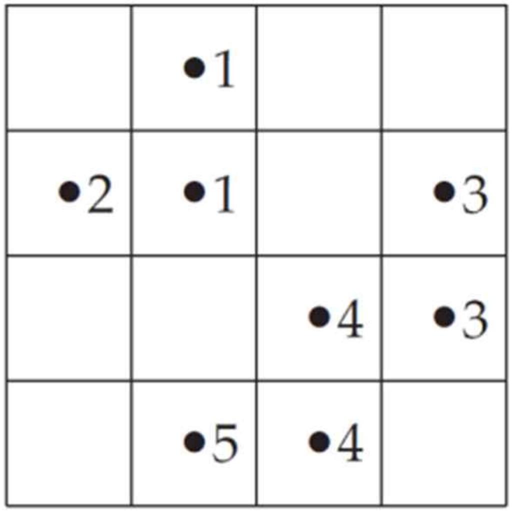
elseif (u and l are fg pixels but have different labels),
assign either of those labels to p and make a note that two labels are equivalent.
(= u and l belong to the same 4-component connected through p)

Step 2. Sort the labels into equivalence classes and assign a different label to each class.

Step 3. Replace the label on each fg pixel with the label assigned to its equivalence class in the Step 2.

Component Labeling





Step 1. Check the state of a pixel p

for (from first pixel index to the last).

If (p is a fg pixel),

If (u and l are bg pixels), assign a new label to p.

elseif (one of u or l is a fg pixel), assign its label to p.

elseif (u and l are fg pixels and have the same label),

assign that label to p.

elseif (u and l are fg pixels but have different labels),

assign either of those labels to p and make a note that two labels are equivalent.

(= u and l belong to the same 4-component connected through p)

endif

endif

endfor

Note
- Label 1 and 2 are equivalent.
- Label 3 and 4 are equivalent.
- Label 4 and 5 are equivalent.

Component Labeling

Note
- Label 1 and 2 are equivalent.
- Label 3 and 4 are equivalent.
- Label 4 and 5 are equivalent.

	•1		
•2	•1		•3
		•4	•3
	•5	•4	

Step 2. Sort the labels into equivalence classes and assign a different label to each class.

- equivalence classes of labels: {1, 2} and {3, 4, 5}
- Assign label 1 to the first class. {1, 2} ⟶ **1**
- Assign label 2 to the second class.: {3, 4, 5} ⟶ **2**

	•1		
•1	•1		•2
		•2	•2
	•2	•2	

Step 3. Replace the label on each fg pixel with the label assigned to its equivalence class in the Step 2.

Component Labeling with 8-Components

- ✓ Diagonal elements of p used to label 8-components of an image.
- ✓ See pp.314 for more details.

d	u	e
l	p	

Component Labeling in Matlab

- Sample image: two 8-components and three 4-components

```
>> i=zeros(8,8);
>> i(2:4,3:6)=1;
>> i(5:7,2)=1;
>> i(6:7,5:8)=1;
>> i(8,4:5)=1;
>> i
```

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

- bwlabel(i, 4) and bwlabel(i, 8) for 4- and 8-components labeling

0	0	0	0	0	0	0	0
0	0	2	2	2	2	0	0
0	0	2	2	2	2	0	0
0	0	2	2	2	2	0	0
0	1	0	0	0	0	0	0
0	1	0	0	3	3	3	3
0	1	0	0	3	3	3	3
0	0	0	3	3	0	0	0

0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	0	0
0	1	0	0	2	2	2	2
0	1	0	0	2	2	2	2
0	0	0	2	2	0	0	0

Component Labeling Example

```
>> b=imread('bacteria.tif');
>> bt=b<100;
>> bl=bwlabel(bt);
>> max(bl(:))
```

```
>> 21
```
: means number of connected components is 21

```
figure, imshow(bt)
```

Distances and Metrics

- It is necessary to define a function that provides a measure of distance between two points *x* and *y* on a grid
- A distance function $d(x, y)$ is called a **metric** if it satisfies the following:
  1. $d(x, y) = d(y, x)$ (symmetry)
  2. $d(x, y) \geq 0$ and $d(x, y) = 0$ if and only if $x = y$ (positivity)
  3. $d(x, y) + d(y, z) \leq d(x, z)$ (the triangle inequality)

$$\begin{array}{ccccc} & & 2 & & \\ & 2 & 1 & 2 & \\ 2 & 1 & 0 & 1 & 2 \\ & 2 & 1 & 2 & \\ & & 2 & & \end{array} \qquad \begin{array}{ccccc} 2 & 2 & 2 & 2 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 1 & 0 & 1 & 2 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{array}$$

- Euclidean distance

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

- D_4 and D_8 distances

$$d_4(x,y) = |x_1 - y_1| + |x_2 - y_2|$$

$$d_8(x,y) = \max\{|x_1 - y_1|, |x_2 - y_2|\}$$

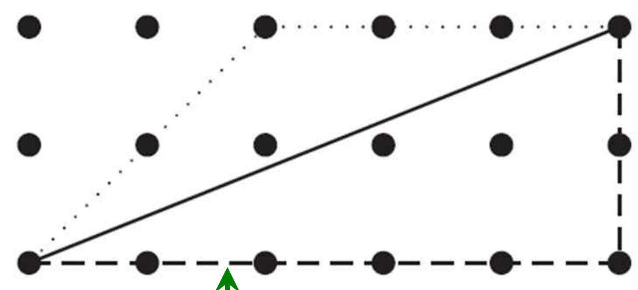


FIGURE 11.4 Comparison of three metrics.

d4 distance or taxicab metric

Distance Transform

- How to find the distance of a pixel p(x,y) from a region R ?
  - **Method 1**: Calculate Euclidean distance between p(x,y) and all pixels in R and take the smallest value.

$$md(x,y) = \sqrt{\min_{(p,q)\in R}\left((x-p)^2 + (y-q)^2\right)}$$

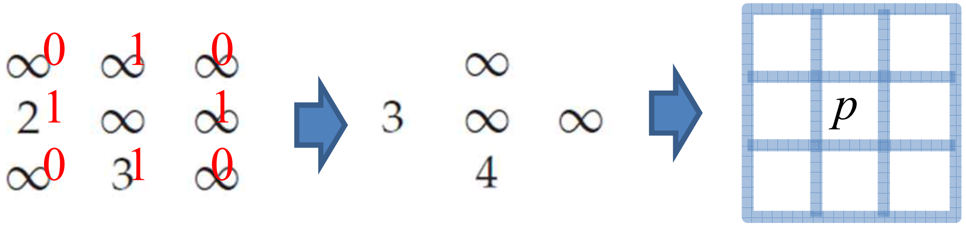
high computational cost

  - **Method 2**: Use distance transform for more efficient computation.
    - **Step 1**: For each pixel (x, y), label with 0 if it is in R, and ∞ if it is not in R.
    - **Step 2**: For each pixel (x, y), replace its label with the minimum distance md(x,y) calculated by $md(x,y) = min\{d(x, y), d(x+1, y)+1, d(x-1, y)+1, d(x, y-1)+1, d(x, y+1)+1\}$ with using $\infty + 1 = \infty$
    - **Step 3**: Repeat **Step 2** until all labels have been converted to finite values.

Distance Transform Example

- Use the matrix to calculate md(x,y) at p(x,y).

0	+1	0
+1	p	+1
0	+1	0



output at p(x,y) =
min(∞, 3, 4) = 3

- Example

∞	∞	∞	∞	∞	∞
∞	∞	0	0	∞	∞
∞	∞	∞	0	∞	∞
∞	∞	∞	0	∞	∞
∞	∞	∞	0	0	∞
∞	∞	∞	∞	∞	∞

Step 1

∞	∞	1	1	∞	∞
∞	1	0	0	1	∞
∞	∞	1	0	1	∞
∞	∞	1	0	1	∞
∞	∞	1	0	0	1
∞	∞	∞	1	1	∞

Step 2 (first pass)

∞	2	1	1	2	∞
2	1	0	0	1	2
∞	2	1	0	1	2
∞	2	1	0	1	2
∞	2	1	0	0	1
∞	∞	2	1	1	2

Step 2 (second pass)

3	2	1	1	2	3
2	1	0	0	1	2
3	2	1	0	1	2
3	2	1	0	1	2
3	2	1	0	0	1
∞	3	2	1	1	2

Step 2 (third pass)

3	2	1	1	2	3
2	1	0	0	1	2
3	2	1	0	1	2
3	2	1	0	1	2
3	2	1	0	0	1
4	3	2	1	1	2

Step 2 (final pass)

Distance Transform Example

- ✓ Different type of mask

4	3	4
3	0	3
4	3	4

∞	4	3	3	4	∞
∞	3	0	0	3	∞
∞	4	3	0	3	∞
∞	∞	3	0	3	∞
∞	∞	3	0	0	3
∞	∞	4	3	3	4

Step 2 (first pass)

7	4	3	3	4	7
6	3	0	0	3	6
7	4	3	0	3	6
8	6	3	0	3	6
∞	6	3	0	0	3
∞	7	4	3	3	4

Step 2 (second)

7	4	3	3	4	7
6	3	0	0	3	6
7	4	3	0	3	6
8	6	3	0	3	6
9	6	3	0	0	3
10	7	4	3	3	4

Step 2 (third)

Divide all values by 3

2.3	1.3	1	1	1.3	2.3
2	1	0	0	1	2
2.3	1.3	1	0	1	2
2.7	2	1	0	1	2
3	2	1	0	0	1
3.3	2.3	1.3	1	1	1.3

Distance Transform Example

✓ Different type of mask

	11		11	
11	7	5	7	11
	5	0	5	
11	7	5	7	11
	11		11	

11	7	5	5	7	11
10	5	0	0	5	10
11	7	5	0	5	10
14	10	5	0	5	7
16	10	5	0	0	5
16	11	7	5	5	7

Result of transform

2.2	1.4	1	1	1.4	2.2
2	1	0	0	1	2
2.2	1.4	1	0	1	2
2.8	2	1	0	1	1.4
3.2	2	1	0	0	1
3.2	2.2	1.4	1	1	1.4

After division by 5

Distance Transform with Two Separated Masks

- ✓ A quicker method requires two passes only:
  - The first pass starts at the top left of the image and moves left to right, top to bottom.
  - The second pass starts at the bottom right of the image and moves right to left, from bottom to top

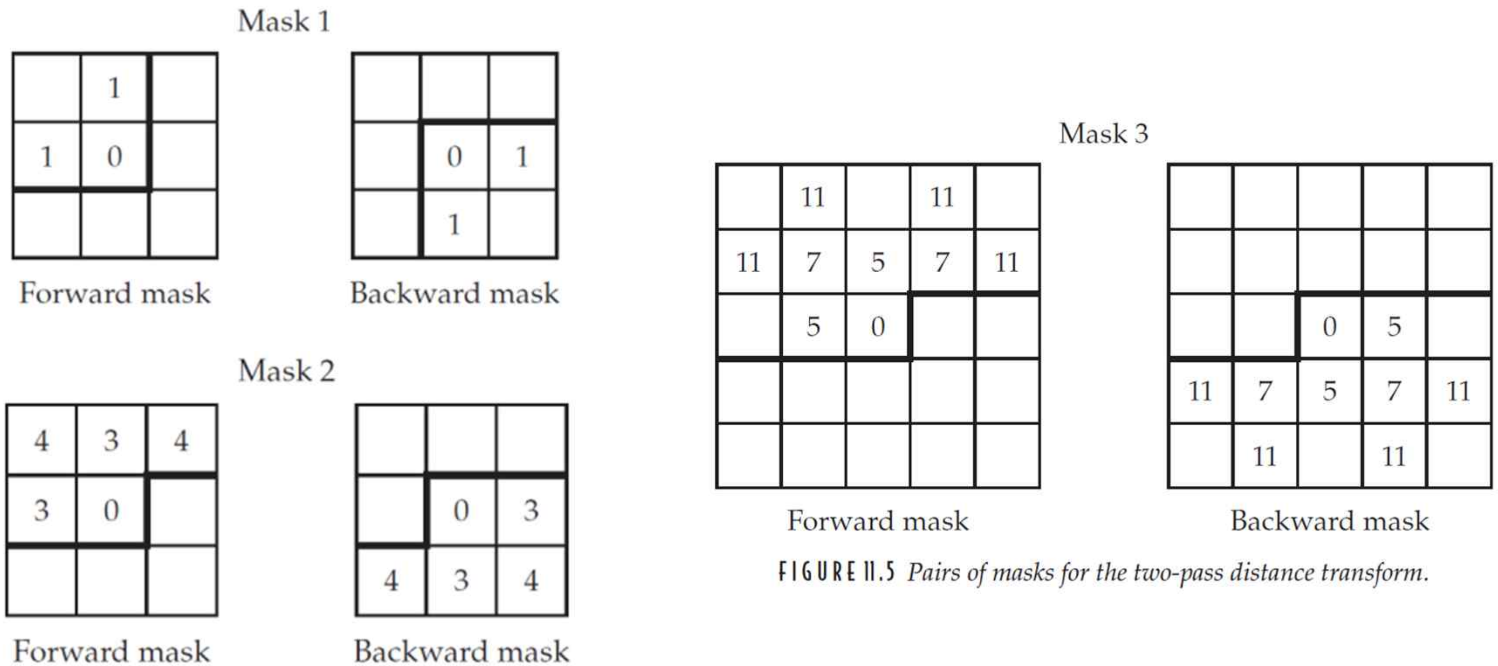


FIGURE 11.5 *Pairs of masks for the two-pass distance transform.*

Distance Transform in Matlab

✓ **How to implement: step-by-step**

1. Using the size of the mask, pad the image with an appropriate number of 0s.
2. Change each 0 to infinity, and each 1 to 0. (Step 1)
3. Create forward and backward masks
4. Perform a forward pass. Replace each label with the **minimum** of its neighborhood plus the forward mask. (Step 2)
5. Perform a backward pass. Replace each label with the **minimum** of its neighborhood plus the backward mask. (Step 2)

Distance Transform in Matlab

- function distrans()

```
function res=disttrans(image,mask)
backmask=rot90(rot90(mask));
[mr,mc]=size(mask);
if ((floor(mr/2)==ceil(mr/2)) |(floor(mc/2)==ceil(mc/2))) then
     error('The mask must have odd dimensions.')
     end;
[r,c]=size(image);
nr=(mr-1)/2;
nc=(mc-1)/2;
image2=zeros(r+mr-1,c+mc-1);
image2(nr+1:r+nr,nc+1:c+nc)=image;
image2(find(image2==0))=Inf;
image2(find(image2==1))=0;
for i=nr+1:r+nr,
  for j=nc+1:c+nc,
    image2(i,j)=min(min(image2(i-nr:i+nr,j-nc:j+nc)+mask));
  end;
end;
for i=r+nr:-1:nr+1,
  for j=c+nc:-1:nc+1,
    image2(i,j)=min(min(image2(i-nr:i+nr,j-nc:j+nc)+backmask));
  end;
end;

res=image2(nr+1:r+nr,nc+1:c+nc);
```

Distance Transform in Matlab

```
>> im=[0 0 0 0 0 0;...
0 0 1 1 0 0;...
0 0 0 1 0 0;...
0 0 0 1 0 0;...
0 0 0 1 1 0;...
0 0 0 0 0 0]

im =

     0     0     0     0     0     0
     0     0     1     1     0     0
     0     0     0     1     0     0
     0     0     0     1     0     0
     0     0     0     1     1     0
     0     0     0     0     0     0
```

generation of a sample image

```
>> mask1=[Inf 1 Inf;1 0 Inf;Inf Inf Inf]

mask1 =

   Inf     1   Inf
     1     0   Inf
   Inf   Inf   Inf
```

generation of the distance transform 'mask1'

Distance Transform in Matlab

- Generation of the distance transform 'mask2' and 'mask3'

```
>> mask2=[4 3 4;3 0 Inf;Inf Inf Inf]

mask2 =

     4     3     4
     3     0   Inf
   Inf   Inf   Inf
```

generation of the distance transform 'mask2'

```
>> mask3=[Inf 11 Inf 11 Inf;...
11 7 5 7 11;...
Inf 5 0 Inf Inf;...
Inf Inf Inf Inf Inf;...
Inf Inf Inf Inf Inf]

mask3 =

   Inf    11   Inf    11   Inf
    11     7     5     7    11
   Inf     5     0   Inf   Inf
   Inf   Inf   Inf   Inf   Inf
   Inf   Inf   Inf   Inf   Inf
```

generation of the distance transform 'mask3'

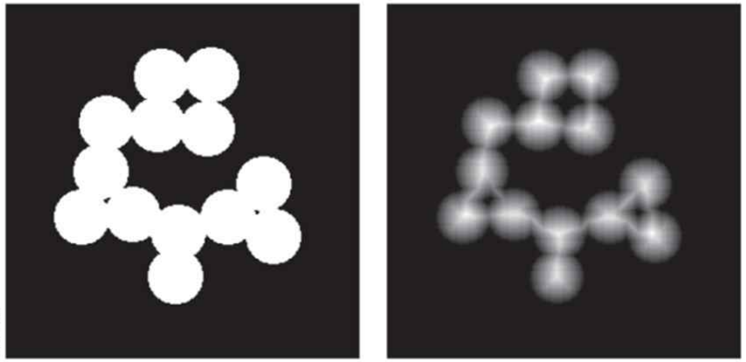
Distance Transform in Matlab

- Performing distance transform with 'mask1', 'mask2' and 'mask3'

```
>> disttrans(im,mask1)

ans =

     3     2     1     1     2     3
     2     1     0     0     1     2
     3     2     1     0     1     2
     3     2     1     0     1     2
     3     2     1     0     0     1
     4     3     2     1     1     2

>> disttrans(im,mask2)

ans =

     7     4     3     3     4     7
     6     3     0     0     3     6
     7     4     3     0     3     6
     8     6     3     0     3     4
     9     6     3     0     0     3
    10     7     4     3     3     4

>> disttrans(im,mask3)

ans =

    11     7     5     5     7    11
    10     5     0     0     5    10
    11     7     5     0     5    10
    14    10     5     0     5     7
    15    10     5     0     0     5
    16    11     7     5     5     7
```

Distance Transform Example

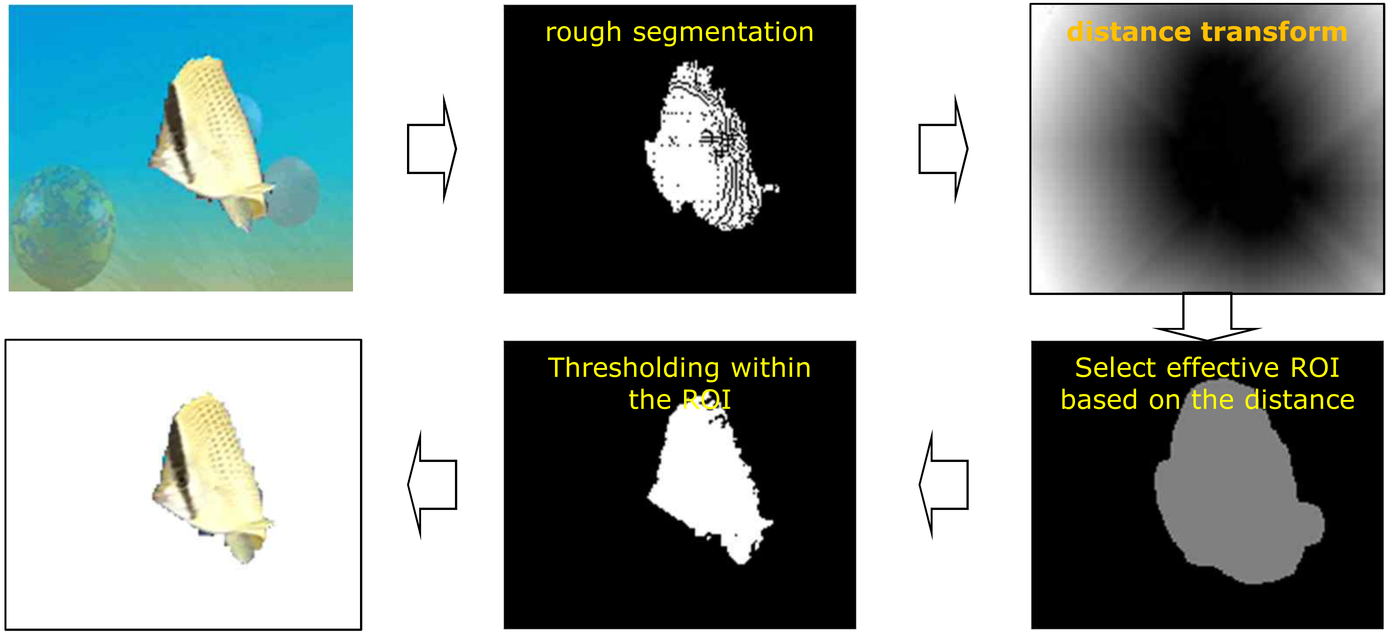
```
>> c=~imread('circles.tif');
>> imshow(c)
>> cd=disttrans(c,mask1);
>> figure,imshow(mat2gray(cd))
```

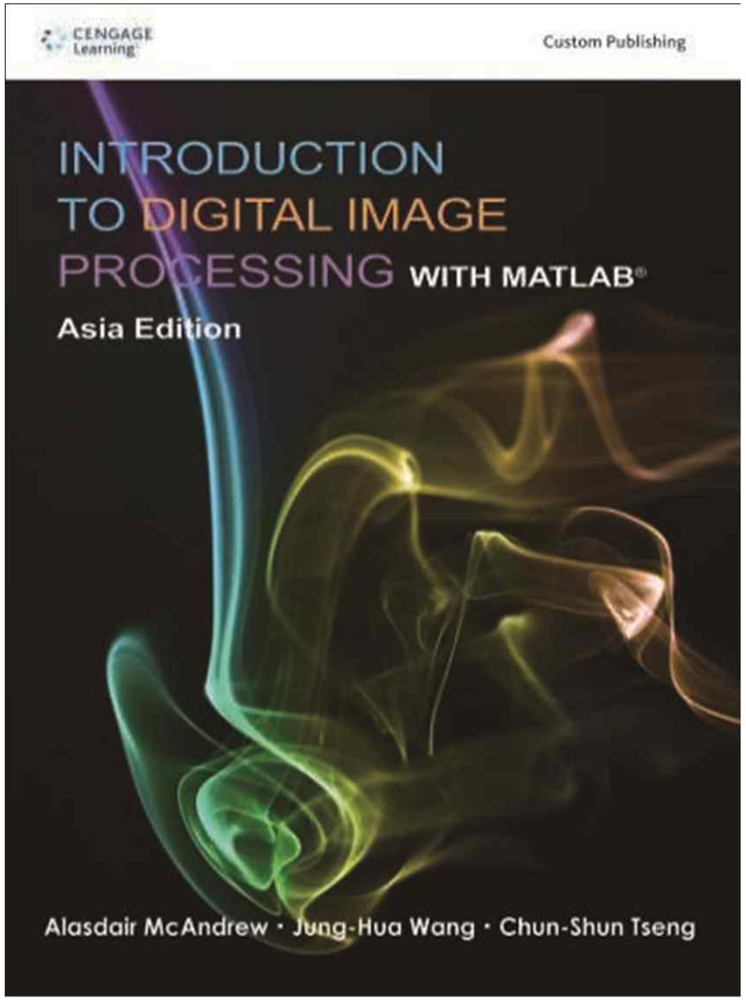


CRC Press

(a) (b)

- Used in image segmentation [Kim, et. al, IEEE CSVT, 2000]





Chapter 12:

Shapes and Boundaries

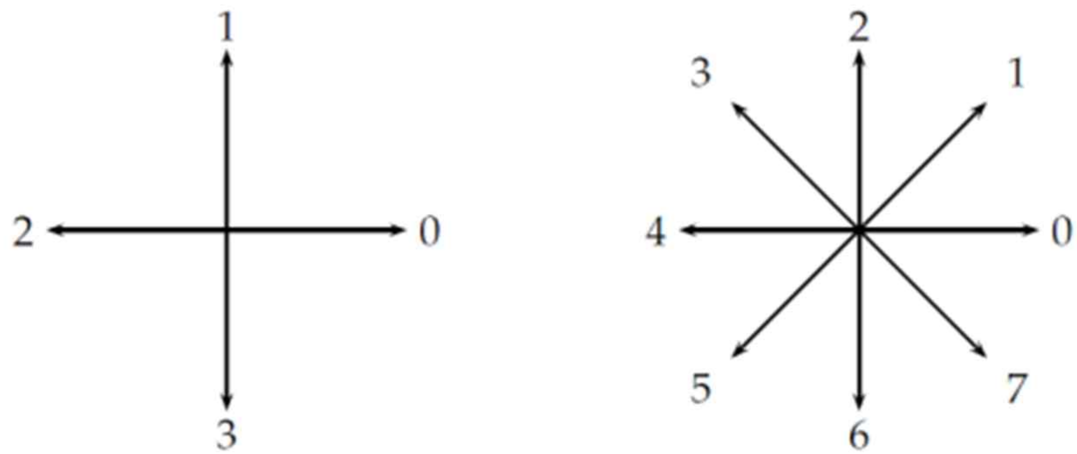
Introduction

- How do we tell if two objects have the same shape?
- How can we classify shape?
- How can we describe the shape of an object?

→ By using the **shape descriptors** which may include size, symmetry, and length of perimeter, etc.

Chain Codes

- The idea of a chain code is quite straightforward:
  - ✓ We walk around the boundary of an object, taking note of the direction we take.
  - ✓ The resulting list of directions is the chain code.



Directions for 4-connectedness   Directions for 8-connectedness

Freeman chain code

Chain Code Example

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

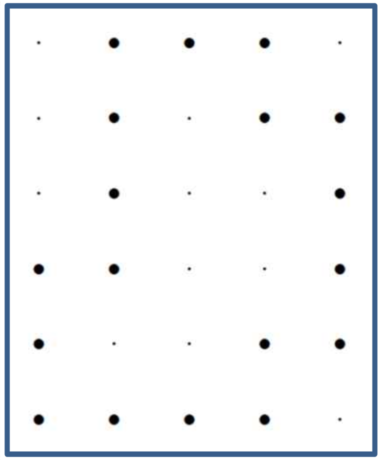


FIGURE 12.2 *A 4-connected object and its boundary.*

- Suppose we walk along the boundary in a **counter-clockwise** direction starting at the leftmost point in the top row and list the directions as we go like:

  3 3 3 2 3 3 0 0 0 1 0 1 1 1 2 1 2 2

  6 6 5 6 6 0 0 0 1 2 2 2 3 4 4

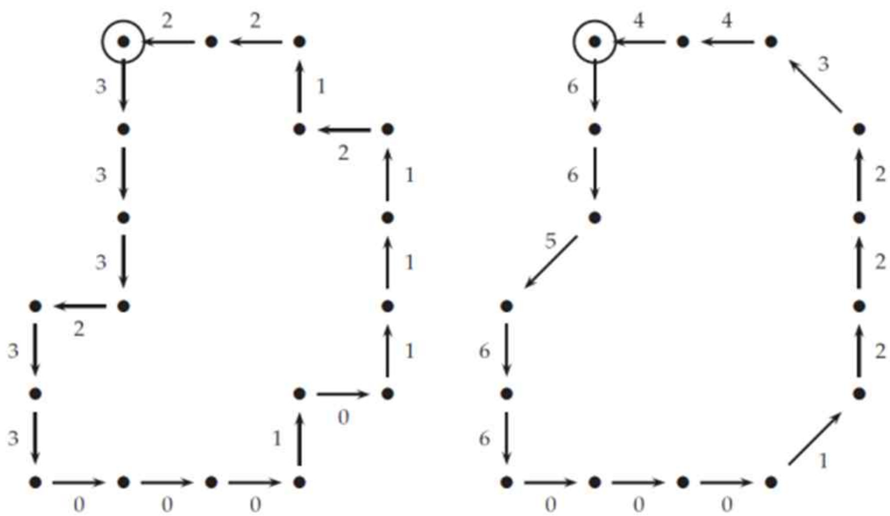
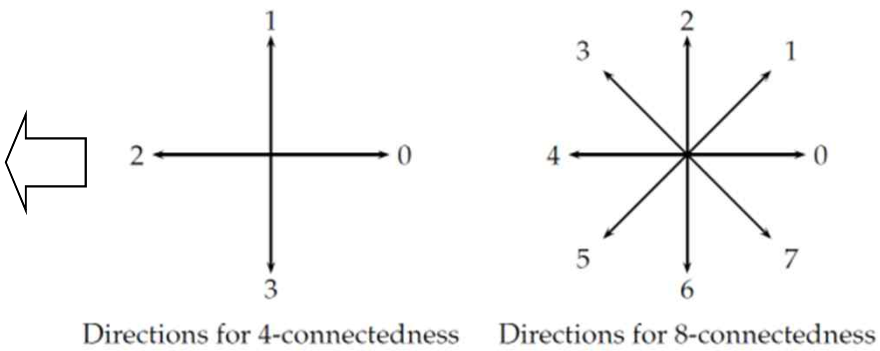


FIGURE 12.3 *Obtaining the chain code from the object in Figure 12.2.*



Freeman chain code

Chain Code in Matlab: chaincode4()

```
function out=chaincode4(image)

n=[0 1;-1 0;0 -1;1 0];

flag=1;
cc=[];
[x y]=find(image==1);
x=min(x);
imx=image(x,:);
y=min(find(imx==1));
first=[x y];
dir=3;

while flag==1,
  tt=zeros(1,4);
  newdir=mod(dir+3,4);
  for i=0:3,
    j=mod(newdir+i,4)+1;
    tt(i+1)=image(x+n(j,1),y+n(j,2));
  end
  d=min(find(tt==1));
  dir=mod(newdir+d-1,4);
  cc=[cc,dir];
  x=x+n(dir+1,1);y=y+n(dir+1,2);
  if x==first(1) & y==first(2)
    flag=0;
  end;
end;

out=cc;
```

FIGURE 12.5 *A MATLAB function for obtaining the chain code of a 4-connected object.*

Chain Codes in Matlab

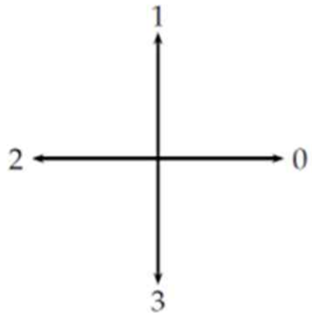
```
>> test=[0 0 0 0 0 0 0;...
         0 0 1 1 1 0 0;...
         0 0 1 1 1 1 0;...
         0 0 1 1 1 1 0;...
         0 1 1 1 1 1 0;...
         0 1 1 1 1 1 0;...
         0 1 1 1 1 0 0;...
         0 0 0 0 0 0 0];

>> chaincode4(test)

  ans =

    Columns 1 through 12

       3   3   3   2   3   3   0   0   0   1   0   1

    Columns 13 through 18

       1   1   2   1   2   2
```

For chaincode8(), see the text book pp.362

First Difference of Chain Codes

- There are two problems with the definition of the chain code as given in previous sections:
  1. The chain code is dependent on the starting pixel.
  2. The chain code is dependent on the orientation of the object.

  → needs rotation normalization



✓ Solution: first difference of chain codes

1. Count the direction difference in counter-clockwise of Freeman chain code between two adjacent chain codes elements.

   e.g. 1st difference of the chain codes 10103322 is 3133030.

2. in front of the 1st difference sequence, add the direction difference of last and first elements.

   e.g. direction difference between 2 and 1 → 3. Thus final codes are 33133030

First Difference of Chain Codes in Circular Representation

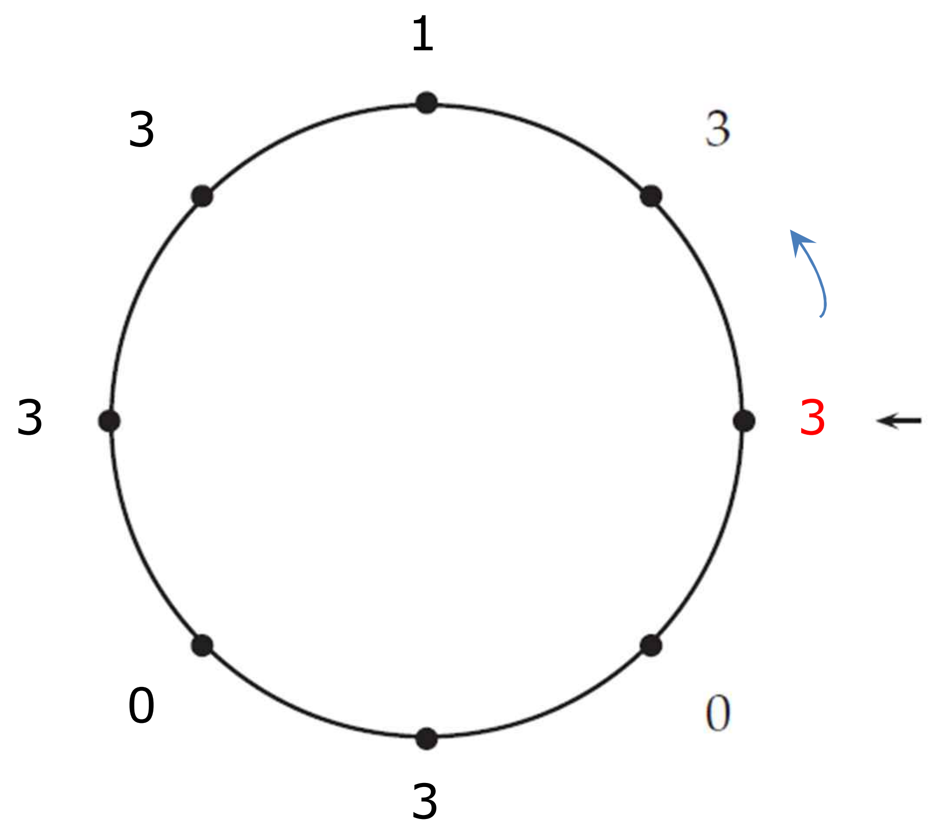


FIGURE 12.7 *A chain code written cyclically.*

Drawback of 1st Diff.

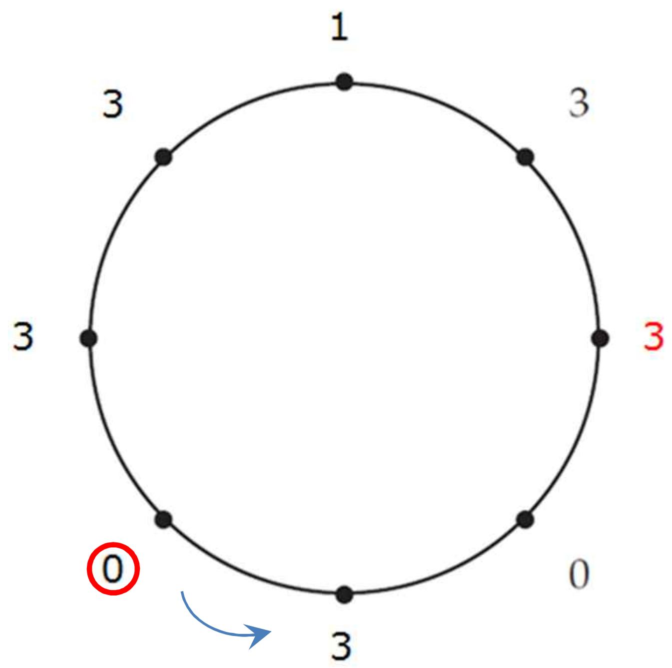
- This method is exact only if boundaries are invariant to rotation and scale, which is very rare in the real world applications.
- If the orientation of object is changed, the starting point is also changed, which two objects are not recognized as the same objects.

→ needs improvement

→ shape number

Shape Numbers

- ✓ We now consider the problem of defining a chain code that is **independent** on the **orientation** of the object.
- ✓ We define a shape number by circular-shifting so that the first difference of chain codes becomes the minimum value as a whole.

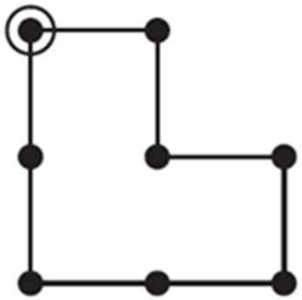


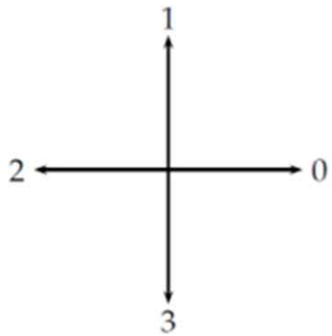
10103322

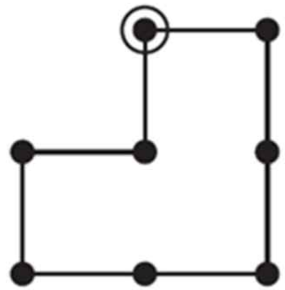
33133030

03033133

Shape Number Example







3 3 0 0 1 2 1 2	chaincode	3 2 3 0 0 1 1 2
0 1 0 1 1 3 1 1	1st difference	3 1 1 0 1 0 1 1
0 1 0 1 1 3 1 1	shape no.	0 1 0 1 1 3 1 1

SAME SHAPE !

Shape Numbers in Matlab

The function normalize(c) circular-shifts the sequence 'c' so that it becomes the minimum value as a whole.

```
function out=normalize(c)
%
% NORMALIZE returns the vector which is the least integer of all cyclic
% shifts of V.
%
m=c;
lc=length(c);
for i=2:lc,m=[m;[m(i-1,lc),m(i-1,1:lc-1)]];end
ms=sortrows(m);
out=ms(1,:);
```

FIGURE 12.8 *A* MATLAB *function for normalizing a chain code.*

Shape Numbers in Matlab

```
>> L=zeros(7,6);L(2:6,2:3)=1;L(5:6,4:5)=1

L =

     0     0     0     0     0     0
     0     1     1     0     0     0
     0     1     1     0     0     0
     0     1     1     0     0     0
     0     1     1     1     1     0
     0     1     1     1     1     0
     0     0     0     0     0     0
```

```
>> L2=rot90(L)
```
% rotate L by 90°

```
L2 =

     0     0     0     0     0     0     0
     0     0     0     0     1     1     0
     0     0     0     0     1     1     0
     0     1     1     1     1     1     0
     0     1     1     1     1     1     0
     0     0     0     0     0     0     0
```

```
>> c=chaincode4(L)

c =

  Columns 1 through 11

     3     3     3     3     0     0     0     1     2     2     1

  Columns 12 through 14

     1     1     2
```

```
>> c2=chaincode4(L2)

c2 =

  Columns 1 through 11

     3     3     2     2     2     3     0     0     0     0     1

  Columns 12 through 14

     1     1     2
```

Chain codes are different !

Shape Numbers in Matlab

```
>> c=chaincode4(L);
>> lc=length(c);
>> c1=[c(2:lc) c(1)];
>> mod(c1-c,4)
```

1st difference

```
ans =

  Columns 1 through 11

     0     0     0     1     0     0     1     1     0     3     0

  Columns 12 through 14

     0     1     1

>> normalize(ans)
```

% shape number

```
  Columns 1 through 11

     0     0     0     1     0     0     1     1     0     3     0

  Columns 12 through 14

     0     1     1
```

```
>> c=chaincode4(L2);
>> lc=length(c);
>> c1=[c(2:lc) c(1)];
>> mod(c1-c,4)

ans =

  Columns 1 through 11

     0     3     0     0     1     1     0     0     0     1     0

  Columns 12 through 14

     0     1     1

>> normalize(ans)

  Columns 1 through 11

     0     0     0     1     0     0     1     1     0     3     0

  Columns 12 through 14

     0     1     1
```

SAME SHAPE !

Fourier Descriptors

- Suppose we walk around an object.

1. Instead of writing down the directions just like in chain codes, we write down the **boundary coordinates**.
2. The list of boundary coordinates (x,y) are converted into a list of **complex numbers** $z = x + yi$.
3. The Fourier transform of this list of complex numbers is a **Fourier descriptor** of the object.

- We can easily modify our function **`chaincode4.m`** to **`boundary4.m`** by replacing the lines.

```
cc=[cc,dir];
x=x+n(dir+1,1);y=y+n(dir+1,2);
```

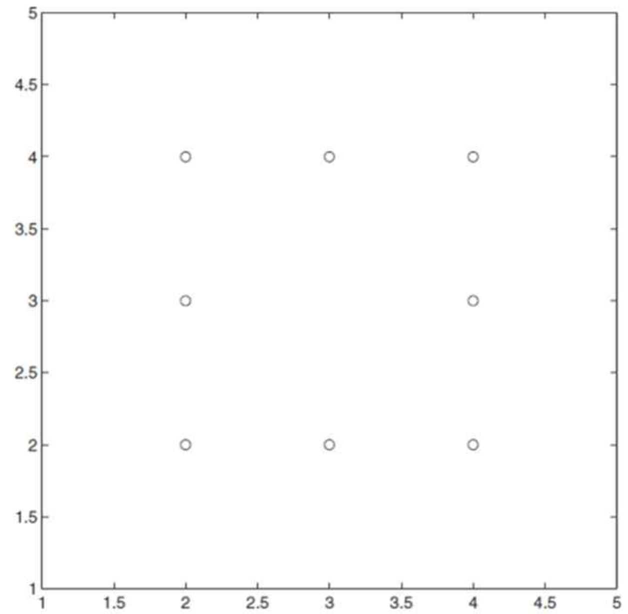
```
x=x+n(dir+1,1);y=y+n(dir+1,2);
cc=[cc;x y];
```

Fourier Descriptors:
1. boundary coord.

```
>> a=zeros(5,5);a(2:4,2:4)=1

a =

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

>> b=boundary4(a)

b =

     3     2
     4     2
     4     3
     4     4
     3     4
     2     4
     2     3
     2     2
```

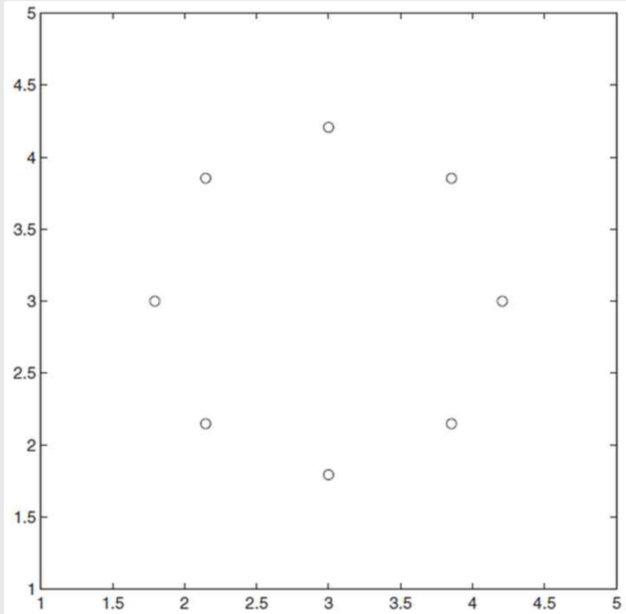
```
>> c=complex(b(:,1),b(:,2))

c =

   3.0000 + 2.0000i
   4.0000 + 2.0000i
   4.0000 + 3.0000i
   4.0000 + 4.0000i
   3.0000 + 4.0000i
   2.0000 + 4.0000i
   2.0000 + 3.0000i
   2.0000 + 2.0000i
```

```
>> plot(c,'o'),axis([1,5,1,5]),axis equal
```



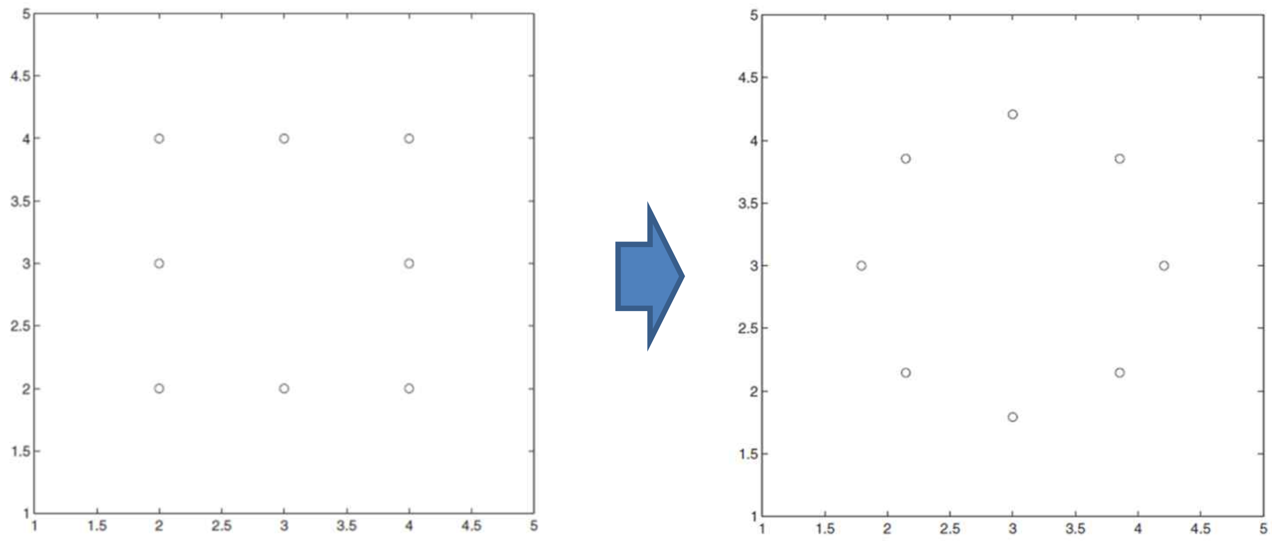
Fourier Descriptors:
2. fft of boundary coord.

```
>> f=fft(c)

f =

   24.0000 +24.0000i
         0 - 9.6569i
         0
         0
         0
         0 + 1.6569i
         0
         0

>> f1=zeros(size(f));
>> f1(1:2)=f(1:2);
>> plot(ifft(f1),'o'),axis([1.0,5.0,1.0,5.0]),axis square
```



Fourier Descriptors: 3. Discussion

- The Fourier transform of **c** contains only three nonzero terms.
- Only **two terms** of the transform are enough to begin to get some idea of the shape, size, and symmetry of the object.
- Even though the shape itself has been greatly changed(a square became a circle), many shape features are not changed. (such as size and symmetricity)

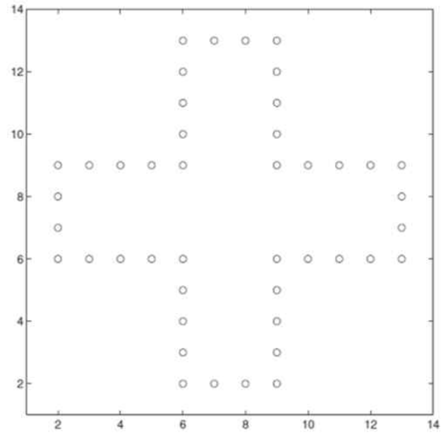


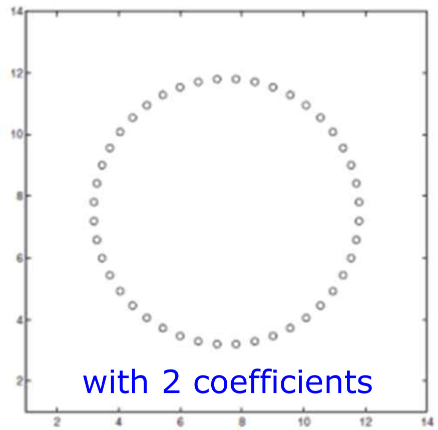
FD Example

```
>> a=zeros(14,14);a(6:9,2:13)=1;a(2:13,6:9)=1;
>> b=boundary4(a);
>> c=complex(b(:,1),b(:,2));
>> plot(c,'o'),axis([1,14,1,14]),axis square
```

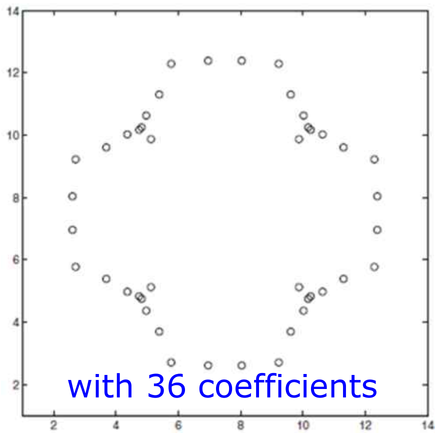
```
>> f=fft(c);
>> f1=zeros(size(f));
>> f1(1:2)=f(1:2);
>> plot(ifft(f1),'o'),axis([1,14,1,14]),axis square
```

```
>> f1(1:8)=f(1:8);
>> plot(ifft(f1),'o'),axis([1,14,1,14]),axis square
```

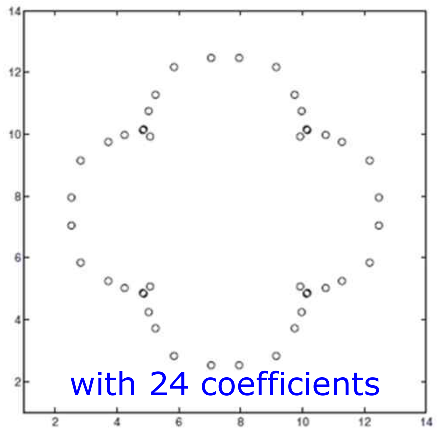




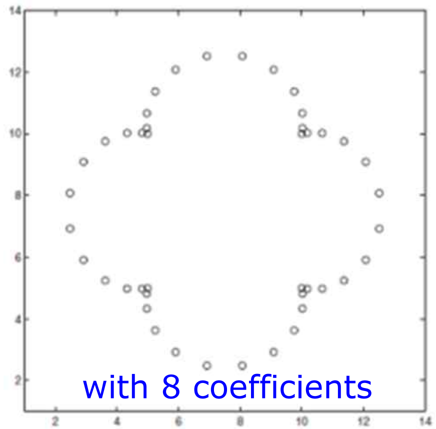
with 2 coefficients



with 36 coefficients



with 24 coefficients



with 8 coefficients

Summary

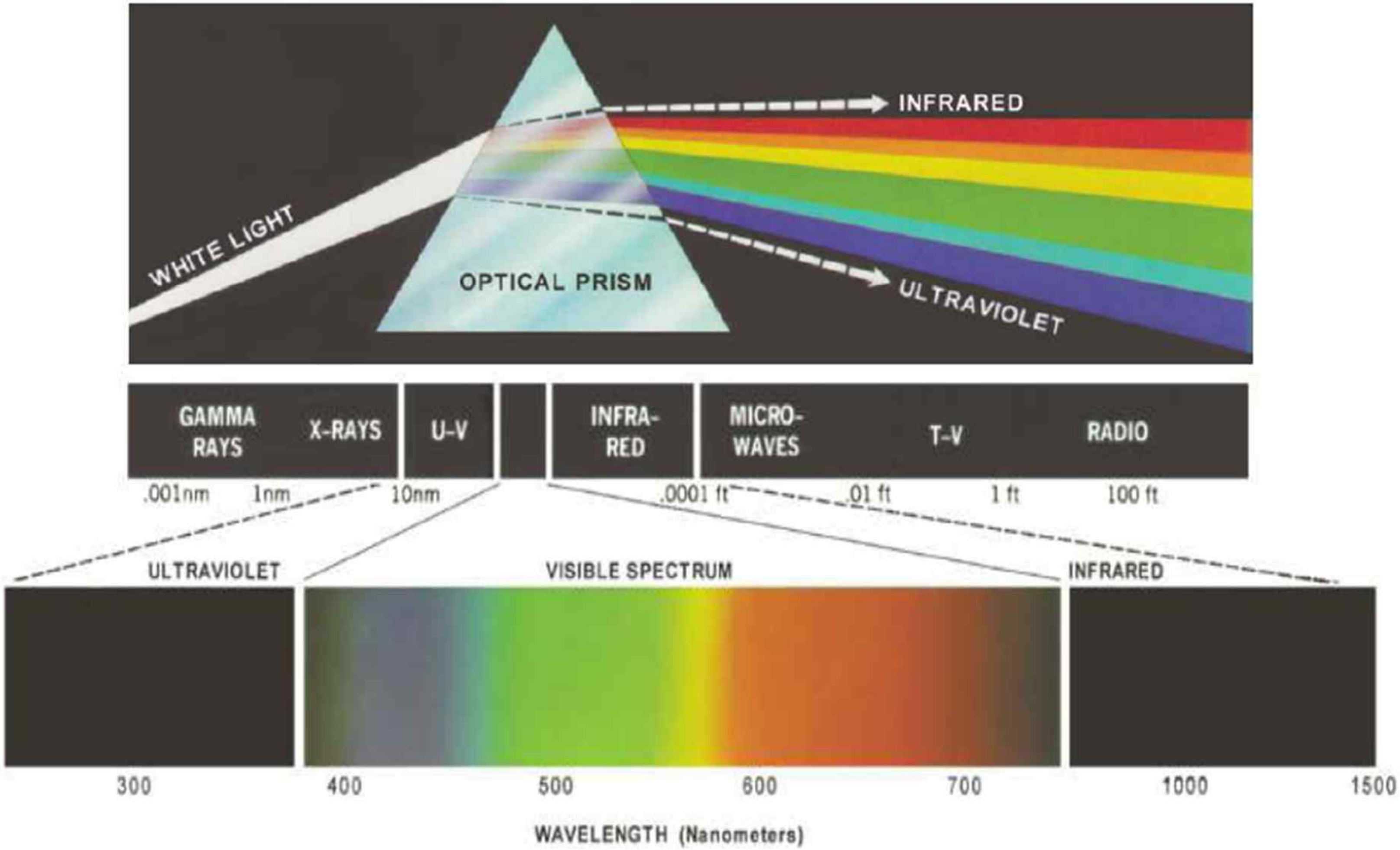
- **Chap 11. Image Topology**
  - 4- and 8-components
  - Component labeling
  - Distance transform

- **Chap 12. Shapes and Boundaries**
  - Chain codes
  - First difference of chain codes
  - Shape numbers
  - Fourier descriptors

Chapter 13:

Color Processing

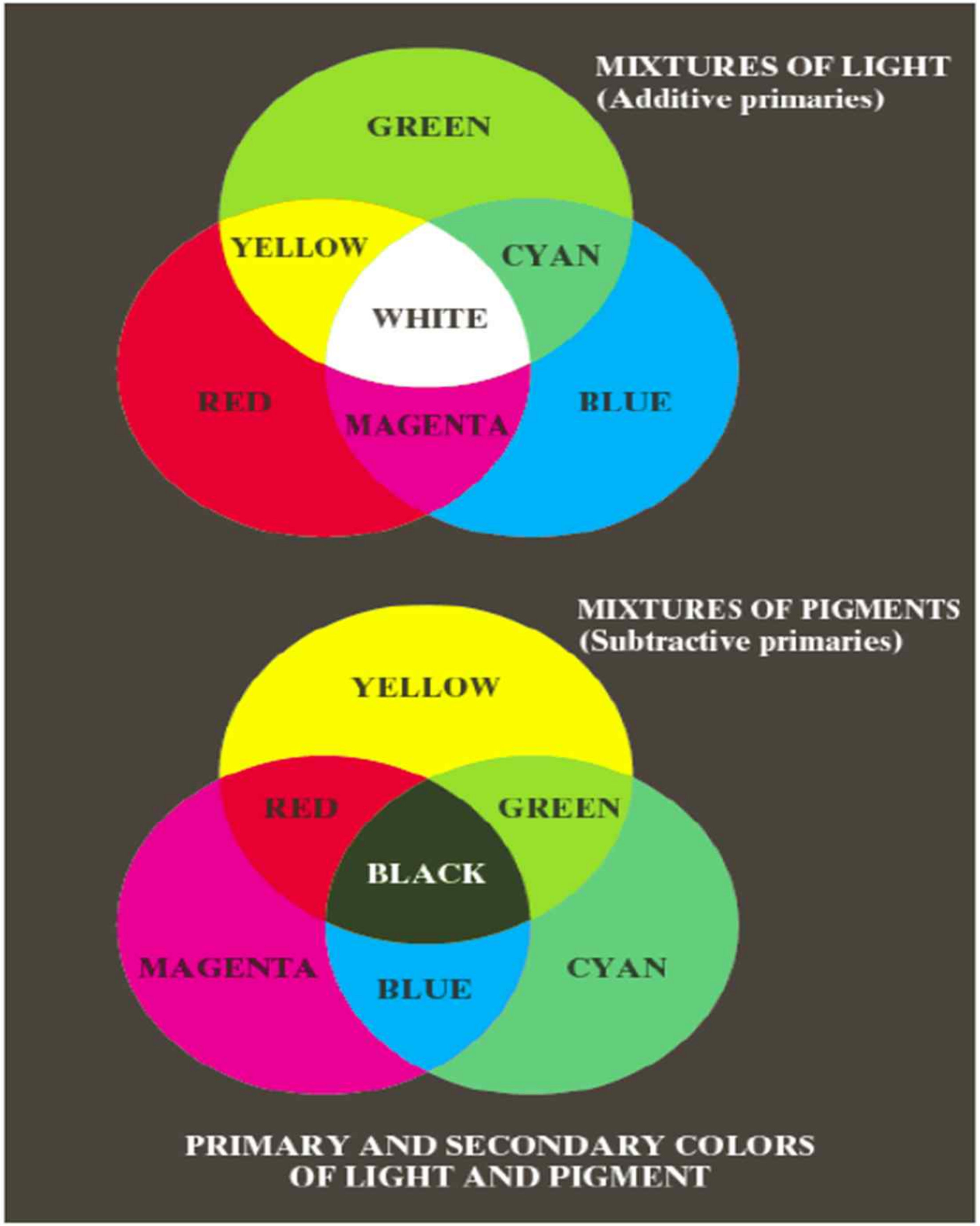
Color Spectrum



Commission Internationale de l'Eclairage (CIE) --
International Commission on Illumination adopted three primary colors:
R=700 nm, G=546.1 nm, B=435.8 nm.

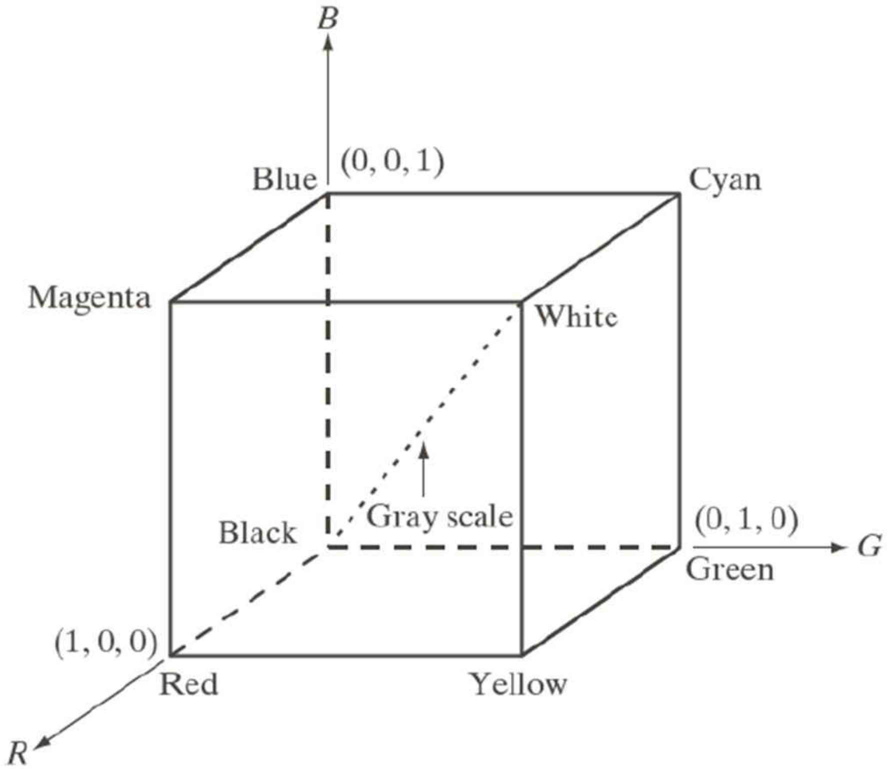
Primary & Secondary Colors of Light

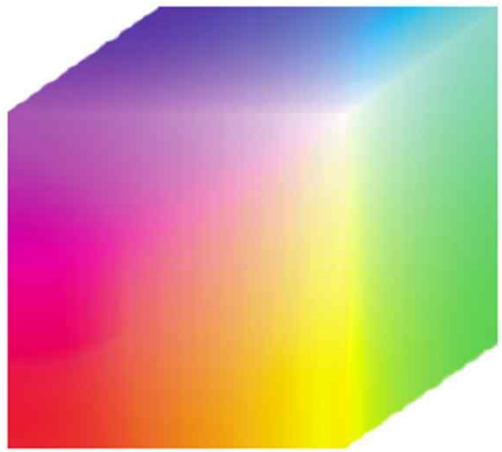
Primary & Secondary Colors of Pigments

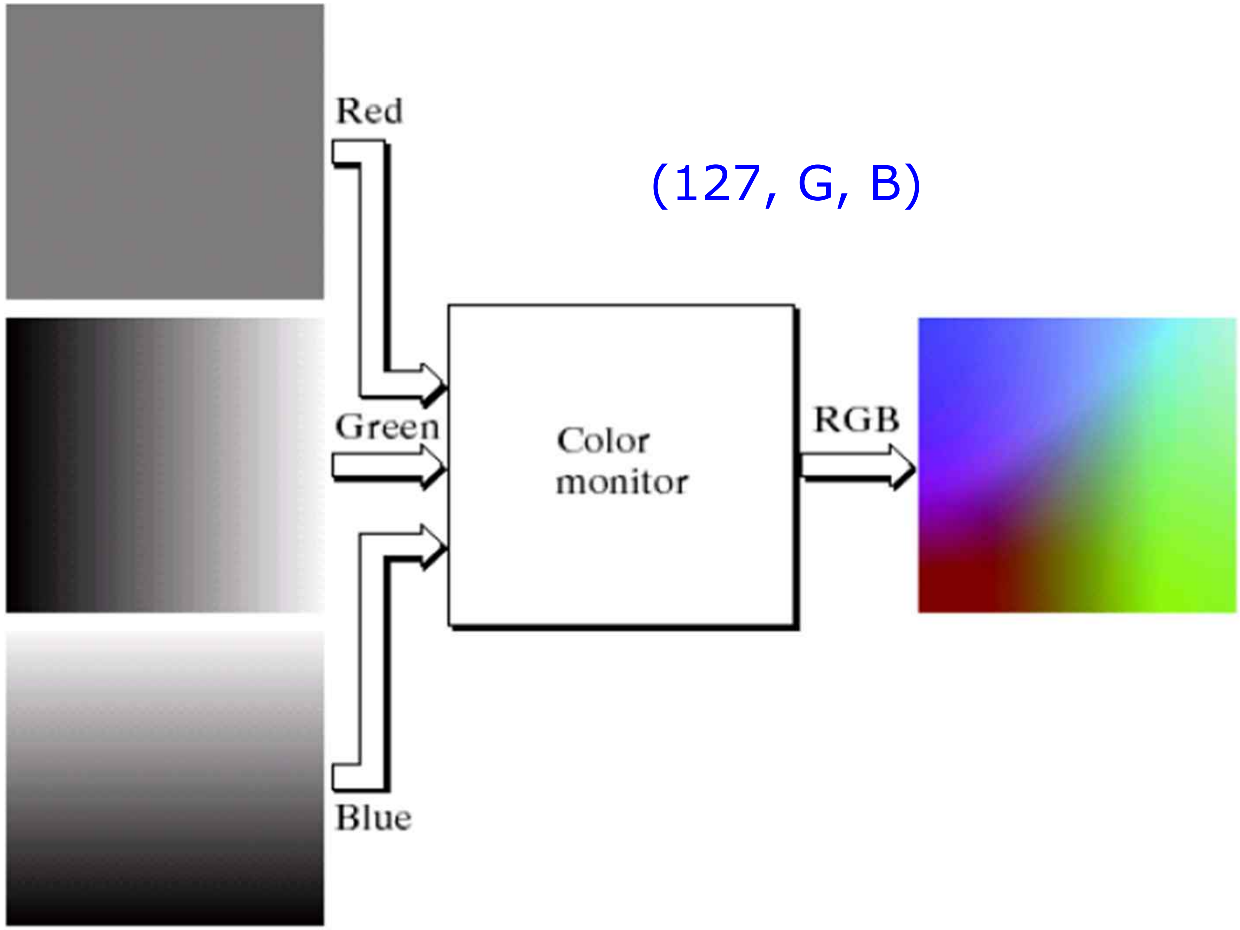


RGB Color Model

- The most widely used system (additive).
- Each pixel is represented as a **triplet, e.g., Red** (255,0,0), Yellow (255,255,0), Green (0,255,0), Cyan (0,255,255), Blue (0,0,255), Magenta (255,0,255).
- (0,0,0) denotes black, and (255,255,255) denotes white.
- (63,63,63), (127,127,127), etc, represent different shades of **gray**







(127, G, B)

3 hidden surface planes in the color cube of the previous slide

$(R = 0)$ $(G = 0)$ $(B = 0)$

HSV Color Model

Hue
- The “true color” attribute
- e.g. red, green, blue, orange, etc

Saturation
- The amount by which the color has been diluted with white.
- The more white in the color, the lower the saturation.

Value:
- The degree of brightness.
- A well-lit color has high intensity; a dark color has low intensity.

Closer to human perception than the other color models

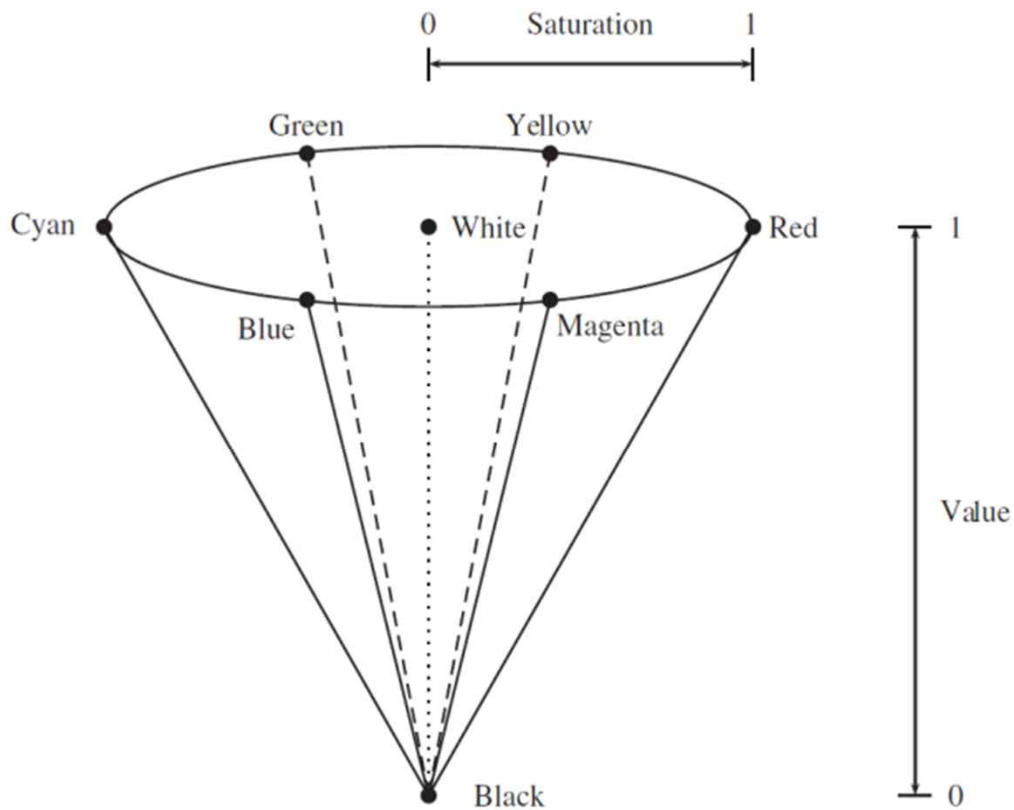


FIGURE 13.6 *The color space HSV as a cone.*

RGB to HSV Conversion

V and S first !

$$V = \max\{R, G, B\}$$

$$\delta = V - \min\{R, G, B\}$$

$$S = \frac{\delta}{V}$$

Then H.

1. If $R = V$ then $H = \frac{1}{6}\frac{G-B}{\delta}$,
2. If $G = V$ then $H = \frac{1}{6}\left(2 + \frac{B-R}{\delta}\right)$,
3. If $B = V$ then $H = \frac{1}{6}\left(4 + \frac{R-G}{\delta}\right)$.

```
>> rgb2hsv([0.2 0.4 0.6])

ans =

    0.5833      0.6667      0.6000
```

Color	Hue
Red	0
Yellow	0.1667
Green	0.3333
Cyan	0.5
Blue	0.6667
Magenta	0.8333

HSV to RGB Conversion

$H' = \lfloor 6H \rfloor$: floor(), 0 <= H < 1.0

$F = 6H - H'$

$P = V(1 - S)$

$Q = V(1 - SF)$

$T = V(1 - S(1 - F))$

H'	R	G	B
0	V	T	P
1	Q	V	P
2	P	V	T
3	P	Q	V
4	T	P	V
5	V	P	Q

Example:

```
>> rgb2hsv([0.2 0.4 0.6])

ans =

    0.5833    0.6667    0.6000
```

$H' = \lfloor 6(0.5833) \rfloor = 3$

$F = 6(0.5833) - 3 = 0.5$

$P = 0.6(1 - 0.6667) = 0.2$

$Q = 0.6(1 - (0.6667)(0.5)) = 0.4$

$T = 0.6(1 - 0.6667(1 - 0.5)) = 0.4$

$$(R, G, B) = (P, Q, V) = (0.2, 0.4, 0.6)$$

YIQ Color Model

- ✓ This color space is used for TV and video in the United States and other countries where NTSC sets the video standard.
- ✓ In this scheme, *Y* is the luminance (this corresponds roughly with intensity).
- ✓ *I* and *Q* carry the color information.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

Color Images in MATLAB

```
>> x=imread('lily.tif');
>> size(x)

ans =

   186   230     3
```

```
>> figure,imshow(x(:,:,1))
>> figure,imshow(x(:,:,2))
>> figure,imshow(x(:,:,3))
```

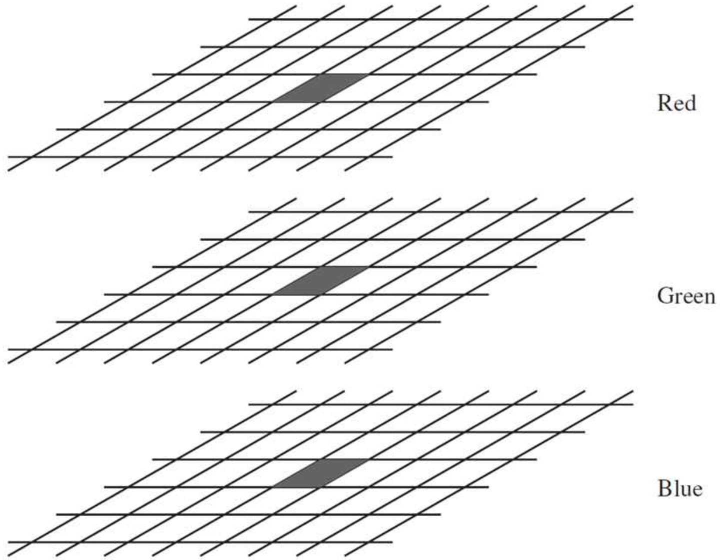


FIGURE 13.8 *A three-dimensional array for an RGB image.*

Red component

Green component

Blue component

Color Images in MATLAB

```
>> xh=rgb2hsv(x);
>> imshow(xh(:,:,1))
>> figure,imshow(xh(:,:,2))
>> figure,imshow(xh(:,:,3))
```

Hue Saturation Value

```
>> xn=rgb2ntsc(x); % RGB -> YIQ
>> imshow(xn(:,:,1))
>> figure,imshow(xn(:,:,2))
>> figure,imshow(xn(:,:,3))
```

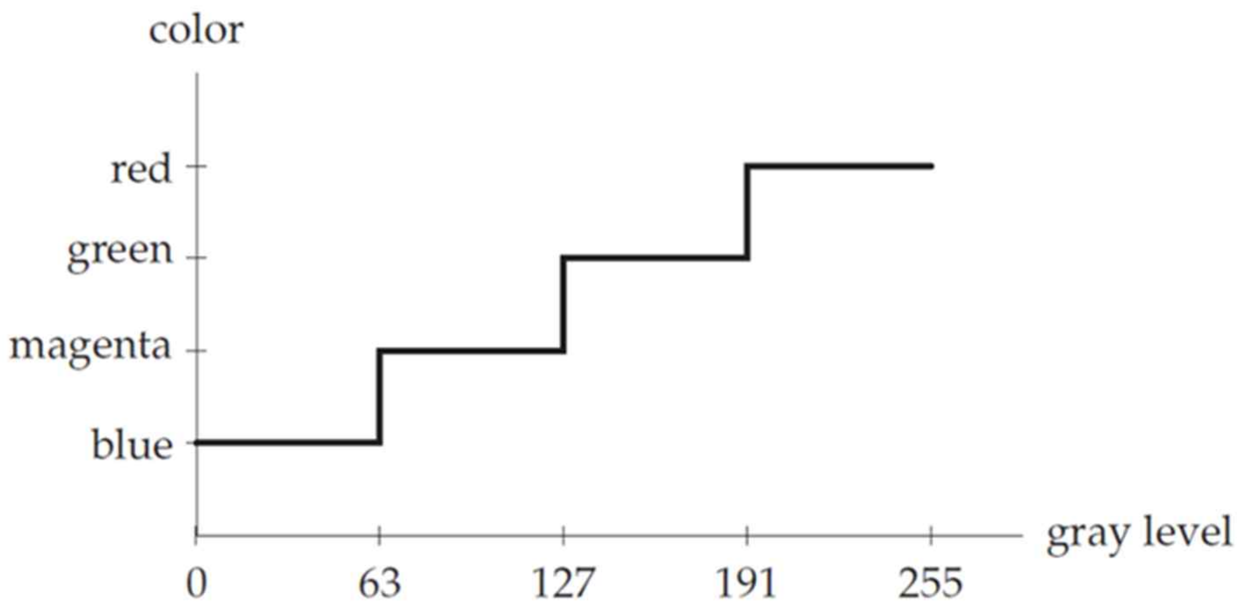
Y I Q

Pseudo-coloring: Intensity Slicing

1. Break up the image into various gray level ranges
2. Simply assign a different color to each range.

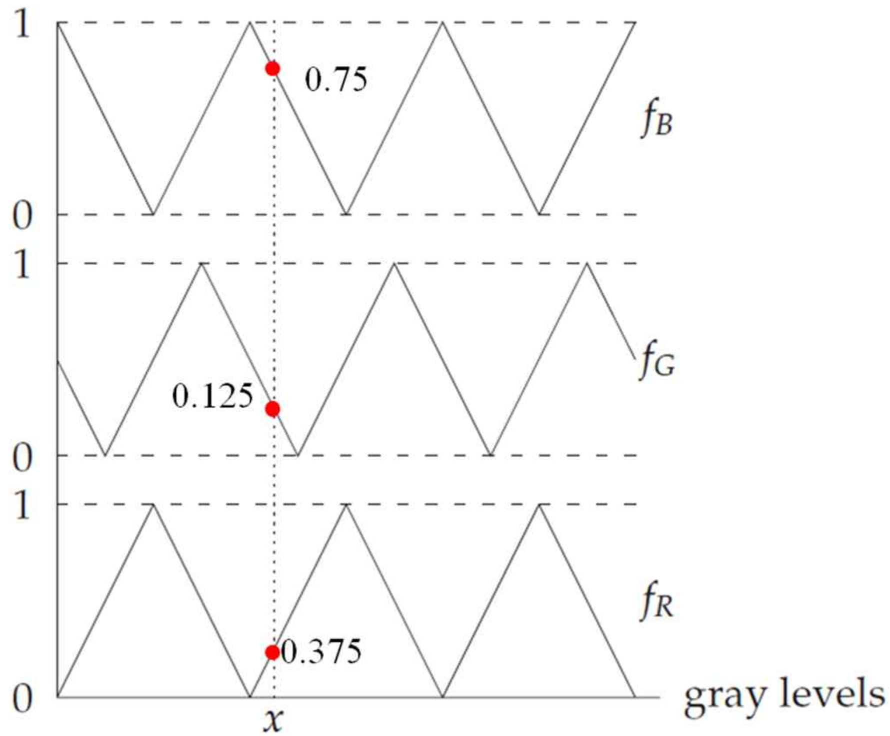
✓ For example,

gray level:	0–63	64–127	128–191	192–255
color:	blue	magenta	green	red



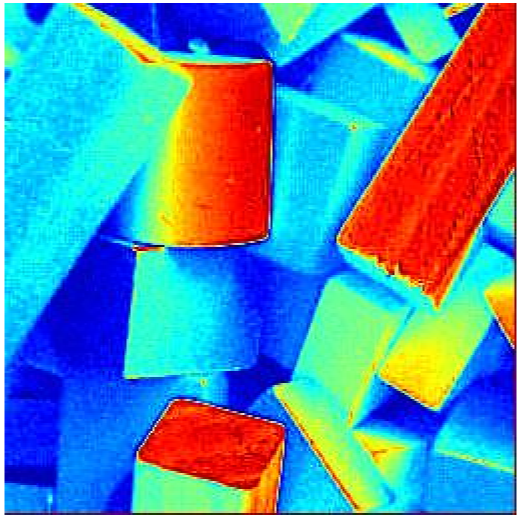
Pseudocoloring: Gray to Color Transformations

- ✓ We have three functions, $f_R(x)$, $f_G(x)$, $f_B(x)$, that assign red, green, and blue values to each gray level x.
- ✓ These values (with appropriate scaling, if necessary) are then used for display.
- ✓ Using an appropriate set of functions can enhance a grayscale image with impressive results.

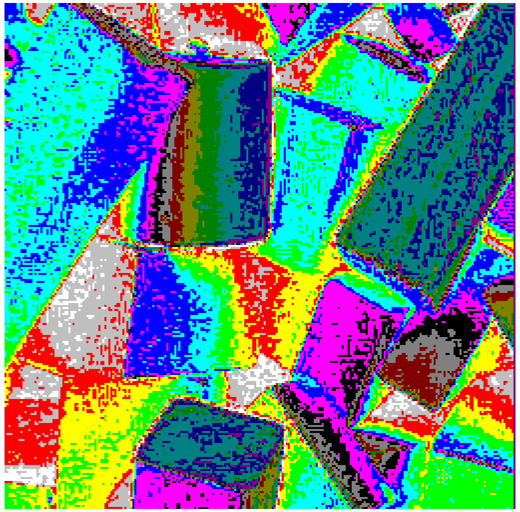


Pseudocoloring

```
>> b=imread('blocks.tif');
>> imshow(b,colormap(jet(256))
```



```
>> b16=grayslice(b,16);
>> figure,imshow(b16,colormap(vga))
```



not always working

```
>> help jet
```

JET(M), a variant of HSV(M), is an M-by-3 matrix containing the default colormap used by CONTOUR, SURF and PCOLOR.

The colors begin with dark blue, range through shades of blue, cyan, green, yellow and red, and end with dark red.

```
>> help vga
```

VGA The Windows color map for 16 colors

VGA returns a 16-by-3 matrix containing the colormap used by Windows for 4-bit color.

For example, to reset the colormap of the current figure: colormap(vga)

Pseudocoloring

- The available color maps are listed in the help file for `graph3d()`

```
hsv         - Hue-saturation-value color map.
hot         - Black-red-yellow-white color map.
gray        - Linear grayscale color map.
bone        - Grayscale with tinge of blue color map.
copper      - Linear copper-tone color map.
pink        - Pastel shades of pink color map.
white       - All white color map.
flag        - Alternating red, white, blue, and black color map.
lines       - Color map with the line colors.
colorcube   - Enhanced color-cube color map.
vga         - Windows color map for 16 colors.
jet         - Variant of HSV.
prism       - Prism color map.
cool        - Shades of cyan and magenta color map.
autumn      - Shades of red and yellow color map.
spring      - Shades of magenta and yellow color map.
winter      - Shades of blue and green color map.
summer      - Shades of green and yellow color map.
```

Processing of Color Images

- Method 1:

  We can process each RGB matrix separately.

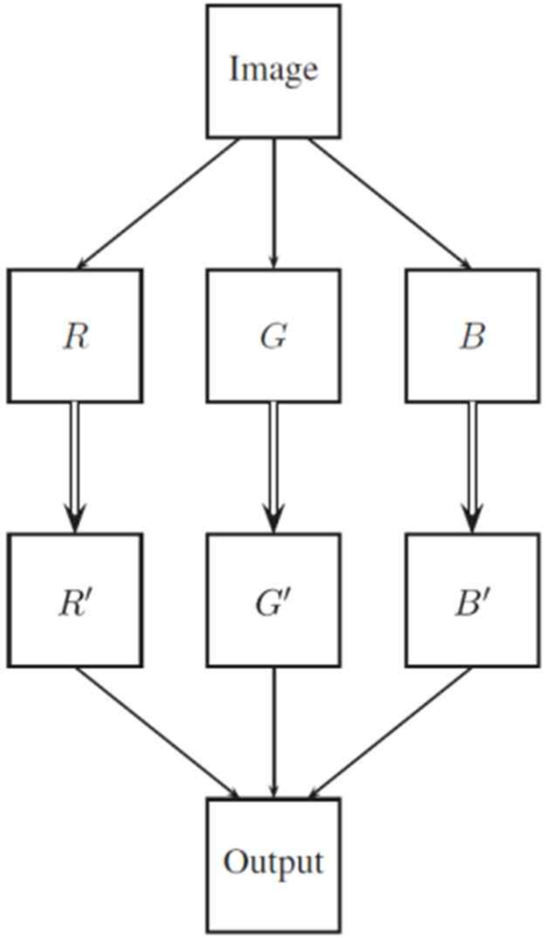


FIGURE 13.13 *RGB processing.*

Processing of Color Images

- Method 2:

1. RGB to YIQ:

   The intensity is separated from the color information.

2. Process the intensity component only.
3. Y'IQ to R'G'B'

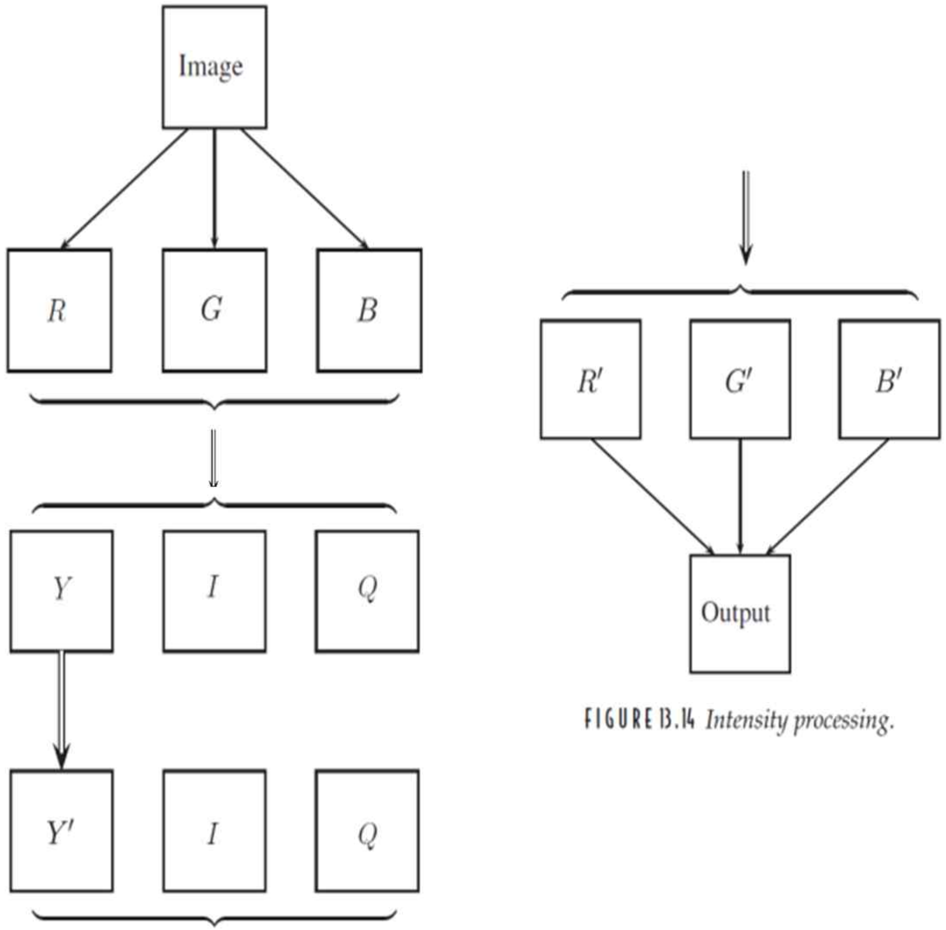


FIGURE 13.14 *Intensity processing.*

Processing of Color Images: Contrast Enhancement by HE

✓ Best done by processing the intensity component

```
>> [x,map]=imread('canoe.tif');
>> c=ind2rgb(x,map);
```

RGB conversion

```
>> cn=rgb2ntsc(c);
```

Intensity conversion


```
>> cn(:,:,1)=histeq(cn(:,:,1));
>> c2=ntsc2rgb(cn);
>> imshow(c2)
```

Contrast Enhancement by Method 1

```
>> cr=histeq(c(:,:,1));
>> cg=histeq(c(:,:,2));
>> cb=histeq(c(:,:,3));
```

```
>> c3=cat(3,cr,cg,cb);
>> imshow(c3)
```

```
CAT Concatenate arrays.
CAT(DIM,A,B) concatenates the
arrays A and B along the
dimension DIM.
```

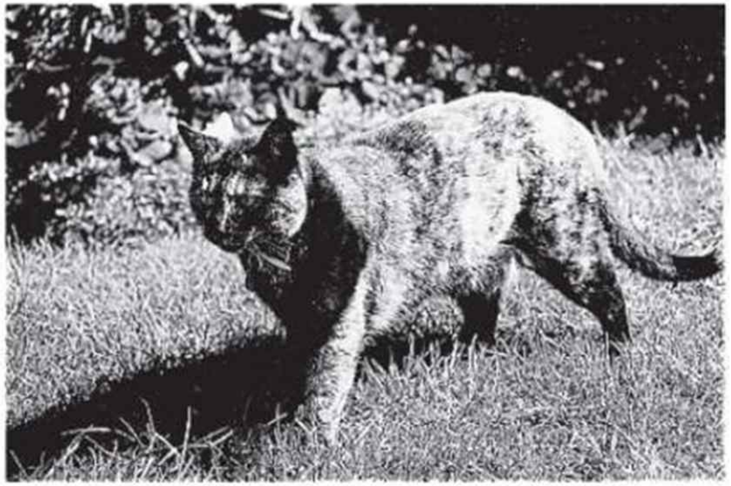
Processing of Color Images: **Spatial Filtering**

- ✓ The schema we use depends on the filter.
- ✓ For a low-pass filter, say a blurring filter, we can apply the filter to each RGB component. -> Method 1
- ✓ For a HPF, Method 2 works better. <- Edge at three planes can change the color.

```
>> a15=fspecial('average',15);
>> cr=filter2(a15,c(:,:,1));
>> cg=filter2(a15,c(:,:,2));
>> cb=filter2(a15,c(:,:,3));
>> blur=cat(3,cr,cg,cb);
>> imshow(blur)
```

```
>> cn=rgb2ntsc(c);
>> a=fspecial('unsharp');
>> cn(:,:,1)=filter2(a,cn(:,:,1));
>> cu=ntsc2rgb(cn);
>> imshow(cu)
```

Low pass filtering



High pass filtering

FIGURE 13.15 *Spatial filtering of a color image (shown in grayscale).*

Processing of Color Images: **Noise Reduction**

```
>> tw=imread('twins.tif');
```

```
>> tn=imnoise(tw,'salt & pepper');
>> figure,imshow(tn(:,:,1))
>> figure,imshow(tn(:,:,2))
>> figure,imshow(tn(:,:,3))
```

(a) (b) (c)

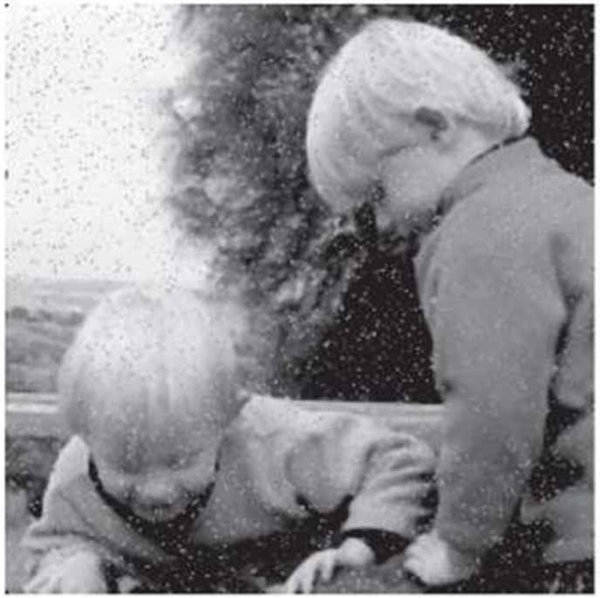
FIGURE 13.16 *Components of a noisy color image. (a) The red component. (b) The green component. (c) The blue component.*

Processing of Color Images: **Noise Reduction**

```
>> trm=medfilt2(tn(:,:,1));
>> tgm=medfilt2(tn(:,:,2));
>> tbm=medfilt2(tn(:,:,3));
>> tm=cat(3,trm,tgm,tbm);
>> imshow(tm)
```

```
>> tnn=rgb2ntsc(tn);
>> tnn(:,:,1)=medfilt2(tnn(:,:,1));
>> tm2=ntsc2rgb(tnn);
>> imshow(tm2)
```

(a) Method 1: better !



(b) Method 2

FIGURE 13.17 *Attempts at de-noising a color image. (a) De-noising each RGB component. (b) De-noising Y only.*

Processing of Color Images: **Edge Detection**

- ✓ Method 1: Apply the `edge` function to each of the RGB components and join the results.
- ✓ Method 2: Take the intensity component only and apply the edge function to it.

```
>> fg=rgb2gray(f);
>> fe1=edge(fg);
>> imshow(fe1)
```

```
>> f1=edge(f(:,:,1));
>> f2=edge(f(:,:,2));
>> f3=edge(f(:,:,3));
>> fe2=f1 | f2 | f3;
>> figure,imshow(fe2)
```

CRC Press

`fe1`: Edges after `rgb2gray`

Method 2

`fe2`: Edges of each RGB component

Method 1

FIGURE 13.18 *The edges of a color image.*