FPT Education

FPT UNIVERSITY

PART I
IMAGE PROCESSING

INSTRUCTOR: DR. NGUYEN NGOC TRUONG MINH

SCHOOL OF ELECTRICAL ENGINEERING, INTERNATIONAL UNIVERSITY (VNU-HCMC)

Ho Chi Minh City, June 2023

LECTURE VI – GRAY-LEVEL TRANSFORMATIONS, HISTOGRAM AND NEIGHBORHOOD PROCESSING

INSTRUCTOR: DR. NGUYEN NGOC TRUONG MINH

SCHOOL OF ELECTRICAL ENGINEERING, INTERNATIONAL UNIVERSITY (VNU-HCMC)

Ho Chi Minh City, June 2023

LECTURE CONTENT

- What does it mean to "enhance" an image?
- How can image enhancement be achieved using gray-level transformations?
- What are the most commonly used gray-level transformations and how can they be implemented using MATLAB?
- What is the histogram of an image?
- How can the histogram of an image be computed?
- How much information does the histogram provide about the image?

LECTURE CONTENT

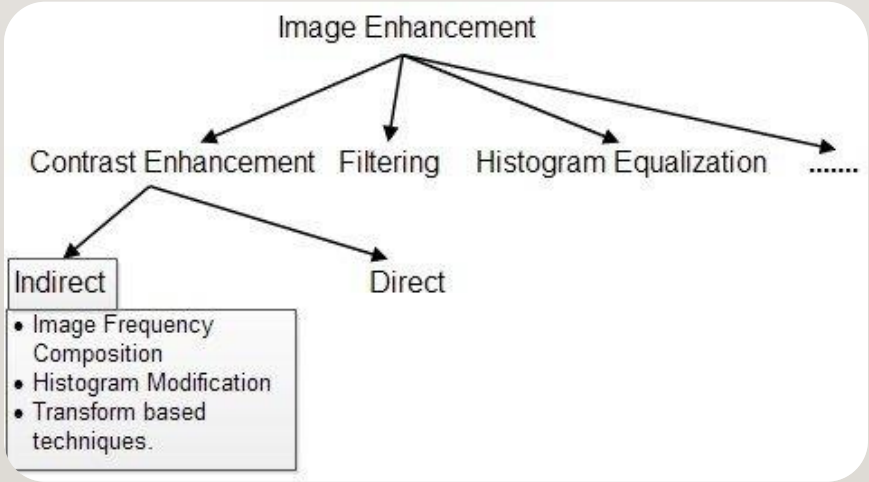
- What is histogram equalization and what happens to an image whose histogram is equalized?
- How can the histogram be modified through direct histogram specification and what happens to an image when we do it?
- What other histogram modification techniques can be applied to digital images and what is the result of applying such techniques?
- What is neighborhood processing and how does it differ from point processing?

LECTURE CONTENT

- What is convolution and how is it used to process digital images?
- What is a low-pass linear filter, what is it used for, and how can it be implemented using 2D convolution?
- What is a median filter and what is it used for?
- What is a high-pass linear filter, what is it used for, and how can it be implemented using 2D convolution?
- Chapter Summary – What have we learned?
- Problems

6.1 INTRODUCTION

- **Image enhancement techniques** usually have one of these two goals:
  - *To improve the subjective quality of an image for human viewing.*
  - *To modify the image in such a way as to make it more suitable for further analysis and automatic extraction of its contents.*



6.1 INTRODUCTION

- In the first case, the ultimate goal is *an improved version* of the original image, whose interpretation will be left to a human expert—for example, an enhanced X-ray image that will be used by a medical doctor to evaluate the possibility of a fractured bone.
- In the second scenario, the goal is to serve as *an intermediate step* toward an automated solution that will be able to derive the semantic contents of the image—for example, by improving the contrast between characters and background on a page of text before it is examined by an OCR algorithm.
- Sometimes these goals can be at *odds with each other*.

6.1 INTRODUCTION

Image enhancement techniques are used when either (1) an image needs improvement, or (2) the low-level features must be detected.

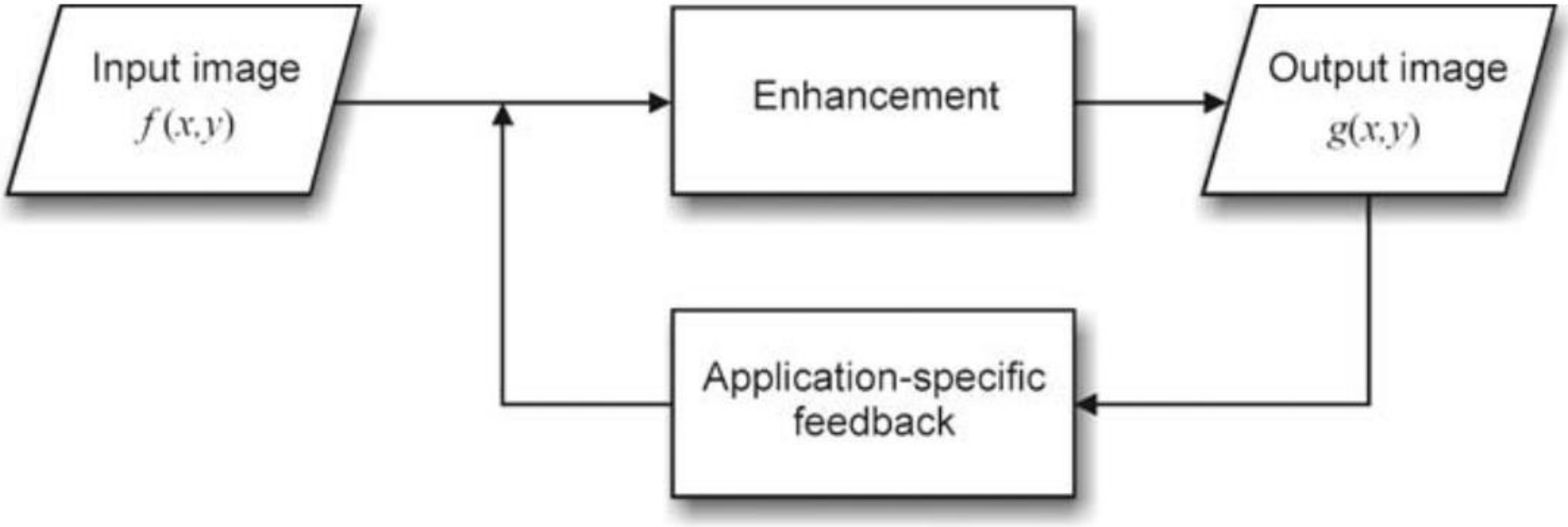


FIGURE 8.1 The image enhancement process. Adapted and redrawn from [Umb05].

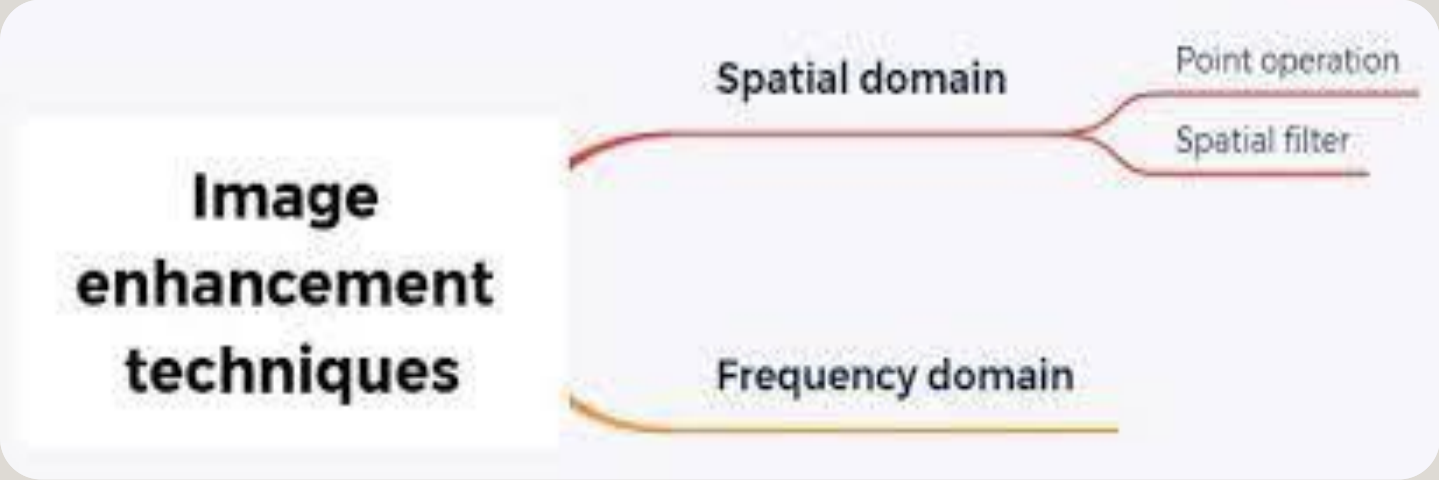
6.1 INTRODUCTION

- It is important to mention that image enhancement algorithms are usually *goal specific*. Rather, it is typically *an interactive process* in which different techniques and algorithms are tried and parameters are fine-tuned, until an acceptable result is obtained.

- Moreover, it is often *a subjective process* in which the human observer's specialized skills and prior problem-domain knowledge play a significant role (e.g., a radiology expert may find significant differences in the quality of two X-ray images that would go undetected by a lay person).

6.1 INTRODUCTION

- This lecture focuses on *point operations* whose common goal is to *enhance an input image*. In some cases, the enhancement results are clearly targeted at a human viewer, while in some other cases the results may be more suitable for subsequent stages of processing in a machine vision system.



6.2 OVERVIEW OF GRAY-LEVEL (POINT) TRANSFORMATIONS

Point operations are also referred to as *gray-level transformations* or *spatial transformations*. They can be expressed as

$$g(x, y) = T[f(x, y)] \qquad (6.1)$$

where g(x, y) is the processed image, f(x, y) is the original image, and T is an operator on f(x, y).

Since the actual coordinates do not play any role in the way the transformation function processes the original image, equation (6.1) can be rewritten as

$$s = T[r] \qquad (6.2)$$

where r is the original gray level of a pixel and s is the resulting gray level after processing.

6.2 OVERVIEW OF GRAY-LEVEL (POINT) TRANSFORMATIONS

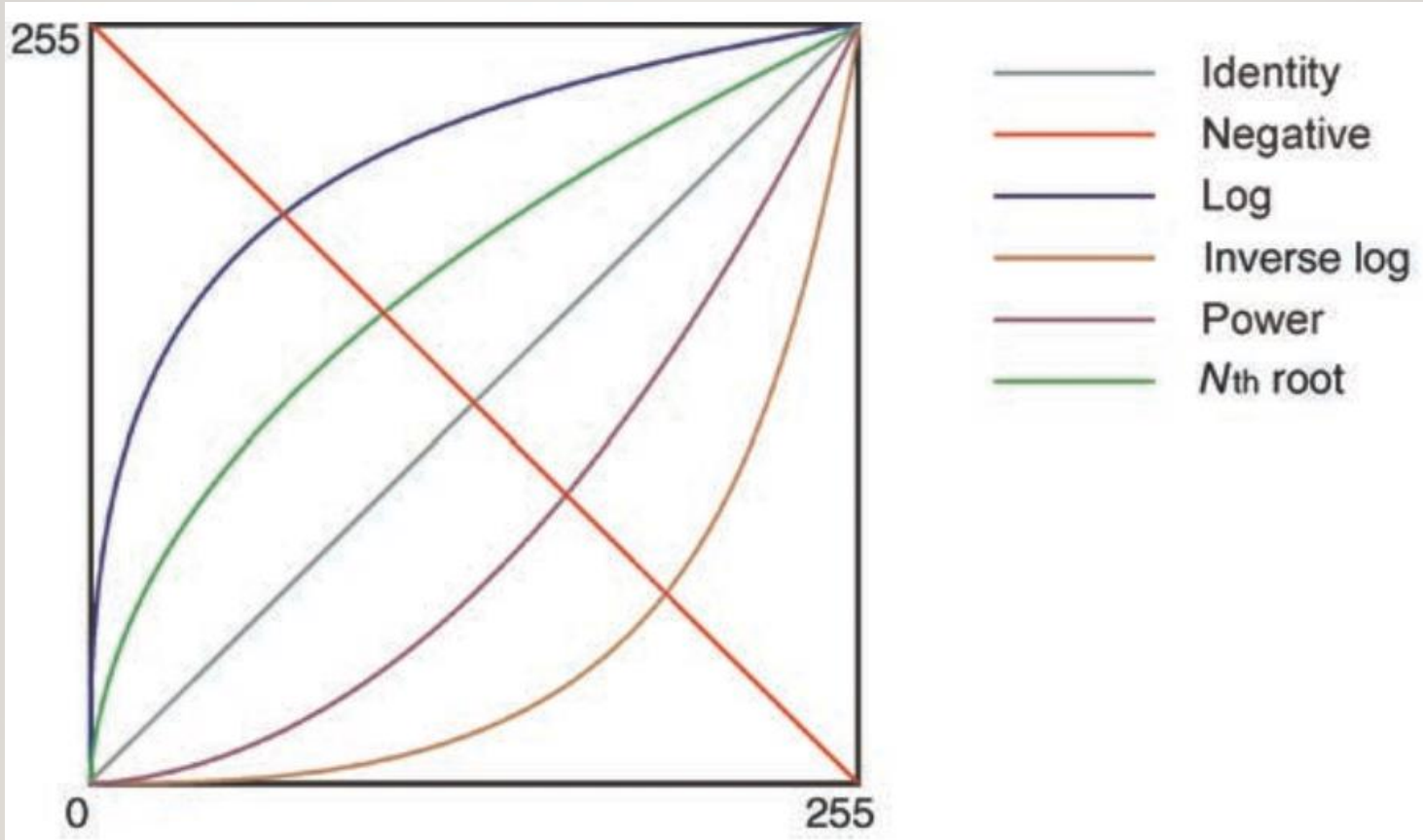


FIGURE 8.2 Basic gray-level transformation functions.

6.2 OVERVIEW OF GRAY-LEVEL (POINT) TRANSFORMATIONS

- Point transformations may be *linear* (e.g., negative), *piecewise linear* (e.g., gray-level slicing), or *nonlinear* (e.g., gamma correction). Figure 8.2 shows examples of basic linear (*identity* and *negative*) and nonlinear (*log*, *inverse log*, *power*—also known as *gamma*—, and *nth root*) transformation functions.
- Point operations are usually treated as simple mapping operations whereby the new pixel value at a certain location (x_0, y_0) *depends only on the original pixel value at the same location and the mapping function*.
- In other words, the resulting image does not exhibit any change in size, geometry, or local structure if compared with the original image.

6.2 OVERVIEW OF GRAY-LEVEL (POINT) TRANSFORMATIONS

- In this lecture, we will call *linear* point transformations those that can be mathematically described by a single linear equation:

$$s = c \cdot r + b \qquad (6.3)$$

  where r is the original pixel value, s is the resulting pixel value, and c is a constant—responsible for controlling the contrast of the output image—, whereas b is another constant whose value impacts the output image's overall brightness.

- From a graphical perspective, a plot of s as a function of r will show a straight line, whose slope (or gradient) is determined by the constant c; the constant term b determines the point at which the line crosses the y-axis.

EXAMPLE 6.1

FIGURE 8.3 Linear point transformations example: input image.

EXAMPLE 6.1

Figure 8.4 shows the results of applying three different linear point transformations to the input image in Figure 8.3.

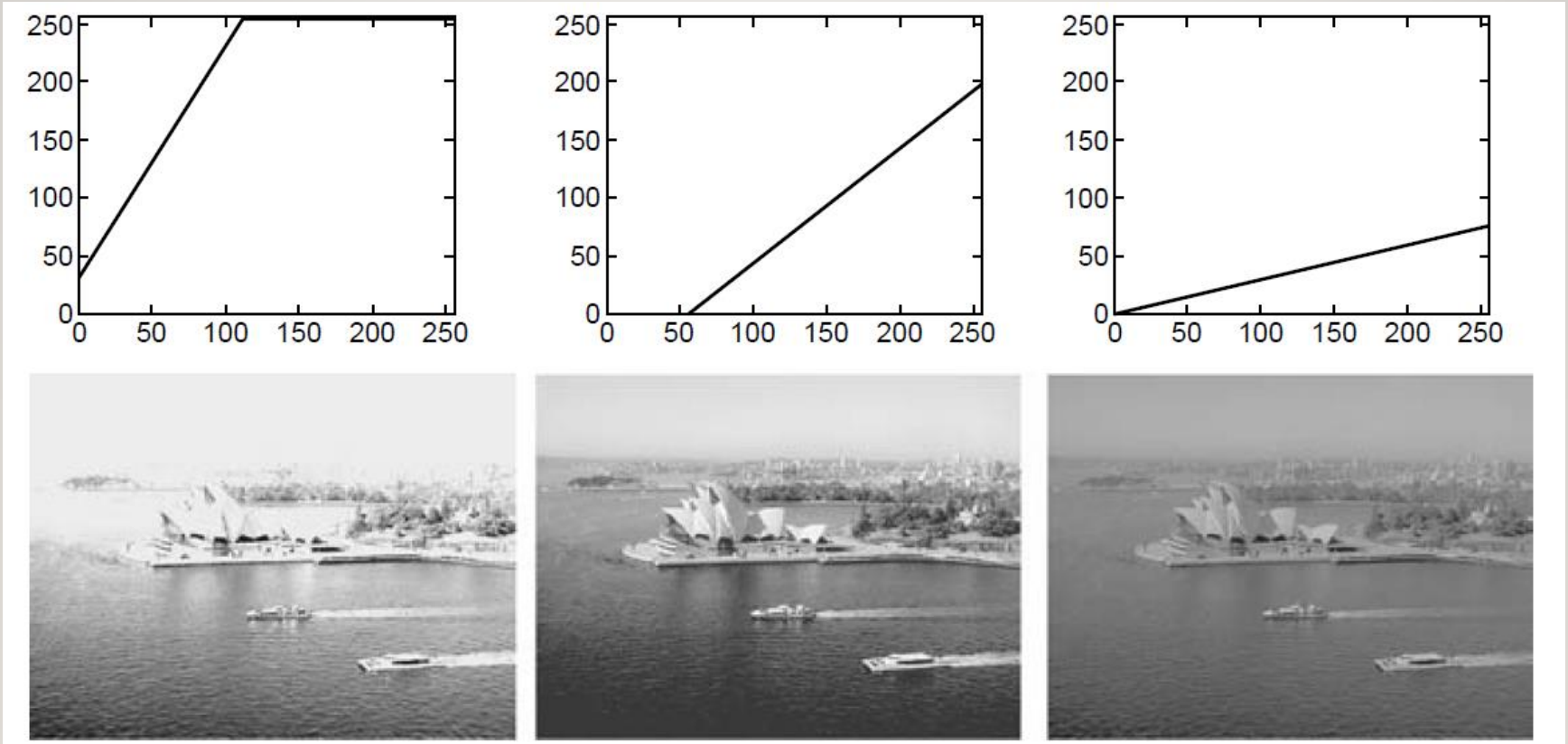


FIGURE 8.4 Linear point transformations and their impact on the overall brightness and contrast of an image: brightening (left), darkening (middle), and contrast reduction (right).

EXAMPLE 6.1

TABLE 8.1 Examples of Linear Point Transformations (Images and Curves in Figure 8.4)

Column	c	b	Effect on Image
Left	2	32	Overall brightening, including many saturated pixels
Middle	1	−56	Overall darkening
Right	0.3	0	Significant contrast reduction and overall darkening

6.3 EXAMPLES OF POINT TRANSFORMATIONS

- In this section, we show examples of some of the most widely used point transformations.

6.3.1 CONTRAST MANIPULATION

- One of the most common applications of point transformation functions is *contrast manipulation* (also known as *contrast stretching*, *gray-level stretching*, *contrast adjustment*, and *amplitude scaling*).
- These functions often exhibit a curve that resembles *the curve of a sigmoid function* (Figure 8.5a): pixel values of $f < m$ are compressed toward darker values in the output image, whereas values of $f > m$ are mapped to brighter pixel values in the resulting image. The slope of the curve indicates how dramatic the contrast changes will be; in its most extreme case, a contrast manipulation function degenerates into a binary thresholding function (Figure 8.5b), where pixels in the input image whose value is $f < m$ become black and pixels whose value is $f > m$ are converted to white.

6.3.1 CONTRAST MANIPULATION

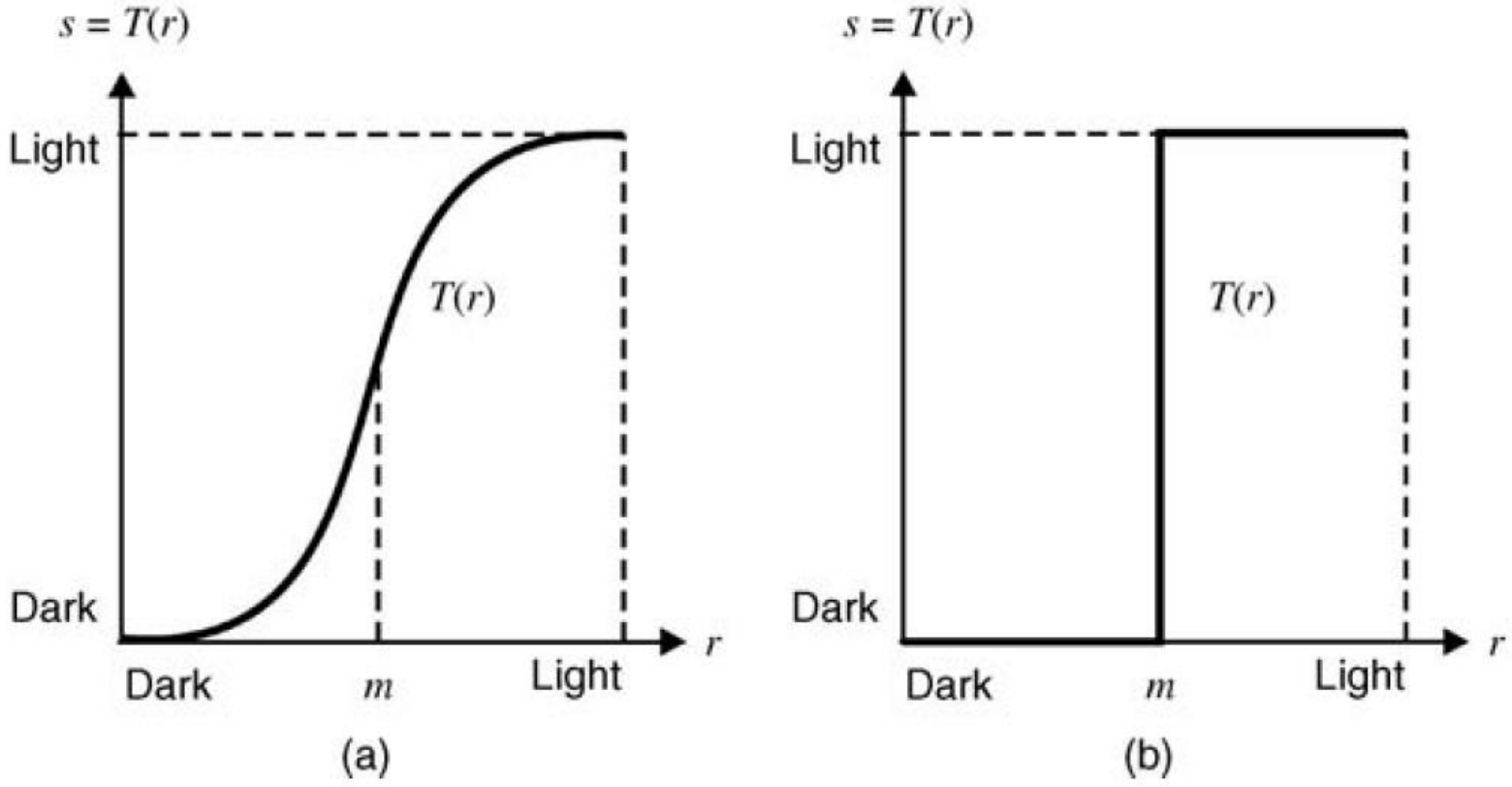


FIGURE 8.5 Examples of gray-level transformations for contrast enhancement. Redrawn from [GW08].

6.3.1 CONTRAST MANIPULATION

- One of the most useful variants of contrast adjustment functions is the *automatic contrast adjustment* (or simply *autocontrast*), a point transformation that—for images of class *uint8* in MATLAB—maps the darkest pixel value in the input image to 0 and the brightest pixel value to 255 and redistributes the intermediate values linearly.
- The autocontrast function can be described as follows:

$$s = \frac{L-1}{r_{max}-r_{min}} \cdot (r - r_{min}) \qquad (6.4)$$

where r is the pixel value in the original image (in the [0, 255] range), r_{max} and r_{min} are the values of its brightest and darkest pixels, respectively, s is the resulting pixel value, and $L - 1$ is the highest gray value in the input image (usually L = 256).

6.3.1 CONTRAST MANIPULATION

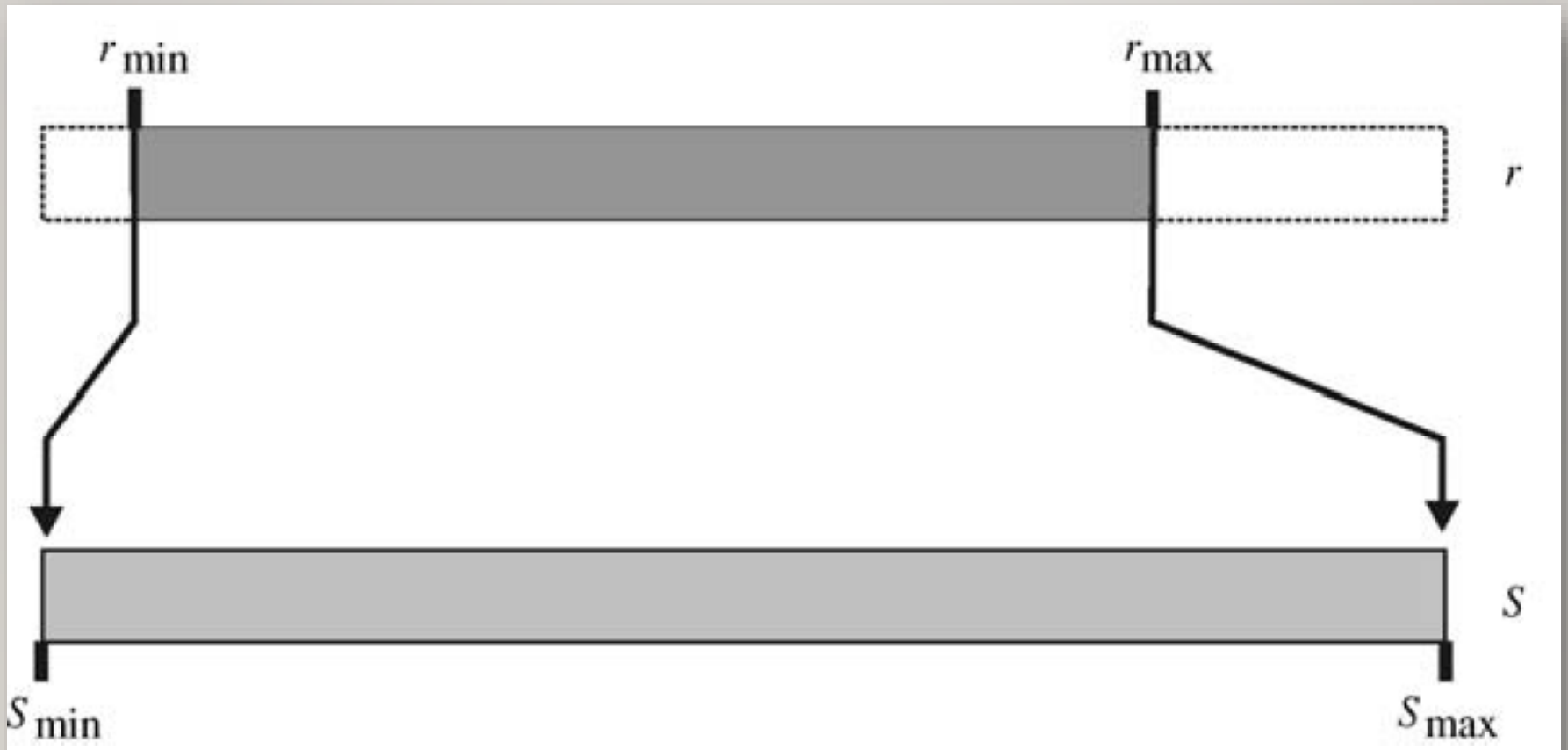


FIGURE 8.6 Autocontrast operation. Redrawn from [BB08].

6.3.1 CONTRAST MANIPULATION

(a) (b)

FIGURE 8.7 (a) Example of an image whose original gray-level range was [90, 162]; (b) the result of applying the autocontrast transformation (equation (8.4)).

6.3.1 CONTRAST MANIPULATION

In MATLAB

MATLAB's IPT has a built-in function *imadjust* to perform contrast adjustments (including autocontrast).

In MATLAB, *interactive* brightness and contrast adjustments can also be performed using *imcontrast* that opens the Adjust Contrast tool.

6.3.2 NEGATIVE

- The negative point transformation function (also known as *contrast reverse*). The negative transformation is used to make the output more suitable for the task at hand (e.g., by making it easier to notice interesting details in the image).

6.3.2 NEGATIVE

In MATLAB

MATLAB's IPT has a built-in function to compute the negative of an image:

imcomplement.

6.3.3 POWER LAW (GAMMA) TRANSFORMATIONS

The power law transformation function is described by

$$s = c \cdot r^{\gamma} \qquad (6.5)$$

where r is the original pixel value, s is the resulting pixel value, c is a scaling constant,

and γ is a positive value. Figure 8.8 shows a plot of equation (6.5) for several values of γ.

6.3.3 POWER LAW (GAMMA) TRANSFORMATIONS

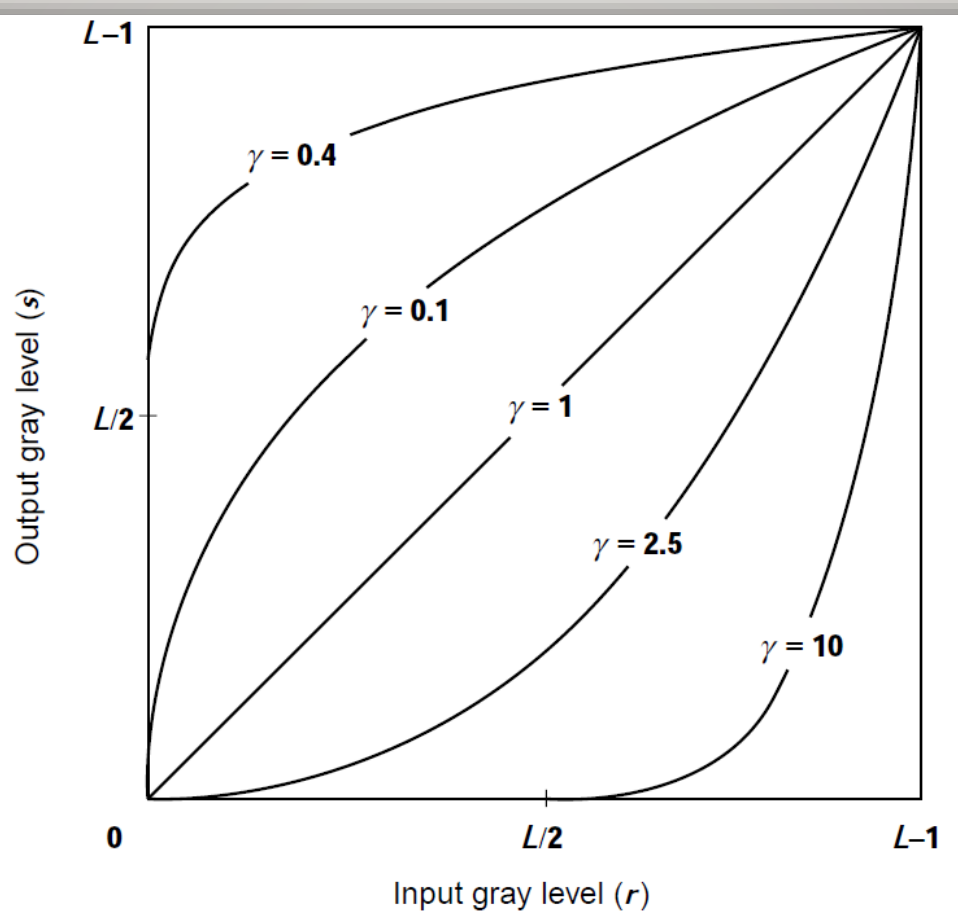


FIGURE 8.8 Examples of power law transformations for different values of γ.

EXAMPLE 6.2

- Figure 8.9 shows the results of applying gamma correction to an input image using two different values of γ. It should be clear from the figure that when γ < 1, the resulting image is darker than the original one; whereas for γ > 1, the output image is brighter than the input image.

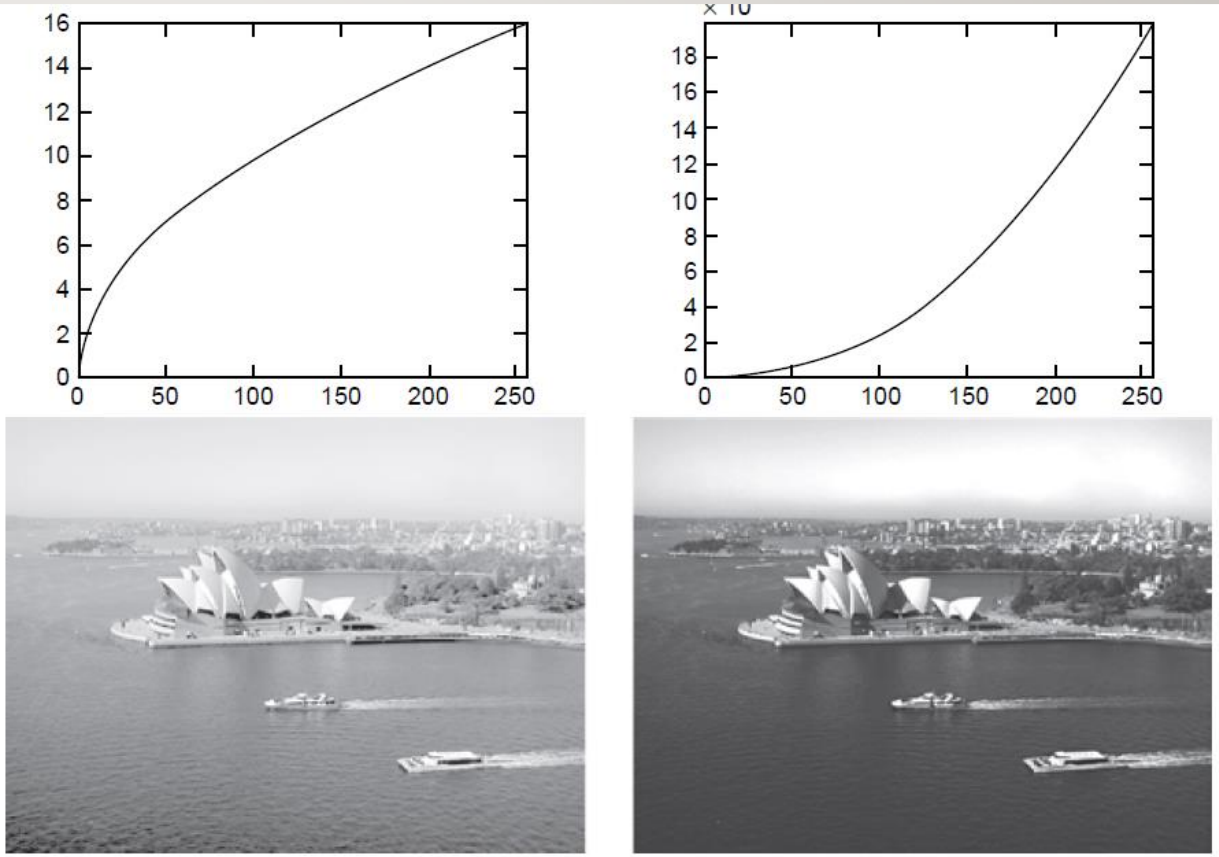


FIGURE 8.9 Examples of gamma correction for two different values of γ: 0.5 (left) and 2.2 (right).

6.3.3 POWER LAW (GAMMA) TRANSFORMATIONS

In MATLAB

The *imadjust* function in the IPT can be used to perform gamma correction with the syntax: *g = imadjust(f,[],[],gamma)*;

6.3.4 LOG TRANSFORMATIONS

- The log transformation and its inverse are nonlinear transformations used, respectively, when we want to compress or expand the dynamic range of pixel values in an image.
- Log transformations can be mathematically described as

$$s = c \cdot log(1 + r) \qquad (6.6)$$

where r is the original pixel value, s is the resulting pixel value, and c is a constant.

- Be aware that in many applications of the log transformation, the input "image" is actually a 2D array with values that might lie outside the usual range for gray levels that we usually associate with *monochrome images* (e.g., [0, 255]).

EXAMPLE 6.3

- This example uses the log transformation to improve the visualization and display of Fourier transform (FT) results (Figure 8.10).
- The range of values in the matrix in part (a) is [0, 2.8591×10^4], which—when displayed on a linearly scaled 8-bit system—makes it hard to see anything but the bright spot at the center. Applying a log transform, the dynamic range is compressed to [0, 10.26].
- Using the proper autocontrast transformation to linearly extend the compressed range to [0, 255], we obtain the image in part (b), where significant additional details (e.g., thin vertical line at the center, concentric circles) become noticeable.

EXAMPLE 6.3

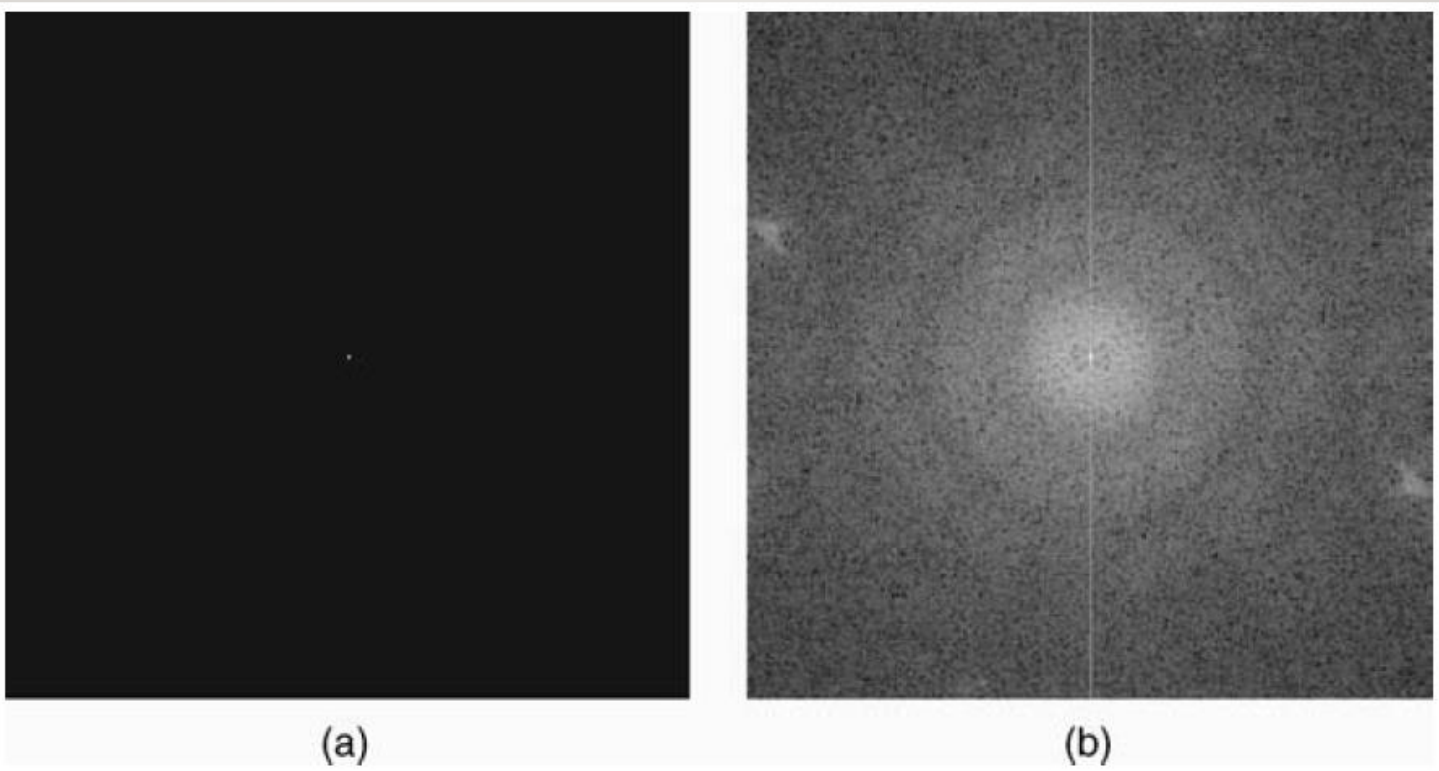


FIGURE 8.10 Example of using log transformation: (a) Fourier spectrum (amplitude only) of the rice image (available in MATLAB); (b) result of applying equation (8.6) with $c = 1$ followed by autocontrast.

6.3.5 PIECEWISE LINEAR TRANSFORMATIONS

- Piecewise linear transformations can be described by several linear equations, one for each interval of gray-level values in the input image.
- The main advantage of piecewise linear functions is that they can be arbitrarily complex; the main disadvantage is that *they require additional user input*.

6.3.5 PIECEWISE LINEAR TRANSFORMATIONS

Figure 8.11 shows an example of an arbitrary piecewise linear transformation function used to improve the contrast of the input image. The function is specified interactively using a GUI-based MATLAB tool glsdemo.

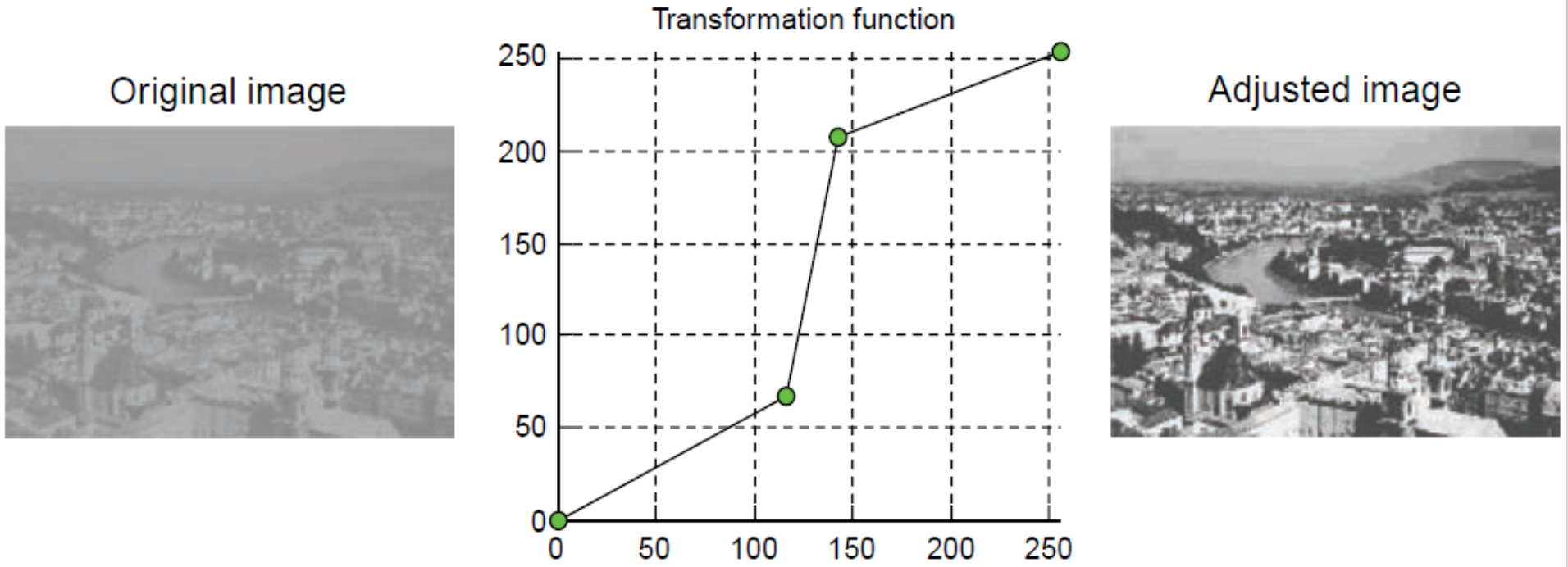


FIGURE 8.11 Piecewise linear transformation using `glsdemo`.

6.3.5 PIECEWISE LINEAR TRANSFORMATIONS

Figure 8.12 shows an example of gray-level slicing, a particular case of piecewise linear transformation in which a specific range of intensity levels (in this case, the [100, 150] range) is highlighted in the output image, while all other values remain untouched.

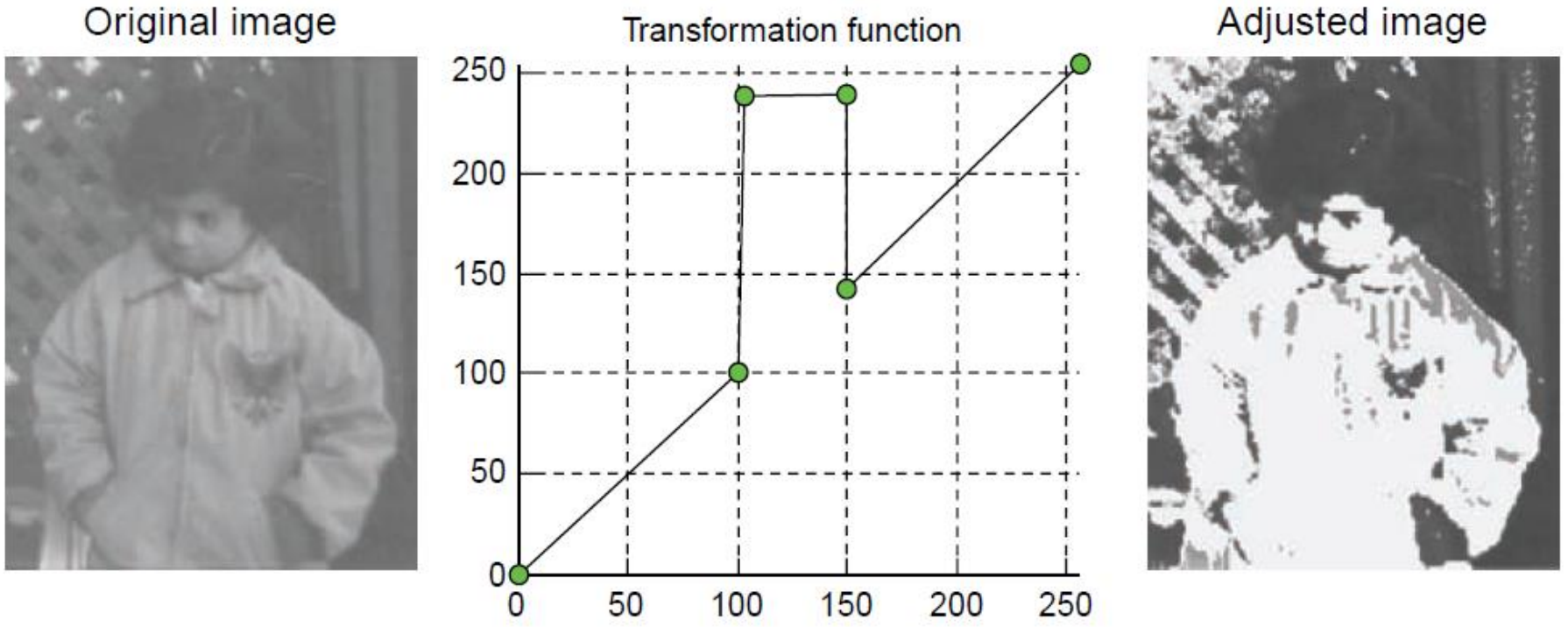


FIGURE 8.12 Gray-level slicing using `glsdemo`. Original image: courtesy of MathWorks.

6.4 SPECIFYING THE TRANSFORMATION FUNCTION

- All the transformation functions presented in this lecture have been described mathematically in a way that is *elegant and appropriate for input variables in the continuous or discrete domain*.
- In this section, we show that in spite of its elegance, the mathematical formulation is not always useful in practice, for two different—but relevant—reasons:

1. *From a user interaction viewpoint, it is often preferable to specify the desired point transformation function interactively using the mouse and a GUI-based application.*

6.4 SPECIFYING THE TRANSFORMATION FUNCTION

- All the transformation functions presented in this lecture have been described mathematically in a way that is *elegant and appropriate for input variables in the continuous or discrete domain*.
- In this section, we show that in spite of its elegance, the mathematical formulation is not always useful in practice, for two different—but relevant—reasons:

2. *From the perspective of computational efficiency, point operations can be executed at significantly higher speed using lookup tables (LUTs). For images of type uint8 (i.e., monochrome images with 256 gray levels), the LUT will consist of a 1D array of length 256. LUTs can be easily implemented in MATLAB as demonstrated in the following examples.*

EXAMPLE 6.4

$$s = \begin{cases} 2 \cdot f & \text{for } 0 < r \leq 64 \\ 128 & \text{for } 64 < r \leq 128 \\ f & \text{for } r > 128 \end{cases}$$

```
LUT = uint8(zeros([1 256]));
LUT(1:65) = 2*(0:64);
LUT(66:129) = 128;
LUT(130:256) = (130:256)-1;
```

Next, we will test the LUT using a 3x3 test image.

A = uint8([20 40 0; 178 198 64; 77 128 1])

B = intlut(A, LUT)

As expected, if the input array is

$$\begin{bmatrix} 20 & 40 & 0 \\ 178 & 198 & 64 \\ 77 & 128 & 1 \end{bmatrix}$$

the resulting array will be

$$\begin{bmatrix} 40 & 80 & 0 \\ 178 & 198 & 128 \\ 128 & 128 & 2 \end{bmatrix}$$

EXAMPLE 6.4

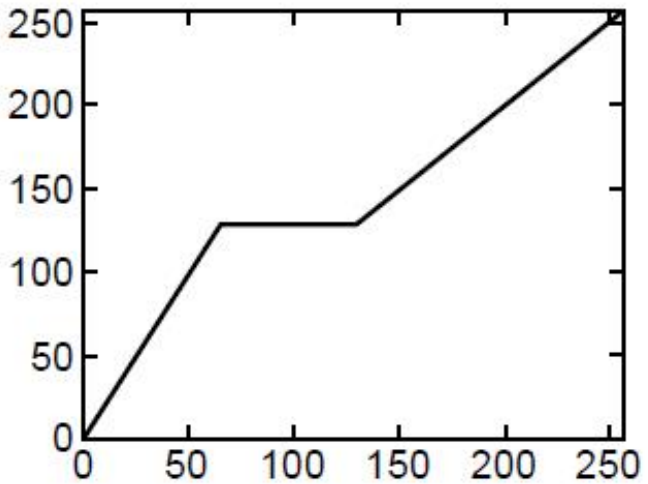
Finally, we will apply the LUT to a real gray-level image. The result appears in Figure 8.13. Note how many pixels in the original image have been "flattened" to the average gray level (128) in the output image.

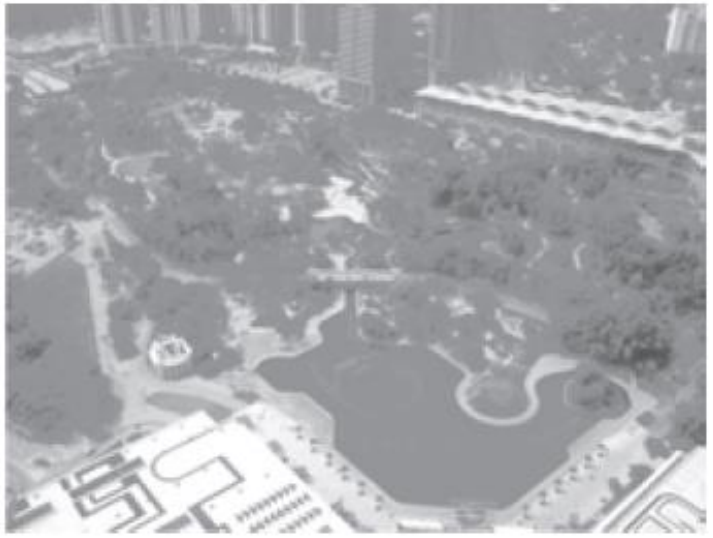
```
I = imread('klcc_gray.png');

O = intlut(I,LUT);

figure, subplot(1,2,1), imshow(I), subplot(1,2,2), imshow(O)
```

EXAMPLE 6.4





(a) (b) (c)

FIGURE 8.13 Example of using a lookup table: (a) input image; (b) transformation function specified by equation (8.7); (c) output image.

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

Goal

- The goal of this tutorial is to learn how to perform basic point transformations on grayscale images.

Objectives

- Explore linear transformations including the identity function and the negative function.
- Learn how to perform logarithmic grayscale transformations.
- Learn how to perform power law (gamma) grayscale transformations.
- Explore gray (intensity)-level slicing.

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

The most basic transformation function is the identity function, which simply maps each pixel value to the same value.

1. Create an identity transformation function.

```
x = uint8(0:255);
plot(x); xlim([0 255]); ylim([0 255]);
```

2. Use the transformation function on the moon image to see how the identity function works.

```
I = imread('moon.tif');
I_adj = x(I + 1);
figure, subplot(1,2,1), imshow(I), title('Original Image');
subplot(1,2,2), imshow(I_adj), title('Adjusted Image');
```

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

Question 1: Why were we required to use I+1 when performing the transformation instead of just I?

Question 2: How can we show that the adjusted image and the original image are equivalent?

The negative transformation function generates the negative of an image.

3. Create a negative transformation function and show the result after applied to the moon image.

```
y = uint8(255:-1:0); I_neg = y(I + 1);
figure, subplot(1,3,1), plot(y), ...
title('Transformation Function'), xlim([0 255]), ylim([0 255]);
subplot(1,3,2), imshow(I), title('Original Image');
subplot(1,3,3), imshow(I_neg), title('Negative Image');
```

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

Question 3: How did we create the negative transformation function?

A negative transformation results in the same image as if we were to complement the image logically.

4. Complement the original image and show that it is equivalent to the negative image generated in the previous step.

```
I_cmp = imcomplement(I);
I_dif = imabsdiff(I_cmp,I_neg);
figure, imshow(I_cmp);
figure, imshow(I_dif,[]);
```

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

Logarithmic transformation functions can be used to compress the dynamic range of an image in order to bring out features that were not originally as clear. The log transformation function can be calculated using equation (6.6). In our case, x represents the value of any particular pixel, and the constant c is used to scale the output within grayscale range [0, 255].

5. Close all open figures and clear all workspace variables.
6. Generate a logarithmic transformation function.

```
x = 0:255; c = 255 / log(256);
y=c*log(x + 1);
figure, subplot(2,2,1), plot(y), ...
title('Log Mapping Function'), axis tight, axis square
```

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

7. Use the transformation function to generate the adjusted image.

```
I = imread('radio.tif');
I_log = uint8(y(I + 1));
subplot(2,2,2), imshow(I), title('Original Image');
subplot(2,2,3), imshow(I_log), title('Adjusted Image');
```

In the second line of code, you will note that we convert the image to *uint8*. This is necessary because the only way the function *imshow* will recognize a matrix with a range of [0, 255] as an image is if it is of class *uint8*. In the next step, we will see that simply brightening the image will not show the missing detail in the image.

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

8. Show a brightened version of the image.

```
I_br = imadd(I,100);
subplot(2,2,4), imshow(I_br), title('Original Image Scaled');
```

Question 4: Why does the log-transformed image display the hidden detail in the radio image, but the brightened image does not?

The inverse of the log function is as follows.

$$y(x) = \exp(x/c) - 1;$$

Again, here c is the scaling constant and x is the pixel value. We can demonstrate that applying this equation to the image we previously created (image I_log, which was transformed using the log transformation) will result in the original image.

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

9. Use inverse log transformation to undo our previous transformation.

```
% Inverse log transformation
z = exp(x/c) - 1;
I_invlog = uint8(z(I_log + 1));
figure, subplot(2,1,1), plot(z), title('Inverse-log Mapping Function');
subplot(2,1,2), imshow(I_invlog), title('Adjusted Image');
```

Power law transformations include nth root and nth power mapping functions. These functions are more versatile than the log transformation functions because we can specify the value of n, which ultimately changes the shape of the curve to meet our particular needs.

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

10. Close all open figures and clear all workspace variables.
11. Generate an nth root function where n equals 2.

```
x = 0:255;n=2;c=255/(255 ^ n);
root = nthroot((x/c), n);
figure, subplot(2,2,1), plot(root), ...
title('2nd-root transformation'), axis tight, axis square
```

Question 5: How does the shape of the curve change if we were to use a different value for n?

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

12. Use the transformation function to generate the adjusted image.

```
I = imread('drill.tif');
I_root = uint8(root(I + 1));
subplot(2,2,2), imshow(I), title('Original Image');
subplot(2,2,[3 4]), imshow(I_root), title('Nth Root Image');
```

We can see that the adjusted image shows details that were not visible in the original image.

The nth power transformation function is the inverse of the nth root.

13. Generate an nth power transformation function.

```
power=c*(x.^n);
figure, subplot(1,2,1), plot(power), ...
title('2nd-power transformation');
axis tight, axis square
```

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

14. Use the nth power transformation to undo our previous transformation.

```
I_power = uint8(power(I_root + 1));
subplot(1,2,2), imshow(I_power), title('Adjusted Image');
```

Question 6: Show that the I_power image and the original image I are (almost) identical.

The transformation functions we explored thus far have been defined by mathematical equations. During the next steps, we shall explore the creation and application of piecewise linear transformation functions to perform specific tasks. Our first example will be gray-level slicing, a process by which we can enhance a particular range of the gray scale for further analysis.

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

15. Close all open figures and clear all workspace variables.
16. Load the micro image and display it.

```
I = imread('micro.tif');
figure, subplot(1,3,1), imshow(I), title('Original Image');
```

17. Create the transformation function.

```
y(1:175) = 0:174;
y(176:200) = 255;
y(201:256) = 200:255;
subplot(1,3,2), plot(y), axis tight, axis square
```

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

Question 7: Based on the previous step, what do you expect to be the visual effect of applying this transformation function to the original image?

18. Generate the adjusted image.

```
I2 = uint8(y(I + 1));
subplot(1,3,3), imshow(I2), title('Adjusted Image');
```

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

19. Create a new transformation function and display the adjusted image.

```
z(1:175) = 50;
z(176:200) = 250;
z(201:256) = 50;
I3 = uint8(z(I + 1));
figure, subplot(1,2,1), plot(z), ...
xlim([0 255]), ylim([0 255]), axis square
subplot(1,2,2), imshow(I3)
```

6.5 TUTORIAL: GRAY-LEVEL TRANSFORMATIONS

Although it is possible to create any transformation function in MATLAB by defining a vector of values, as we did above, this can be tedious and time consuming.

Through the use of a GUI, we can dynamically generate a function that the user defines visually. The following demo illustrates this.

20. Review the help information for *glsdemo*.

21. Run *glsdemo* with the image *micro.tif* and recreate the transformation functions that we previously used in steps 17 and 19.

WHAT HAVE WE LEARNED?

- Image enhancement is the process of modifying the pixel values within an image in such a way that the resulting image is an improved version of the original image for a particular purpose, whether it is the human perception of subjective quality or further processing by machine vision algorithms.
- Image enhancement can be achieved in many different ways, including the use of certain gray-level transformations, whose chief characteristic is the fact that the resulting gray level of a pixel depends only on the original pixel value and the transformation function. For this reason, gray-level transformations are also referred to as *point transformations*.
- Gray-level transformations can be implemented in MATLAB using the *imadjust* function.

WHAT HAVE WE LEARNED?

- Gray-level transformations are often used for brightness and contrast adjustments. In MATLAB, interactive brightness and contrast adjustments can also be performed using *imcontrast*.
- Some of the most commonly used point transformations are negative, power law (gamma), logarithmic, and piecewise linear transformations.
- Point transformation functions can be specified interactively using tools such as *glsdemo*.
- The use of a lookup table speeds up the processing of point transformation functions.

PROBLEMS

Problem 1. Write a MATLAB function to perform a piecewise linear brightness and contrast adjustment on monochrome images using the generic method described in equation (8.3). It should take as arguments a monochrome image, the c coefficient (slope), and the b coefficient (offset).

Problem 2. Write a MATLAB function to perform a simple version of image solarization technique (also known as *Sabatier effect*), a point transformation that processes an image by leaving all pixels brighter than a certain value (T) untouched, while extracting the negative of all pixels darker than T.

PROBLEMS

Problem 3. Write a MATLAB function to perform a point transformation by which each pixel value in an input image of class uint8 is replaced by the square of its original value and answer the following questions:

1. Do you have to explicitly make provisions for clamping the results (so that they stay within range)? Why (not)?

2. Is the resulting image brighter or darker than the original image? Explain.

Problem 4. Repeat Problem 8.3, this time for an input image of class double. Does the answer to any of the questions change? Why?

6.6 IMAGE HISTOGRAM: DEFINITION AND EXAMPLE

- *The histogram of a monochrome image* is a graphical representation of the frequency of occurrence of each gray level in the image. The data structure that stores the frequency values is a 1D array of numerical values, *h*, whose individual elements store the number (or percentage) of image pixels that correspond to each possible gray level.
- Each individual histogram entry can be expressed mathematically as

$$h(k) = n_k = card\{(x, y) | f(x, y) = k\} \qquad (6.7)$$

- Here, $k = 0, 1, \ldots, L - 1$, where *L* is the number of gray levels of the digitized image, and $card\{(x, y)\}$ denotes the cardinality of a set, that is, the number of elements in that set (n_k).

6.6 IMAGE HISTOGRAM: DEFINITION AND EXAMPLE

- A normalized histogram can be mathematically defined as

$$p(r_k) = \frac{n_k}{n} \qquad (6.8)$$

where n is the total number of pixels in the image and $p(r_k)$ is the probability (percentage) of the kth gray level (r_k).

- Histograms are normally represented using a bar chart, with one bar per gray level, in which the height of the bar is proportional to the number (or percentage) of pixels that correspond to that particular gray level.

6.6 IMAGE HISTOGRAM: DEFINITION AND EXAMPLE

In MATLAB

MATLAB's IPT has a built-in function to calculate and display the histogram of a monochrome image: *imhist*.

Alternatively, other MATLAB plotting functions such as *bar*, *plot*, and *stem* can also be used to display histograms.

EXAMPLE 6.5

- Table 9.1 shows the pixel counts for a hypothetical image containing 128x128 pixels, with eight gray levels. The number of pixels that correspond to a given gray level is indicated in the second column and the corresponding percentages (probabilities), $p(r_k)$, are given in the third column. Its bar graph representation is shown in Figure 9.1.
- Each value of $p(r_k)$ represents the percentage of pixels in the image whose gray level is r_k. In other words, a histogram can be interpreted as a probability mass function of a random variable (r_k) and as such it follows all the axioms and theorems of elementary probability theory. For instance, it is easy to verify from Table 9.1 that the sum of the values for $p(r_k)$ is 1, as expected.

EXAMPLE 6.5

TABLE 9.1 Example of a Histogram

Gray Level (r_k)	n_k	$p(r_k)$
0	1120	0.068
1	3214	0.196
2	4850	0.296
3	3425	0.209
4	1995	0.122
5	784	0.048
6	541	0.033
7	455	0.028
Total	16,384	1.000

EXAMPLE 6.5

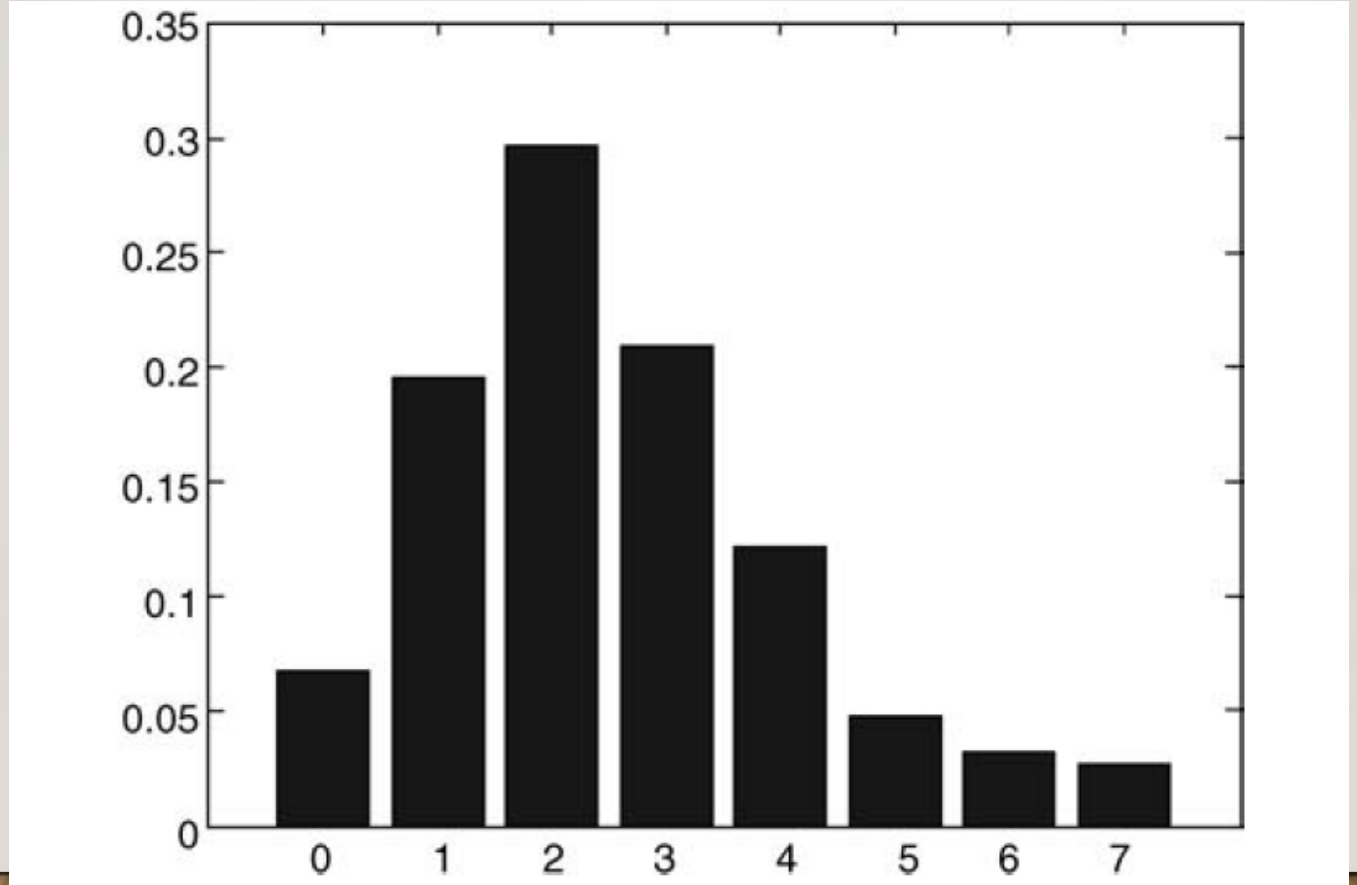


FIGURE 9.1 Example of histogram for an image with eight gray levels.

6.7 COMPUTING IMAGE HISTOGRAMS

- To compute the histogram of an 8-bit (256 gray levels) monochrome image, an array of 256 elements (each of which acts as a counter) is created and initialized with zeros. The image is then read, one pixel at a time, and for each pixel the array position corresponding to its gray level is incremented.

- After the whole image is processed, each array element will contain the number of pixels whose gray level corresponds to the element's index. These values can then be normalized, dividing each of them by the total number of pixels in the image.

6.7 COMPUTING IMAGE HISTOGRAMS

- For images with more than 8 bits per pixel, it is not practical to map each possible gray level k ($0 \leq k \leq K$) to an array element: the resulting array would be too large and unwieldy. Instead, we use a technique known as *binning*, by which an array of B elements—where B is the number of histogram *bins* (or *buckets*), $B \leq K$—is created and initialized with zeros. In this case, each bin $h(j)$ stores the number of pixels having values within the interval $r_j \leq r < r_{j+1}$ and equation (6.7) can be rewritten as

$$h(j) = card\{(x,y) | r_j \leq f(x,y) < r_j + 1\} for\ 0 \leq j < B \qquad (6.8)$$

- The lower limit of each bin can be obtained by the following expression:

$$r_j = j \cdot \frac{K}{B} = j \cdot k_B \qquad (6.9)$$

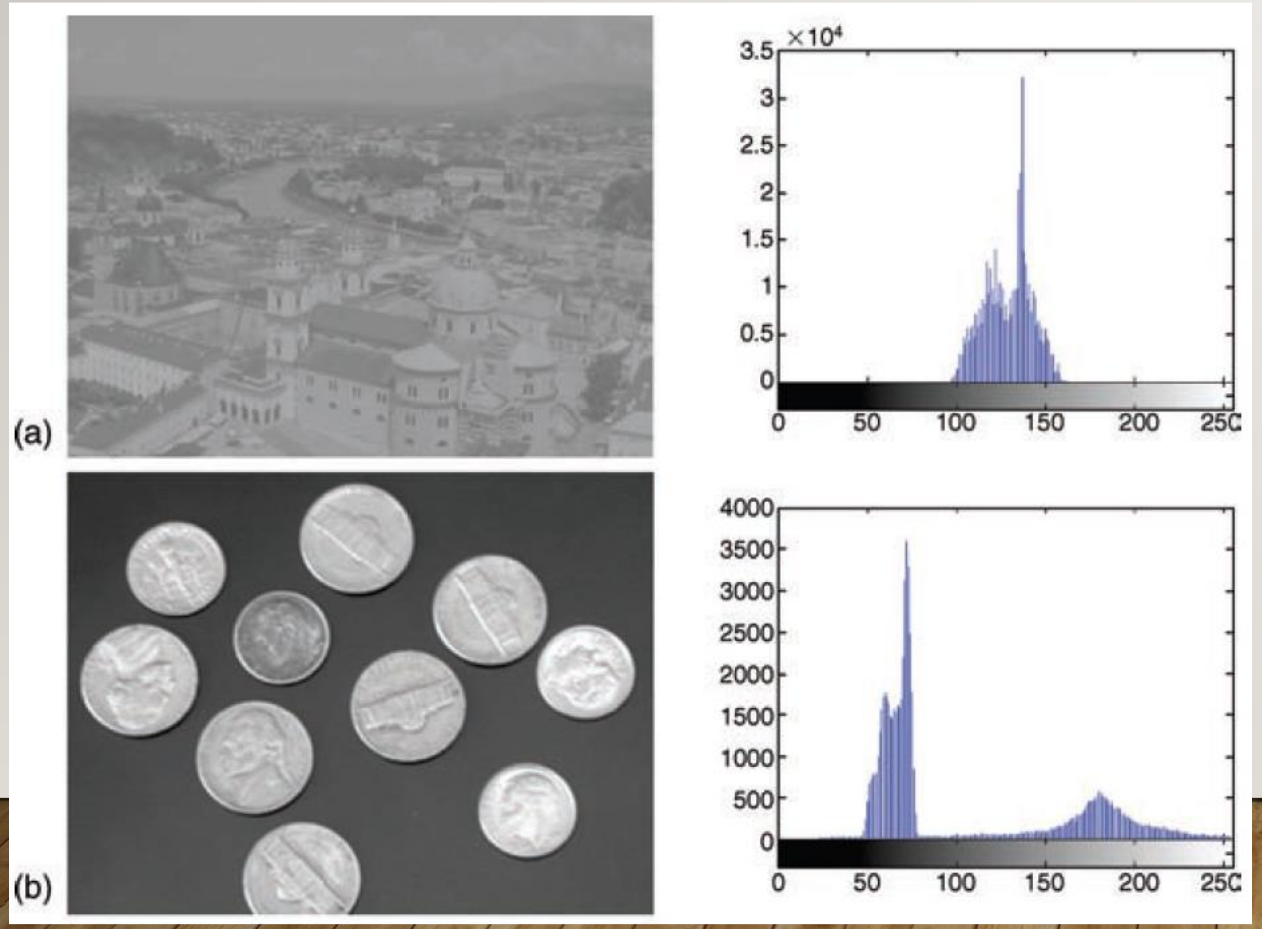
where k_B is the length of each interval.

6.8 INTERPRETING IMAGE HISTOGRAMS

- Histograms provide an easy, practical, and straightforward way of evaluating image attributes, such as overall contrast and average brightness. The histogram of a predominantly dark image contains a concentration of bars on the lower end of the gray-level range (Figure 9.2c), whereas the histogram for a bright image is mostly concentrated on the opposite end (Figure 9.2d).
- For a low contrast image, the histogram is clustered within a narrow range of gray levels (Figure 9.2a), whereas a high contrast image usually exhibits a bimodal histogram with clear separation between the two predominant modes (Figure 9.2b).

6.8 INTERPRETING IMAGE HISTOGRAMS

FIGURE 9.2 Examples of images and corresponding histograms. Original image in part (b): courtesy of MathWorks.



EXAMPLE 6.6

- Figure 9.2 shows four examples of monochrome images and their corresponding histograms (obtained using the *imhist* function in MATLAB).
- The histogram in Figure 9.2a shows that the pixels are grouped around intermediate gray-level values (mostly in the [100, 150] range), which corresponds to an image with low contrast. Figure 9.2b shows a typical bimodal histogram, one that has two distinctive hills, a more pronounced one in the dark region (background) and the other smaller one in the light region of the histogram (foreground objects).
- In these situations, it can be said that the corresponding image has high contrast since the two modes are well spaced from each other.

EXAMPLE 6.6

- In part (c), the histogram exhibits a large concentration of pixels in the lower gray levels, which corresponds to a mostly dark image.
- Finally, in Figure 9.2d, the pixel values are grouped close to the higher gray-level values, which corresponds to a bright image.
- Histograms have become a popular tool for conveying image statistics and helping determine certain problems in an image. Their usefulness can be demonstrated by the fact that many contemporary digital cameras have an optional real-time histogram overlay in the viewfinder to prevent taking underexposed or overexposed pictures.

EXAMPLE 6.6

- Histograms can be used whenever a statistical representation of the gray-level distribution in an image or video frame is desired. Histograms can also be used to enhance or modify the characteristics of an image, particularly its contrast.

- Some of these techniques, generally called *histogram modification* (or *modeling*) *techniques*, are *histogram equalization*, *histogram specification* (*matching*), *histogram stretching* (*input cropping*), and *histogram shrinking* (*output cropping*).

6.9 HISTOGRAM EQUALIZATION

- Histogram equalization is a technique by which the gray-level distribution of an image is changed in such a way as to obtain a uniform (flat) resulting histogram, in which the percentage of pixels of every gray level is the same. To perform histogram equalization, it is necessary to use an auxiliary function, called the *transformation function*, $T(r)$. Such transformation function must satisfy two criteria:
  1. T(r) must be a monotonically increasing function in the interval 0 ≤ r ≤ L – 1.
  2. 0 ≤ T(r) ≤ L – 1 for 0 ≤ r ≤ L – 1.
- The most usual transformation function is the cumulative distribution function (cdf) of the original probability mass function, given by

$$s_k = T(r_k) = \sum_{j=0}^{k} \frac{n_j}{n} = \sum_{j=0}^{k} p(r_j)$$

where s_k is the new (mapped) gray level for all pixels whose original gray level used to be r_k.

6.9 HISTOGRAM EQUALIZATION

In MATLAB

MATLAB's IPT has a built-in function to perform histogram equalization of a monochrome image: *histeq*. For the sake of histogram equalization, the syntax for *histeq* is usually J = histeq(I,n), where n (whose default value is 64) is the number of desired gray levels in the output image. In addition to histogram equalization, this function can also be used to perform histogram matching.

EXAMPLE 6.7

Assume the histogram data from Table 9.1 and its graphic representation (Figure 9.1). Calculate the equalized histogram using the cdf of the original probability mass function as a transformation function and plot the resulting histogram.

EXAMPLE 6.7

Solution

Using the cdf as the transformation function, we can calculate

$$s_0 = T(r_0) = \sum_{j=0}^{0} p(r_j) = p(r_0) = 0.068$$

Similarly,

$$s_1 = T(r_1) = \sum_{j=0}^{1} p(r_j) = p(r_0) + p(r_1) = 0.264$$

and $s_2 = 0.560$, $s_3 = 0.769$, $s_4 = 0.891$, $s_5 = 0.939$, $s_6 = 0.972$, and $s_7 = 1$. The transformation function is plotted in Figure 9.3.

EXAMPLE 6.7

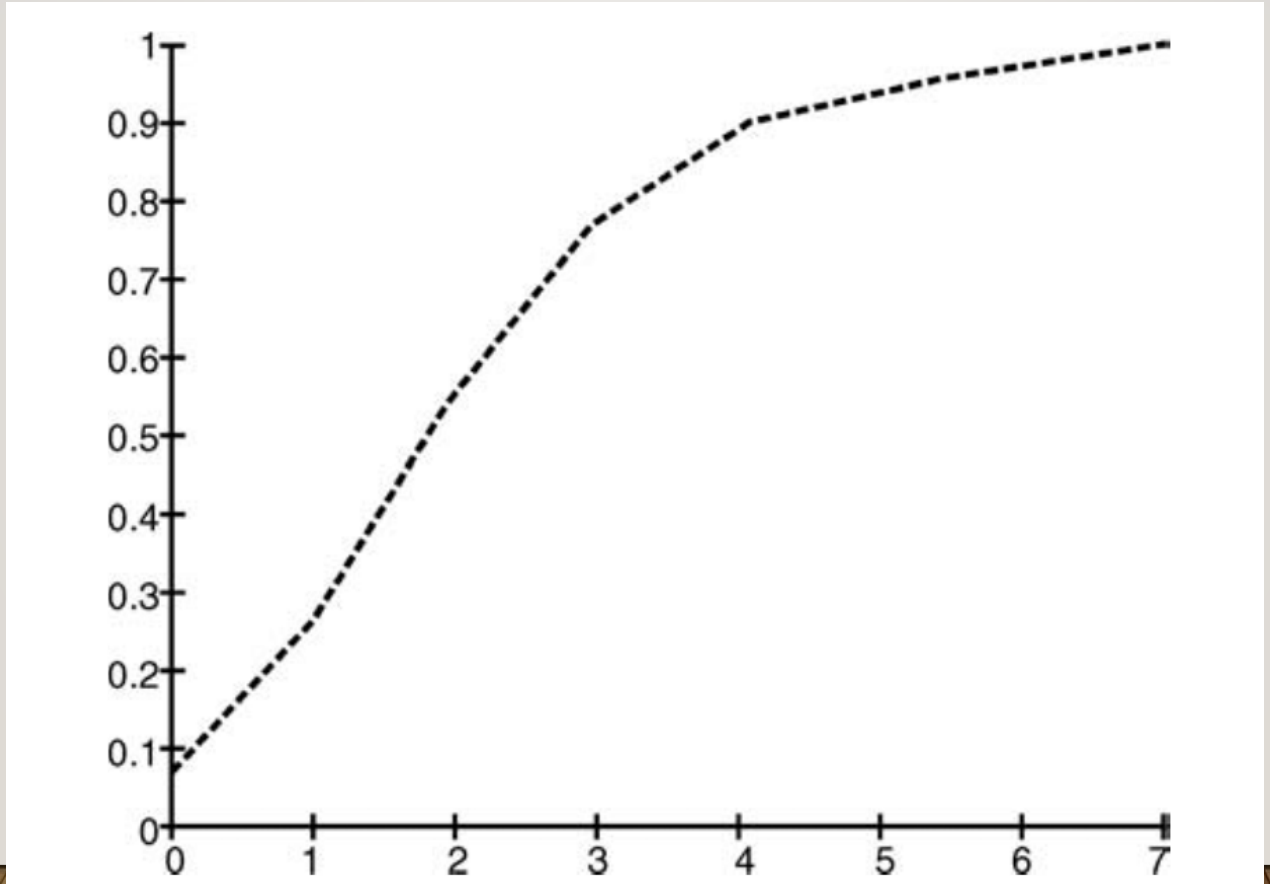


FIGURE 9.3 Transformation function used for histogram equalization.

EXAMPLE 6.7

- Since the image was quantized with only eight gray levels, each value of s_k must be rounded to the closest valid (multiple of 1/7) value. Thus, $s_0 \approx 0$, $s_1 \approx 2$, $s_2 \approx 4$, $s_3 \approx 5$, $s_4 \approx 6$, $s_5 \approx 7$, $s_6 \approx 7$, and $s_7 \approx 7$.
- The above values indicate that the original histogram bars must be shifted (and occasionally grouped) according to the following mapping: the original level $r_0 = 0$ should be mapped to the new level $s_0 = 0$, which means the corresponding bar should not change. The 3214 pixels whose original gray level was (1/7) should be mapped to $s_1 = 2(1/7)$. Similarly, those pixels whose gray level was 2 should be mapped to 4, those with r = 3 should be mapped to 5, and those with gray level 4 should be mapped to 6. Finally, the three bins that correspond to pixels with gray levels 5, 6, and 7 should be added and mapped to 7.

EXAMPLE 6.7

- The resulting (equalized) histogram is shown in Figure 9.4 and the corresponding values are presented in Table 9.2.
- It should be noted that in the equalized histogram (Figure 9.4), the pixels are more evenly distributed along the gray scale than in the original one (Figure 9.1).
- Although clearly not a perfectly flat result, it should be interpreted as "the best possible result that can be obtained for this particular image using this transformation function."

EXAMPLE 6.7

TABLE 9.2 Equalized Histogram: Values

Gray Level (s_k)	n_k	$p(s_k)$
0	1120	0.068
1	0	0.000
2	3214	0.196
3	0	0.000
4	4850	0.296
5	3425	0.209
6	1995	0.122
7	1780	0.109
Total	16,384	1.000

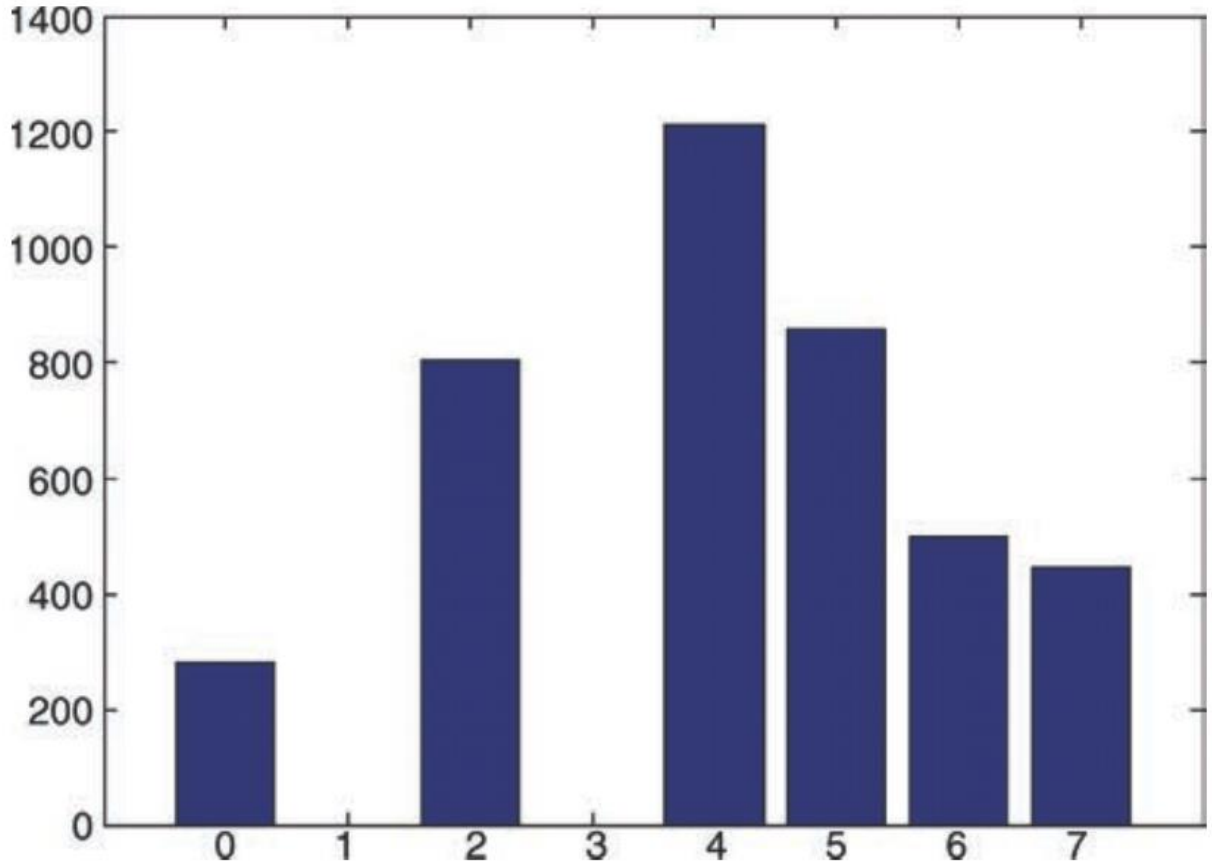
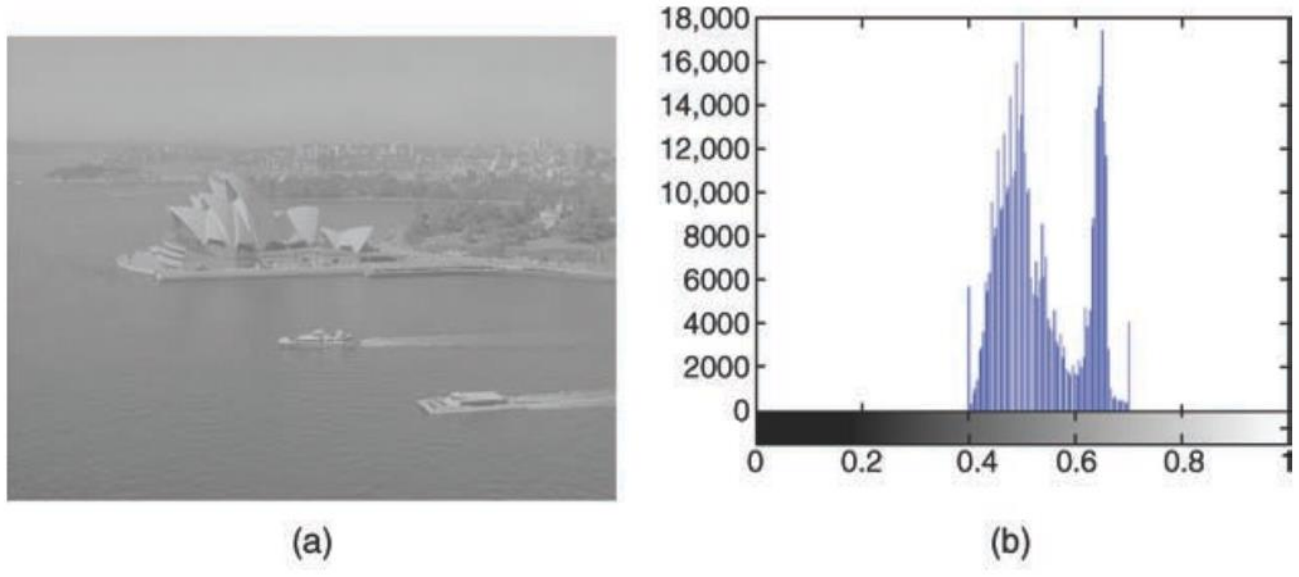


FIGURE 9.4 Equalized histogram—graph.

EXAMPLE 6.8

- Figure 9.5 shows an example (obtained using the *histeq* function in MATLAB) of employing histogram equalization to improve the contrast of a 600 x 800 image with 256 gray levels.
- Part (a) shows the original image, whose histogram is plotted in Figure 9.5b. Part (d) shows the equalized histogram, corresponding to the image in Figure 9.5c. You may have noticed that the image after equalization exhibits a little bit of *false contouring*, particularly in the sky portion of the image. This can be seen as an inevitable "side effect" of the histogram equalization process.
- The histogram equalization algorithm described above is *global*; that is, once the mapping (transformation) function is computed, it is applied (as a lookup table) to all pixels of the input image.

EXAMPLE 6.8



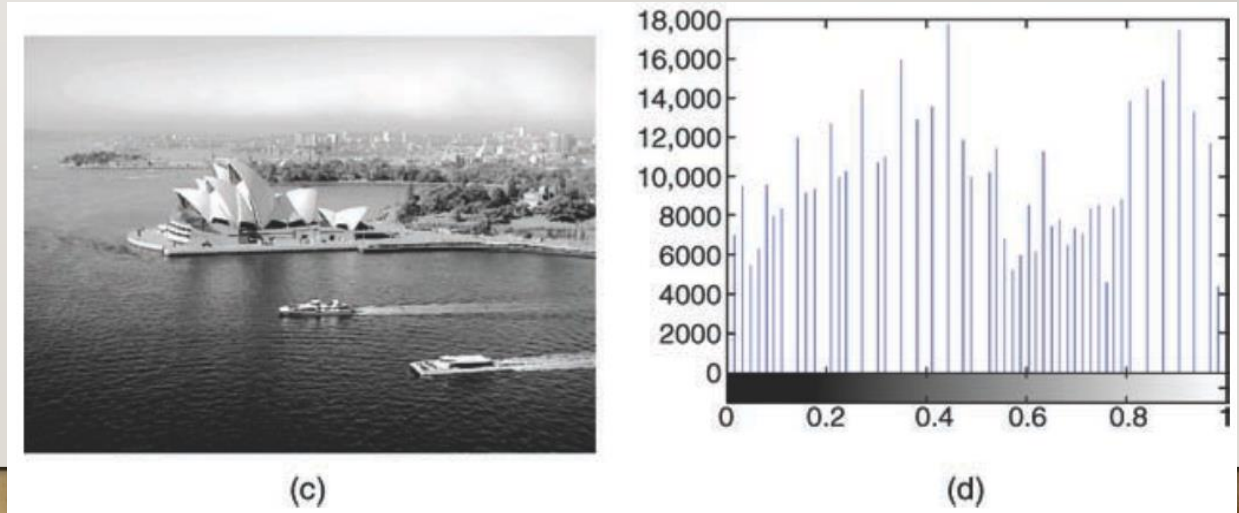


FIGURE 9.5 Use of histogram equalization to improve image contrast.

EXAMPLE 6.8

- When the goal is to enhance details in small areas within an image, it is sometimes necessary to apply a *local* version of the algorithm. The local variant of histogram equalization consists of adopting a rectangular (usually square) sliding window (also called a *tile*) and moving it across the entire image.
- For each image pixel (aligned with the center of the sliding window), the histogram in the neighborhood delimited by the window is computed, the mapping function is calculated, and the reference pixel is remapped to the new value as determined by the transformation function. This process is clearly much more computationally expensive than its global variant and often results in a noisy-looking output image.

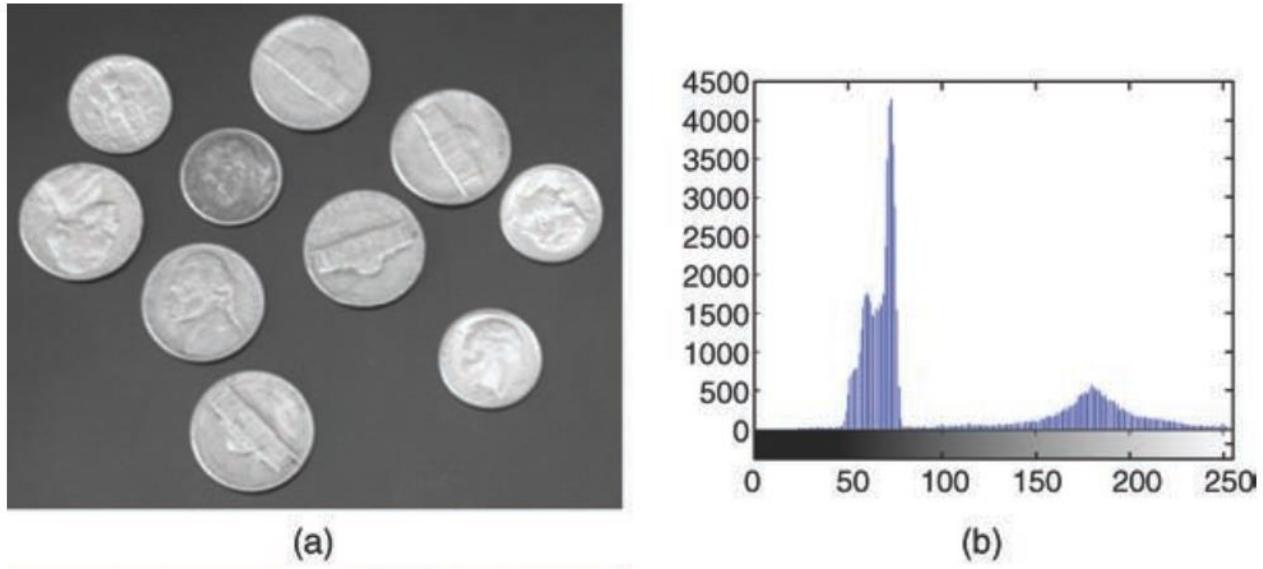
EXAMPLE 6.9

- Figure 9.6 shows a comparison between local and global histogram equalization (obtained using the *adapthisteq* and *histeq* functions in MATLAB, respectively). Parts (a) and (b) show the original image and its histogram, parts (c) and (d) show the results of global histogram equalization, and parts (e) and (f) show the results of local histogram equalization that preserves the bimodal nature of the original histogram while still improving the contrast of the image.

In MATLAB

- MATLAB's IPT has a built-in function to perform local histogram equalization of a monochrome image, *adapthisteq*, which, unlike *histeq*, operates on small data regions (tiles), rather than the entire image.

EXAMPLE 6.9



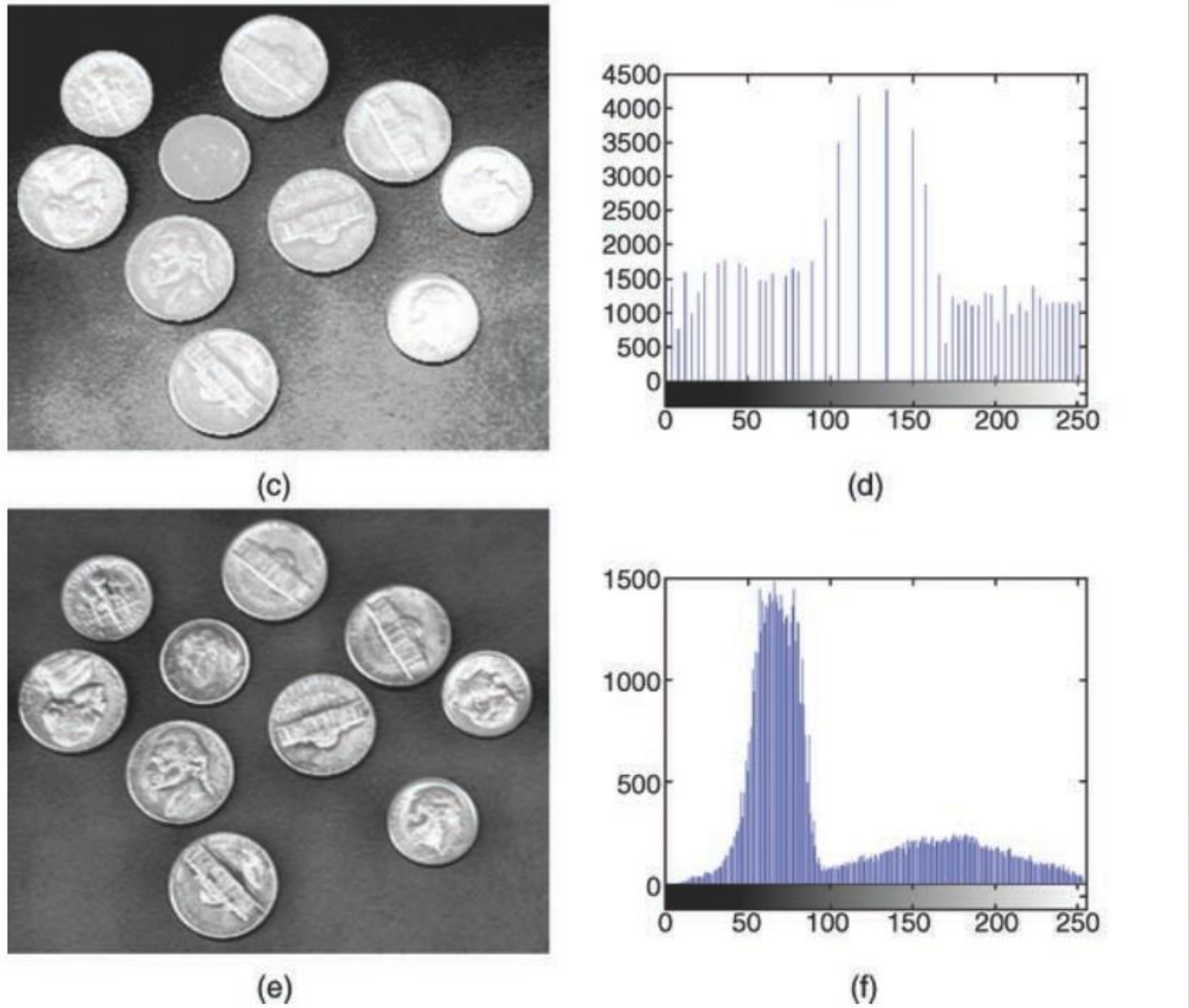


FIGURE 9.6 Global versus local histogram equalization. Original image: courtesy of MathWorks.

6.10 DIRECT HISTOGRAM SPECIFICATION

- Despite its usefulness in contrast enhancement, histogram equalization is a rather inflexible technique. Its only modifiable parameter is the choice of transformation function, which is normally chosen to be the cdf of the original probability mass function (a convenient choice since the information needed to compute the transformation function can be extracted directly from the pixel values).
- There might be situations, however, in which one wants to be able to perform specific changes on the original histogram. In these situations, a useful technique is the direct histogram specification, also known as histogram matching.

6.10 DIRECT HISTOGRAM SPECIFICATION

Given an image (and its original histogram) and the desired resulting histogram, the direct histogram specification consists of the following:

1. Equalizing the original image's histogram using the cdf as a transformation function:

$$s_k = T(r_k) = \sum_{j=0}^{k} \frac{n_j}{n} = \sum_{j=0}^{k} p(r_j)$$

2. Equalizing the desired probability mass function (in other words, equalizing the desired histogram):

$$v_k = G(z_k) = \sum_{j=0}^{k} p(z_j)$$

3. Applying the inverse transformation function

$$z = G^{-1}(s)$$

to the values obtained in step 1.

6.10 DIRECT HISTOGRAM SPECIFICATION

In MATLAB

The *histeq* function introduced previously can also be used for histogram matching. In this case, the syntax for histeq usually changes to J = histeq(I,h), where h (a 1D array of integers) represents the specified histogram.

6.10 DIRECT HISTOGRAM SPECIFICATION

Let us use the histogram from Table 9.1 one more time. Assume that we want to modify this histogram in such a way as to have the resulting pixel distribution as shown in Table 9.3 and plotted in Figure 9.7a.

Following the steps outlined above, calculate and plot the resulting histogram that best matches the desired characteristics.

TABLE 9.3 Desired Histogram

Gray Level (s_k)	n_k	$p(s_k)$
0	0	0.0
1	0	0.0
2	0	0.0
3	1638	0.1
4	3277	0.2
5	6554	0.4
6	3277	0.2
7	1638	0.1
Total	16,384	1.0

6.10 DIRECT HISTOGRAM SPECIFICATION

Solution

The equalized histogram has already been calculated before and its results are shown in Table 9.2. The next step consists in obtaining the cdf of the desired probability mass function.

Using equation (9.10), we find the following values: $v_0 = 0$, $v_1 = 0$, $v_2 = 0$, $v_3 = 0.1$, $v_4 = 0.3$, $v_5 = 0.7$, $v_6 = 0.9$, and $v_7 = 1$.

The last step—and the most difficult to understand when studying this technique for the first time—is obtaining the inverse function. Since we are dealing with discrete values, the inverse function can be obtained simply by searching, for each value of s_k, the closest value of v_k.

For instance, for $s_1 = 2/7 \approx 0.286$, the closest value of v_k is $v_4 = G(z_4) = 0.3$. In inverse function notation, $G^{-1}(0.3) = z_4$. Therefore, pixels that were shifted to gray level s_1 after the original histogram equalization should be mapped to gray level z_4.

6.10 DIRECT HISTOGRAM SPECIFICATION

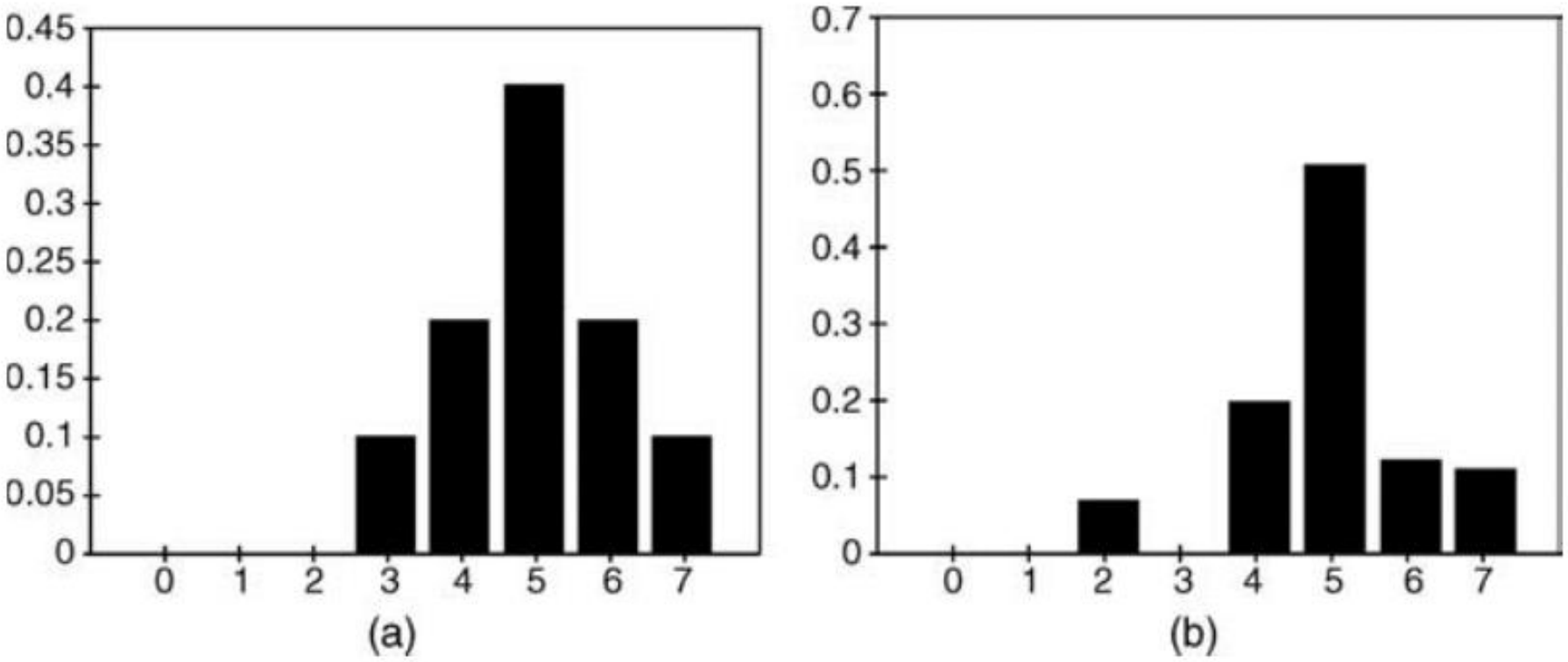


FIGURE 9.7 Histogram matching: (a) desired (specified) histogram; (b) resulting histogram.

6.10 DIRECT HISTOGRAM SPECIFICATION

TABLE 9.4 Direct Histogram Specification: Summary

k	$p(r_k)$	s_k ($\times 1/7$)	Maps to	v_k	$p(z_k)$
0	0.068	0	z_2	0.00	0.000
1	0.196	2	z_4	0.00	0.000
2	0.296	4	z_5	0.00	0.000
3	0.209	5	z_5	0.10	0.100
4	0.122	6	z_6	0.30	0.200
5	0.048	7	z_7	0.70	0.400
6	0.033	7	z_7	0.90	0.200
7	0.028	7	z_7	1.00	0.100

END OF LECTURE 6

LECTURE 7 – IMAGE RESTORATION