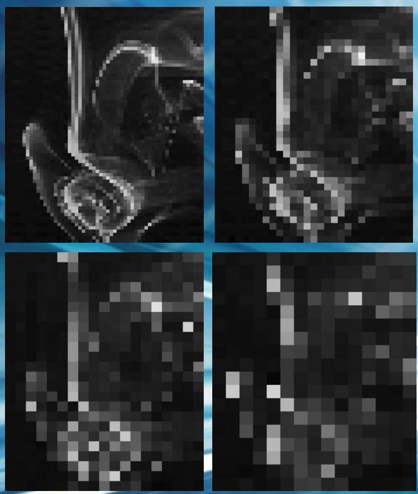
Introduction to Digital Image Processing with MATLAB® Asia Edition

McAndrew‧Wang‧Tseng



Chapter 3: Image Display

CENGAGE Learning

3.1 Introduction

- We take a closer look at the use of the `imshow` function
- We look at image quality and how that may be affected by various image attributes
- For human vision in general, the preference is for images to be sharp and detailed

Ch3-p.41

CENGAGE Learning

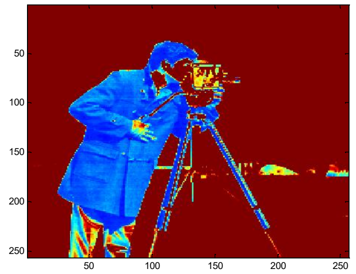
3.2 Basics of Image Display

- There are many factors that will affect the display
  - ✓ ambient lighting,
  - ✓ the monitor type and settings,
  - ✓ the graphics card, and
  - ✓ monitor resolution

3.2 Basics of Image Display

- This function `image` simply displays a matrix as an image

```
>> c=imread('cameraman.tif');
>> image(c)
```



3.2 Basics of Image Display

- To display the image properly, we need to add several extra commands to the `image` line

```
>> size(unique(c))

ans =

   247     1
```

```
>> image(c),truesize,axis off, colormap(gray(247))
```

3.2 Basics of Image Display

- We may to adjust the color map to use fewer or more colors; however, this can have a dramatic effect on the result

`colormap(gray(512))`

`colormap(gray(128))`

3.2 Basics of Image Display

- use `imread` to pick up the color map

```
>> [x,map]=imread('trees.tif');
>> image(x),truesize,axis off,colormap(map)
```

✓ `map` is <256×3 double> in the **workspace**

3.2 Basics of Image Display

- True color image will be read (by `imread`) as a three-dimensional array

✓ In such a case, `image` will ignore the current color map and assign colors to the display based on the values in the array

```
>> t=imread('twins.tif');
>> image(t),truesize,axis off
```

3.3 The imshow Function

- We have two choices with a matrix of type `double`:
  - ✓ Convert to type `uint8` and then display
  - ✓ Display the matrix directly
- `imshow` will display a matrix of type `double` as a grayscale image (matrix elements are between 0 and 1)

FIGURE 3.1

```
>> c=imread('caribou.tif');
>> cd=double(c);
>> imshow(c),figure,imshow(cd)
```

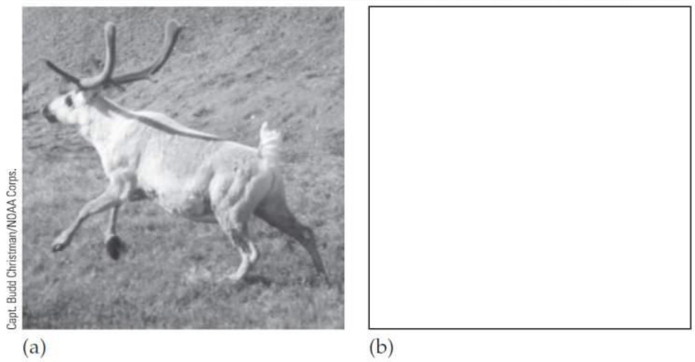


FIGURE 3.1 *An attempt at data type conversion. (a) The original image. (b) After conversion to type* `double`.

FIGURE 3.2

```
>> imshow(cd/512)
>> imshow(cd/128)
```

Capt. Budd Christman/NOAA Corps.

(a) (b)

FIGURE 3.2 *Scaling by dividing an image matrix by a scalar. (a) The matrix* cd *divided by 512. (b) The matrix* cd *divided by 128.*

3.3 The imshow Function

- We can convert the original image to `double` more properly using the function `im2double`

```
>> cd=im2double(c);
```

- If we take `cd` of type `double`, properly scaled so that all elements are between 0 and 1, we can convert it back to an image of type `uint8` in two ways:

```
>> c2=uint8(255*cd);
>> c3=im2uint8(cd);
```

3.3 The imshow Function

- **BINARY IMAGES** MATLAB have a `logical` flag, where `uint8` values 0 and 1 can be interpreted as logical data

```
>> cl=c>120;
```

- Check `c1` with `whos`

```
Name        Size          Bytes        Class        Attributes
c1          256x256       65536        logical
```

13 **Ch3-p.46**

CENGAGE Learning

FIGURE 3.3

```
>> imshow(cl)          >> imshow(uint8(cl))
```

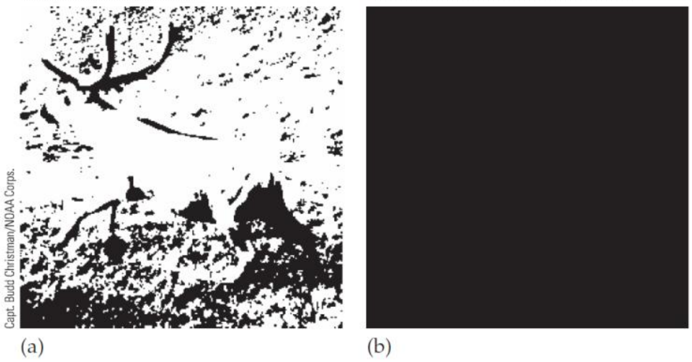


FIGURE 3.3 *Making the image binary. (a) The caribou image turned binary. (b) After conversion to type* `uint8`.

14 **Ch3-p.47**

CENGAGE Learning

3.4 Bit Planes

- Grayscale images can be transformed into a sequence of binary images by breaking them up into their **bitplanes**
- **The zeroth bit plane**
  - ✓ the least significant bit plane
- **The seventh bit plane**
  - ✓ the most significant bit plane

15 **Ch3-p.48**

3.4 Bit Planes

- We start by making it a matrix of type `double`; this means we can perform arithmetic on the values

```
>> c=imread('cameraman.tif');
>> cd=double(c);
>> c0=mod(cd,2);
>> c1=mod(floor(cd/2),2);
>> c2=mod(floor(cd/4),2);
>> c3=mod(floor(cd/8),2);
>> c4=mod(floor(cd/16),2);
>> c5=mod(floor(cd/32),2);
>> c6=mod(floor(cd/64),2);
>> c7=mod(floor(cd/128),2);
```

16 **Ch3-p.48**

FIGURE 3.4

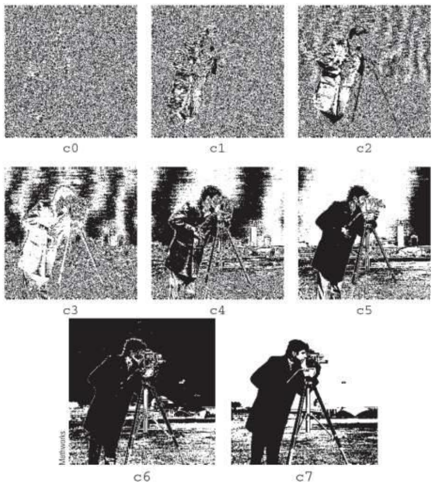


FIGURE 3.4 *The bit planes of an 8-bit grayscale image.*

17 **Ch3-p.49**

CENGAGE Learning

3.5 Spatial Resolution

- The greater the spatial resolution, the more pixels are used to display the image
- We can experiment with spatial resolution with MATLAB's `imresize` function

$$\begin{array}{ccccccc} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & \cdots \\ x_{21} & \boxed{x_{22}} & x_{23} & \boxed{x_{24}} & x_{25} & \boxed{x_{26}} & \cdots \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & \cdots \\ x_{41} & \boxed{x_{42}} & x_{43} & \boxed{x_{44}} & x_{45} & \boxed{x_{46}} & \cdots \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & \cdots \\ x_{61} & \boxed{x_{62}} & x_{63} & \boxed{x_{64}} & x_{65} & \boxed{x_{66}} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array} \rightarrow \texttt{imresize(x,1/2)} \rightarrow \begin{array}{cccc} x_{22} & x_{24} & x_{26} & \cdots \\ x_{42} & x_{44} & x_{46} & \cdots \\ x_{62} & x_{64} & x_{66} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{array}$$

18 **Ch3-p.50**

© 2010 Cengage Learning Engineering. All Rights Reserved.

CENGAGE Learning

3.5 Spatial Resolution

```
x2=imresize(imresize(x,1/2),2);
```

$$\rightarrow \begin{array}{cccc} \boxed{\begin{array}{cc} x_{22} & x_{22} \\ x_{22} & x_{22} \end{array}} & \boxed{\begin{array}{cc} x_{24} & x_{24} \\ x_{24} & x_{24} \end{array}} & \boxed{\begin{array}{cc} x_{26} & x_{26} \\ x_{26} & x_{26} \end{array}} & \cdots \\ \boxed{\begin{array}{cc} x_{42} & x_{42} \\ x_{42} & x_{42} \end{array}} & \boxed{\begin{array}{cc} x_{44} & x_{44} \\ x_{44} & x_{44} \end{array}} & \boxed{\begin{array}{cc} x_{46} & x_{46} \\ x_{46} & x_{46} \end{array}} & \cdots \\ \boxed{\begin{array}{cc} x_{62} & x_{62} \\ x_{62} & x_{62} \end{array}} & \boxed{\begin{array}{cc} x_{64} & x_{64} \\ x_{64} & x_{64} \end{array}} & \boxed{\begin{array}{cc} x_{66} & x_{66} \\ x_{66} & x_{66} \end{array}} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{array}$$

Command	Effective resolution
`imresize(imresize(x,1/4),4);`	64×64
`imresize(imresize(x,1/8),8);`	32×32
`imresize(imresize(x,1/16),16);`	16×16
`imresize(imresize(x,1/32),32);`	8×8

```
imresize(imresize(x, factor, 'nearest'), factor, 'nearest');
```


FIGURE 3.5

(a) (b)

FIGURE 3.5 *Reducing resolution of an image. (a) The original image. (b) Image at 128 × 128 resolution.*

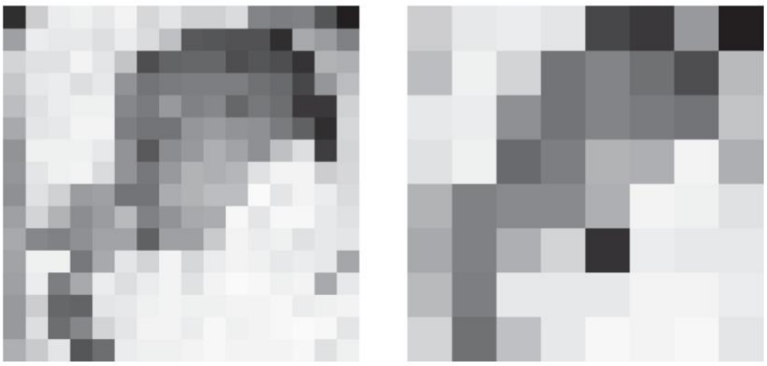
FIGURE 3.6

(a) (b)

FIGURE 3.6 *Further reducing the resolution of an image. (a) Image at 64 × 64 resolution. (b) Image at 32 × 32 resolution.*

21 **Ch3-p.52**

FIGURE 3.7



(a) (b)

FIGURE 3.7 *Even more reducing the resolution of an image. (a) Image at 16 × 16 resolution. (b) Image at 8 × 8 resolution.*

22 **Ch3-p.52**

3.6 Quantization and Dithering

- **Uniform quantization**

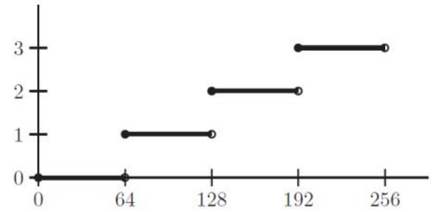


FIGURE 3.8 *A mapping for uniform quantization.*

Original values	Output value
0–63	0
64–127	1
128–191	2
192–255	3

23 **Ch3-p.52**

CENGAGE Learning

3.6 Quantization and Dithering

- To perform such a mapping in MATLAB, we can perform the following operations, supposing `x` to be a matrix of type `uint8`

```
f=floor(double(x)/64);
q=uint8(f*64);
```

- There is, a more elegant method of reducing the grayscales in an image, and it involves using the `grayslice` function

24 **Ch3-p.53**

CENGAGE Learning

3.6 Quantization and Dithering

Command	Number of grayscales
`imshow(grayslice(x,128),gray(128))`	128
`imshow(grayslice(x,64),gray(64))`	64
`imshow(grayslice(x,32),gray(32))`	32
`imshow(grayslice(x,16),gray(16))`	16
`imshow(grayslice(x,8),gray(8))`	8
`imshow(grayslice(x,4),gray(4))`	4
`imshow(grayslice(x,2),gray(2))`	2

FIGURE 3.9

(a) (b)

FIGURE 3.9 *Quantization (1). (a) The image quantized to 128 grayscales. (b) The image quantized to 64 grayscales.*

FIGURE 3.10

(a) (b)

FIGURE 3.10 *Quantization (2). (a) The image quantized to 32 grayscales. (b) The image quantized to 16 grayscales.*

FIGURE 3.11

(a) (b)

FIGURE 3.11 *Quantization (3). (a) The image quantized to eight grayscales. (b) The image quantized to four grayscales.*

FIGURE 3.12

FIGURE 3.12 *The image quantized to two grayscales.*

29 Ch3-p.56

3.6 Quantization and Dithering

- **DITHERING** In general terms, refers to the process of reducing the number of colors in an image
- Representing an image with only two tones is also known as **halftoning**

30 Ch3-p.56

3.6 Quantization and Dithering

- Dithering matrix

$$D = \begin{bmatrix} 0 & 128 \\ 192 & 64 \end{bmatrix} \qquad D_2 = \begin{bmatrix} 0 & 128 & 32 & 160 \\ 192 & 64 & 224 & 96 \\ 48 & 176 & 16 & 144 \\ 240 & 112 & 208 & 80 \end{bmatrix}$$

- D or D_2 is repeated until it is as big as the image matrix, when the two are compared

31 **Ch3-p.56**

3.6 Quantization and Dithering

- Suppose $d(i, j)$ is the matrix obtain by replicating the dithering matrix, then an output pixel $p(i, j)$ is defined by

$$p(i,j) = \begin{cases} 1 & \text{if } x(i,j) > d(i,j) \\ 0 & \text{if } x(i,j) \le d(i,j) \end{cases}$$

32 **Ch3-p.56**

FIGURE 3.13

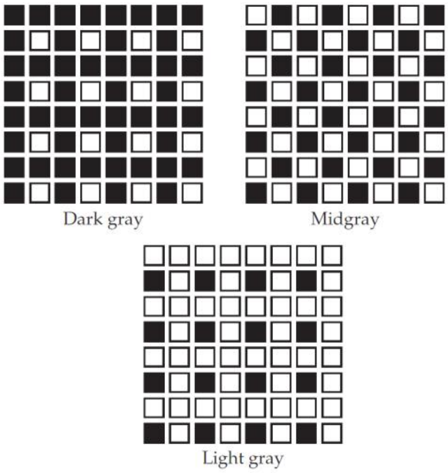
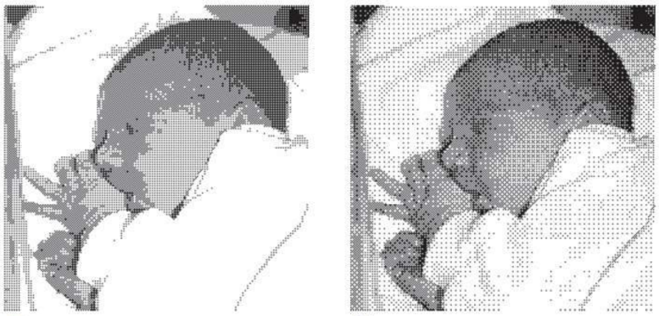


FIGURE 3.13 *Patterns for dithering output.*

33 **Ch3-p.57**

FIGURE 3.14

```
>> D=[0 128;192 64]
>> r=repmat(D,128,128);
>> x2=x>r;imshow(x2)
>> D2=[0 128 32 160;192 64 224 96;48 176 16 144;240 112 208 80];
>> r2=repmat(D2,64,64);
>> x4=x>r2;imshow(x4)
```



(a) (b)

FIGURE 3.14 *Examples of dithering. (a) The newborn baby image dithered using* D *(b) The newborn baby image dithered using* D2

34 **Ch3-p.58**

3.6 Quantization and Dithering

- Dithering can be extended easily to more than two output gray values
- For example, we wish to quantize to four output levels 0, 1, 2, and 3

$$q(i,j) = \lfloor x(i,j)/85 \rfloor \quad \text{(Since 255/3 = 85)}$$

$$p(i,j) = q(i,j) + \begin{cases} 1 & \text{if } x(i,j) - 85q(i,j) > d(i,j) \\ 0 & \text{if } x(i,j) - 85q(i,j) \le d(i,j) \end{cases}$$

FIGURE 3.15

```
>> D = [0 56; 84 28];
>> r = repmat(D, 128, 128);
>> x = double(x);
>> q = floor(x/85);
>> x4 = q+(x-85*q>r);
>> imshow(uint8(85*x4))
```

```
>> D = [0 24; 36 12];
>> r = repmat(D, 128, 128);
>> x = double(x);
>> q = floor(x/37);
>> x8 = q+(x-37*q>r);
>> imshow(uint8(37*x8))
```

(a)

(b)

FIGURE 3.15 *Dithering to more than two grayscales. (a) Dithering to four output grayscales. (b) Dithering to eight output grayscales.*

3.6 Quantization and Dithering

- **ERROR DIFFUSION**
  - ✓ The image is quantized at two levels
  - ✓ For each pixel we take into account the error between its gray value and its quantized value
  - ✓ The idea is to spread this error over neighboring pixels

37 Ch3-p.59

CENGAGE Learning

3.6 Quantization and Dithering

- Floyd and Steinberg method
  - ✓ For each pixel $p(i, j)$ in the image we perform the following sequence of steps:

    1. Perform the quantization

    2. Calculate the quantization error

$$E = \begin{cases} p(i,j) & \text{if } p(i,j) < 128 \\ p(i,j) - 255 & \text{if } p(i,j) >= 128 \end{cases}$$

38 Ch3-p.59

CENGAGE Learning

3.6 Quantization and Dithering

3. Spread this error E over pixels to the right and below according to this table

		$+\frac{7}{16}E$
$+\frac{3}{16}E$	$+\frac{5}{16}E$	$+\frac{1}{16}E$

FIGURE 3.16

```
function y = fl_stein(x)
%
% FL_STEIN applies Floyd-Steinberg error diffusion to an image x, which is
% assumed to be of type uint8.
%
height=size(x,1);
width=size(x,2);
y=uint8(zeros(height,width));
z=zeros(height+2,width+2);
z(2:height+1,2:width+1)=x;
for i=2:height+1,
  for j=2:width+1,
    if z(i,j) < 128
      y(i-1,j-1) = 0;
      e = z(i,j);
    else
      y(i-1,j-1) = 255;
      e = z(i,j)-255;
    end
    z(i,j+1)=z(i,j+1)+7*e/16;
    z(i+1,j-1)=z(i+1,j-1)+3*e/16;
    z(i+1,j)=z(i+1,j)+5*e/16;
    z(i+1,j+1)=z(i+1,j+1)+e/16;
  end
end
```

FIGURE 3.16 *A MATLAB function for applying Floyd-Steinberg error diffusion to a grayscale image.*

FIGURE 3.17

FIGURE 3.17 *The newborn baby image after Floyd-Steinberg error diffusion.*

FIGURE 3.18

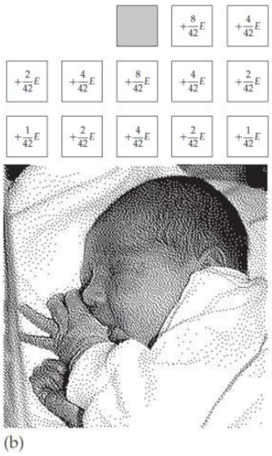


FIGURE 3.18 *Using other error-diffusion schemes. (a) Result of Jarvis-Judice-Ninke error diffusion. (b) Result of Stucki error diffusion.*