國立臺灣大學電機資訊學院電信工程學研究所

碩士論文

Graduate Institute of Communication Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

以影像分割及邊界描述為基礎的影像壓縮技術

Image Compression by Segmentation and Boundary Description

黃俊德

Jiun-De Huang

指導教授：丁建均 博士

Advisor: Jian-Jiun Ding, Ph.D.

中華民國 97 年 6 月

June, 2008

誌謝

能完成這篇論文，我要特別感謝我的指導教授丁建均老師，從一開始訓練我們的英文寫作能力，接著幫助我們找到論文的方向，直到最後幾個月的密集meeting，老師總是很有耐心的指導我們，並且在適當的時候都能提供我們一些新的想法與在我們遇到瓶頸的時候騰出時間來幫忙解決。

另外也要感謝實驗室的同學們：國銓、于哲、育思、汝川，在這兩年來一起努力時對我的幫助與鼓勵。還有謝謝學弟們在口試時的幫忙，以及自恆在口試前的幫忙審稿。

最後要感謝我的父母，讓我在經濟上沒有後顧之憂，可以專心的學習與研究。

中文摘要

在現有的影像壓縮技術上，如 JPEG，皆是對整張圖做相同的處理，而不會對於不同的影像內容而有所調整，所以壓縮率有其極限。而在新一代的影像壓縮技術中，是以影像分割為基礎，將影像盡可能的切割成數個特性或色彩值近似的區塊，每個區塊分別有不同的形狀與色彩值。由於同區塊中的色彩值通常會有高度相關，所以理論上可以產生更高的壓縮率。

對於影像分割，大致上是根據像素值的兩種性質：不連續性與相似性。為了找到不連續的像素值，我們將會介紹影像的邊緣偵測的基本技術並且提出一種結合了傳統的微分法與希爾伯轉換法的可適性方法，叫做短時響應的希爾伯轉換。我們也將會介紹許多其他的方法來做影像切割。我們主要的目的是找出適合的分割結果讓壓縮可以更有效率。

做好影像切割之後，我們會介紹 JPEG 標準中使用到的壓縮演算法，並應用在我們提出的方法中。為了有效率的記錄每個影像區塊的輪廓，我們先介紹一些常用的邊界描述子並提出兩種改進過後的邊界描述子。為了壓縮影像區塊的色彩值，我們將會討論如何將一個不規則形狀的影像區塊轉換到頻域，接著我們便可以對轉換後的頻率係數做量化及編碼來降低資料量。最後我們將結果與使用 JPEG 標準壓縮的圖片做比較，證實在可接受的失真範圍下，壓縮率的確可以增加很多。

ABSTRACT

The present technique of image compression, like the JPEG standard, makes the same process to whole image and does not adjust the parameters based on the local characteristics of the image. Therefore, it has a limit to its compression ratio. However, a new compression technique called segmentation-based image compression has been developed. It segments an image to several regions with similar characteristic or color. Because each image segment has different shapes and color values, we compress these regions individually. Due to the high correlation of the color values in an image segment, we could achieve higher compression ratio in theory.

The technique of image segmentation is based on two properties of color values: discontinuity and similarity. To find the discontinuity of the color values, we will introduce the image edge detection technique and propose an adaptive method called the short response Hilbert transform (SRHLT) which combines the traditional differential method and the Hilbert transform method. We will also discuss many other ways to segment an image. The main object is to find a suitable segmented result that can be compressed efficiency.

After Segmentation, we will introduce the basic image compression algorithms in JPEG standard and apply them in our proposed methods. To record the boundary of an image segment efficiently, we will introduce some popular boundary descriptors and propose two improved boundary descriptors. To compress the color values of an image segment, we will discuss how to transform an arbitrary-shape image segment to frequency domain. Then we can quantize and encode the frequency coefficients to decrease the information quantity. Finally, we will compare the result with that of JPEG standard and prove that the compression ratio could be increase a lot under acceptable distortion.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Because of the explosively increasing information of image and video in various storage devices and Internet, the image compression technique becomes more and more important. For example, if we have a 4×6 inch photo with 300 dpi, the required data quantity can be counted as $4\times6\times(300)^2 \times 3$ bytes per pixel = 6.48 megabytes. In other words, if we do not compress these images, a CD of 650MB can only store about 100 photos. However, if we compress these photos by the general JPEG standard whose compression ratio is about 15:1, a compressed photo could have only 432 kilobytes and a CD can store up to 1500 photos.

Although the compression ratio of the JPEG standard seems large, it still can increase. The present technique of image compression makes the same process to a whole image and does not adjust the parameters based on the local characteristics of the image. Therefore, it has a limit to its compression ratio. However, a new compression technique, called the segmentation-based image compression has been developed. It segments an image to several regions with similar characteristic or color. Because each image segment has different shapes and color values, we compress these regions individually. Due to the high correlation of the color values in an image segment, it could reach higher compression ratio in theory.

The diagram of the segmentation-based image compression model is shown in Fig. 1.1. It can be divided into three parts. First of all, the image segmentation process input an image and output its boundary and image segment of each segmented region. Then the two data are encoded by different compression processes. The two encoded data are combined to the bit stream and the bit stream is stored in some storage.

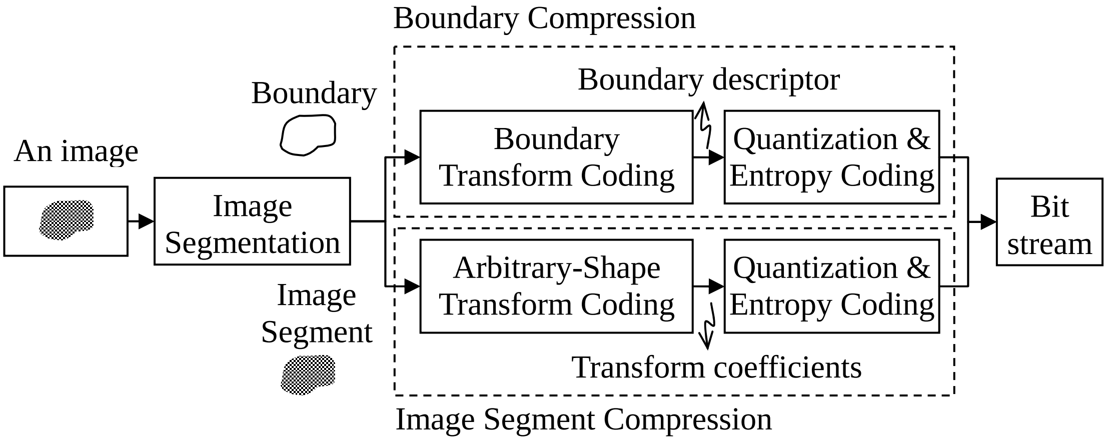

Boundary Compression
Boundary descriptor
Boundary
An image
Boundary Transform Coding
Quantization & Entropy Coding
Image Segmentation
Bit stream
Arbitrary-Shape Transform Coding
Quantization & Entropy Coding
Image Segment
Transform coefficients
Image Segment Compression

Fig. 1.1 Segmentation-based image compression model.

The technique of image segmentation is based on two properties of color values: discontinuity and similarity. To find the discontinuity of the color values, we will introduce the image edge detection technique in the Chapter 2. We will propose an adaptive method called the short response Hilbert transform (SRHLT) which combines the traditional differential method and the Hilbert transform method. In Chapter 3, we will discuss many other ways to segment an image. The main object is to find a suitable segmented result that can be compressed efficiently.

In Chapter 4, we will introduce the basic image compression algorithms in JPEG standard and apply them in our proposed methods.

In Chapter 5, we will introduce some popular boundary descriptors to record the boundary of an image segments efficiently. In Chapter 6, we will propose two improved boundary descriptors.

In Chapter 7, we will discuss how an arbitrary-shape image segment is compressed. In Chapter 8, we implement the arbitrary-shape image segment compression by modified DCT. We transform the color values to frequency domain and then we quantize and encode them to decrease the information quantity. Finally, we will compare the result with that of JPEG standard and prove that the compression ratio could be increase a lot under acceptable distortion.

Chapter 2

Edge Detection

Local discontinuities in image intensity from one pixel to another are called edges, which are considered as the boundaries between different textures. The edges for an image are always the important characteristics that offer an indication for a higher frequency. Detection of edges for an image may help for image segmentation described in the next chapter, and also help for well matching, such as image reconstruction and so on.

Edge detection is an approach for detecting meaningful discontinuities in gray level. The most common method for edge detection is to calculate the differentiation of an image. The first-order derivatives in an image are computed using the gradient, and the second-order derivatives are obtained using the Laplacian. Another method for edge detection uses Hilbert Transform. And we have proposed a new method called short response Hilbert transform (SRHLT) that combines the differentiation method and the Hilbert transform method.

2.1 Differentiation Method for Edge Detection

2.1.1 First-Order Derivative Edge Detection

We define a point in an image which is an edge point if its two-dimensional first-order derivative is greater than a specified threshold. First-order derivatives of a digital image are based on various approximations of the 2-D gradient. The gradient of an image $f(x, y)$ at location (x, y) is defined as the vector:

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix}. \tag{2.1}$$

The gradient vector points are in the direction of maximum rate of change of f at coordinates (x, y).

An important quantity in edge detection is the magnitude of this vector, denoted ∇f, where

$$\nabla f = \left| \nabla \mathbf{f} \right| = \sqrt{G_x^2 + G_y^2}\ . \tag{2.2}$$

The magnitude gives the maximum rate of increase of f (x, y) per unit distance in the direction of $\nabla \mathbf{f}$.

Another important quantity is the direction of the gradient vector. That is,

$$\text{angle of } \nabla \mathbf{f} = \tan^{-1}\left(\frac{G_y}{G_x} \right), \tag{2.3}$$

where the angle is measured with respect to the x-axis. The direction of an edge at (x, y) is perpendicular to the direction of the gradient vector at that point.

Computation of the gradient of an image is based on obtaining the partial derivatives of $\partial f / \partial x$ and $\partial f / \partial y$ at every pixel location. Let the 3×3 area shown in Fig. 2.1 represent the gray levels in a neighborhood of an image. One of the simplest ways to implement a first-order partial derivative at point z_5 is to use the following Roberts cross-gradient operators [32]:

$$G_x = (z_9 - z_5) \tag{2.4}$$

and

$$G_y = (z_8 - z_6)\ . \tag{2.5}$$

These derivatives can be implemented for an entire image by using the masks shown in

Fig. 2.2 with the procedure of convolution.

Another approach using masks of size 3×3 shown in Fig. 2.3 which is given by

$$G_x = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3) \tag{2.6}$$

and

$$G_y = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7) \tag{2.7}$$

is called Prewitt operators [30]. In this operator, the difference between the first and third rows of the 3×3 image region approximates the derivative in the *x*-direction, and the difference between the third and first columns approximates the derivative in the *y*-direction.

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Fig. 2.1 A 3×3 area of an image.

0	0	0
0	–1	0
0	0	1

0	0	0
0	0	–1
0	1	0

Fig. 2.2 The Roberts operators.

–1	–1	–1
0	0	0
1	1	1

–1	0	1
–1	0	1
–1	0	1

Fig. 2.3 The Prewitt operators.

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

Fig. 2.4 The Sobel operators.

In [19], a slight variation of these two equations uses a weight of 2 in the center coefficient:

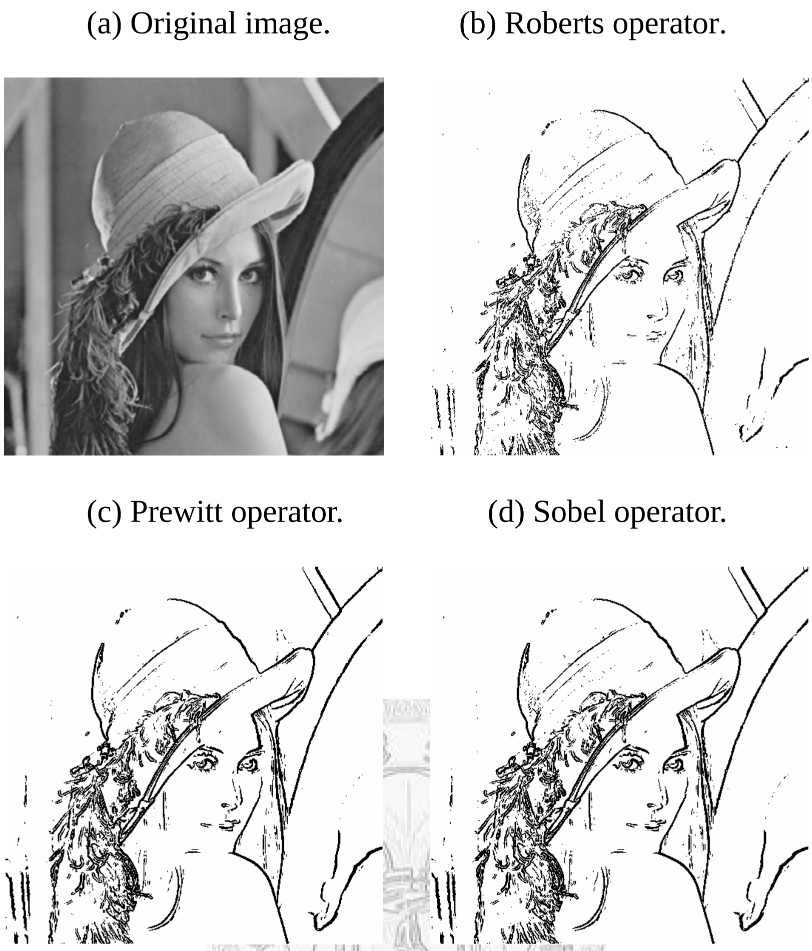
$$G_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{2.8}$$

$$G_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \tag{2.9}$$

A weight value of 2 is used to achieve some smoothing by giving more importance to the center point. Fig. 2.4, called the Sobel operators, is used to implement these two equations.

There are several operators developed such as Frei-Chen operator [20], Kirsch operator [23], Robinson operator [34], and Nevatia-Babu operator [28]. They have their own property and have good effect on different edge, respectively.

After the edge gradient is formed for the differential edge detection methods, the gradient is compared to a threshold to determine if an edge exists. The threshold value determines the sensitivity of the edge detector. For noise-free images, the threshold can be chosen such that all amplitude discontinuities of a minimum contrast level are detected as edges, and all others are called non-edges. With noisy images, threshold selection becomes a trade-off between missing valid edges and creating noise-induced false edges. Edge detection can be regarded as a hypothesis-testing problem to determine if an image region contains an edge or contains no edge, see [13].



(a) Original image. (b) Roberts operator.

(c) Prewitt operator. (d) Sobel operator.

Fig. 2.5 Examples of edge detection using three different operators.

2.1.2 Second-Order Derivative Edge Detection

The Laplacian of a 2-D function $f(x, y)$ is a second-order derivative defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\,. \tag{2.10}$$

There are two digital approximations to the Laplacian for a 3×3 region:

$$\nabla^2 f = 4z_5 - (z_2 + z_4 + z_6 + z_8) \tag{2.11}$$

$$\nabla^2 f = 8z_5 - (z_1 + z_2 + z_3 + z_4 + z_6 + z_7 + z_8 + z_9)\,, \tag{2.12}$$

where the z's are defined in Fig. 2.1. Masks for implementing these two equations are shown in Fig. 2.6.

0	–1	0
–1	4	–1
0	–1	0

(a)

–1	–1	–1
–1	8	–1
–1	–1	–1

(b)

Fig. 2.6 Two kind of 3×3 Laplacian mask.

The Laplacian is usually combined with smoothing as a precursor to finding edges via zero-crossings. The 2-D Gaussian function

$$h(x,y) = -e^{-\frac{x^2+y^2}{2\sigma^2}}\,, \tag{2.13}$$

where σ is the standard deviation, blurs the image with the degree of blurring being determined by the value of σ. The Laplacian of *h* is

$$\nabla^2 h(x,y) = -\left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] e^{-\frac{r^2}{2\sigma^2}}\,. \tag{2.14}$$

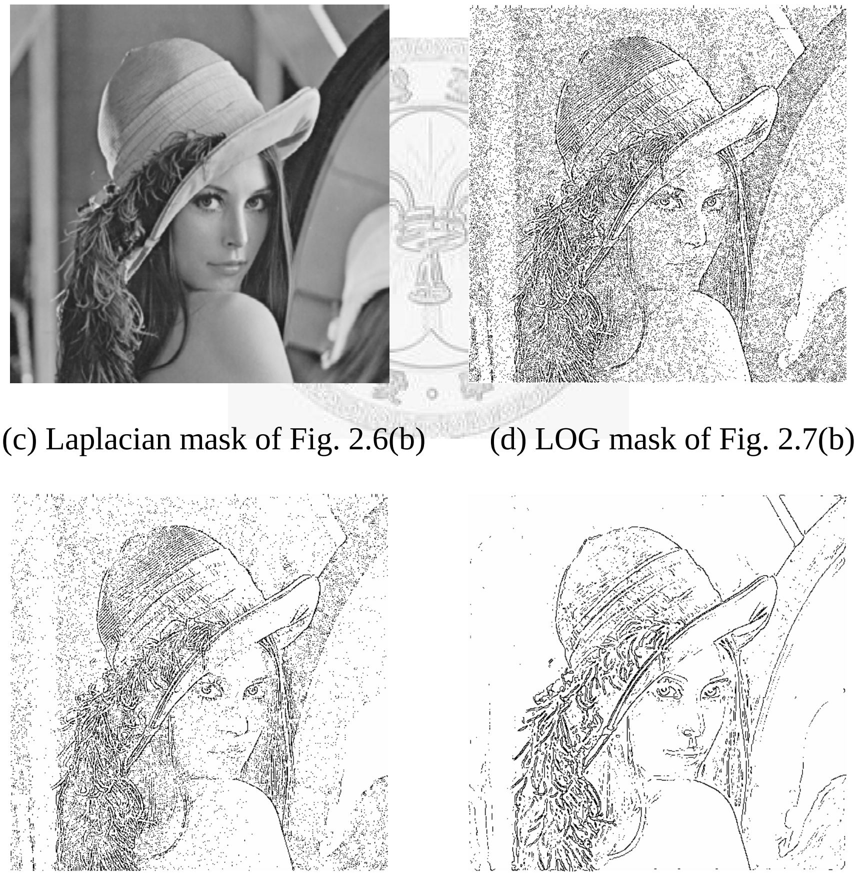
This function is commonly referred to as the Laplacian of Gaussian (LOG).

0	0	0	-1	-1	-2	-1	-1	0	0	0
0	0	-2	-4	-8	-9	-8	-4	-2	0	0
0	-2	-7	-15	-22	-23	-22	-15	-7	-2	0
-1	-4	-15	-24	-14	-1	-14	-24	-15	-4	-1
-1	-8	-22	-14	52	103	52	-14	-22	-8	-1
-2	-9	-23	-1	103	178	103	-1	-23	-9	-2
-1	-8	-22	-14	52	103	52	-14	-22	-8	-1
-1	-4	-15	-24	-14	-1	-14	-24	-15	-4	-1
0	-2	-7	-15	-22	-23	-22	-15	-7	-2	0
0	0	-2	-4	-8	-9	-8	-4	-2	0	0
0	0	0	-1	-1	-2	-1	-1	0	0	0

Fig. 2.7 An 11×11 mask approximation to Laplacian of Gaussian (LOG).

After calculating the two-dimensional second-order derivative of an image, we find the value of a point which is greater than a specified threshold and one of its neighbors is less than the negative of the threshold. The property of this point is called zero-crossing and we can denote it as an edge point.

We note two additional properties of the second derivative around an edge: (1) It produces two values for every edge in an image (an undesirable feature); and (2) an imaginary straight line joining the extreme positive and negative values of the second derivative would cross zero near the midpoint of the edge. This zero-crossing property of the second derivative is quite useful for locating the centers of thick edges.



(a) Original image. (b) Laplacian mask of Fig. 2.6(a)

(c) Laplacian mask of Fig. 2.6(b) (d) LOG mask of Fig. 2.7(b)

Fig. 2.8 Examples of edge detection using three different Laplacian masks.

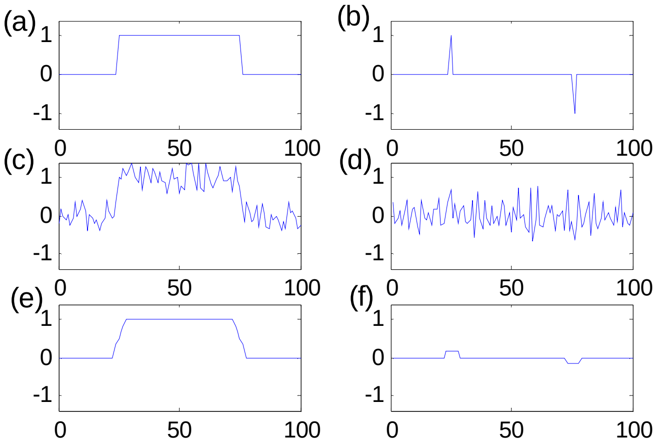


Fig. 2.9 Using differentiation to detect (a) the sharp edges, (c) the step edges with noise, and (e) the ramp edges. (b)(d)(e) are the results of differentiation of (a)(c)(e).

2.1.3 The Drawbacks of the Differentiation Method for Edge Detection

Differentiation is the simplest way to find the edges of a signal or an image. After differentiation, the edge will become a peak (or a dip) and the plain will be near to zero, as the experiments in Fig. 2.9(a)(b).

Although using the method of differentiation or the 3×3 mask for edge detection is very simpler, they have several drawbacks, such as:

(1) Sensitivity to noise. For example, in Fig. 2.9(c), the input is interfered by noise. After doing differentiation, the output (in Fig. 2.9(d)) does not have obvious peaks at the points of edges. Instead, many peaks occur at the non-edge region. Since noise usually has higher frequency and differentiation may magnify the high frequency components, using differentiation for edge detection has worse performance when the input is interfered by noise.

(2) Not good for ramp edges. This can be seen from Fig. 2.9(e)(f), where the input is not suddenly changed from 0 to 1 or 1 to 0. Instead, the edges have the form of:

$$g[n] = \pm(n - n_0)/B + 1/2 \quad \text{for } n_0 - B/2 < n < n_0 + B/2. \tag{2.15}$$

In this case, from the results in Fig. 2.9(f), the outputs at the ramp edges will be much smaller than those of the sharp edges in Fig. 2.9(b). If we choose the threshold as a smaller value, the two ramp edges cannot be detected.

(3) Make no difference between the significant edge and the detailed edge. Since differentiation only observes the variations among the adjacent pixels, it cannot observe the longer trend of variations. When using differentiation, too many details are treated as edges and the edges that are really significant are not stressed.

2.2 Hilbert Transform for Edge Detection

There is another method for edge detection that uses the Hilbert transform (HLT). The HLT is

$$g_H(\tau) = h(x) * g(x), \quad \text{where} \;\; h(x) = \frac{1}{\pi x}, \tag{2.16}$$

and * means convolution. Alternatively,

$$G_H(f) = H(f)G(f) \tag{2.17}$$

where $G(f) = FT\,[g(x)]$ (FT means the Fourier transform), $G_H(f) = FT\,[g_H\,(x)]$, and

$$H(f) = -j\,\mathrm{sgn}(f), \tag{2.18}$$

where the sign function is defined as

$$\begin{aligned} &\mathrm{sgn}(f) = 1 \text{ when } f > 0, \\ &\mathrm{sgn}(f) = -1 \text{ when } f < 0, \\ &\mathrm{sgn}(0) = 0 \end{aligned} \tag{2.19}$$

Since the HLT has longer impulse response, using it for edge detection can much reduce the effect of noise. For the example in Fig. 2.10(c), even if the input is seriously affected by noise, after doing the HLT, there is still a peak and a dip at the location of edge, as Fig. 2.10(d). Thus, we can use the HLT to detect the edges that are affected by noise

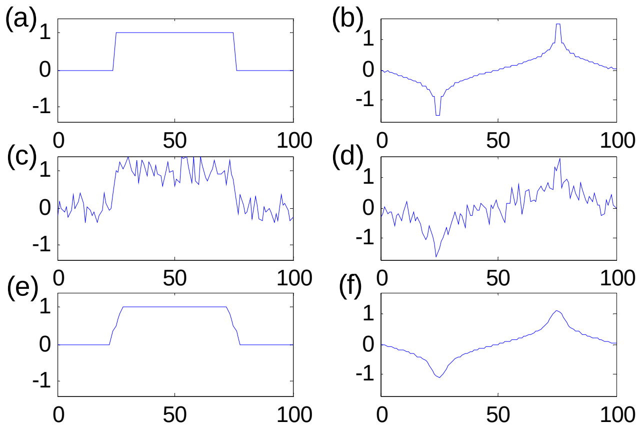


Fig. 2.10 Using HLTs to detect (a) the sharp edges, (c) the step edges with noise, and (e) the ramp edges. (b)(d)(e) are the results of the HLTs of (a)(c)(e).

successfully. Moreover, also due to the longer impulse response, the HLT can also successfully detect the ramp edges, as in Fig. 2.10(f).

However, using the HLT for edge detection has a drawback. From the experiments in Fig. 2.10(b), although after doing the HLT, there is still a peak or a dip at the location of the edge, the difference between the outputs of the edge and the non-edge points is not so obvious as Fig. 2.9(b), where differentiation is used. Thus, when the SNR is high and the edge is sharp, using the HLT for edge detection may not be better than using differentiation.

For a natural image, the edge is usually of the ramp form and inevitably affected by noise. Thus, generally speaking, using the HLT for edge detection will obtain better performance than using differentiation. It can be seen from the experiments in [39][42].

The discrete version of the HLT is

$$g_H[n] = IDFT\left(H[p]DFT\left(g[n]\right)\right), \tag{2.20}$$

where

$$\begin{aligned} &DFT\left(g[n]\right) = \sum_{n=0}^{N-1} g\left[n\right] e^{-j2\pi pn/M}, \\ &IDFT\left(F[p]\right) = \frac{1}{N}\sum_{p=0}^{N-1} e^{j2\pi pn/N} F[p], \end{aligned} \tag{2.21}$$

$$\begin{aligned} &H[p] = -j \quad \text{for } 0 < p < N/2, \\ &H[p] = j \quad\;\; \text{for } N/2 < p < N, \\ &H[0] = H[N/2] = 0. \end{aligned} \tag{2.22}$$

The 2-D form of the discrete HLT (also called as the discrete radial Hilbert transform (DRHLT) [41][42]) is

$$g_H\left[m,n\right] = IDFT_{2D}\left(H[p,q] DFT_{2D}\left(g[m,n]\right)\right), \tag{2.23}$$

where

$$DFT_{2D}\left(g[m,n]\right) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} g\left[m,n\right] e^{-j2\pi\frac{pm}{M}} e^{-j2\pi\frac{qn}{N}}, \tag{2.24}$$

and

$$\begin{aligned} &H\left[p,q\right] = \Phi\left(\theta\right), \\ &\quad \theta = \cos^{-1}(p/r) = \sin^{-1}(q/r),\ \ r = \sqrt{p^2+q^2} \\ &\quad \text{when } [p,q] \neq [0,0],\ p \neq M/2,\ q \neq N/2, \\ &H[0,0] = H[M/2,n] = H[m,N/2] = 0, \end{aligned} \tag{2.25}$$

and $\Phi(\theta)$ is any function that satisfies

$$\Phi(\theta+\pi) = -\Phi(-\theta). \tag{2.26}$$

For example, according to [39], we can choose $\Phi(\theta)$ as:

$$\Phi(\theta) = \exp(j\theta). \tag{2.27}$$

In [42], S. C. Pei and J. J. Ding discussed how to use the DRHLT for edge detection in detail and showed that the DRHLT is not only effective for edge detection but also useful for directional edge detection and corner detection.

2.3 Short Response Hilbert Transform for Edge Detection

Based on Canny’s criterion [16], we develop the short response Hilbert transform (SRHLT), which is the intermediate of the original HLT and the differentiation operation. For edge detection, the SRHLT can compromise the advantages of the HLT and differentiation. It can well distinguish the edges from the non-edge regions and at the same time are robust to noise. We also find that there are many ways to define the SRHLT. If the constraints which come from Canny’s criterion are satisfied, the resultant SRHLT will have good performance in edge detection.

2.3.1 The Definition of the SRHLT

We combine the HLT and differentiation to define the SRHLT. From the theorem of the Fourier Transform,

$$\operatorname{csch}(\pi x) \xrightarrow{FT} -j\tanh(\pi f)\,, \tag{2.28}$$

where

$$\begin{aligned} &\operatorname{csch}(x) = 2/(e^{x} - e^{-x}) \\ &\tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x}) \end{aligned}, \tag{2.29}$$

Therefore, from the scaling property of the FT:

$$g(bx) \xrightarrow{FT} |b|^{-1}\, G(f/b)\,, \tag{2.30}$$

we obtain

$$|b|\operatorname{csch}(\pi b x) \xrightarrow{FT} -j\tanh(\pi f/b)\,. \tag{2.31}$$

From (2.31), we can define the **short response Hilbert transform** (**SRHLT**) as:

(1) $$g_H(\tau) = h_b(x) * g(x), \quad \text{where } h_b(x) = |b|\operatorname{csch}(\pi b x)\,, \tag{2.32}$$

or

$$\begin{aligned}&G_H(f) = H_b(f)G(f) \text{ where } G_H(f) = FT[g_H(\tau)],\\&G(f) = FT[g(\tau)], \qquad H_b(f) = -j\tanh(\pi f/b).\end{aligned} \tag{2.33}$$

[Theorem 1] When $\boldsymbol{b \to 0^+}$ (0^+ is a positive number very near to 0), the SRHLT becomes the **HLT**. When $\boldsymbol{b} \to \infty$, the SRHLT tends to the **differentiation** operation.

(proof): Note that

$$\tanh(\pi f/b) = (e^{\pi f/b} - e^{-\pi f/b})/(e^{\pi f/b} + e^{-\pi f/b}). \tag{2.34}$$

When $b \to 0^+$, then

$$e^{-\pi f/b} \to 0 \quad \text{where } f > 0, \qquad e^{\pi f/b} \to 0 \quad \text{when } f < 0. \tag{2.35}$$

Therefore, when $b \to 0^+$

$$\tanh(\pi f/b) \approx e^{\pi f/b}/e^{\pi f/b} = 1 \qquad \text{when } f > 0, \tag{2.36}$$

$$\tanh(\pi f/b) \approx -e^{-\pi f/b}/e^{-\pi f/b} = -1 \qquad \text{when } f < 0. \tag{2.37}$$

That is,

$$-j\tanh(\pi f/b) \approx -j\,\mathrm{sgn}(f) \qquad \text{where } b \to 0^+. \tag{2.38}$$

Moreover, since

$$e^x \approx 1 + x \quad \text{where } x \approx 0, \tag{2.39}$$

therefore, from (2.34), when $b \to \infty$,

$$\tanh(\pi f/b) \approx \frac{1 + \pi f/b - 1 + \pi f/b}{1 + \pi f/b + 1 - \pi f/b} = \frac{\pi f}{b}, \tag{2.40}$$

$$-j\tanh(\pi f/b) \approx -j\frac{\pi f}{b} \qquad \text{where } b \to \infty. \tag{2.41}$$

Note that the transfer function of the Hilbert transform is $-j$ sgn(f). The FT of the differentiation operation is

$$-d/dx \xrightarrow{FT} -j2\pi f \tag{2.42}$$

Therefore, from (2.38) and (2.42) when $b \to 0^+$ and ∞, the SRHLT in (2.33) becomes the HLT and the differentiation multiplied by some constant, respectively.

From Theorem 1, we can conclude that when

$$0 < b < \infty, \tag{2.43}$$

the characters of the SRHLT are partially like the HLT and partially like the differentiation operation. When *b* is small, the SRHLT is more similar to the HLT. When *b* is large, the SRHLT is more similar to differentiation.

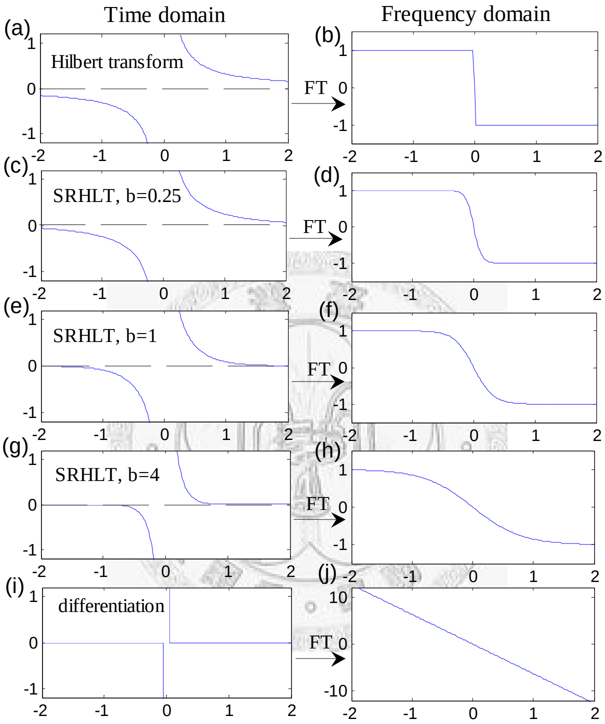


Fig. 2.11 Impulse responses and their FTs of the SRHLT for different *b*. We can compare them with the impulse response of the differential operation and the original HLT.

These facts can be seen from the experiments in Fig. 2.11, where we plot the impulse response of the SRHLT for different *b*. In the frequency domain, the transfer function of the SRHLT gradually changes from the step form into the linear form as *b* grows. Moreover, in the time domain, when *b* is small, the SRHLT has a long impulse response. When *b* is large, the SRHLT has a short impulse response.

The edge detection filter with longer impulse response has higher robustness to noise and has better ability to detection the ramp edges. The edge detection filter with shorter impulse response has sharper outputs for edges. Thus, the SRHLT with smaller *b* is suitable for detecting the noise-interfered edges and ramp edges. The SRHLT with higher *b* is suitable for detecting the step edge.

In Fig. 2.12, we do several experiments that use the SRHLT to detect the step, the step + noise, the ramp edges in Fig. 2.9(a)(c)(e).

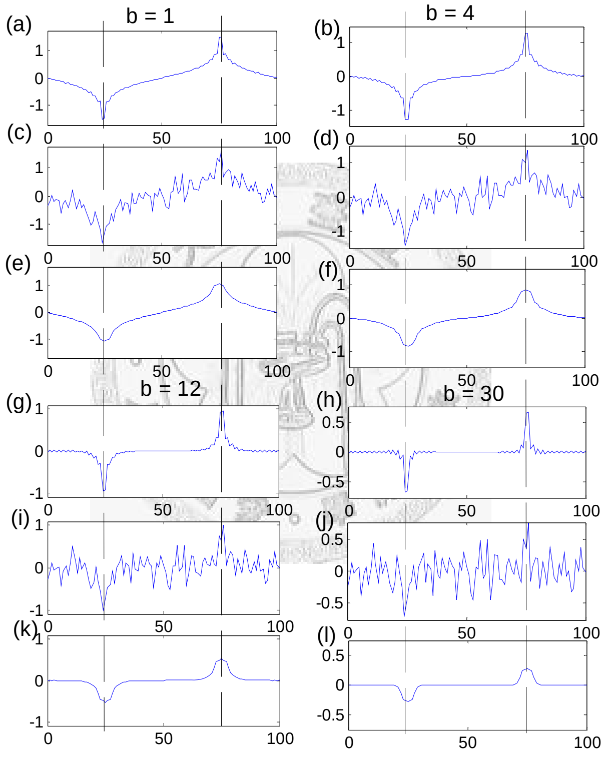


Fig. 2.12 Using SRHLTs to detect the sharp edges, the step edges with noise, and the ramp edges. Here we choose *b* = 1, 4, 12, and 30.

Note that when *b* is small, the SRHLT will have higher ability to detect the edges that are interfered by noise but the outputs of the edge and the non-edge region have less difference, as the results in Fig. 2.12(a)(c)(e) and Fig. 2.12(b)(d)(f). When *b* is large, the edge corresponds to a very narrow impulse in the output (as in Fig. 2.12(g)(h)) and can well distinguish an edge from the non-edge region. However, the results of edge detection are highly sensitive to noise, as in Fig. 2.12(i)(j). Moreover, for ramp edge detection, it is proper to choose *b* neither too large nor too small, as the results in Fig. 2.12(e)(f)(k)(l).

2.3.2 SRHLT of the 2-D Form

Then, for the purpose of detecting the edge of a 2-D digital image, we convert the SRHLT in Section 3 into the discrete form and the multiple dimensional forms.

Analogous to the discrete Hilbert transform in (2.20)-(2.22), the discrete form of the SRHLT can be defined as

$$g_H\left[n\right] = IDFT\left(H[p]DFT\left(g[n]\right)\right), \tag{2.44}$$

where

$$\begin{aligned} &H[p] = -j\tanh(\pi p/b) \ \text{ for } 0 < p < N/2, \\ &H[N-p] = j\tanh(\pi p/b) \ \text{ for } 0 < p < N/2, \\ &H[0] = H[N/2] = 0. \end{aligned} \tag{2.45}$$

Moreover, analogous to the works in [41][42](see (2.23)~(2.27)), we define the 2-D discrete SRHLT as:

$$g_H\left[m,n\right] = IDFT_{2D}\left(H[p,q]DFT_{2D}\left(g[m,n]\right)\right), \tag{2.46}$$

where the 2-D DFT is defined in (2.24) and

$$\begin{aligned} &H[p,q] = \Phi(\theta)R(r) \\ &\quad \theta = \cos^{-1}(p/r) = \sin^{-1}(q/r), \quad r = \sqrt{p^2+q^2} \\ &\quad \text{when } [p,q] \neq [0,0], \quad p \neq M/2, q \neq N/2 \\ &H[p,q] = H[M/2,n] = H[m,N/2] = 0 \end{aligned} \tag{2.47}$$

$\Phi(\theta)$ is any odd symmetric function that satisfies

$$\Phi(\theta+\pi) = -\Phi(-\theta). \tag{2.48}$$

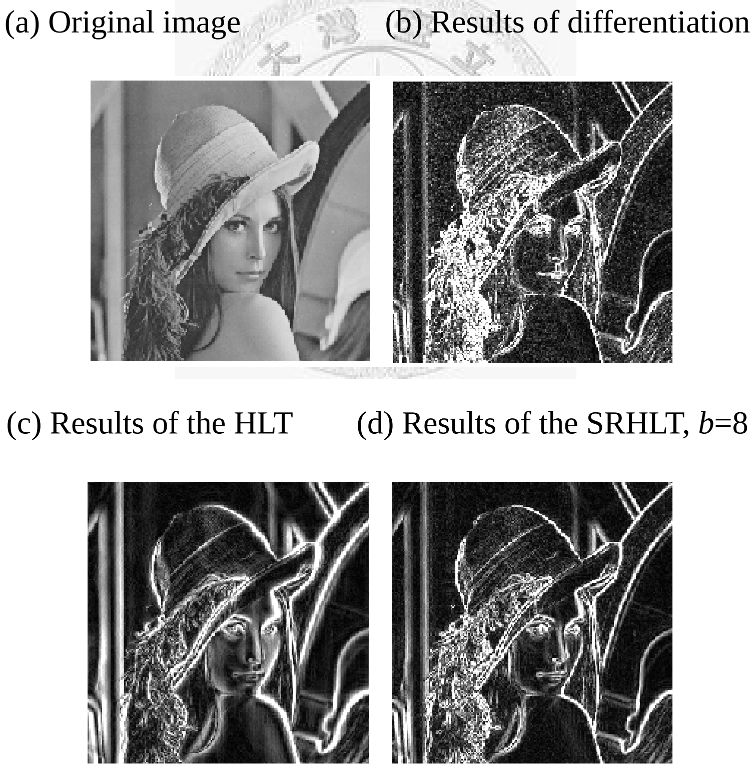
For example, we can also choose $\Phi(\theta)$ as

$$\Phi(\theta) = \exp(j\theta)\,. \tag{2.49}$$

Moreover

$$R\left(r\right) = \tanh(\pi r\,/\,b) = (e^{\pi\, r/b} - e^{-\pi\, r/b})\,/\,(e^{\pi\, r/b} + e^{-\pi\, r/b})\,. \tag{2.50}$$

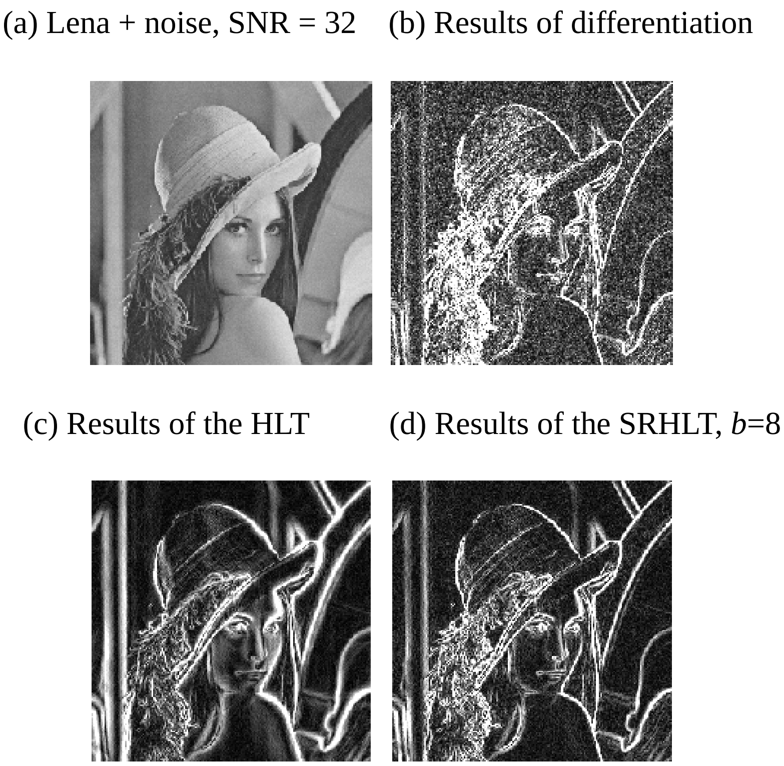
Note that the definition of the 2-D discrete SRHLT is similar to that of the 2-D discrete HLT in (2.23)~(2.27). The only difference is now the transfer function $H[p,\ q]$ is dependent on r. We can use the parameter b to control $R(r)$ and hence control the performance. When b is large, the ability of detecting detail edges is well. When b is small, the effect of noise can be reduced.



(a) Original image (b) Results of differentiation

(c) Results of the HLT (d) Results of the SRHLT, b=8

Fig. 2.13 Experiments that use differentiation, the HLT, and the SRHLT (b=8) to do edge detection for Lena image.



(a) Lena + noise, SNR = 32 (b) Results of differentiation

(c) Results of the HLT (d) Results of the SRHLT, *b*=8

Fig. 2.14 Experiments that use differentiation, the HLT, and the SRHLT (*b*=8) to detect the edges of Lena image interfered by noise.

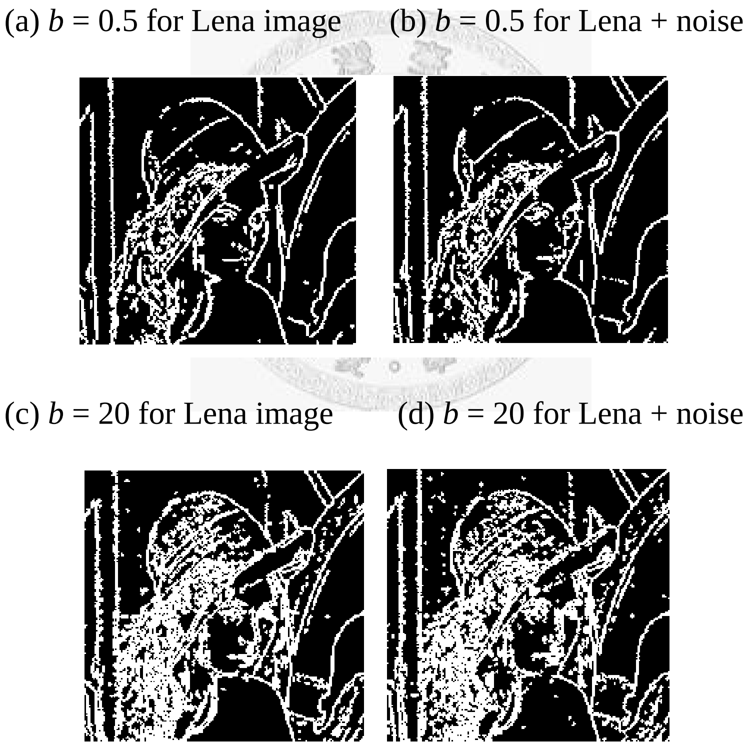
2.3.3 Experiments of Edge Detection Using SRHLT

In Fig. 2.13~Fig. 2.18, we do several experiments to show the results of using the SRHLT to detect the edges of natural and complicated images.

First, we do a comparison with differentiation and the HLT. When using differentiation for edge detection, as Fig. 2.13(b), although some edges are detected, many isolated dots are regarded as edges and some ramp edges cannot be well detected. Since differentiation observes only the amplitude difference among the adjective pixels, it cannot accurately determine whether a pixel is on an edge. Moreover, when the input is interfered by noise, as Fig. 2.14(a), the result of differentiation is worse (see Fig. 2.14(b)) even if the noise is small (in Fig. 2.14(a) SNR = 32).

When we use the HLT for edge detection, as Fig. 2.13(c), the problems that isolated dots is recognized as edges is avoided and the performance is better than differentiation, as Fig. 2.13(c). Even if the input is interfered by noise the HLT still has better performance, as Fig. 2.14(c). However, it can be seen from Fig. 2.13(c) that many pixels that are near to edges will have larger outputs after doing the HLT (such as the face region).

If we use the proposed SRHLT for edge detection, as Fig. 2.13(d), the edges are clearer than the case where we use the HLT and the non-edge pixel seldom has large outputs after doing the SRHLT. At the same time, he ramp edges can still be detected successfully and the noise robustness is still higher, as Fig. 2.14(d).

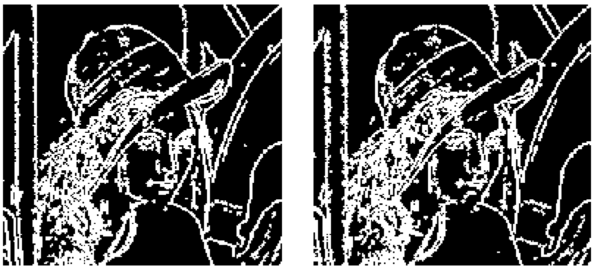


(a) *b* = 0.5 for Lena image (b) *b* = 0.5 for Lena + noise

(c) *b* = 20 for Lena image (d) *b* = 20 for Lena + noise

Fig. 2.15 Using the SRHLT to detect the edges of Lena image or Lena + noise when *b* is chosen as 0.5 and 20.

Thus, if we use the SRHLT for edge detection, the advantages of differentiation and the HLT can be compromised. Compare with differentiation and the HLT, the SRHLT can more accurately detect the edges of an image.



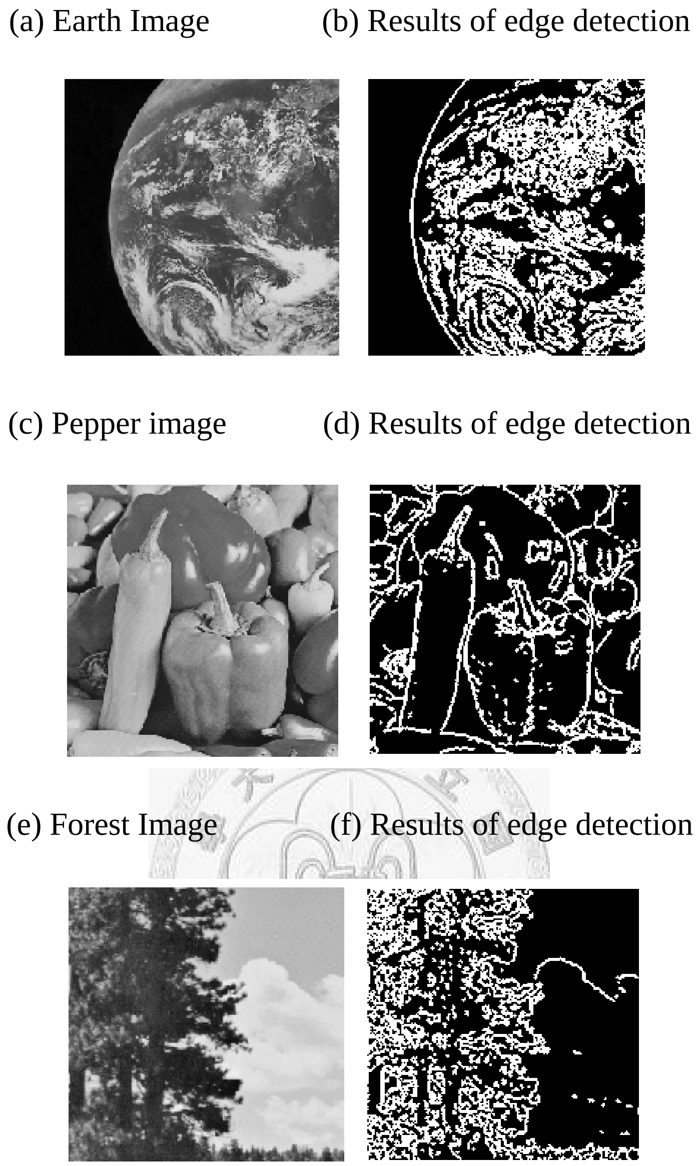
(a) *b* = 8 for Lena image (b) *b* = 8 for Lena + noise

Fig. 2.16 Using adaptive threshold and overlapped section to further refined the results of edge detection by SRHLTs. (a) is the refined results of Fig. 2.13(d). (b) is the refined results of Fig. 2.14(d).

To further improve the performance, we can apply the assistant techniques such as overlapped section and adaptive thresholds. In Fig. 2.16(a)(b), we use these techniques to make the edges detected by the SRHLT clearer. It can be seen that almost all the edges of the original image are detected successfully and the noise nearly has no effect when we use the SRHLT for edge detection.

We then show the results that we adjust the parameter *b* of the SRHLT for edge detection. In Fig. 2.13, Fig. 2.14, and Fig. 2.16, we choose *b* = 8. If we choose *b* = 0.5, as Fig. 2.15(a)(b), some detailed edges are not detected successfully, If we choose *b* = 20, as Fig. 2.15(c)(d), the ability of noise immunity is reduced. Therefore, to achieve the goals of detecting detailed edges and noise robustness at the same time, it is improper to choose *b* neither too small nor too large.

Then, we do several experiments in Fig. 2.17 and Fig. 2.18 to show the results of using the SRHLT (*b* = 8) for detecting the edges of complicated images. We find that the SRHLT can successfully detect most of the edges of an image, even if the image is as complicated as Earth, Pepper, Tiffany, Ken, Shuttle, and Forest images.

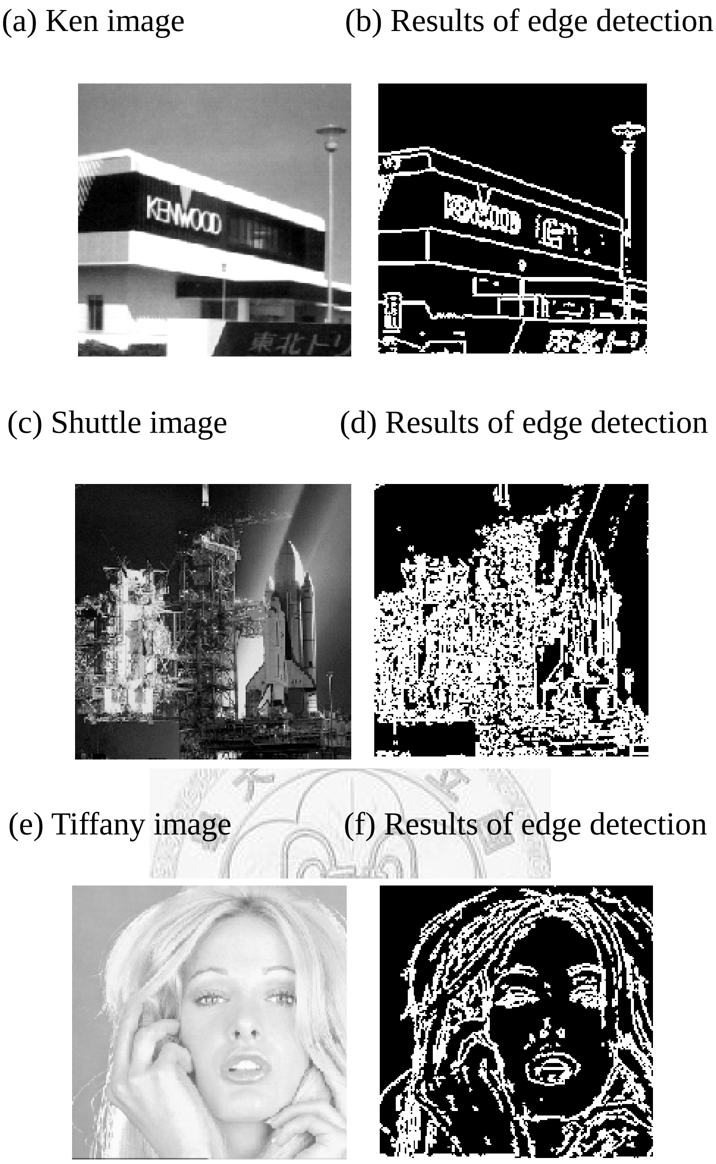


(a) Earth Image (b) Results of edge detection

(c) Pepper image (d) Results of edge detection

(e) Forest Image (f) Results of edge detection

Fig. 2.17 Using the SRHLT (b = 8) to detect the edges of Earth, Pepper, and Forest images. Most of the edges are detected successfully.



(a) Ken image (b) Results of edge detection

(c) Shuttle image (d) Results of edge detection

(e) Tiffany image (f) Results of edge detection

Fig. 2.18 Using the SRHLT (b = 8) to detect the edges of Ken, Shuttle, and Tiffany images. Most of the edges are detected successfully.

2.3.4 Illustrated by Canny's Theorem and Mathematical Analysis

From Canny's theorem [16], the following three factors are used for measuring the performance of edge detection:

1. Good detection,

2. Good localization,

3. Single response.

To satisfy the requirement of "Good detection", the edge detection filter should satisfy

(a) Higher distinction: lead to very larger outputs for edges and smaller outputs for non-edge regions. That is, if $g(x)$ is the input and $h(x)$ is the impulse response of the edge detection filter,

$$g_H(\tau) = \int_{-\infty}^{\infty} h(\tau - x) g(x) dx, \tag{2.51}$$

then

$$g_H(x_0) \gg g_H(x_1) \tag{2.52}$$

if x_0 is at an edge and x_1 is not at an edge.

(b) Noise immunity: i.e., less sensitive to noise. If $g(x)$ is interfered by $n(t)$ where $n(t)$ is noise and

$$g_{H,n}(\tau) = \int_{-\infty}^{\infty} h(\tau - x)[g(x) + n(x)] dx, \tag{2.53}$$

then

$$g_{H,n}(x_0) \gg g_{H,n}(x_1) \tag{2.54}$$

is still satisfied.

However, **(a)** and **(b)** are a tradeoff. When we use the method of differentiation, the edge and the non-edge regions can be distinguished very well. Nevertheless, the method is higher sensitive to noise. When we use the original HLT, the effect of noise can be much reduced but the method cannot distinguish the edge from the non-edge region very well.

In fact, to compromise the goals of "higher distinction" and "noise immunity" and achieve the requirement of "good detection" proposed by Canny, the impulse response of the edge detection filter should satisfy

(Constraint 1) The impulse response $h(x)$ is neither too short nor too long. If we define

$$T = \int_{-\infty}^{\infty} |x|\,|h(x)|^2\,dx, \tag{2.55}$$

then T should satisfy

$$A_1 < T < A_2, \tag{2.56}$$

where A_1 and A_2 are some thresholds. To achieve higher immunity to noise, T should be larger than a certain threshold A_1. To make the filter have higher ability for distinguishing the edge from the non-edge region, T should be smaller than a certain threshold A_2.

Now, we propose the SRHLT. We can adjust the parameter b in (2.33) and (2.50) to control the length of impulse response. If b is chosen properly, (2.55) would be satisfied and the resultant SRHLT can compromise the goals of "higher distinction" and "noise immunity" at the same time. Therefore, the SRHLT can achieve the goal of "good detection" proposed by Canny.

In addition to "good detection", there are another two requirements for edge detection proposed by Canny.

To achieve the goal of "Good localization", the impulse response of the edge detection filter should have a peak around $x = 0$:

(Constraint 2)

$$Max\{|h(x)|\} = h(x_0) \tag{2.57}$$

where $x_0 = 0$ or x_0 is very close to 0.

To achieve the goal of "Single response", the absolute value of the impulse response of the edge detection filter should be a strictly descending function (or near to a strictly descending function):

(Constraint 3)

$$|h(x_1)| > |h(x_2)| \qquad \text{if } |x_2| > |x_1| \geq |x_0|, \tag{2.58}$$

or although in some conditions the impulse response is not strictly descending but the local peak is much smaller than the global peak $|h(x_0)|$.

$$\begin{aligned}&\left|h\left(x_1\right)\right| < \left|h\left(x_2\right)\right|, \quad \left|x_2\right| > \left|x_1\right| \\ &\text{but } \left|h(x_2)\right| \ll \left|h(x_0)\right|.\end{aligned} \tag{2.59}$$

Moreover, an edge detection filter should be an odd operation:

(Constraint 4) $$h(x) = -h(-x). \tag{2.60}$$

If the impulse response of an edge detection filter satisfies all the four constraints in (2.55), (2.57), (2.58), and (2.60), then the edge detection filter will satisfy the three requirements proposed by Canny [16] and have good performance for edge detection.

Now, we can observe the impulse response of the SRHLT. If b is chosen properly, $h_b(x)$ in (2.32) satisfies (2.55). Moreover, no matter what the value of b is,

$$\text{Max}\{|\,h_b(x)\,|\} = \left|h\left(0^+\right)\right| = \left|h\left(0^-\right)\right| \to \infty. \tag{2.61}$$

Furthermore, it is obviously that $h_b(x)$ in (2.32) is an odd function and strictly decreases with $|x|$. Therefore, if b is chosen properly, the SRHLT satisfy all the constraints in (2.55), (2.57), (2.58), and (2.60). From Canny's theorem, the SRHLT can be a very effective tool for edge detection.

2.3.5 Other Possible Ways to Define the SRHLT

In Section 6, we illustrated that the proposed SRHLT in (2.32) satisfies all the four constraints in (2.55), (2.57), (2.58), and (2.60) and hence satisfies Canny's criterions. It makes the SRHLT defined in (2.32) good for edge detection.

In fact, there are many alternative ways to define the SRHLT. In addition to (2.32), there are also other functions that satisfy Canny's criterions and can be treated as the impulse responses of SRHLTs. For example, from the theorem of Fourier Transform,

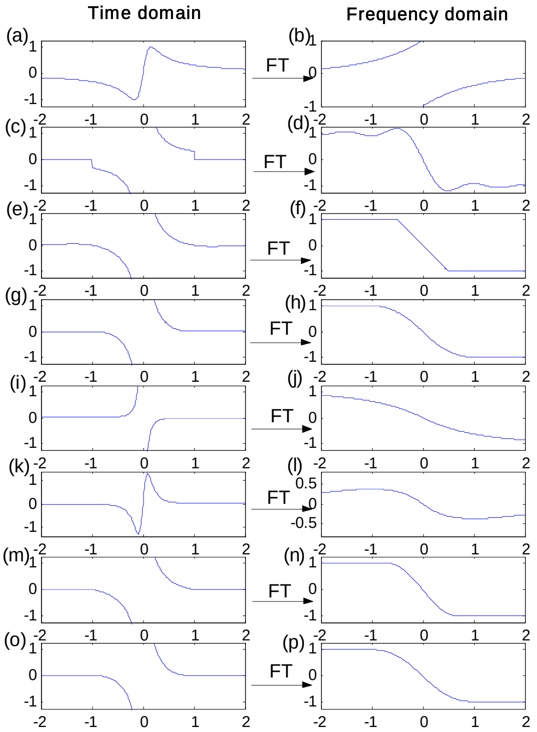


Fig. 2.19 Other transform pairs (defined in (2.62)~(2.74)) that can be treated as the impulse response and the transfer function of the SRHLT. All of them satisfy Canny's criterion and are useful for edge detection. Here we all set *b* =1.

(2)

$$\frac{4\pi b^{-2}x}{1+(2\pi b^{-1}x)^2} \xrightarrow{FT} -je^{-|b\,f|}\operatorname{sgn}(f) \tag{2.62}$$

(The transform pair is plotted in Fig. 2.19(a)(b)),

where sgn(*f*) is defined in (2.19), it is no hard to prove that the impulse response $4\pi b^{-2}x/[1+(2\pi\, b^{-1}x)2]$ satisfies all the constraints in (2.55), (2.57), (2.58), and (2.60). Therefore, we can also define the SRHLT as the form of (2.62) and use it for edge detection. Moreover,

$$\frac{4\pi b^{-2}x}{1+(2\pi b^{-1}x)^2} \approx \frac{1}{\pi x} \qquad \text{when } b \approx 0, \tag{2.63}$$

That is, when $b \to 0$, (2.62) becomes the original HLT pair.

Moreover, in addition to (2.32) and (2.62), from pages 575-591 in [16] and the scaling property of the FT,

$$g(bx)\xrightarrow{FT}|b|^{-1}G(f/b), \tag{2.64}$$

we find that the following transform pairs satisfy all the four constraints in (2.55), (2.57), (2.58), and (2.60) (i.e., satisfy Canny’s criterions) and can be treated as the SRHLTs are (Here we constrain *b* to be a positive real number):

(3)

$$\frac{1}{\pi x}\Pi\left(bx/2\right)\xrightarrow{FT}-i\frac{2}{\pi}Si\left(2\pi b^{-1}f\right), \tag{2.65}$$

where

$$Si(x)=\int_0^x\frac{\sin t}{t}dt\ . \tag{2.66}$$

(The transform pair is plotted in Fig. 2.19(c)(d))

(4)

$$\frac{\sin c(bx)}{b\pi x}\xrightarrow{FT}H_5(f), \tag{2.67}$$

where

$$\mathrm{sinc}(x) = \sin(\pi x)/(\pi x), \tag{2.68}$$

$$\begin{aligned}&H_5(f)=j\ \ \text{for}\ f<-b/2,\ \ H_5(f)=-j\ \ \text{for}\ f>b/2,\\&H_5(f)=-j2f/b\ \ \text{for}\ -b/2\le f\le b/2\end{aligned} \tag{2.69}$$

(The transform pair is plotted in Fig. 2.19(e)(f))

(5)

$$\frac{\sin c^2(bx)}{b\pi x}\xrightarrow{FT}H_6(f), \tag{2.70}$$

where

$$\begin{aligned}&H_6(f)=j\ \ \text{for}\ f<-b,\ \ H_6(f)=-j\ \ \text{for}\ f>b,\\&H_6(f)=j-j(1+b^{-1}f)^2\ \ \text{for}\ -b\le f<0,\\&H_6(f)=-j+j(1-b^{-1}f)^2\ \ \text{for}\ 0\le f<b,\end{aligned} \tag{2.71}$$

(The transform pair is plotted in Fig. 2.19(g)(h))

(6)

$$\frac{2\pi b^2x}{1+(2\pi bx)^2}-\frac{1}{2\pi x}\xrightarrow{FT}-i(1-e^{-|f/b|})\mathrm{sgn}(f)\ . \tag{2.72}$$

(The transform pair is plotted in Fig. 2.19(i)(j))

(7)

$$\frac{8\pi b^3 x}{(1+4\pi^2 b^2 x^2)^2} \xrightarrow{FT} -i\, f\, e^{-|f/b|}. \tag{2.73}$$

(The transform pair is plotted in Fig. 2.19(k)(l))

(8)

$$\frac{\sin c(bx)}{\pi x}\Pi\left(\frac{x}{2b}\right) \xrightarrow{FT} 2\,|\,b\,|\sin c(2bf) * H_5\left(f\right), \tag{2.74}$$

where $\Pi(f) = 1$ for $-1/2 < f < 1/2$, $\Pi(\pm 1/2) = 1/2$, (2.75)

$\Pi(f) = 0$ otherwise,

and * means convolution. (The transform pair is plotted in Fig. 2.19(m)(n))

(9)

$$|\,b\,|\,x e^{-\pi b^2 x^2} \xrightarrow{FT} -i\frac{f}{b^2}e^{-\pi f^2/b^2} \tag{2.76}$$

(10)

$$\frac{1}{\pi x}\left(1-\frac{\sin(2\pi bx)}{2\pi bx}\right) \xrightarrow{FT} -j\Lambda\left(x/b\right)\mathrm{sgn}(x), \tag{2.77}$$

where Λ(x)= 1– |x| when |x| < 1, Λ(x)= 0 otherwise.

The edge detection filters defined in (2.73) and (2.74) have the advantages of finite impulse response, which is an important benefit for implementation.

It is no hard to prove that in all the cases the constraints in (2.55), (2.57), (2.58), and (2.60) are satisfied. That is, all of the above forms of SRHLTs satisfy Canny’s criterions and can be used for edge detection.

Note that, in all the cases we can use the parameter b to control the performance. If b is near to zero, the resultant edge detection filter will have higher ability for noise immunity but the have lower ability to distinguish the edge and the non-edge regions. When b is large the edge detection filter can well distinguish the edge and the non-edge regions but is highly affected by noise.

We can extend the above 1-D edge detection filter into the 2-D form. We can also define the 2-D edge detection filter as the form in (2.46)~(2.50). The only difference is that $R(r)$ can be changed into other forms.

2.4 Conclusion

In this chapter, we introduce the SRHLT which can be viewed as the medium between the differentiation operation and the Hilbert transform (HLT) for edge detection. As the HLT, the SRHLT has higher robustness for noise and can successfully detect ramp edges. At the same time, the SRHLT can avoid the pixels that near to an edge be recognized as an edge pixel, which is usually an important problem when using the HLT for edge detection. We also do several experiments and show that the SRHLT can successfully detect the edges of a complicated image. Moreover, directional edge detection (i.e., detect the edges with certain direction) are also the possible applications of the SRHLT.

Chapter 3

Image Segmentation

The segmentation technique divides an image into several regions for some applications. The algorithms generally are based on two properties which are the discontinuity and the similarity of color values. We can partition an image based on sudden changes in color values, such as edges in an image. On the other hand, we partition an image into regions that are similar according to a set of predefined criteria.

3.1 Thresholding

Because of the intuitive properties and simplicity of implementation, image thresholding is important in image segmentation. The gray-level histogram of an image *f*(*x*, *y*) is shown in Fig. 3.1(b), where the dark part is background and the light part is object. One obvious way to extract the objects from the background is to select a threshold *T* that separates these two parts. Then any points (*x*, *y*) for which (*x*, *y*) > *T* is called an object point and the other points is called background point.

A thresholded image *g*(*x*, *y*) is defined as

$$g(x,y)=\begin{cases}1 & \text{if } f(x,y)>T\\ 0 & \text{if } f(x,y)\le T\end{cases} \tag{3.1}$$

Thus, pixels labeled 1 correspond to objects, and pixels labeled 0 correspond to the background.

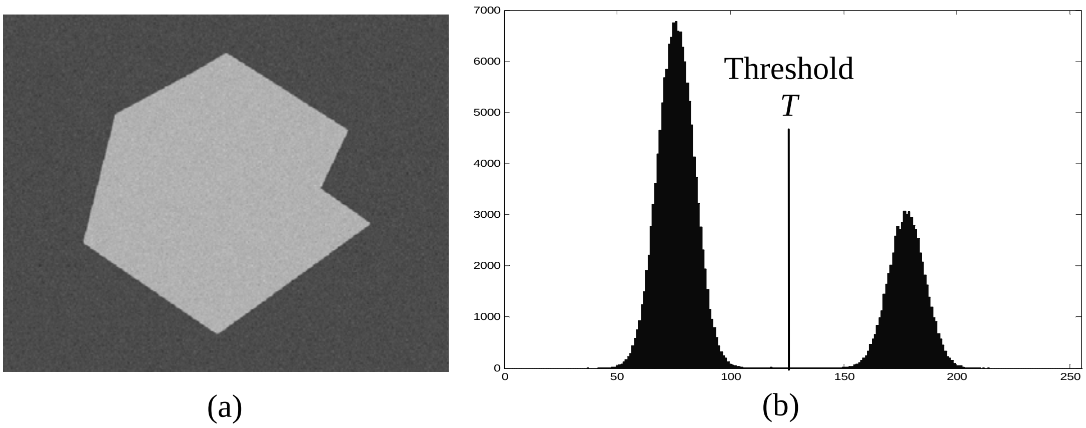


Fig. 3.1 (a) An example image $f(x, y)$ and its (b) gray-scale histogram.

The simplest of all thresholding techniques is to partition the image histogram by using a single global threshold T. Segmentation is then accomplished by scanning the image pixel by pixel and labeling each pixel as object or background depending on whether the gray level of that pixel is greater or less than the value of T.

The following algorithm can be used to obtain T automatically:

1. Select an initial estimate for T.
2. Segment the image using T. This will produce two groups of pixels: G_1 consisting of all pixels with gray level values $>T$ and G_2 consisting of pixels with values $\leq T$.
3. Compute the average gray level values μ_1 and μ_2, for the pixels in regions G_1 and G_2.
4. Compute a new threshold value: $T = (\mu_1 + \mu_2)/2$.
5. Repeat steps 2 through 4 until the difference in T in successive iterations is smaller than a predefined parameter T_0.

An appropriate initial value for T is the average value of the maximum and minimum gray levels. The parameter T_0, is used to stop the algorithm after changes become small in terms of this parameter.

3.2 Edge Linking

The term edge segment generally is used if the edge is short in relation to the dimensions of the image. A key problem in segmentation is to assemble edge segments into longer edges.

3.2.1 Local Processing

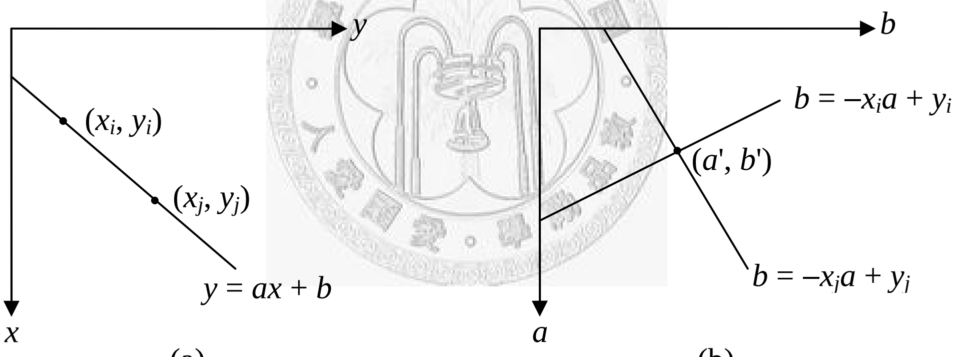
One of the simplest approaches for linking edge points is to analyze the characteristics of pixels in a small neighborhood about every point (x, y) in an image that has been labeled an edge point. Al1 points that are similar are linked, forming an edge of pixels that share those criteria.

The two principal properties used for establishing similarity of edge pixels in this kind of analysis are (1) the strength of the response of the gradient operator used to produce the edge pixel, and (2) the direction of the gradient vector. The two property is given by the value of $\nabla \mathbf{f}$, as defined in (2.2). If the difference of the gradient of an edge pixel and its neighbor is less than a threshold, they are similar in gradient. The direction of the edge at (x, y) is perpendicular to the direction of the gradient vector at that point.

A point in the predefined neighborhood of (x, y) is linked to the pixel at (x, y) if both magnitude and direction criteria are satisfied. This process is repeated at every location in the image. A record must be kept of linked points as the center of the neighborhood is moved from pixel to pixel. A simple bookkeeping procedure is to assign a different gray level to each set of linked edge pixels.

3.2.2 Hough Transform

If there are n edge points in an image, we want to find subsets of these points that lie on straight lines. Hough proposed an alternative approach, commonly referred to as the Hough transform. Consider a point (x_i, y_i) and the general equation of a straight line in slope-intercept form, $y_i = a\ x_i + b$. There are infinite lines pass through (x_i, y_i), but they all satisfy the equation $y_i = a\ x_i + b$ for varying values of a and b. However, writing this equation as $b = -x_i\ a + y_i$ and considering the *ab*-plane (also called parameter space) yields the equation of a single line for a fixed pair (x_i, y_i). Furthermore, a second point (x_j, y_j) also has a line in parameter space associated with it, and this line intersects the line associated with (x_i, y_i) at (a', b'), where a' is the slope and b' the intercept of the line containing both (x_i, y_i) and (x_j, y_j) in the *xy*-plane. In fact, all points contained on this line have lines in parameter space that intersect at (a', b'). Fig. 3.2 illustrates these concepts.



(a) (b)

Fig. 3.2 (a) *xy*-plain. (b) *ab*-plain (parameter space).

A problem with using the equation $y = a\ x + b$ to represent a line is that the slope approaches infinity as the line approaches the vertical. One way around this difficulty is to use the normal representation of a line:

$$x\cos\theta + y\sin\theta = \rho \tag{3.2}$$

Fig. 3.3(a) illustrates the geometrical interpretation of the parameters used in this equation. Instead of a straight line, however, the location is a sinusoidal curve in the

$\rho\theta$-plane.

The computational attractiveness of the Hough transform arises from subdividing the parameter space into so-called accumulator cells, as illustrated in Fig. 3.3(b), where ($\rho_{\max}$, $\rho_{\min}$) and ($\theta_{\max}$, $\theta_{\min}$) are the expected range of ρ and θ. Initially, we set all cells with an accumulative value to 0. If a sinusoidal curve passes across the cell, its accumulative value will be added to 1.

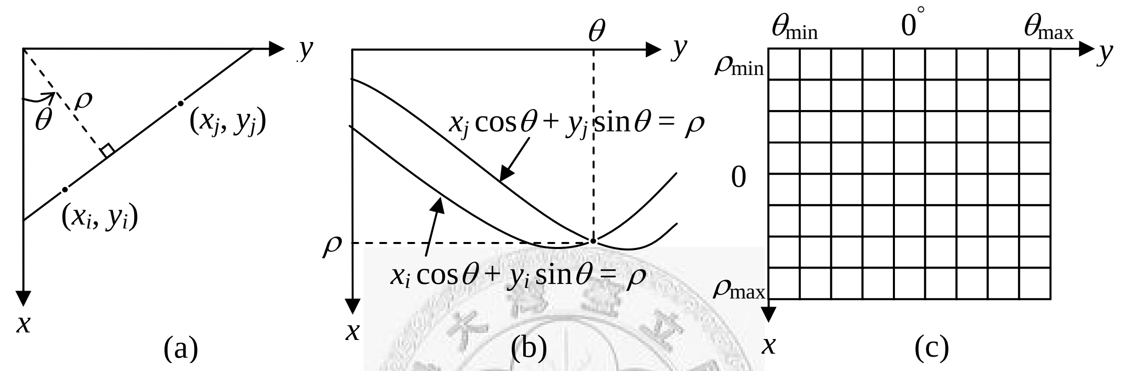


Fig. 3.3 (a) Normal representation of a line. (b) $\rho\theta$-plain. (b) $\rho\theta$-plain with cell.

Finally, we find the local maximum cumulative value in the $\rho\theta$-plain. The position (ρ, θ) with a local maximum cumulative value means a line of (3.2) in the xy-plain that pass through relatively more edge pixels in an image.

3.3 Region-Based Segmentation

Let R represent the entire image region. We may view segmentation as a process that partitions R into n sub-regions, R$_1$, R$_2$, ..., R$_n$, such that

(a) $\bigcup_{i=1}^{n} R_i = R$.

(b) R_i is a connected region, i = 1, 2, ... , n.

(c) $R_i \cap R_j = \varnothing$ for all i and j, $i \neq j$.

(d) $P(R_i)$ = TRUE for i = 1, 2, ... , n.

(e) $P(R_i \cup R_j)$ = FASLE for any adjacent region R_i and R_j.

Where $P(R_k)$ is a logical predicate defined over the points in set R_k and $\varnothing$ is the null set.

3.3.1 Region Growing

Region growing is a procedure that group pixels or sub-regions into larger regions based on predefined criteria. A simple method is to start with a set of seed points and groups its neighboring pixels that have properties similar to the seed. It is an iterative process where each seed pixel grows iteratively until every pixel is processed. Finally it generates different regions and it can be a result of segmentation.

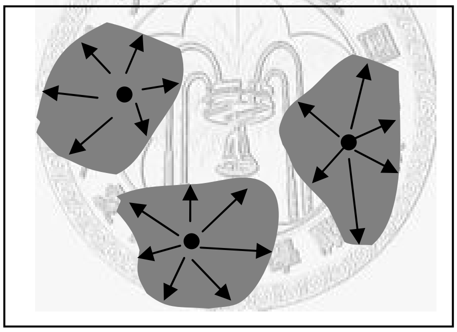


Fig. 3.4 Region growing.

The main problem is how to find the seed points correctly. If the image has pure background color like black, the seeds can be chosen from the points with the intensity value larger than a threshold. That is, we have to analysis the characteristic of the image first and it is hard to apply to arbitrary images.

3.3.2 Segmentation by Morphological Watersheds

The main idea of watersheds is based on visualizing an image *f*(*x*, *y*) in three dimensions. Therefore, we get a topographic interpretation of the image *f*(*x*, *y*). The object is to find the watershed lines. If we see the regional minimum locations as valleys, we could let each valley be flooded with water. When the flooding water reaches some ridges, a dam is built to prevent the two valleys be merged. After the entire topography is flooded under the water, the built dams are the watersheds we want. Fig. 3.5 illustrates the flooding process in a cross-sectional view.

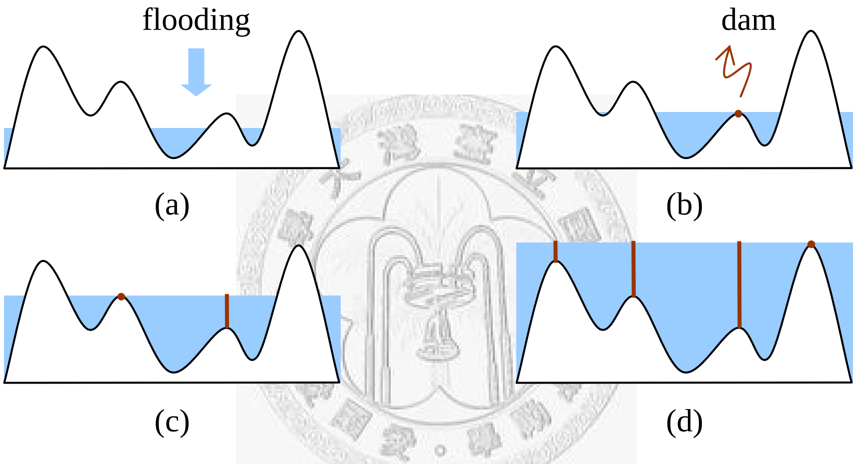
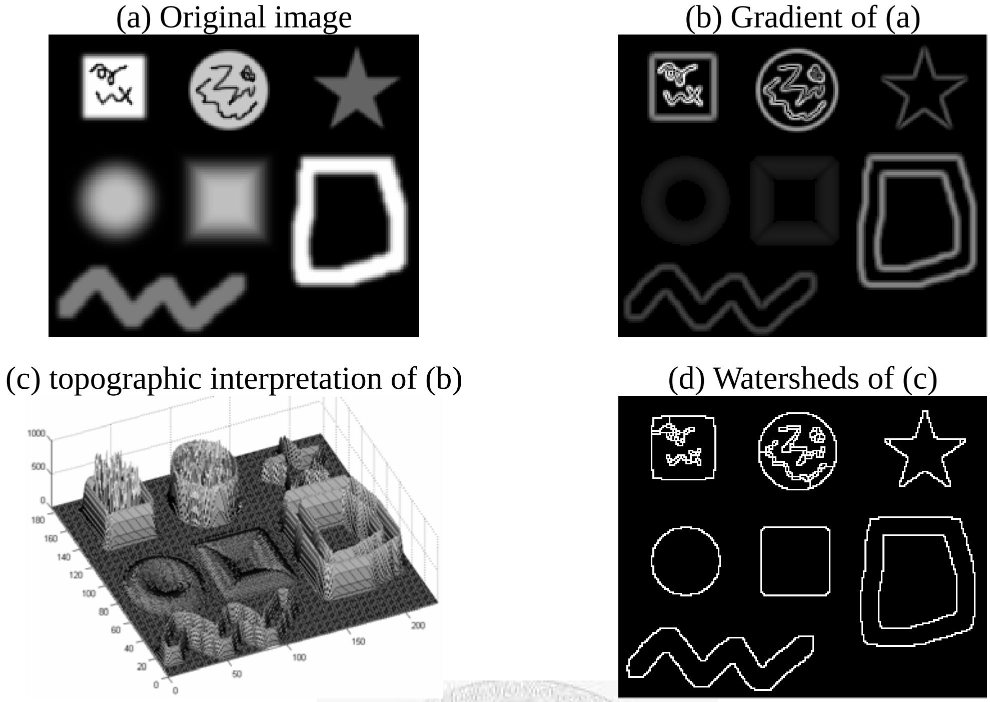


Fig. 3.5 The flooding process of the morphological watershed algorithm.

In the watershed based segmentation, we often apply the watershed segmentation to the gradient of an image rather than to the image itself. It is because gradient is an important feature when extracting edges or discontinuities.

Because the algorithm provides promising efficiency and accuracy, it is one of the most popular methods in image segmentation. However, it is sensitive to noise and cause the over segmenting problem.

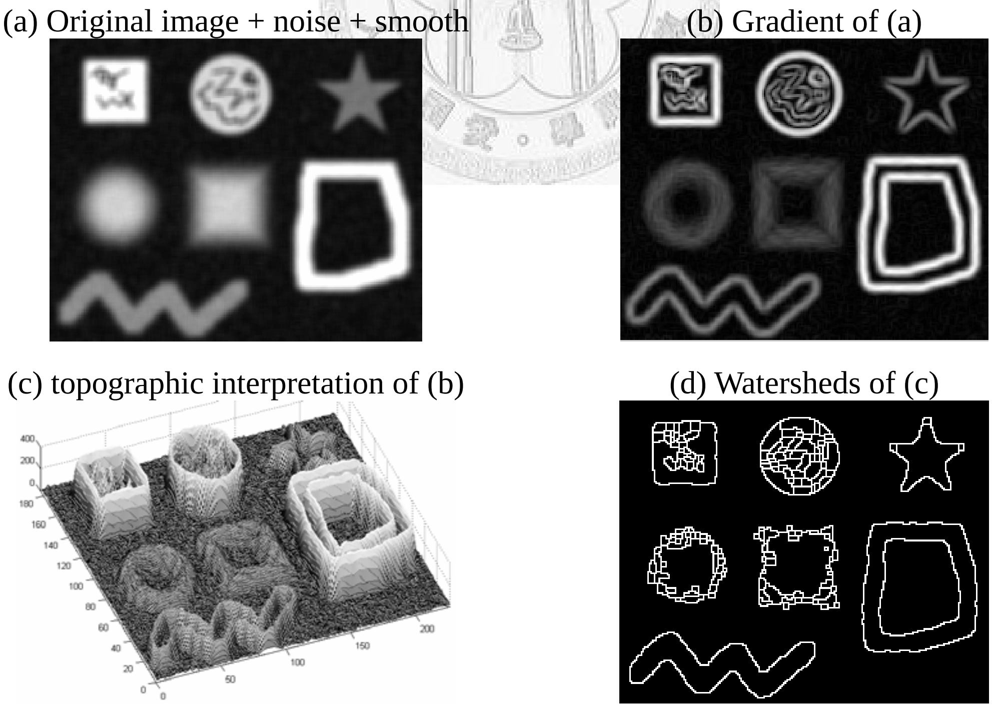

(a) Original image

(b) Gradient of (a)

(c) topographic interpretation of (b)

(d) Watersheds of (c)

Fig. 3.6 An example of watershed algorithm.


(a) Original image + noise + smooth

(b) Gradient of (a)

(c) topographic interpretation of (b)

(d) Watersheds of (c)

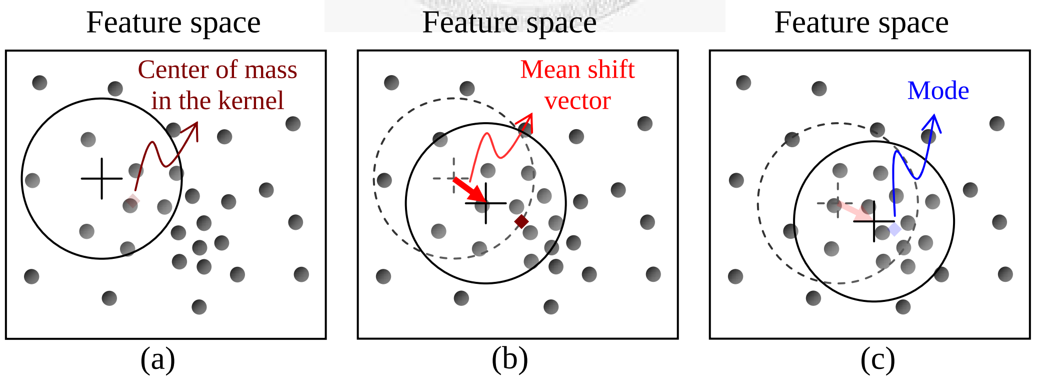
Fig. 3.7 An example of watershed algorithm with noise.

3.3.3 Mean Shift Based Image Segmentation

We would describe the mean shift based image segmentation because its robust performance and convenience of specifying the parameters. It is widely used in computer vision. Comaniciu and Meer [52] proposed an image segmentation method using the mean-shift Algorithm, which is proposed in 1975 by Fukunaga, and Hostetler [53].

The mean shift algorithm technique is introduced in [52] and is a technique based on the feature space analysis. It includes two steps: a mean shift filtering of the original image data, and clustering of the filtered data points.

The filtering step of the algorithm consists of analyzing the probability density function (PDF) underlying the image data in the feature space. One example of the feature space is a five dimension space consists of the image location (x, y) of each pixel and the pixel color (L^*,u^*,v^*) in the L*u*v* color space [76]. The modes of the PDF underlying the image data in this space are the locations with highest data density, and data points close to these modes can be clustered together to form a segmentation.



(a) (b) (c)

Fig. 3.8 The mean shift iterations to find the mode.

As shown in Fig. 3.8, for every data point in a uniform kernel, the gradient estimate of the PDF is computed and the center point of the kernel is moved in that direction. This movement in position is called the mean shift vector. The kernel is moved iterating until the gradient is less than a threshold. The resulting points are the modes which have gradient approximately equal to zero. Each data point is then replaced by its corresponding mode. The region for which all trajectories lead to the same mode is called the attraction basin as illustrated in Fig. 3.9. The process helps us to smooth the image while preserving discontinuities.

The clustering step is to reduce the number of modes. A simple way is to group the modes that are less than one kernel radius apart and merge their attraction basins. This step converts the filtered points into a segmented image.

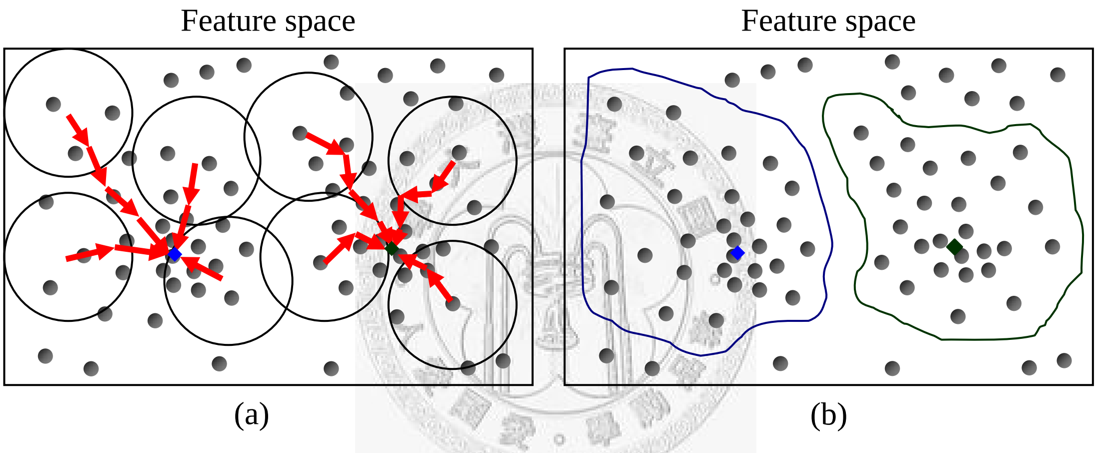


Fig. 3.9 Attraction basins of two modes. The diamonds indicate the modes.

Fig. 3.10 is an example in [52]. Fig. 3.10(a) is an image whose feature space is a three dimension space consists of the position (x, y) and the gray-level. Fig. 3.10(b)(c) show the filtering result. Note that the number of modes is too much on the plateau and on the line. Fig. 3.10(d) show the clustering result to group these redundant modes and it can be the segmentation of the image.

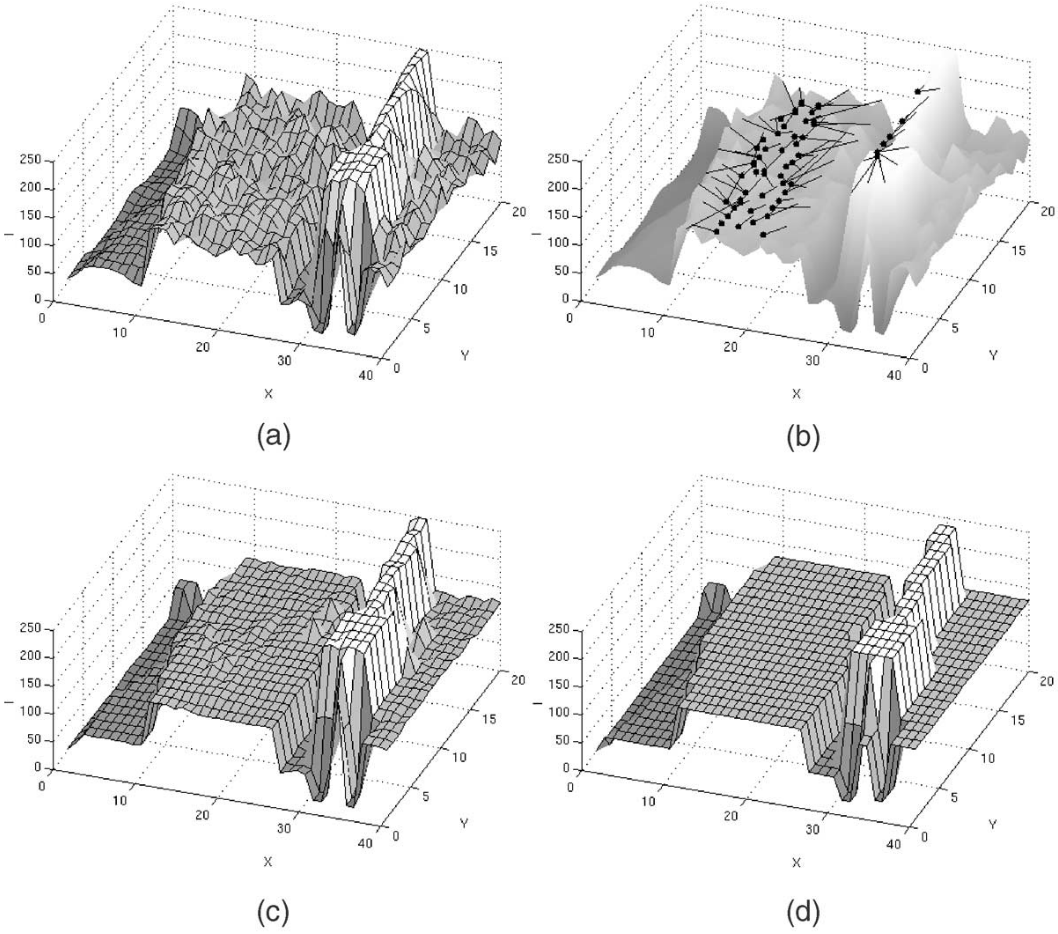


Fig. 3.10 Mean shift based filtering and segmentation for a gray-level image from [52]. (a) Original image. (b) Mean shift paths for the pixels on the plateau and on the line. The black dots indicate the modes. (c) Filtering result. (d) Clustering result.

3.4 Conclusion

We have introduced many methods to segment an image. They are design for images with some characteristic individually. It is hard to find a common way to segment various images and all of the results have good performance. However, the reason for us to segment an image in this thesis is only to apply to compression. The incorrect segments do not cause serious problem. Therefore, we concentrated on compression and paid scant attention to segmentation. In the following chapters, we only use simple image which has pure background color and some geometric image segments to ignore the effect of segmentation.

Chapter 4

Basic Image Compression Algorithm

Image compression is a process of reducing the quantity of data required to represent an image. In digital image compression, the process eliminates the correlation component and truncates the part that is unobvious to human vision. Then encode the result to reduce the code redundancy. The compressed code would be store in storage or use to transfer on channel. When we need to load this image, the code is decompressed to reconstruct the original image or an approximation of it.

Data redundancy is a central issue in digital image compression. It is not an abstract concept can be a mathematically quantification. If N_1 and N_2 denote the number of bytes in two data sets that represent the uncompressed image and the compressed code, the compression ratio can be defined as

$$C_R = \frac{N_1}{N_2}\,. \tag{4.1}$$

The distortion of the original image f_1 and the reconstructive image f_2 can be calculated as

$$RMSE = \sqrt{\sum_{y=1}^{H}\sum_{x=1}^{W}\left[\, f_1(x,y) - f_2(x,y)\right]^2 / (H \times W)} \tag{4.2}$$

where H and W are the height and the width of the images respectively.

JPEG (Joint Photographic Experts Group) is an international compression standard for continuous-tone still image, both grayscale and color. This standard is designed to support a wide variety of applications for continuous-tone images. The JPEG standard specifies the DCT-based method for lossy image compression and has been widely used today. In this chapter, we will introduce the basic compression algorithms in JPEG standard. Fig. 4.1 shows the diagram of the image compression model and we will discuss each step in the following sections.

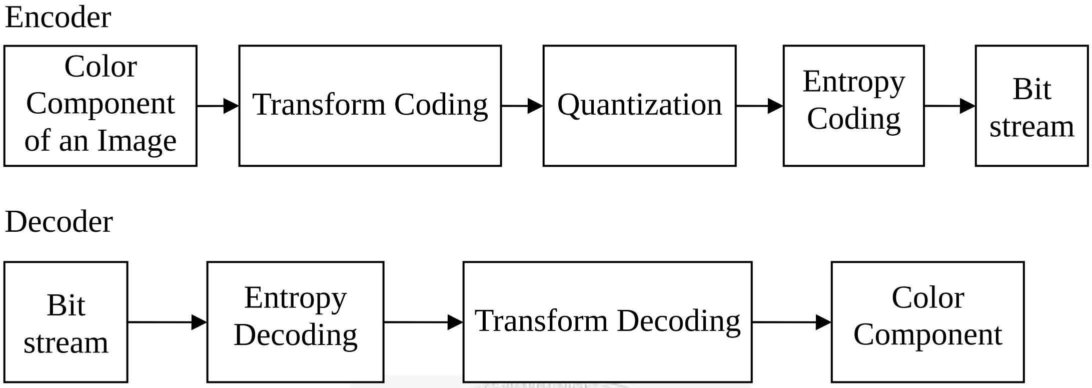


Fig. 4.1 Image compression model.

4.1 Color Space Conversion and Downsampling

In order to achieve good compression performance, correlation between the color components is first reduced by converting the *RGB* color space into a uncorrelated color space. In JPEG standard, a *RGB* image is first transformed into a luminance-chrominance color space such as YC_bC_r. The advantage of converting the image into luminance-chrominance color space is that they are much uncorrelated between each other. Moreover, the chrominance channels contain much redundant information and can easily be sampled without sacrificing any visual quality for the reconstructed image. The transformation from *RGB* to YC_bC_r is

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299000 & 0.587000 & 0.114000 \\ -0.168736 & -0.331264 & 0.500002 \\ 0.500000 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}. \tag{4.3}$$

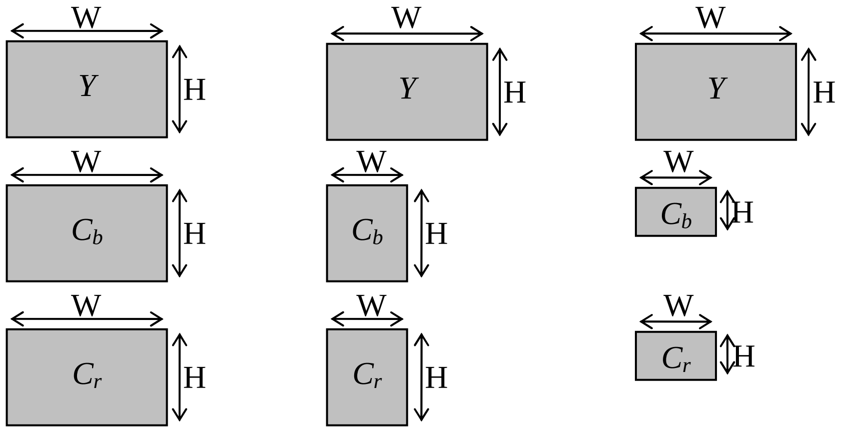
The value $Y = 0.299R + 0.587G + 0.114B$ is called the luminance. It is the value used by monochrome monitors to represent an *RGB* color. Physiologically, it represents the intensity of an *RGB* color perceived by the eye. The formula is like a weighted-filter with different weights for each spectral component. The eye is most sensitive to the Green component then it follows the Red component and the last is the Blue component. The values C_b and C_r are called chrominance values and indicate how much blue and how much red are in that color, respectively. Accordingly, the inverse transformation from YC_bC_r to *RGB* is:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 1.40210 \\ 1.0 & -0.34414 & -0.71414 \\ 1.0 & 1.77180 & 0.0 \end{bmatrix} \begin{bmatrix} Y \\ C_b - 128 \\ C_r - 128 \end{bmatrix} \tag{4.4}$$

Because the eye seems to be more sensitive at the luminance than the chrominance, luminance is taken in every pixel while the chrominance is taken as a medium value for a 2×2 block of pixels. And this way will result a good compression ratio with almost no loss in visual perception of the new sampled image. There are three color formats in JPEG standard as illustrated in Fig. 4.2:

1. **4:4:4 format:** The chrominance components have identical vertical and horizontal resolution as the luminance component.
2. **4:2:2 format:** The chrominance components have the same vertical resolution as the luminance component, but the horizontal resolution is half one.
3. **4:2:0 format:** Both vertical and horizontal resolution of the chrominance component is half of the luminance component.

Note that, in 4:2:2 format, the data quantity can be one half of the original image and the distortion is unobvious in human vision.



(a) 4:4:4 (b) 4:2:2 (c) 4:2:0

Fig. 4.2 Three color formats.

4.2 Transform Coding

To apply the transform coding in the JPEG standard, the image is divided into 8×8 blocks of pixels and then calculate the discrete cosine transform (DCT) indivdually. If the width or height of the original image is not divisible by 8, the encoder should make it divisible. The 8×8 blocks are processed from left-to-right and from top-to-bottom.

The purpose of the DCT is to transform the value of pixels to the spatial frequencies. These spatial frequencies are much related to the level of detail present in an image. High spatial frequencies correspond to high levels of detail, while lower frequencies correspond to lower levels of detail. The mathematical definition of the forward DCT is

$$F(u,v)=\frac{2}{N}C(u)C(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}f(x,y)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

$$\text{for } u=0,\ldots,N-1 \text{ and } v=0,\ldots,N-1\,, \qquad (4.5)$$

$$\text{where } N=8 \text{ and } C(k)=\begin{cases}1/\sqrt{2} & \text{for } k=0\\ 1 & \text{otherwise}\end{cases}.$$

The inverse DCT is

$$f(x,y)=\frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}C(u)C(v)F(u,v)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right] \qquad (4.6)$$

for $x=0,\ldots,N-1$ and $y=0,\ldots,N-1$ where $N=8$.

The $F(u,v)$ is called the DCT coefficient, and the DCT basis is

$$\omega_{x,y}(u,v)=\frac{2C(u)C(v)}{N}\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right]. \qquad (4.7)$$

Then we can rewrite the inverse DCT to

$$f(x,y)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}F(u,v)\omega_{x,y}(u,v) \quad \text{for } x=0,\ldots,7 \ \text{ and } \ y=0,\ldots,7\,. \qquad (4.8)$$

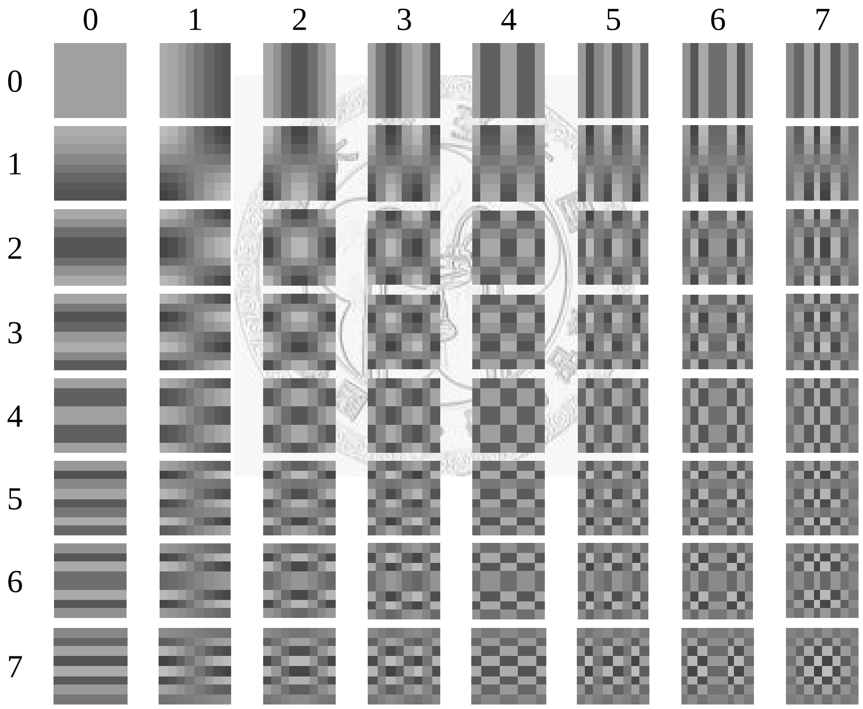


Fig. 4.3 The 8×8 DCT bases $\omega_{x,y}(u,v)$

48	39	40	68	60	38	50	121
149	82	79	101	113	106	27	62
58	63	77	69	124	107	74	125
80	97	74	54	59	71	91	66
18	34	33	46	64	61	32	37
149	108	80	106	116	61	73	92
211	233	159	88	107	158	161	109
212	104	40	44	71	136	113	66

699.25	43.18	55.25	72.11	24.00	-25.51	11.21	-4.14
-129.78	-71.50	-70.26	-73.35	59.43	-24.02	22.61	-2.05
85.71	30.32	61.78	44.87	14.84	17.35	15.51	-13.19
-40.81	10.17	-17.53	-55.81	30.50	-2.28	-21.00	-1.26
-157.50	-49.39	13.27	-1.78	-8.75	22.47	-8.47	-9.23
92.49	-9.03	45.72	-48.13	-58.51	-9.01	-28.54	10.38
-53.09	-62.97	-3.49	-19.62	56.09	-2.25	-3.28	11.91
-20.54	-55.90	-20.59	-18.19	-26.58	-27.07	8.47	0.31

(a) $f\,(x, y)$: 8×8 values of luminance. (b) $F(u,v)$: 8×8 DCT coefficients.

Fig. 4.4 An example of DCT coefficients for a 8×8 block.

The transformed 8×8 block now consists of 64 DCT coefficients. The first coefficient $F(0,0)$ is the DC component and the other 63 coefficients are AC component. The DC component $F(0,0)$ is essentially the sum of the 64 pixels in the input 8×8 pixel block multiplied by the scaling factor $(1/4)C(0)C(0)=1/8$ as shown in equation 3 for $F(u,v)$.

4.3 Quantization

The next step in the compression process is to quantize the transformed coefficients. In JPEG standard, each of the 64 DCT coefficients is uniformly quantized. The 64 quantization step-size parameters for uniform quantization of the 64 DCT coefficients form an 8×8 quantization matrix. Each element in the quantization matrix is an integer between 1 and 255. Each DCT coefficient $F(u,v)$ is divided by the corresponding quantization matrix $Q(u,v)$ and rounded to the nearest integer as:

$$F_q(u,v) = Round\left(\frac{F(u,v)}{Q(u,v)}\right) \tag{4.9}$$

The JPEG standard does not define any fixed quantization matrix. It is the prerogative of the user to select a quantization matrix. There are two quantization matrices provided in Annex K of the JPEG standard for reference, but not requirement. These two quantization matrices are shown in Fig. 4.5.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

(a) Luminance quantization matrix.

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

(b) Chrominance quantization matrix.

Fig. 4.5 Quantization matrix.

699.25	43.18	55.25	72.11	24.00	-25.51	11.21	-4.14
-129.78	-71.50	-70.26	-73.35	59.43	-24.02	22.61	-2.05
85.71	30.32	61.78	44.87	14.84	17.35	15.51	-13.19
-40.81	10.17	-17.53	-55.81	30.50	-2.28	-21.00	-1.26
-157.50	-49.39	13.27	-1.78	-8.75	22.47	-8.47	-9.23
92.49	-9.03	45.72	-48.13	-58.51	-9.01	-28.54	10.38
-53.09	-62.97	-3.49	-19.62	56.09	-2.25	-3.28	11.91
-20.54	-55.90	-20.59	-18.19	-26.58	-27.07	8.47	0.31

(a) $F(u,v)$: 8×8 DCT coefficients.

44	4	6	5	1	-1	0	0
-11	-6	-5	-4	2	0	0	0
6	2	4	2	0	0	0	0
-3	1	-1	-2	1	0	0	0
-9	-2	0	0	0	0	0	0
4	0	1	-1	-1	0	0	0
-1	-1	0	0	1	0	0	0
0	-1	0	0	0	0	0	0

(b) $F_q(u,v)$: After quantization.

Fig. 4.6 An example of quantization for a 8×8 DCT coefficients.

The quantization process has the key role in the JPEG compression. It is the process which removes the high frequencies present in the original image. We do this because of the fact that the eye is much more sensitive to lower spatial frequencies than to higher frequencies. This is done by dividing values at high indexes in the vector (the amplitudes of higher frequencies) with larger values than the values by which are divided the amplitudes of lower frequencies. The bigger values in the quantization table is the bigger error introduced by this lossy process, and the smaller visual quality.

Another important fact is that in most images the color varies slowly from one pixel to another. Therefore, most images will have a small quantity of high detail to a small amount of high spatial frequencies, and have a lot of image information contained in the low spatial frequencies.

4.4 Entropy Coding Algorithms

After doing the DCT transform and quantization over a block of 8x8 values, we have a new 8×8 block. Then, this 8×8 block is traversed in zig-zag shown in Fig. 4.7:

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

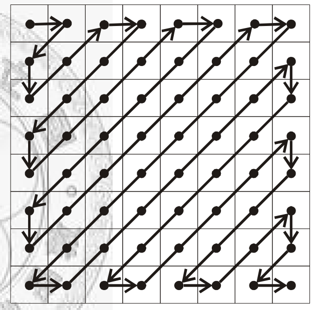


Fig. 4.7 Zig-zag reordering matrix

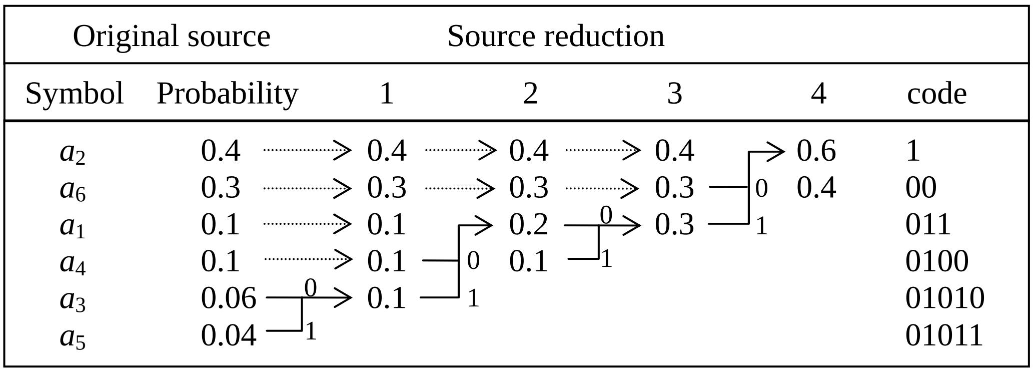
After traversing the 8×8 matrix in zig-zag order, we have now a vector with 64 co-efficients (0,1,...,63). The reason for this zig-zag traversing is that we traverse the 8×8 DCT coefficients in the order of increasing the spatial frequencies. Therefore, we get a vector sorted by the criteria of the spatial frequency. In consequence in the quantized vector at high spatial frequencies, we will have a lot of consecutive zeroes.

We divide the coefficients to the first value (DC-term) and the second to the 64th values (AC-term). The two terms is encoded in different way that will be described in this section.

4.4.1 Huffman Coding

Huffman coding is the most popular technique for removing coding redundancy. When coding the symbols of an information source individually, Huffman coding makes the smallest possible number of code symbols per source symbol. In the noiseless coding theorem, the resulting code is optimal for a fixed value of *n*, while the source symbols are coded one at a time.

The first step in Huffman's approach is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction. Fig. 4.8 illustrates this process for binary coding. The hypothetical set of source symbols and their probabilities are ordered from top to bottom in terms of decreasing probability values. To form the first source reduction, the bottom two probabilities, 0.06 and 0.04, are combined to form a "compound symbol" with probability 0.1. This compound symbol and its associated probability are placed in the first source reduction column so that the probabilities of the reduced source are also ordered from the most to the least probable. This process is then repeated until a reduced source with two symbols is reached.



Original source		Source reduction				
Symbol	Probability	1	2	3	4	code
a_2	0.4	0.4	0.4	0.4	0.6	1
a_6	0.3	0.3	0.3	0.3	0.4	00
a_1	0.1	0.1	0.2	0.3		011
a_4	0.1	0.1	0.1			0100
a_3	0.06	0.1				01010
a_5	0.04					01011

Fig. 4.8 Huffman code procedure.

The second step in Huffman's procedure is to code each reduced source, starting with the smallest source and working back to the original source. The minimal length binary code for a two-symbol source is the symbols 0 and 1. Fig. 4.8 shows that these symbols are assigned to the two symbols on the right. As the reduced source symbol with probability 0.6 was generated by combining two symbols in the reduced source to its left, the 0 used to code it is now assigned to both of these symbols, and a 0 and 1 are arbitrarily appended to each to distinguish them from each other. This operation is then repeated for each reduced source until the original source is reached. The final code appears at the far right in Fig. 4.8. The average length of this code is

$$L_{\text{avg}} = 0.4\times1 + 0.3\times2 + 0.1\times3 + 0.1\times4 + 0.06\times5 + 0.04\times5 = 2.2 \text{ bits/symbol},$$

and the entropy of the source is 2.14 bits/symbol.

4.4.2 Difference Coding

Because the DC coefficients contain a lot of energy, it usually has much larger value than AC coefficients, and we can notice that there is a very close connection between the DC coefficients of adjacent blocks. So, the JPEG standard encode the difference between the DC coefficients of consecutive 8×8 blocks rather than its true value. The mathematical represent of the difference is:

$$\text{Diff}_i = \text{DC}_i - \text{DC}_{i\text{-}1} \tag{4.10}$$

and we set $\text{DC}_0 = 0$. DC of the current block DC_i will be equal to $\text{DC}_{i\text{-}1} + \text{Diff}_i$. Therefore, in the JPEG file, the first coefficient is actually the difference of DCs. Then the difference is Huffman encoded together with the encoding of AC coefficients. The Huffman table of the DC coefficients in JPEG standard is shown in Table 4.1.

Table 4.1 Huffman table of luminance DC coefficients.

category	code length	code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110

4.4.3 Zero Run Length Coding

After quantization, we have the vector with a lot of consecutive zeroes. We can exploit this by run length coding of the consecutive zeroes. Let us consider the 63 AC coefficients in the original 64 quantized vectors first. For example, we have

57, 45, 0, 0, 0, 0, 23, 0, -30, -16, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ..., 0

We encode for each value which is not 0, than add the number of consecutive zeroes preceding that value in front of it. The run length coding is

(0,57) ; (0,45) ; (4,23) ; (1,-30) ; (0,-16) ; (2,1) ; EOB

EOB (End of Block) is a special coded value. If we have reached in a position in the vector from which we have till the end of the vector only zeroes, we'll mark that position with EOB and finish the run length coding of the quantized vector. Note that if the quantized vector does not finishes with zeroes (the last element is not 0), we do not add the EOB marker. Actually, EOB is equivalent to (0,0), so we have

(0,57) ; (0,45) ; (4,23) ; (1,-30) ; (0,-16) ; (2,1) ; (0,0)

We give another example. For the quantized vector as

57, eighteen zeroes, 3, 0, 0, 0, 0, 2, thirty-three zeroes, 895, EOB

The JPEG Huffman coding makes the restriction that the number of previous 0's to be coded as a 4-bit value, so it can not overpass the value 15 (0xF). So, this example would be coded as

(0,57) ; (15,0) ; (2,3) ; (4,2) ; (15,0) ; (15,0) ; (1,895) ; (0,0)

(15,0) is a special coded value which indicates that there are 16 consecutive zeroes.

The JPEG standard specifies that we store the minimum size in bits in which we can keep that value (it is called the category of that value) and then a bit-coded representation of that value like Table 4.2.

Table 4.2 The category and bit-coded values.

Category	Values	Bits for the value
1	-1,1	0,1
2	-3,-2,2,3	00,01,10,11
3	-7,-6,-5,-4,4,5,6,7	000,001,010,011,100,101,110,111
4	-15,...,-8,8,...,15	0000,...,0111,1000,...,1111
5	-31,...,-16,16,...31	00000,...,01111,10000,...,11111
6	-63,...,-32,32,...63	000000,...,011111,100000,...,111111
7	-127,...,-64,64,...,127	0000000,...,0111111,1000000,...,1111111
8	-255,..,-128,128,..,255	...
9	-511,..,-256,256,..,511	...
10	-1023,..,-512,512,..,1023	...
11	-2047,...,-1024,1024,...,2047	...

In consequence for the previous example of AC coefficients

(0,57) ; (0,45) ; (4,23) ; (1,-30) ; (0,-8) ; (2,1) ; (0,0)

We encode only the right value of these pairs as category and bits for the value, except the pairs that are special markers like (0,0) or (15,0). For example, 57 is in the category 6 and it is bit-coded 111001, so we will encode it as 6,111001. And we write again the string of pairs

(0,6,111001) ; (0,6,101101) ; (4,5,10111); (1,5,00001) ; (0,4,0111) ; (2,1,1) ; (0,0)

The first 2 values in bracket parenthesis can be represented on a byte because of the fact that each of the 2 values is 0,1,2,...,15. In this byte, the higher 4-bit represents the number of previous 0s, and the lower 4-bit is the category of the value which is not 0.

Table 4.3 Huffman table of luminance AC coefficients.

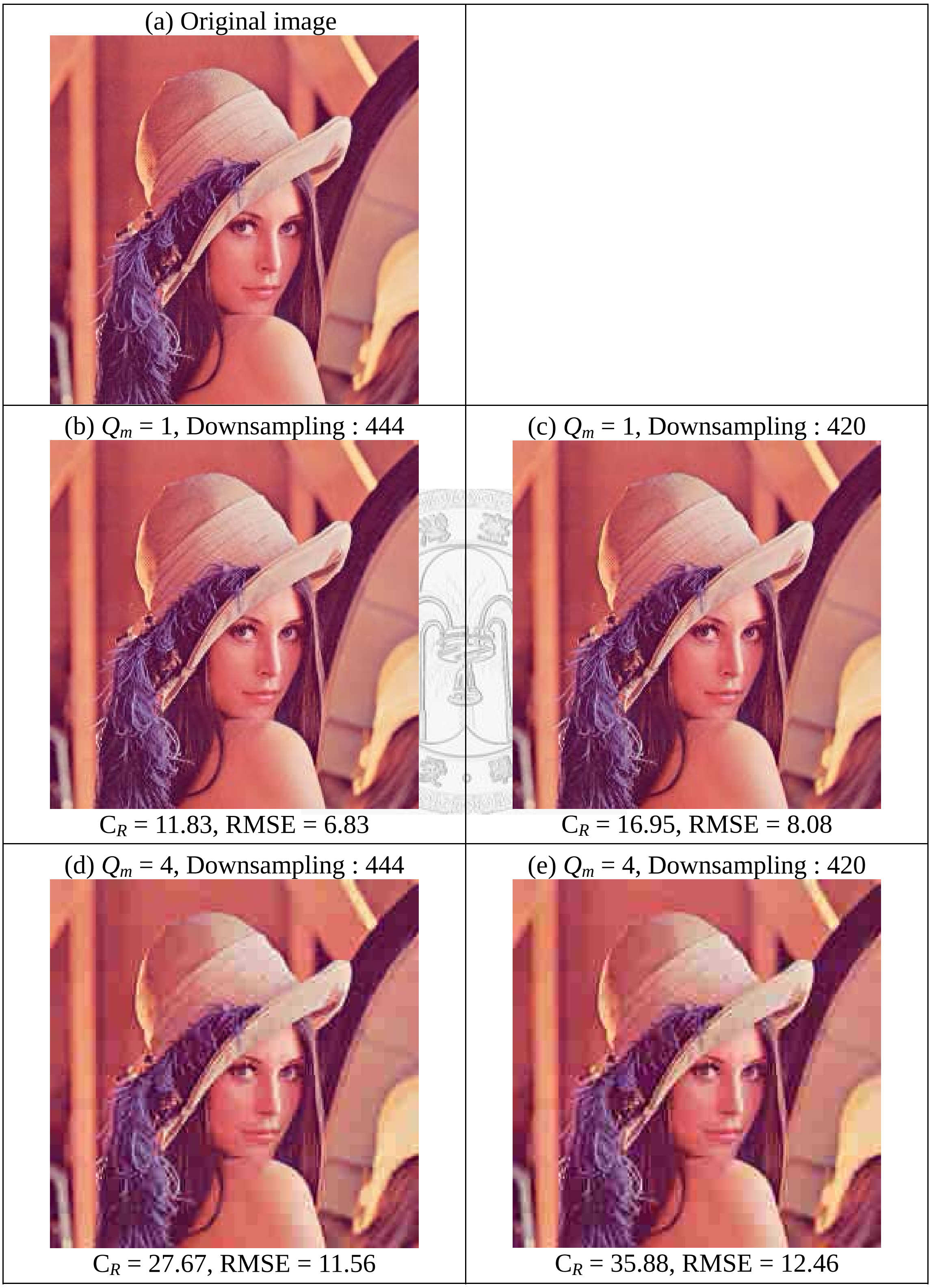
run/category	code length	code word
0/0 (EOB)	4	1010
15/0 (ZRL)	11	11111111001
0/1	2	00
...		
0/6	7	1111000
...		
0/10	16	1111111110000011
1/1	4	1100
1/2	5	11011
...		
1/10	16	1111111110001000
2/1	5	11100
...		
4/5	16	1111111110011000
...		
15/10	16	1111111111111110

The final step is encoding this byte using Huffman coding. For example, if the Huffman code of byte (0,6) is 1111000, and the Huffman code of byte (4,5) is 1111111110011000, and so on. The final stream of bits written in the JPEG file on disk for the previous example of 63 coefficients is

1111000 1111001 , 111000 101101 , 1111111110011000 10111 ,

11111110110 00001 , 1011 0111 , 11100 1 , 1010

4.5 Simulation Result



(a) Original image

(b) Q_m = 1, Downsampling : 444

C_R = 11.83, RMSE = 6.83

(c) Q_m = 1, Downsampling : 420

C_R = 16.95, RMSE = 8.08

(d) Q_m = 4, Downsampling : 444

C_R = 27.67, RMSE = 11.56

(e) Q_m = 4, Downsampling : 420

C_R = 35.88, RMSE = 12.46

Fig. 4.9 Simulation result of the JPEG standard.

Fig. 4.8 shows the simulation result of the JPEG coded image with different compression parameters. The Q_m means the multiple of the quantization matrix. We can see that the type 420 downsampling type can decrease the compression ratio with little vision error. If Q_m increase to 4, it can also decrease the compression ratio but generating the clear characteristic 'blocky and blurry' artifacts.

4.6 Conclusion

We have introduced the basic compression methods of JPEG standard. Although this standard has become the most popular image format, it still has some properties to improvement, especially to the 'blocky and blurry' artifacts problem. The intuitive way is to divide the region by the edges rather than 8×8 block. The method will describe in the following chapters. The algorithms in JPEG standard will be modified to apply in our proposed methods.

Chapter 5

Boundary Description and Compression

The segmentation technique outputs the boundaries in the form of pixels along a shape. These data sometimes are used directly to describe the shape of an image segment. In practice, we use some other representation, called descriptor, to describe the boundaries. In this chapter, we will introduce some popular boundary descriptors to record the boundary of an image segments efficiently.

5.1 Polygonal Approximation

The simplest way to reduce the data size of boundary is polygonal approximation. It only adopts some critical points which can be linked to a polygon to approximate the original boundary. There are two basic techniques to find the critical points: merging technique and splitting technique.

5.1.1 Merging technique

Merging technique is to merge points along a boundary to a straight line until the least square error exceeds a threshold. When this condition occurs, the parameters of the line are stored. Then the procedure is repeated to merging new points along the boundary until the error again exceeds the threshold. At the end of the procedure the intersections of adjacent line segments form the vertices of the polygon. One of the principal difficulties with this method is that vertices in the resulting approximation do not always correspond to the corners of the original boundary, because a new line is not started until the error threshold is exceeded. For instance, if a long straight line were being tracked and it turned a corner, a number of points past the corner would be absorbed before the threshold was exceeded. However, the splitting technique may be used to alleviate this difficulty.

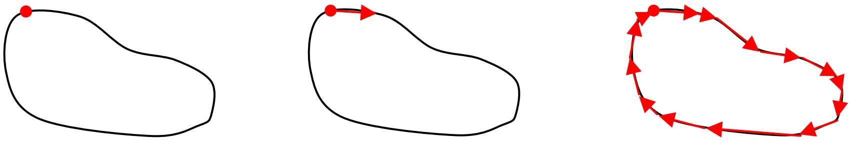


Fig. 5.1 Use merging technique to find the polygonal approximation.

5.1.2 Splitting Technique

The splitting technique is to subdivide a segment successively into two parts until a specified criterion is satisfied. For instance, a requirement might be that the maximum perpendicular distance from a boundary segment to the line joining its two end points not exceeds a preset threshold. If it does, the farthest point from the line becomes a vertex, thus subdividing the initial segment into two sub-segments. This approach has the advantage of seeking prominent corner points. For a closed boundary, the best starting points usually are the two farthest points in the boundary. For example, Fig. 5.2(a) shows an object boundary and a subdivision of this boundary about its farthest points. The point marked *c* is the farthest point from the top boundary segment to line *ab*. Similarly, point *d* is the farthest point in the bottom segment. Fig. 5.2(b) shows the result of the second splitting procedure. Then the procedure is repeated until no point in the new boundary segments has a perpendicular distance that exceeds the threshold. The procedure terminates with the polygon shown in Fig. 5.2(c).

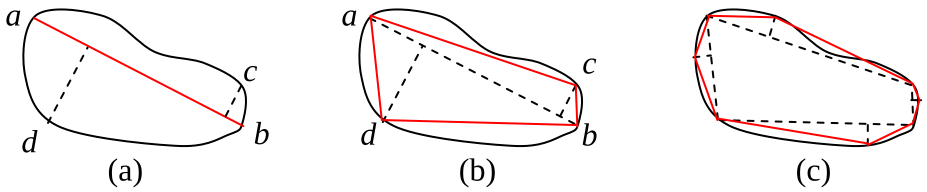


Fig. 5.2 Using splitting techniques to find polygonal approximation.

5.2 Fourier Descriptor

Fig. 5.3 shows a K-point digital boundary in the xy-plane. Starting at an arbitrary point (x_0, y_0), coordinate pairs (x_0, y_0), (x_1, y_1), ..., (x_{K-1}, y_{K-1}) are encountered in traversing the boundary in the counterclockwise direction. These coordinates can be expressed in the form $x(k) = x_k$ and $y(k) = y_k$. With this notation, the boundary itself can be represented as the sequence of coordinates $s(k) = [x(k), y(k)]$, for $k = 0, 1, 2, ..., K-1$. Moreover, each coordinate pair can be treated as a complex number so that

$$s(k) = x(k) + jy(k) \tag{5.1}$$

for $k = 0, 1, 2, ..., K-1$. That is, the x-axis is treated as the real axis and the y-axis as the imaginary axis of a sequence of complex numbers. This representation is great that reduces a 2-D problem to a 1-D problem.

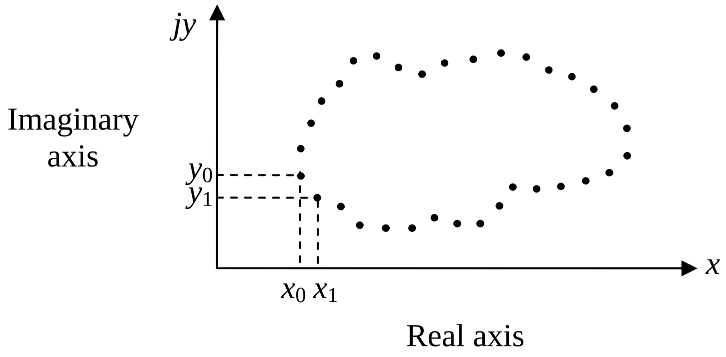


Fig. 5.3 A digital boundary that represented as a complex sequence.

The Discrete Fourier transform (DFT) of $s(k)$ is

$$a(u) = \frac{1}{K}\sum_{k=0}^{K-1} s(k)\, e^{-j2\pi uk/K} \qquad (5.2)$$

for u = 0, 1, 2, ... , K−1. The complex coefficients $a(u)$ are called the Fourier descriptors of the boundary. The inverse Fourier transform of these coefficients restores $s(k)$. That is,

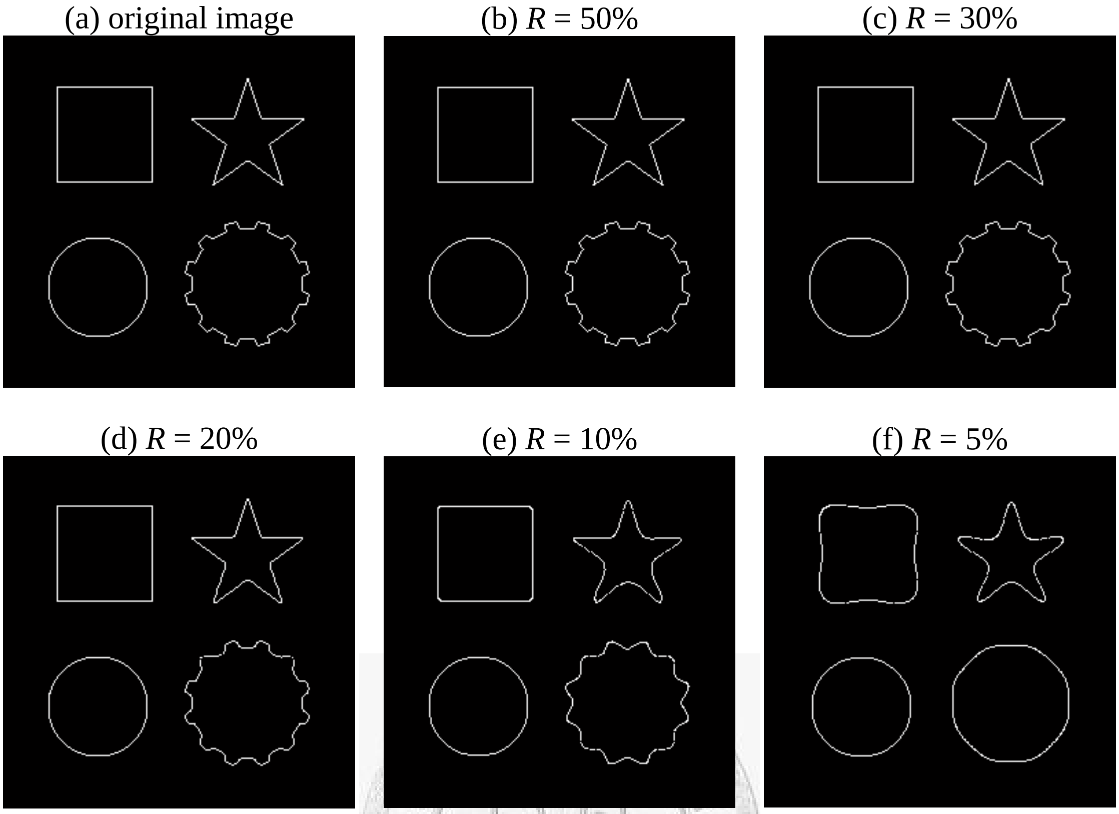
$$s(k) = \sum_{u=0}^{K-1} a(u)\, e^{j2\pi uk/K} \qquad (5.3)$$

for k = 0, 1, 2, ... , K−1. If we do not use of all the Fourier coefficients, only the first P coefficients are used. This is equivalent to setting $a(u)$ = 0 for $u > P-1$ in (5.3). The result is the approximation to $s(k)$:

$$\hat{s}(k) = \sum_{u=0}^{P-1} a(u)\, e^{j2\pi uk/K} \qquad (5.4)$$

for k = 0, 1, 2, ... , K−1. Although only P terms are used to obtain each component of $\hat{s}(k)$, k still ranges from 0 to K−1. That is, the same number of points exists in the approximate boundary, but not as many terms are used in the reconstruction of each point. In Fourier transform theorem, high-frequency components account for fine detail, and low frequency components determine global shape. Thus the smaller P becomes, the more detail that is lost on the boundary.

Fig. 5.4(a) shows four boundaries and Fig. 5.4(b)~(f) is the results of using (5.4) to reconstruct these boundaries for various values of P. Note illustration of that the percentage of reserve descriptors has to be about 20% before the reconstructed boundary looks more like its original. Thus, a few low-order coefficients are able to capture gross shape, but many more high-order terms are required to define accurately sharp features such as corners and straight 1ines. This result is not unexpected in view of the role played by low- and high-frequency components in defining the shape of a region.



(a) original image

(b) *R* = 50%

(c) *R* = 30%

(d) *R* = 20%

(e) *R* = 10%

(f) *R* = 5%

Fig. 5.4 Reconstruction from Fourier descriptors with different reserve rate.

Chapter 6

Proposed Methods for Boundary Description and Compression

We have introduced several methods to describe a boundary in the previous chapter. In this chapter, we propose two methods to improve the vision effect and compression ratio of the boundary description. One is second-order curve descriptor which combined with the splitting technique. The other is modified Fourier descriptor which segments the boundary and then computes the Fourier transform of the boundary segments.

6.1 Second-Order Curve Descriptor

In the splitting techniques for polygonal approximation of a boundary which is described in section 5.1, it finds several nodes in a closed boundary and uses straight lines to link these nodes. This method gives us an idea to propose a new boundary descriptor called second-order curve descriptor. In this method, we add a second-order parameter to each boundary segment and make it a second-order curve instead of a straight line.

6.1.1 Second-Order Polynomial Approximate to a Boundary Segment

First, we divide a closed boundary to several non-closed boundaries. For instant, we can directly cut the boundary to one half. Then, we find second-order polynomials approximate to each boundary segment. For a boundary segment $s_1(k)$, for k = 0, 1, ..., K–1, we shift it to make its first point $s_1(0)$ locate at origin. That is,

$$s_2(k) = s_1(k) - s_1(0), \quad \text{for } k = 0,1,\ldots,K-1. \tag{6.1}$$

And then we rotate $s_2(k)$ an angle θ to make its final point $s_2(K–1)$ locate at x-axis. The angle θ can be calculated by

$$\theta = \tan^{-1}\left(\frac{y_{K-1} - y_0}{x_{K-1} - x_0}\right) \tag{6.2}$$

where (x_0, y_0) is the first point of $s_2(k)$ and $(x_{K–1}, y_{K–1})$ is the final point of $s_2(k)$. And then every point (x_k, y_k) of $s_2(k)$ is multiply a rotation matrix that

$$\begin{bmatrix} x'_k \\ y'_k \end{bmatrix} = \begin{bmatrix} -\sin\theta & \cos\theta \\ \cos\theta & \sin\theta \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix}, \quad \text{for } k = 0,1,\ldots,K-1, \tag{6.3}$$

where (x'_k, y'_k) is the k-th point of the rotated boundary segment $s_3(k)$.

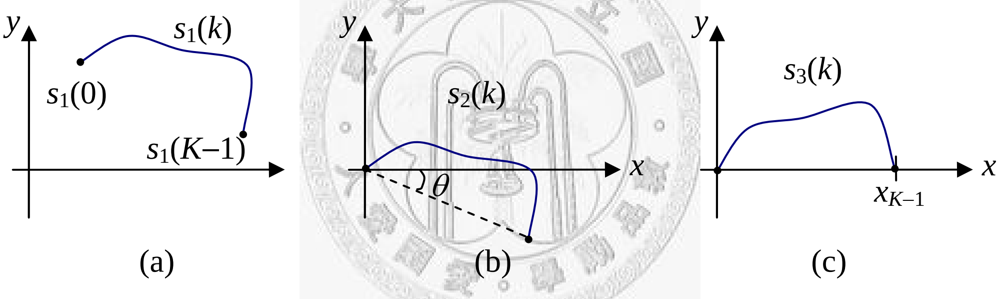


Fig. 6.1 Initial steps to a boundary segment.

After these steps to a boundary segment, it can be treat as a function of x_k and can be approximated as a second-order polynomial that

$$y_k = Ax_k^2 + Bx_k + C, \quad \text{for } k = 0,1,\ldots,K-1.. \tag{6.4}$$

Because the first point is located at the origin, the constant C can be set to zero. If the (x, y) of (6.4) is substituted for the final point $(x_{K–1}, 0)$ of $s_3(k)$, we can get that the coefficient $B = –A\, x_{K–1}$. Then we can rewrite (6.4) to

$$y_k = Ax_k^2 - Ax_{K-1}x_k, \quad \text{for } k = 0,1,\ldots,K-1. \tag{6.5}$$

Now we need to find the coefficient A in order to approximate $s_3(k)$ most closely using (6.5). Therefore, we count the sum of error E that

$$E = \sum_{k=0}^{K-1} \left(s_3(k) - (Ax_k^2 - Ax_{K-1}x_k) \right)^2 . \tag{6.6}$$

The value of E is least when its derivatives of A is zero, that is

$$\begin{aligned} \frac{\partial E}{\partial A} &= 2\sum_{k=0}^{K-1} \left[\left(s_3(k) - Ax_k^2 + Ax_{K-1}x_k \right) \left(-x_k^2 + x_{K-1}x_k \right) \right] \\ &= 2\sum_{k=0}^{K-1} \left[s_3(k) \left(-x_k^2 + x_{K-1}x_k \right) + A \left(-x_k^2 + x_{K-1}x_k \right)^2 \right] \\ &= 2\sum_{k=0}^{K-1} \left[s_3(k) \left(-x_k^2 + x_{K-1}x_k \right) \right] + 2A\sum_{k=0}^{K-1} \left(-x_k^2 + x_{K-1}x_k \right)^2 \\ &= 0 \end{aligned} . \tag{6.7}$$

And we can get

$$A = -\frac{\sum_{k=0}^{K-1} \left[s_3(k) \left(-x_k^2 + x_{K-1}x_k \right) \right]}{\sum_{k=0}^{K-1} \left(-x_k^2 + x_{K-1}x_k \right)^2} . \tag{6.8}$$

Now, we can use only two end points and a parameter A to approximate a boundary segment.

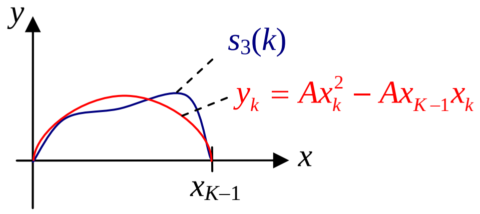
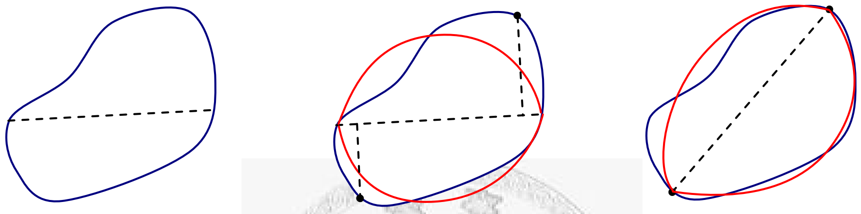


Fig. 6.2 The original boundary segment $s_3(k)$ and the approximate 2nd-order curve y_k.

6.1.2 Splitting Technique with Approximate Second-Order Curve

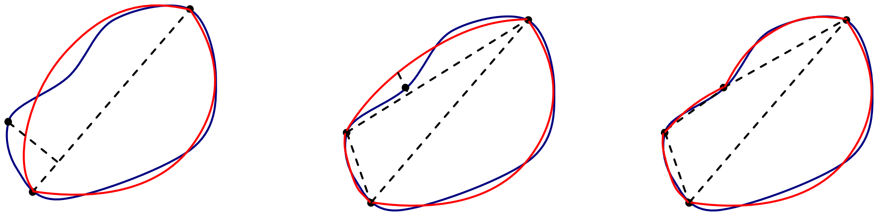
In the splitting technique described in section 5.1, we have to find the longest distance between any two points first. However, in this section, we can use the second-order curve described above to find the two points. Initially, we directly split the closed boundary to one half and produce two non-closed curves. After calculating the two approximate second-order curve of the two non-closed curves, we find two points that have the largest error between the original curve and the approximate curve. Use the two points to split the original closed boundary and find the most two error points again. Repeat the procedure several time and we can get the appropriate two points to split the closed boundary first, as illustrate in Fig. 6.3.



(a) (b) (c)

Fig. 6.3 Find the first two points to split the boundary.

After finding the first two node points, we continue to find the point with largest error between the original boundary and the approximate second-order curve. Then we set this point to the third node that splits the boundary to three segments. Calculate the approximate second-order curve of the two boundary segment by the side of the third node. Repeat the procedure until the largest error is less than a threshold, as illustrate in Fig. 6.4.



(a) (b) (c)

Fig. 6.4 Splitting technique with approximate second-order curve.

6.2 Fourier Descriptor of Non-Closed Boundary Segments

Due to the problem of the Fourier descriptor, that is, the angle of the shape will be smooth when the reserve rate of coefficient is lower than 20%. We proposed a new method to improve the Fourier descriptor. We find the corner points and segment the boundary by these points first. Then we convert these boundary segments to Fourier descriptors which are redefined for a non-closed boundary segment.

6.2.1 Boundary Segmentation

First of all, we have to find the proper corner points. An intuition way is to make use of the original Fourier descriptors described in Section 5.2. We compute the Fourier descriptors and only reserve the first 20% coefficients. Then we reconstruct the new boundary points. As illustrated in Fig. 6.5(a)(b), the blue line is the original boundary and the green line is the reconstructed boundary. Then we compute the error of the two boundary, as illustrated in Fig. 6.5(c)(d). The corner points are at the regional maximum place of the error value. In our experiment, we define the corner points are at the place that error value is greater then 0.5 and is the maximum in the near 10-point region. This definition ensures that the number of corner points is not too many and the obvious corner points can be found. As illustrated in Fig. 6.5, the red points are the corner points we found.

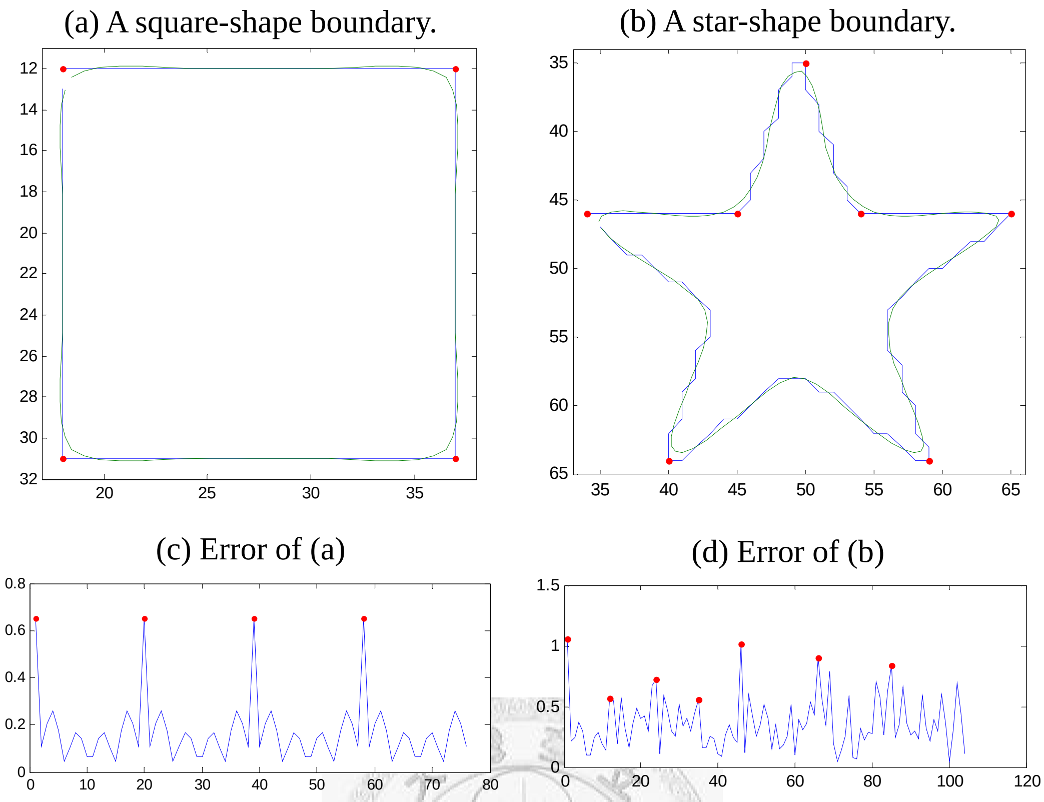


Fig. 6.5 Error of the original boundary points and the reconstruct ones.

6.2.2 Fourier Descriptor of Non-Closed Boundary Segment

We segment the boundary with the corner points and then we compute the Fourier descriptor of the boundary segments. However, if we do not use the whole coefficients, the reconstructed boundary segment will be closed due to the discontinuous two end points which is shown in Fig. 6.6.

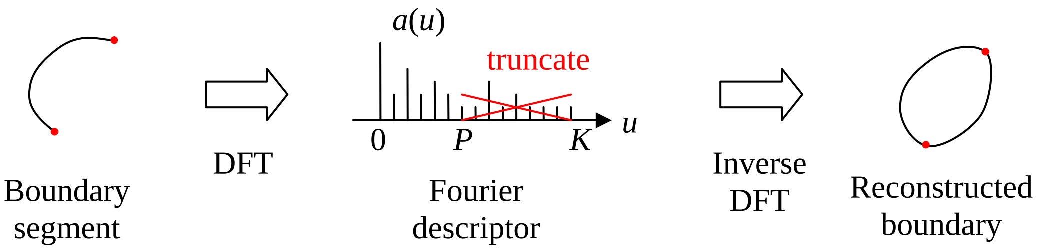


Fig. 6.6 Use Fourier descriptor to a non-close boundary segment.

To solve the non-closed problem, we adapt the following steps:

1. Record the coordinates of the two end points, $s_1(0)$ and $s_1(K-1)$ of the boundary segment $s_1(k)$ for $k = 0,1,\ldots,K-1$.

2. Shift the first point to the origin, and the other points shift the same distance.

$$s_2(k) = s_1(k) - s_1(0), \qquad \text{for } k = 0,1,\ldots,K-1. \tag{6.9}$$

3. Shift the end point to the origin, and the other points shift linearly according to its point number k. If (x_k, y_k) is a point of the boundary segment $s_2(k)$, for $k = 0, 1, \ldots, K-1$, it will be shifted to (x_k', y_k') of $s_3(k)$ where

$$\begin{aligned} x_k' &= x_k - x_{K-1} \times k/(K-1) \\ y_k' &= y_k - y_{K-1} \times k/(K-1) \end{aligned}. \tag{6.10}$$

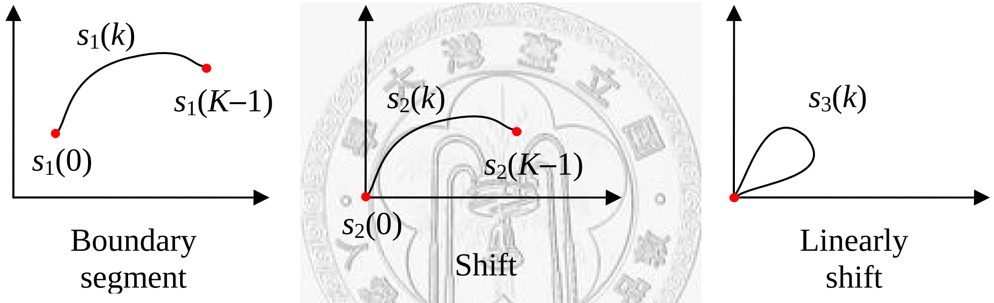


Fig. 6.7 Linearly shift the two end points to the origin.

4. Add a boundary segment which is odd symmetry to the original one. Then the new boundary segment is closed and is smooth in the two end points. If the linearly shifted boundary segment in step 3 is $s_3(k)$, the new boundary segment is

$$s_4(k) = s_3(k) - s_3(-k), \qquad \text{for } k = -K+1, -K+2, \ldots, 0, 1, \ldots, K-1. \tag{6.11}$$

5. Compute the Fourier descriptor to the new boundary segment $s_4(k)$. That is,

$$a(u) = \frac{1}{K} \sum_{k=-K+1}^{K-1} s_4(k)\, e^{-j2\pi uk/K}, \quad \text{for } u = 0,1,\ldots,2K-2. \tag{6.12}$$

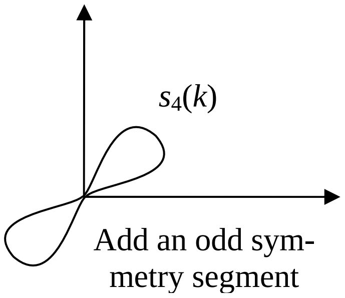

$s_3(k)$
Linearly shifted boundary segment
$s_4(k)$
Add an odd symmetry segment

Fig. 6.8 Add an odd symmetry boundary segment to make the two end points smooth.

By the property of DFT, if $a(u)$ is the DFT of $s(k)$,

$$-s(-k) \xrightarrow{DFT} -a(-u)\,. \tag{6.13}$$

Therefore, if the signal $s(k)$ is odd symmetry, its DFT $a(u)$ is odd symmetry, too.

$$s(k) = -s(-k) \xrightarrow{DFT} a(u) = -a(-u)\,. \tag{6.14}$$

Moreover, because the central point of $s_3(k)$ is origin, the DC-term (the first coefficient of DFT) is zero. There is, we only need to record the second to the K-th coefficient of the Fourier descriptors, as illustrated in Fig. 6.9.

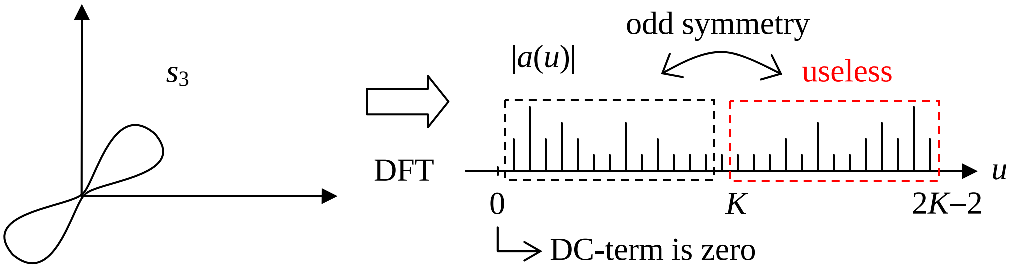


Fig. 6.9 (a) The new boundary segment. (b) Fourier descriptor of (a).

6.2.3 Boundary Compression

Now we can truncate the high frequency coefficients without worrying the discontinuous two end points. As shown in Fig. 6.10(a), we can only reserve the P–1 coefficients and truncate the other coefficients. We recover the whole coefficient by stuffing zeroes and then copy to the odd symmetry part, as shown in Fig. 6.10(b).

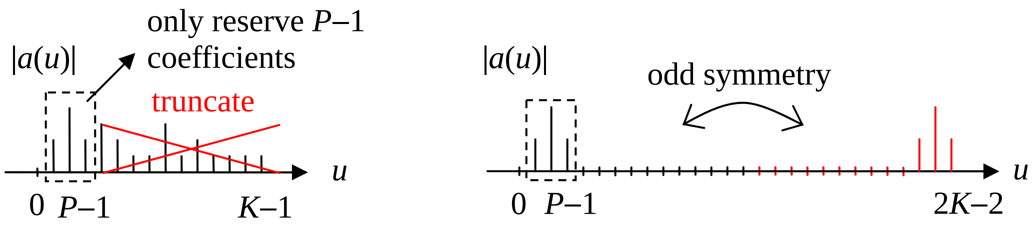


Fig. 6.10 (a) The reserve P−1 coefficients. (b) Recover the whole coefficients from (a).

In Fig. 6.11, we use the modified Fourier descriptor method to an entire boundary and compare to the original Fourier descriptor described in the previous chapter. In the original Fourier descriptor, we can observe that the boundary which has sharp corner will be distortion obviously when R is less than 20%. However, if we use the modified Fourier descriptor method that has split the boundary at the corner point to several boundary segments, the sharp corner can be preserve when R is less.

We can see that, when R = 10%, the result of the original Fourier descriptor method is obvious distortion. However, in the modified Fourier descriptor method, the characteristic of corners can be preserved and some longer boundary segments are not distorted obviously. The problem is that the shorter boundary segments are stretched from a curve due to the fact that the preserve coefficients are less than one. Therefore, we force the reserve coefficient number is greater than three, where the three coefficients can represent the most characteristic in our experiment. The improved result is shown in Fig. 6.12.

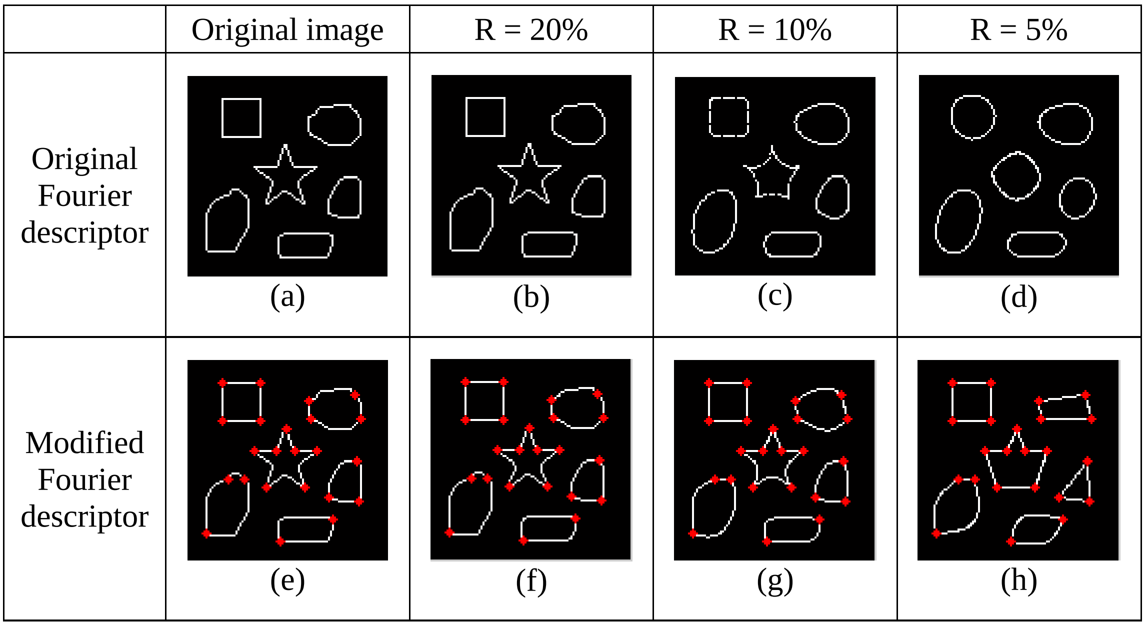
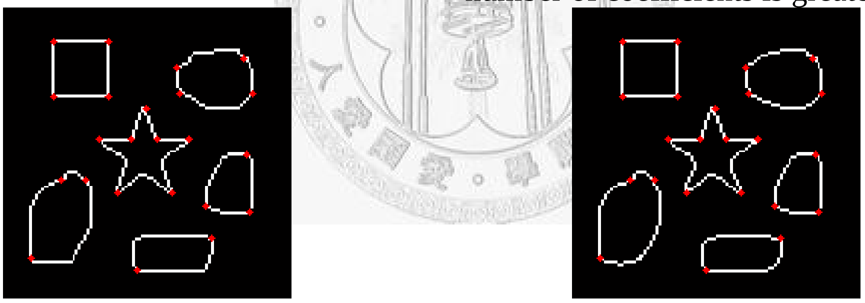


Fig. 6.11 Boundary compression with different coefficient reserve rate.


(a) Original boundary

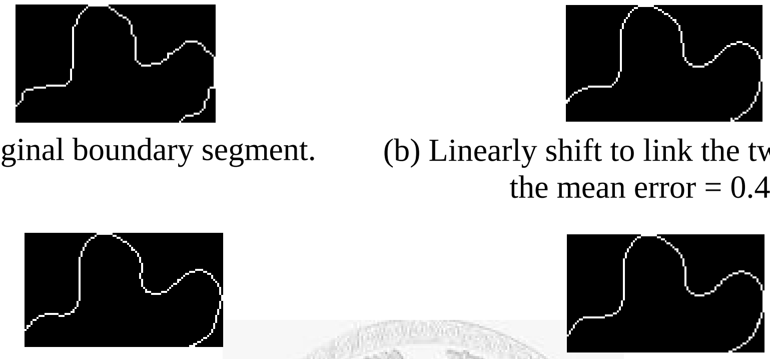
(b) Reconstructed boundary with *R* = 10% and number of coefficients is greater than 3

Fig. 6.12 Result of improved boundary compression.

6.2.4 Other Method for Fourier Descriptor of Boundary Segment

We compare the three methods for Fourier descriptor of a boundary segment with the coefficient reserve rate *R* = 0.1. Fig. 6.13(b) only linearly shifts to links the two end points. We can find that the distortion at the two end points is obvious. Fig. 6.13(c) traces back at the end point. In other words, it only adds another even symmetry boundary segment. It improves the end point distortion but increase the mean error. Fig.

6.13(d) is the method we proposed above that not only linearly shifts to link the two end points but also adds another odd symmetry boundary segment. It has a good effect that not only avoids the end point distortion but also significantly decrease the mean error. Therefore, the method we proposed is truly better than the other methods proposed before.



(a) Original boundary segment.

(b) Linearly shift to link the two end points.
the mean error = 0.4545

(c) Add an even symmetry.
the mean error = 0.4949

(d) Linearly shift and add an odd symmetry.
the mean error = 0.3793

Fig. 6.13 Three methods for Fourier descriptor of a boundary segment with R = 0.1.

6.3 Boundary Encoding of the Boundary Segments

In the previous sections, we introduce two boundary descriptors which are both based on boundary segment. They are similar because they both need to record the coordinates of corners and some coefficients of each boundary segment. In this section, we introduce our method to encode these data in order to save more space. Fig. 6.14 and Fig. 6.15 are the block diagram of encoder and decoder, respectively. We will explain them and focus on the Fourier descriptor.

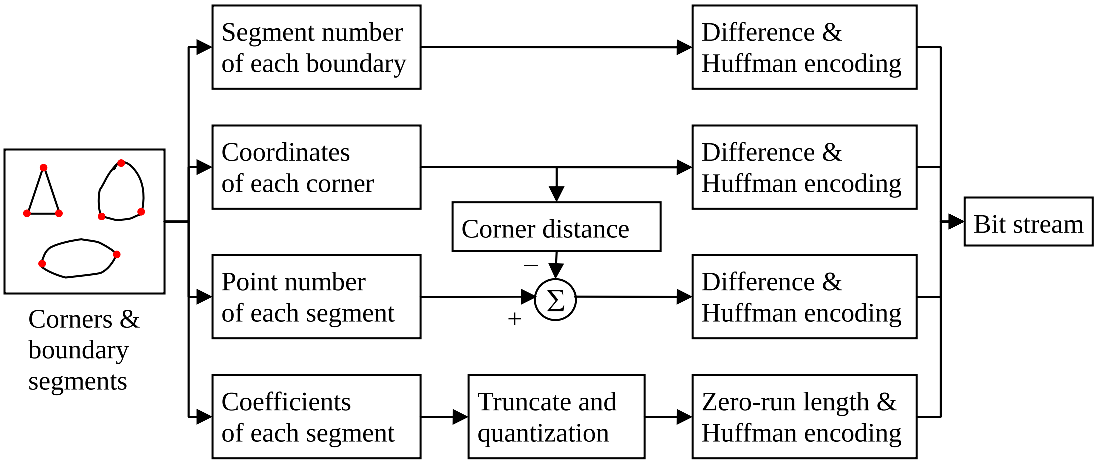

Segment number of each boundary
Difference & Huffman encoding
Coordinates of each corner
Difference & Huffman encoding
Corner distance
–
Bit stream
Point number of each segment
+
Σ
Difference & Huffman encoding
Corners & boundary segments
Coefficients of each segment
Truncate and quantization
Zero-run length & Huffman encoding

Fig. 6.14 Boundary segment encoder.

In the boundary segment encoder, we have four data to record: the segment number of each boundary, the coordinates of each corner, the point number of each boundary segment, and the coefficients of each boundary segment. The first three data are arrays with positive numbers, and we suppose that the adjacent number in these arrays will be near. Therefore, we can use the difference encoding described in Section 4.4.2 and then use the Huffman encoding.

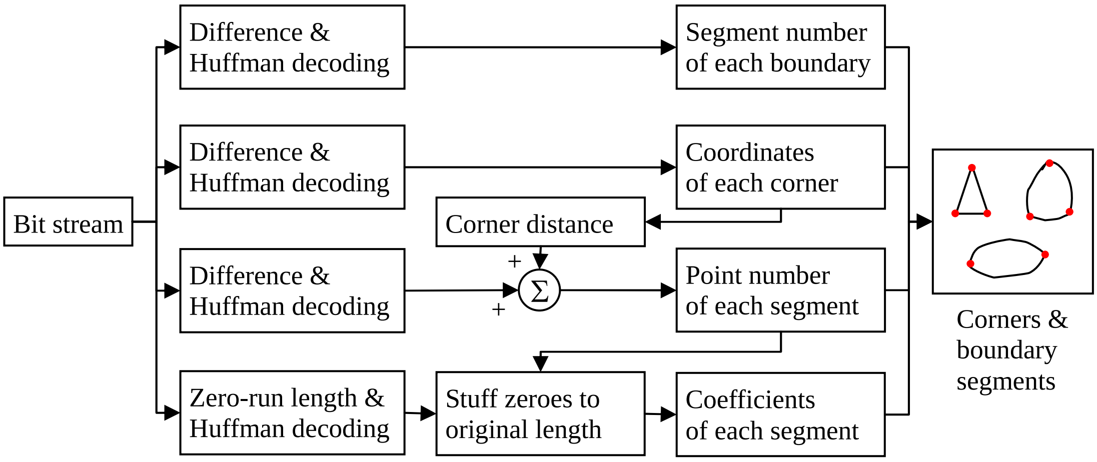

Difference & Huffman decoding
Segment number of each boundary
Difference & Huffman decoding
Coordinates of each corner
Bit stream
Corner distance
Difference & Huffman decoding
+
Σ
+
Point number of each segment
Corners & boundary segments
Zero-run length & Huffman decoding
Stuff zeroes to original length
Coefficients of each segment

Fig. 6.15 Boundary segment decoder.

In the third data, we record the difference of the point number and the distance of two end points of each boundary segment. The distance we used here is the sum of the two distances of the *x*-axis and *y*-axis. Compared with Euclidean distance, it is integer and often closer to point number of segment.

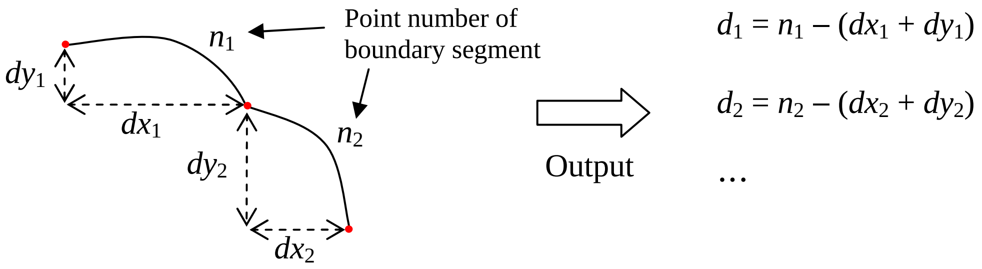

Point number of boundary segment
n_1
n_2
dy_1
dx_1
dy_2
dx_2
Output
$d_1 = n_1 - (dx_1 + dy_1)$
$d_2 = n_2 - (dx_2 + dy_2)$
…

Fig. 6.16 Point number of boundary and distance of two end points.

As illustrated in Fig. 6.16, we have vector **n** of the point number of each boundary segment. Similarly, **dx** and **dy** are vectors that record the distances of the x-axis and y-axis, respectively. Therefore, we can get the difference vector **d** where

$$\mathbf{d} = \mathbf{n} - (\mathbf{dx} + \mathbf{dy})\,. \tag{6.15}$$

The value difference vector **d** is close to zeroes and is appropriate to encode with Huffman encoding. In the decoder as shown in Fig. 6.15, we can recover **n** that

$$\mathbf{n} = \mathbf{d} + (\mathbf{dx} + \mathbf{dy})\,, \tag{6.16}$$

where **dx** and **dy** can be calculated by the corners.

In the fourth data, we combine the coefficients of each boundary segment in a whole boundary and encode them with zero-run length and Huffman coding described in Section 4.4.3. When the boundary segment is a straight line, its coefficients of Fourier descriptor will be all zeroes. Therefore, it is appropriate to use the zero-run length coding when many boundary segments are straight lines.

Because we have recoded the point number of each boundary segment, we can calculate the reserved coefficient number and split the combined coefficient array correctly. And then we can recover the original coefficients by stuffing zeroes to the truncated position.

Finally, we encode the boundary segment of Fig. 6.12(b) as an example. The data size and the bytes after encoding are list in Table 6.1. Note that the coefficients of each segment have been truncated to three complex numbers. The coding technique can only reduce some information quantity. The major compression quantity is truncating coefficients described before.

Table 6.1 Data size and coding bytes of the example of Fig. 6.12(b).

data name	**data size**	**after encoding**
Segment number of each boundary	6	4 bytes
Coordinates of each corner	23×2	38 bytes
Point number of each segment	23	13 bytes
Coefficients of each segment	69×2	102 bytes
sum	223	157 bytes

6.4 Conclusion

We have described the methods to record a boundary efficiently. We have proposed two boundary descriptions: the second-order curve descriptor and the modified Fourier descriptor. In our experiment, the former has more complexity and needs to cut more boundary segment than the latter does. Therefore, we adopt the modified Fourier descriptor in our entire compression system.

Chapter 7

Arbitrary-Shape Image Segment Compression

A complete representation of the arbitrary-shape image segment includes two parts: shape and internal contents. The former represents the boundary information of the image segment, and the latter represents the internal color intensity value. Both these two components are required to represent an arbitrary-shape image segment. In the previous chapter, we described the methods to record the boundary information efficiently. In this chapter, we focus on designing efficient representation of the image contents to achieve good compression and image quality.

In particular, we use transform coding of the image contents, such as the widely used discrete cosine transform (DCT). The advantage of using the transform coding is that existing codec hardware can be used to process arbitrary-shape image segment, as well as traditional rectangular image.

7.1 Block-Filled Method

After segmentation, image segments are placed in blocks. For arbitrary-shape image segments, boundary blocks usually have part of the pixel values defined only. Let $P(x, y)$ represent the pixel values within an $N \times N$ pixel block area, called R. Let B represent the occupied region within the block. An irregular shaped image segment has $P(x, y)$ defined within region B only. The block-filled transform coding technique fills up the redundant area outside the boundary and then utilizes the traditional transform coding.

The approach to filling the region outside the boundary with optimal redundant data provides a freedom for us to optimize the transform spectrum. The simplest method to augment a partially defined image segment into a full block image is by stuffing zeroes outside the image boundary. But it is well known this method may introduce sharp edges on the boundary and high-frequency components in the transform spectrum. A more promising method is to extend the image segment with its “mirror image” outside the boundary. Fig. 7.1 shows a simple example in one dimension. In general, the support of the original image sequence is not exactly one half of the block size, we may need to duplicate the image sequence several times and truncate it at the block boundary. For a 2D image segment, we can apply this 1D mirror image extension technique in one direction first and then once again in another direction.

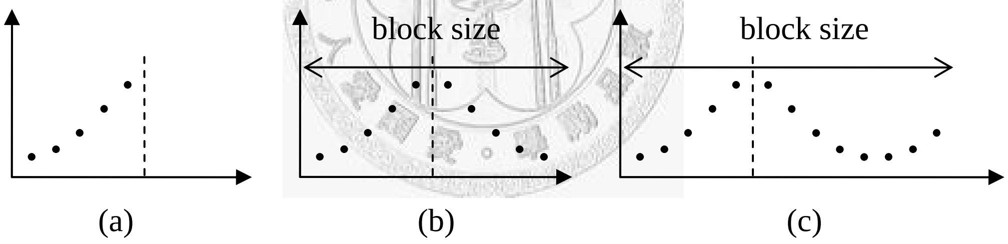


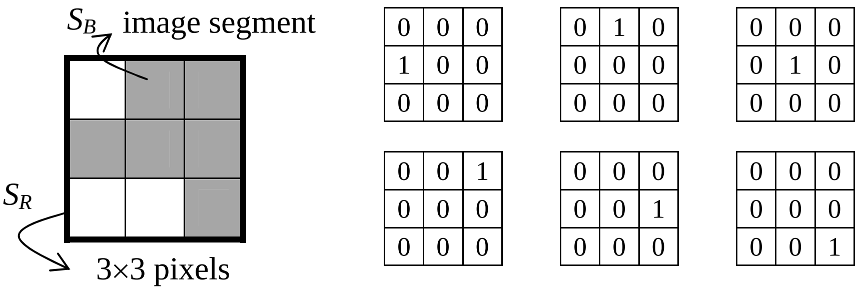
Fig. 7.1 Fill the outside region with the mirror image of the internal contents.

(a) Original segment. (b) The segment size equals one half of the block size.

(c) The segment size is smaller than one half of the block size.

7.2 Arbitrary-Shape Image Transform

Instead of filling data outside the image boundary and applying the full-block rectangular transform, we can focus on the defined image contents only. Mathematically, we define S_R as the linear space spanned over the whole square block R, S_B as the subspace spanned over the irregular region *B* only. As illustrate in Fig. 7.2(a), space S_R has a dimension equal to 9, while the dimension of subspace S_B equals to 6 only. One possible basis for subspace S_B is shown in Fig. 7.2(b). Each basis matrix has a single non-zero element only.



$$\begin{matrix}0&0&0\\1&0&0\\0&0&0\end{matrix}\quad\begin{matrix}0&1&0\\0&0&0\\0&0&0\end{matrix}\quad\begin{matrix}0&0&0\\0&1&0\\0&0&0\end{matrix}$$

$$\begin{matrix}0&0&1\\0&0&0\\0&0&0\end{matrix}\quad\begin{matrix}0&0&0\\0&0&1\\0&0&0\end{matrix}\quad\begin{matrix}0&0&0\\0&0&0\\0&0&1\end{matrix}$$

(a) (b)

Fig. 7.2 (a) An irregular-shape image segment in a 3×3 block area.

(b) A canonical basis of the subspace.

Every arbitrary-shape image segment can be considered as a vector $P(x, y)$ in S_B. To represent this vector completely, we can find a set of independent vectors b_i in S_B and describe the image segment as a linear combination of b_i's. The distinction between this approach and that in the previous section is that the whole problem domain now is confined in the subspace S_B only. If we still want to use traditional block-based transform bases f_i, we can project these basis functions into subspace S_B,

$$\hat{f}_i = \text{Project}(f_i, S_B)\,, \tag{7.1}$$

and describe vector $P(x, y)$ as a linear combination of $\hat{f}_i$'s. Actually, the above projection is very simple. It just removes the components of f_i outside subspace S_B.

An important issue remains now is how to find optimal basis functions in subspace S_B such that we can use the least number of coefficients to reconstruct the image segment vector with satisfactory errors. The above formulation does provide a very flexible platform to derive new transform bases and evaluate their performance.

Chapter 8

Proposed Method for Arbitrary-Shape Image Segment Compression

In this chapter, we implement the process to compress the contents in an arbitrary-shape image segment. We modify the DCT bases to the arbitrary-shape by the Gram-Schmidt process. After transforming, we quantize the coefficients and then encode them. Finally, we combine the encoded coefficients with the code of boundary in the whole compression system.

8.1 Arbitrary-Shape Transform with DCT Bases

The mathematical definition of DCT for an $N \times N$ image block is showed in (4.5) and (4.6). However, the height and width of an image segment is usually not the same. Therefore, we redefine the DCT as

Forward DCT:

$$
\begin{aligned}
&F(u,v) = \frac{2C(u)C(v)}{\sqrt{H*W}} \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} f(x,y) \cos\left[\frac{\pi(2x+1)u}{2W}\right] \cos\left[\frac{\pi(2y+1)v}{2H}\right] \\
&\text{for } u = 0,\ldots,W-1 \text{ and } v = 0,\ldots,H-1 \qquad (8.1) \\
&C(k) = \begin{cases} 1/\sqrt{2} & \text{for } k = 0 \\ 1 & \text{otherwise} \end{cases}
\end{aligned}
$$

Inverse DCT:

$$f(x,y)=\frac{2}{\sqrt{H*W}}\sum_{u=0}^{W-1}\sum_{v=0}^{H-1}C(u)C(v)F(u,v)\cos\left[\frac{\pi(2x+1)u}{2W}\right]\cos\left[\frac{\pi(2y+1)v}{2H}\right]$$

for $x=0,...,W-1$ and $y=0,...,H-1$ (8.2)

$$C(k)=\begin{cases}1/\sqrt{2} & \text{for } k=0\\ 1 & \text{otherwise}\end{cases}$$

where *W* and *H* is the width and height of the image segment, respectively. The *F*(*u*,*v*) is called the DCT coefficient, and the DCT basis is

$$\omega_{x,y}(u,v)=\frac{2C(u)C(v)}{\sqrt{H*W}}\cos\left[\frac{\pi(2x+1)u}{2W}\right]\cos\left[\frac{\pi(2y+1)v}{2H}\right]. \tag{8.3}$$

Then we can rewrite the inverse DCT to

$$f(x,y)=\sum_{u=0}^{W-1}\sum_{v=0}^{H-1}F(u,v)\omega_{x,y}(u,v) \quad \text{for } x=0,...,W-1 \text{ and } y=0,...,H-1. \tag{8.4}$$

To compare with the 8×8 DCT, we use an image segment whose height and width are both eight for an example, as shown in Fig. 8.1(a). Fig. 8.1(b) is its shape matrix by filling with one's in the position inside the boundary of the shape and filling with zeroes otherwise.

				75	96		
105	98	99	101	73	85	66	60
	100	97	89	94	87	64	55
		84	94	90	81	71	66
		93	86	94	81	70	
			86	86	81	72	
			98	97	78		
			105	104			

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

Fig. 8.1 (a) An arbitrary-shape image segment *f* and (b) its shape matrix.

Similar to the 8×8 DCT, we can get the *H*×*W* bases of the image segment *f*. We multiplying the *H*×*W* DCT bases shown in Fig. 4.3 by the shape matrix shown in Fig. 8.1(b) and the result is shown in Fig. 8.2. Because the point number *M* of is less than

H×*W*, we can know that the *H*×*W* bases are not orthogonal. Before we use the Gram-Schmidt process to reduce the bases to *M* orthogonal ones, we reorder the *H*×*W* bases by the zig-zag reordering matrix which is shown in Fig. 8.3. The reason to reorder is that the low frequency components concentrate on the left-top position and is more important to the high frequency components which concentrate on the right-bottom position. After reordering and the Gram-Schmidt process, we get the *M* orthogonal bases $\omega'_{x,y}$ for the image segment *f*, as shown in Fig. 8.4.

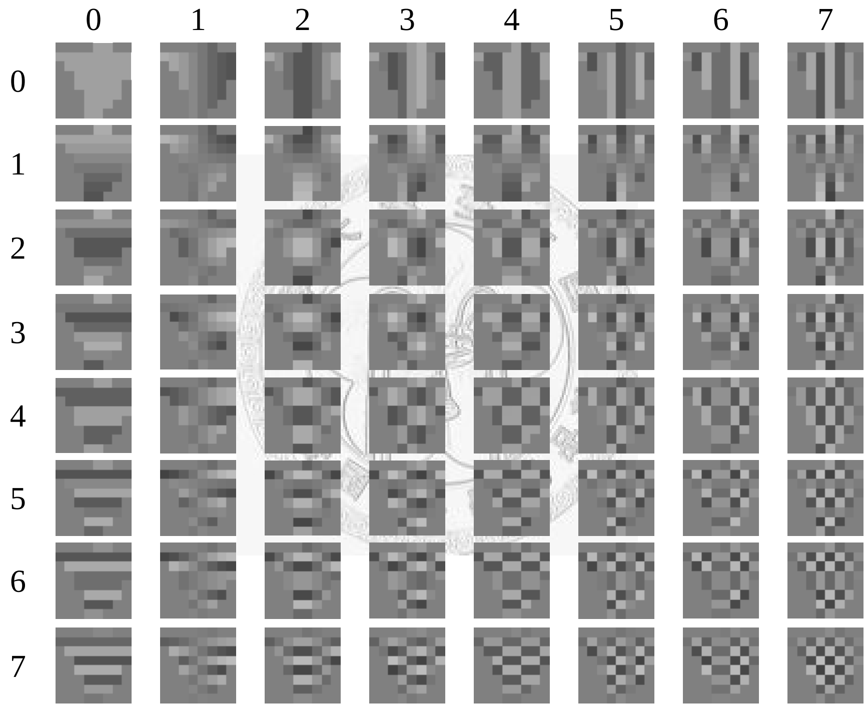


Fig. 8.2 The 8×8 DCT bases with the shape of *f*.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

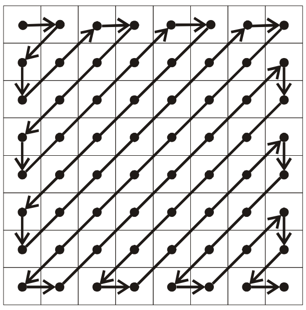


Fig. 8.3 Zig-zag reordering matrix.

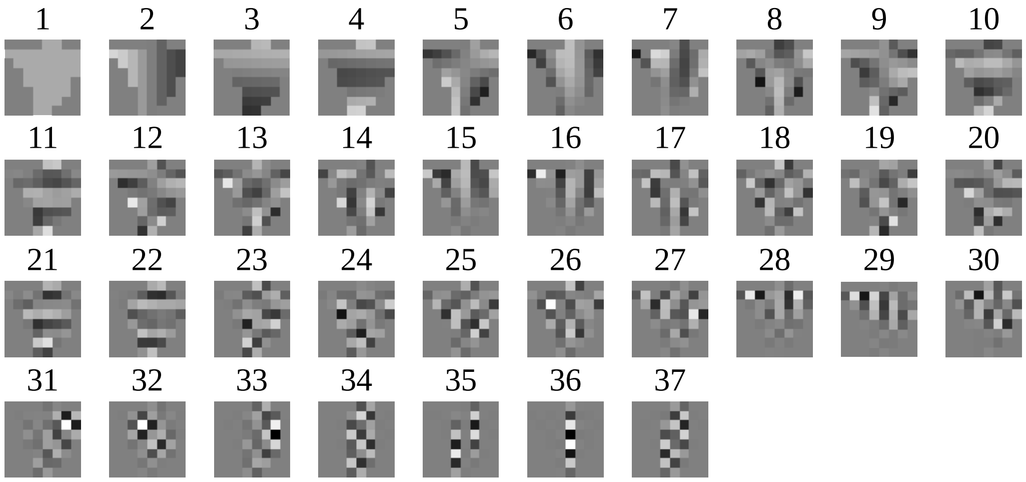


Fig. 8.4 The 37 arbitrary-shape orthonormal DCT bases $\omega'_{x,y}$.

Finally, we define the forward DCT for an arbitrary-shape image segment as

$$F(k)=\sum_{x=0}^{W-1}\sum_{y=0}^{H-1} f(x,y)\omega'_{x,y}(k) \quad \text{for } k=1,2,...,M \ , \tag{8.5}$$

and the inverse DCT for an arbitrary-shape image segment is

$$f(x,y)=\sum_{k=1}^{M} F(k)\omega'_{x,y}(k) \quad \text{for } x=0,1,...,W-1 \text{ and } y=0,1,...,H-1. \tag{8.6}$$

The M DCT coefficients of f are show in Fig. 8.5. Fig. 8.6 has six arbitrary-shape image segments and their DCT coefficients are shown in Fig. 8.7.

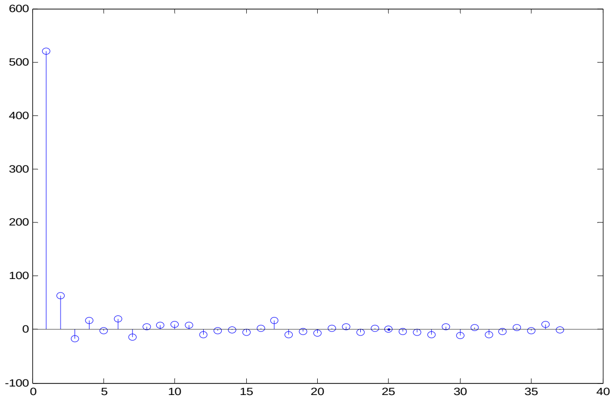


Fig. 8.5 The 37 arbitrary-shape DCT coefficients of f.

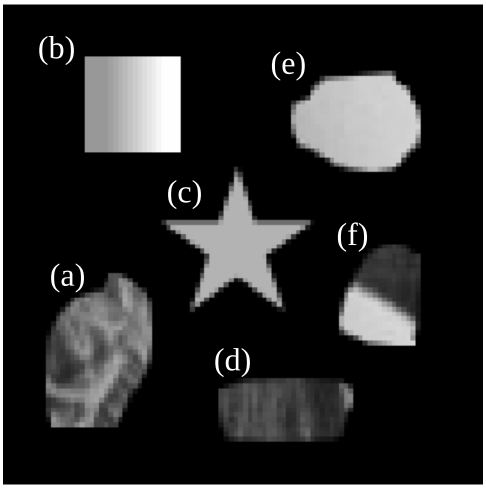


Fig. 8.6 An example of six arbitrary-shape image segments.

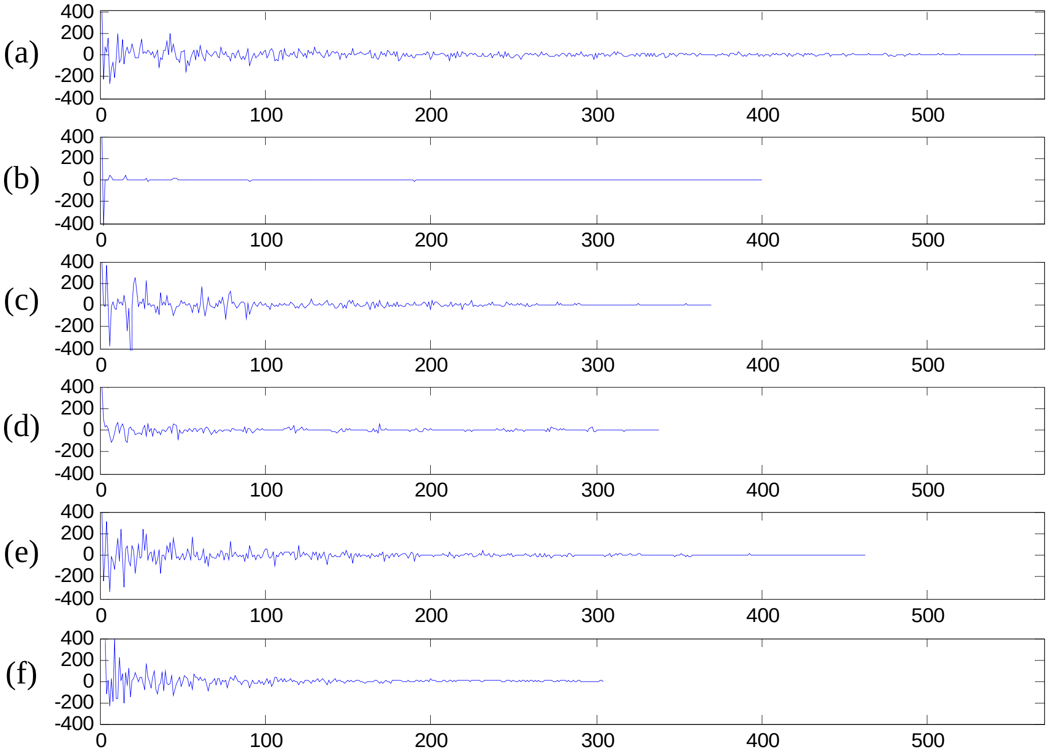


Fig. 8.7 The arbitrary-shape DCT coefficients of Fig. 8.6 (a)~(f).

8.2 Quantization of the DCT Coefficients

After transforming, we quantize the transformed coefficients to integers. On the 8×8 DCT coefficients, we divided them by a quantization matrix. But on the arbitrary-shape DCT coefficients, the length of coefficients is not fixed. Therefore, we define an unfixed quantization array $Q(k)$ as an increasing line:

$$Q(k) = Q_a k + Q_c, \quad \text{for } k = 1, 2, \ldots, M \ , \tag{8.7}$$

where the two parameters Q_a and Q_c are the slope and the intercept of the line respectively and M is the length of the DCT coefficients. Then each DCT coefficient $F(k)$ is divided by the corresponding quantization array $Q(k)$ and rounded to the nearest integer as:

$$F_q(k) = Round\left(\frac{F(k)}{Q(k)}\right), \quad \text{where } k = 1, 2, \ldots, M \ . \tag{8.8}$$

The parameter Q_c is important because it effects the quantization quantity of the DC term and the low frequency components whose value is usually greater than the high frequency components. The other parameter Q_a leads the quantization quantity more and more from the first DCT coefficient to the last one. It is because that the high frequency components of an image segment is not visible in human vision and can tolerate more distortion.

Now we quantize the DCT coefficients of the six image segments in Fig. 8.7. The quantization array $Q(k)$ is shown in Fig. 8.8 where the parameter Q_a is set to 0.06 and Q_c is set to 8. Then we quantize the DCT coefficients by (8.8). The result is shown in Fig. 8.9. We can see that the value of the quantized DCT coefficients is much smaller than the original ones. The other effect is that a lot of coefficients in the higher frequency part are reduced to zeroes.

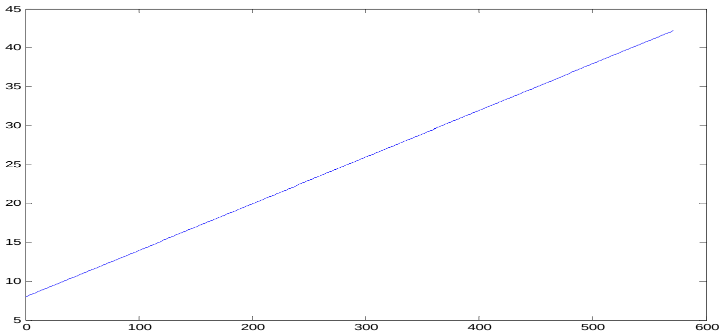


Fig. 8.8 Quantization array $Q(k)$ with $Q_a = 0.06$ and $Q_c = 8$.

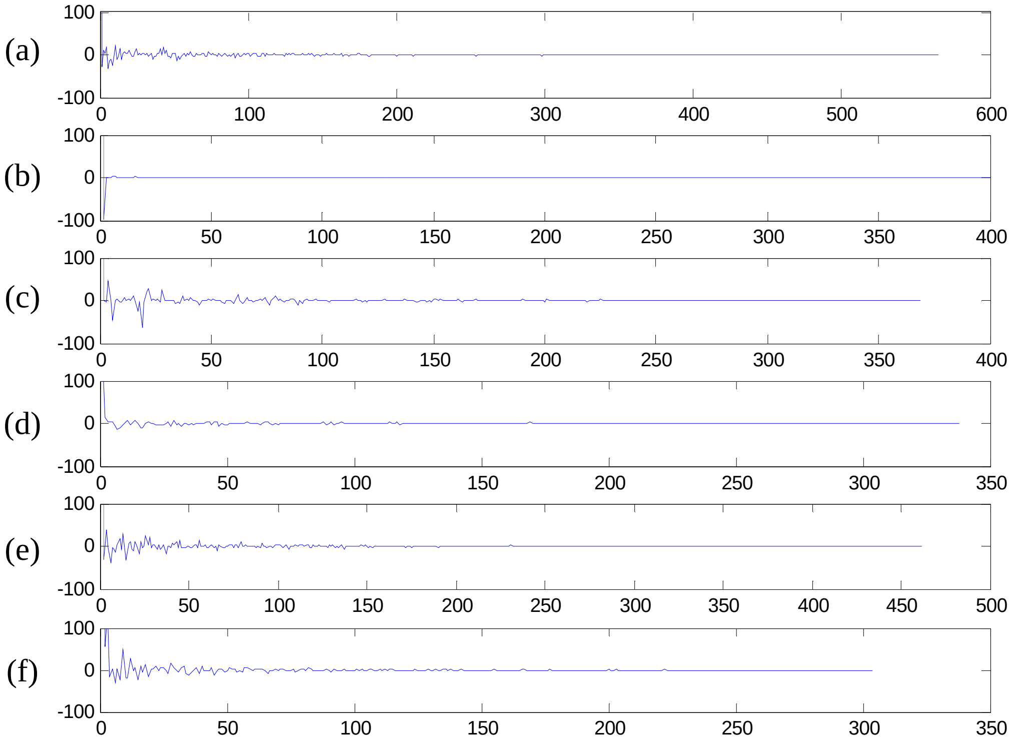


Fig. 8.9 The quantized DCT coefficients of Fig. 8.7(a)~(f).

8.3 Coding Technique of the Image Segment

In the final step, we encode the quantized coefficients to bit stream and combine with the code of boundary in the whole compression process. By the result of the quantization process, the quantized coefficients are serial integer numbers with a large value in the first place and have a lot of zeroes in the back. This characteristic is similar to the coefficients of the 8×8 DCT and we can use the same way to encode them. That is, we divide the first value and the other ones which are called the DC-term and the AC-term, respectively. We encode the difference of the present DC-term and the previous one and encode the AC-term by zero-run length coding and the Huffman coding. The detail process is described in Section 4.4.

After encoding the DCT coefficients of each image segment, we combine them directly to the bit stream of all image segments. Note that in the zero-run length coding, we truncate the successive zeroes in the end of the coefficients, and replace with an end-of-bit (EOB) symbol in this position. Therefore, we can divide the bit stream to each image segment by the EOB symbol in the decoding process. The diagram of the encoding and decoding process is illustrated in Fig. 8.10 and Fig. 8.11.

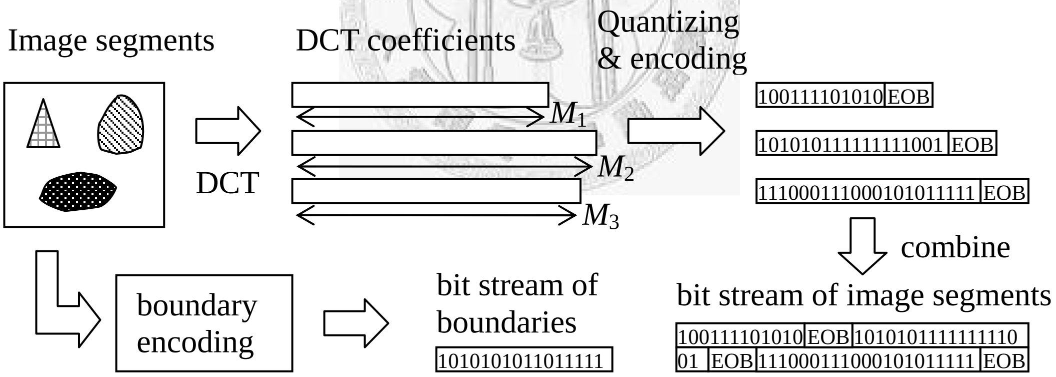


Fig. 8.10 Image segment encoder.

Note that we do not need to record the length *M* of DCT coefficients of each image segment because it can be got by the boundary. Therefore, in the decoding process, we decode the bit stream of boundaries first. Then we count the point number inside a boundary. The number is exactly the length *M* of DCT coefficients to the corresponding image segment. Further, we have to compute the DCT bases based on the shape of each image segment from the decoded boundaries. Finally, we get the reconstructed image segments by combining the DCT bases with the DCT coefficients as illustrated in Fig. 8.11.

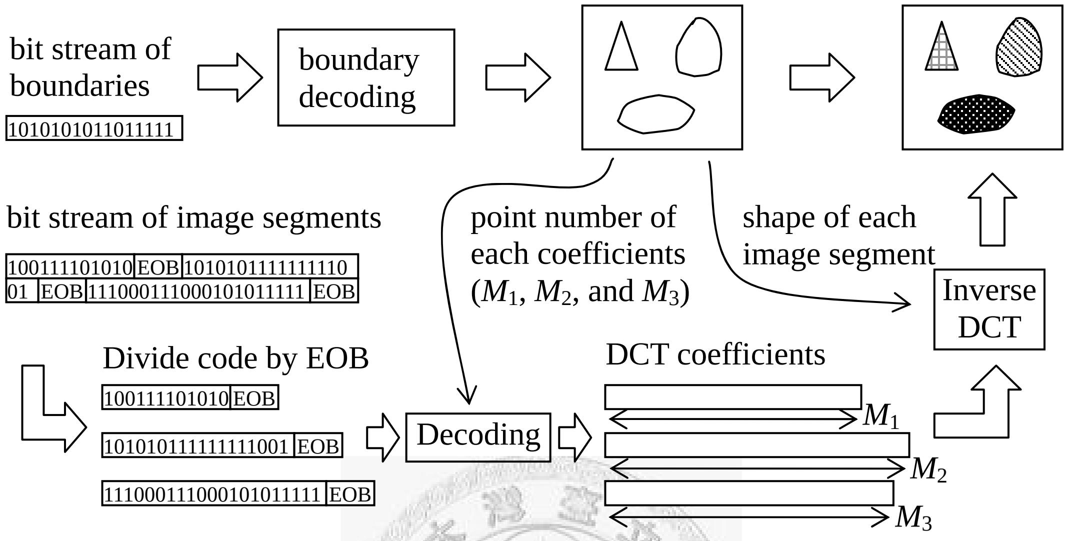
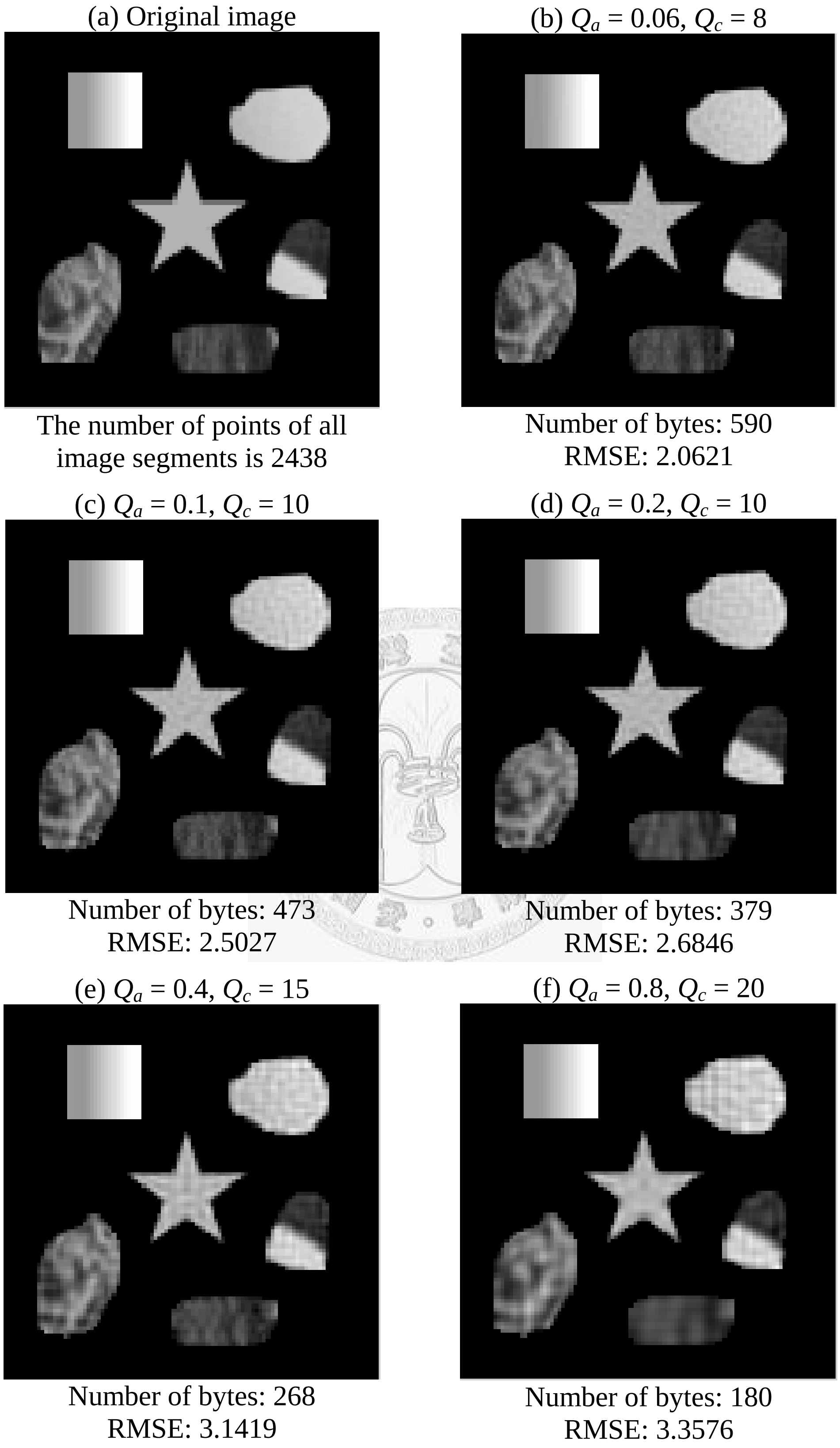


Fig. 8.11 Image segment decoder.

Fig. 8.12(a) is an example image with six image segments. Fig. 8.12(b)~(f) show five reconstructed images with different quantization parameters Q_a and Q_c. The number of bytes after encoding and the root mean square error (RMSE) with the original image are marked below the images. Note that the number of bytes does not include the bit stream of boundaries whose length is 157 bytes calculated in Section 6.3. In these figures, we can observe that when the quantization parameters are larger and larger, the texture in an image segment is smoother and smoother. The smooth effect in a complex color image segment is not visible in the human vision. But when the color in an image segment is almost identical, the reconstructed quantized image segment would produce some noise which is like spots or ripples. The effect is a major problem because it is clear in the human vision and we will solve it in the next section.



(a) Original image

The number of points of all image segments is 2438

(b) $Q_a = 0.06$, $Q_c = 8$

Number of bytes: 590
RMSE: 2.0621

(c) $Q_a = 0.1$, $Q_c = 10$

Number of bytes: 473
RMSE: 2.5027

(d) $Q_a = 0.2$, $Q_c = 10$

Number of bytes: 379
RMSE: 2.6846

(e) $Q_a = 0.4$, $Q_c = 15$

Number of bytes: 268
RMSE: 3.1419

(f) $Q_a = 0.8$, $Q_c = 20$

Number of bytes: 180
RMSE: 3.3576

Fig. 8.12 Number of Bytes and RMSE of the encoded image segments with different quantization parameters.

8.4 Improvement of the Boundary Region by Morphology

The reason of the noise to an identical color image segment is that its color has a lot of variation in the boundary region. When we compute the DCT, its high frequency coefficients would take more components. If we truncate its high frequency components, the noise which is like spots or ripples would appear.

To avoid this effect, we could segment the image more detailed. However, it will produce more boundary information that need to record. Since we know the variation is around the boundary region, we could divide the image segment into boundary region part and the internal part. Then we compute their DCT and encode them separately. Note that we do not need to record the boundary of the internal part because it can be calculated by the original boundary in the decoding process. Fig. 8.13 shows the diagram of the dividing process.

The way that we divide the image segment is by morphological erosion. For two binary image set *A* and *B*, the erosion of *A* by *B*, denote $A \ominus B$, is defined as

$$A \ominus B = \left\{ z \mid (B)_z \subseteq A \right\}, \tag{8.9}$$

where the translation of set *B* by point $z = (z_1, z_2)$, denoted $(B)_z$, is defined as

$$(B)_z = \left\{ c \mid c = b + z, \quad \text{for } b \in B \right\}. \tag{8.10}$$

In words, the erosion of *A* by *B* is the set of all points *z* such that *B* which is translated by *z* is contained in *A*. Base on the definition described above, we erode the shapes of image segments by a 5×5 disk-shape image and we get the shape of the internal region. Then we subtract it from the original shape and we get the shape of boundary region. The process is illustrated in Fig. 8.13.

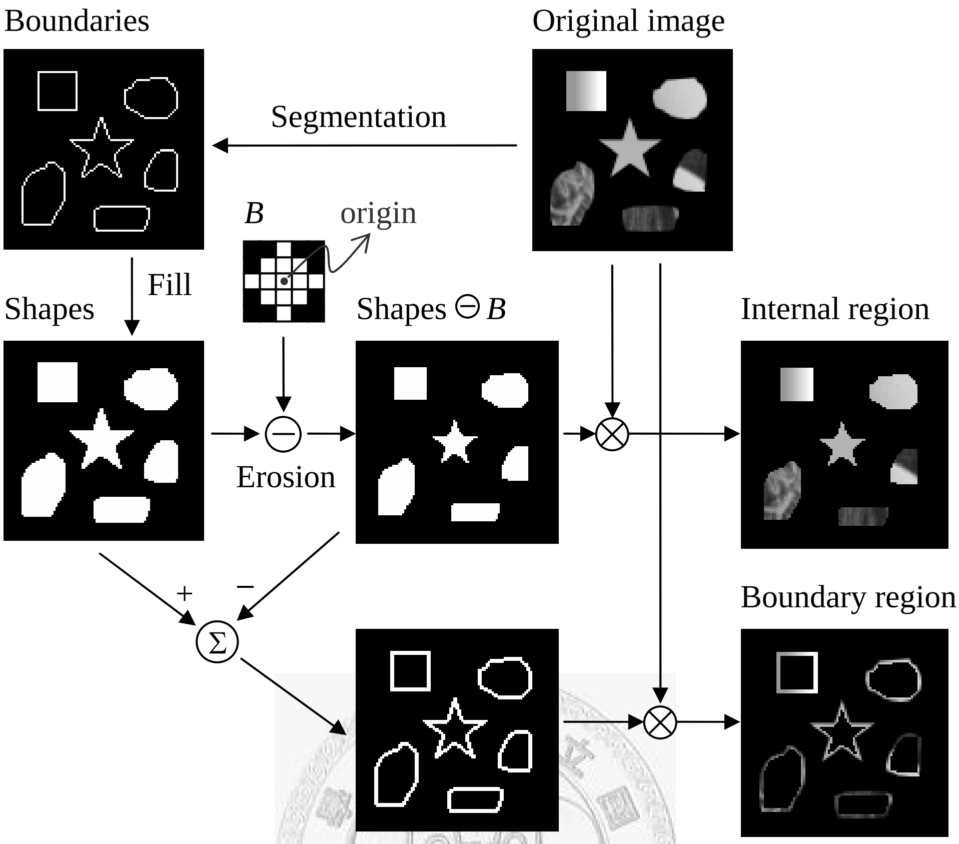
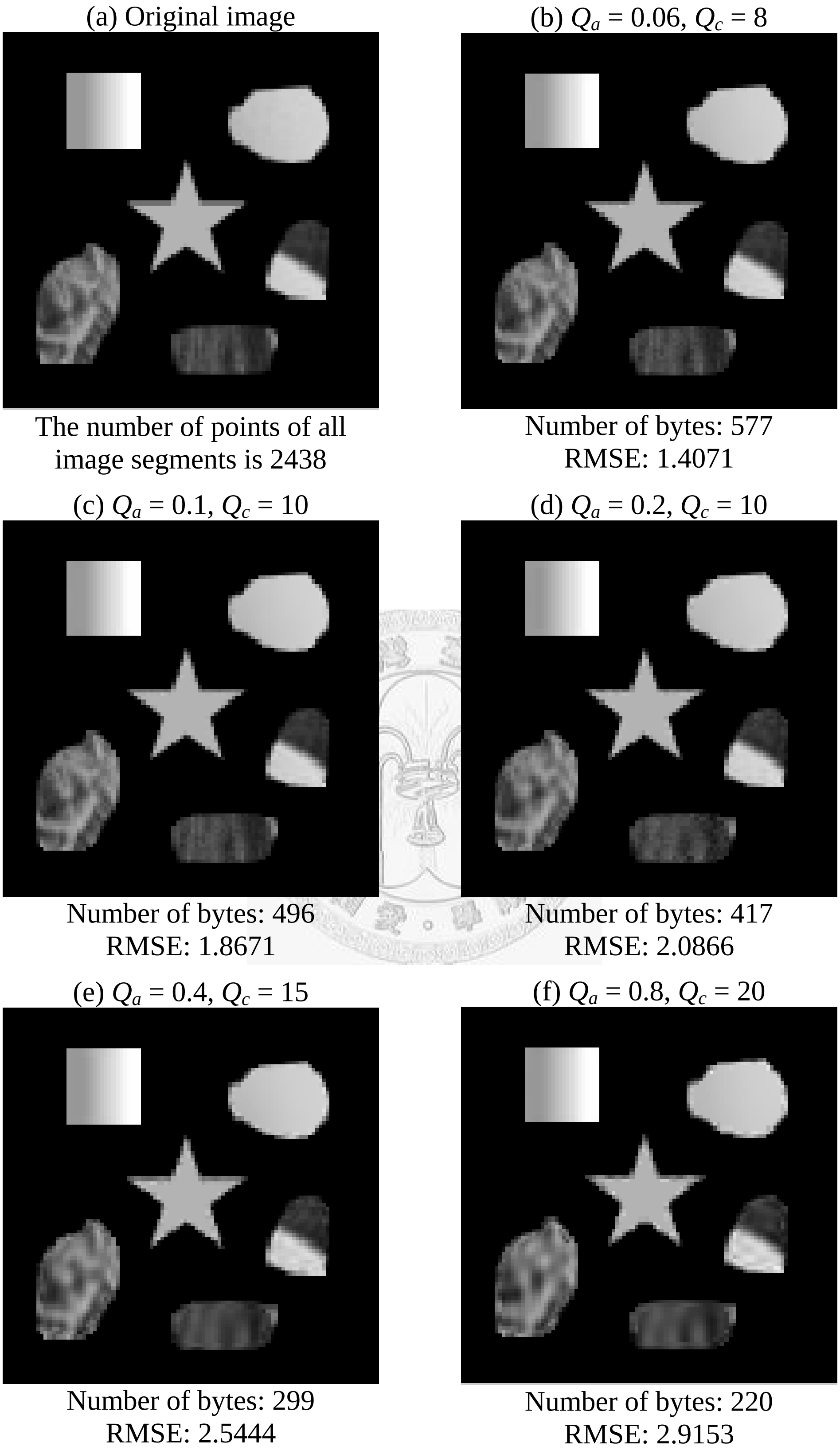


Fig. 8.13 Divide the image segment by morphological erosion.

Fig. 8.14 is the simulation result of the improved image segment compression with the same quantization parameter of Fig. 8.12. We can observe that the noise of the internal regions have been eliminated and the RMSE is much less than the original method. Although the number of bytes with the same quantization parameters is a little more, we can use larger quantization parameters with similar RMSE and reach higher compression ratio.



(a) Original image

The number of points of all image segments is 2438

(b) Q_a = 0.06, Q_c = 8

Number of bytes: 577
RMSE: 1.4071

(c) Q_a = 0.1, Q_c = 10

Number of bytes: 496
RMSE: 1.8671

(d) Q_a = 0.2, Q_c = 10

Number of bytes: 417
RMSE: 2.0866

(e) Q_a = 0.4, Q_c = 15

Number of bytes: 299
RMSE: 2.5444

(f) Q_a = 0.8, Q_c = 20

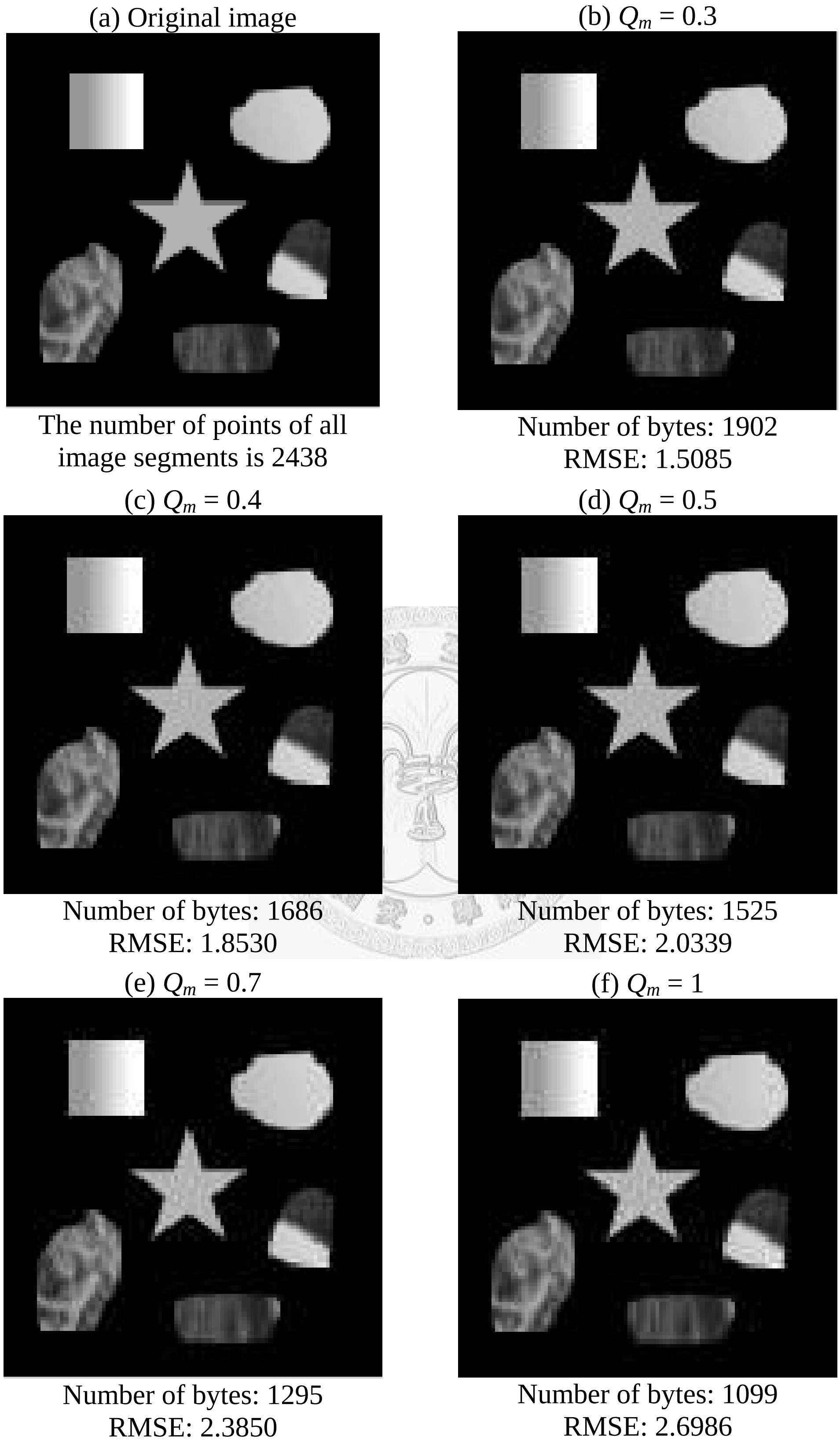
Number of bytes: 220
RMSE: 2.9153

Fig. 8.14 Number of bytes and RMSE of the encoded image segments with different quantization parameters in the improvement method.

8.5 Compare with the JPEG Standard

In this section, we compare the arbitrary-shape image compression method to the JPEG compression method. The JPEG compression process has been described in Chapter 2. We divide the image into 8×8 blocks and transform these blocks to frequency domain by DCT. Then we quantize these 8×8 DCT coefficients by the quantization matrix $Q(u,v)$ which is shown in Fig. 4.5(a). The method to modulate the compression ratio in the JPEG standard is to multiply a quantization parameter Q_m by the quantization matrix. Finally, we encode the quantized DCT coefficients by the same coding algorithm of the arbitrary-shape image compression.

Fig. 8.15(b)~(f) show the results of the decoded image with different quantization parameter Q_m. We can observe that the number of bytes is much larger than the number of bytes of the arbitrary-shape image compression which include boundaries and image segment information. That is because the JPEG method produce a lot of 8×8 blocks and many of them cross the background and the image segments. That causes its high frequency coefficients take more components and need more bytes to record. If we increase the quantization parameter, the high frequency components would be lost. Therefore, there are many obvious blocky and blurry artifacts around the boundary of the image segments.



(a) Original image

The number of points of all image segments is 2438

(b) $Q_m = 0.3$

Number of bytes: 1902
RMSE: 1.5085

(c) $Q_m = 0.4$

Number of bytes: 1686
RMSE: 1.8530

(d) $Q_m = 0.5$

Number of bytes: 1525
RMSE: 2.0339

(e) $Q_m = 0.7$

Number of bytes: 1295
RMSE: 2.3850

(f) $Q_m = 1$

Number of bytes: 1099
RMSE: 2.6986

Fig. 8.15 Number of bytes and RMSE of the encoded image segments with different quantization parameter in the JPEG method.

8.6 Conclusion

We implemented the arbitrary-shape image compression process. However, the modified DCT bases need to use Gram-Schmidt orthogonal process whose complexity is $O(n^2)$. If the number of points of an image segment is large, it would cost a lot of computational time. Therefore, we can segment the image more detail to avoid the number of point of an image segment been too large. On the other hand, if the number of points of an image segment is large, it means that its values of points are approximate and can not be segmented more detailed. Therefore, the number of bases can be chosen smaller than the dimension of the image segment to avoid n being too large in the Gram-Schmidt process.

We showed three kinds of results by different compression method. When we use the image which has pure background and some geometric figures, the result of JPEG method is poorer than the arbitrary-shape method because it can not adapt to the characteristic of the image. If we adopt the improved arbitrary-shape method, we can reach higher compression ratio with acceptable RMSE. Therefore, the arbitrary-shape image compression method is indeed excellent to compression an image.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

We discussed a lot of subjects about the segmentation-based image compression. The first subject is to segment an image with similar characteristic. Edge detection algorithm is one of the methods to segment an image. We proposed an adaptive method called the short response Hilbert transform which combines the traditional differential method and the Hilbert transform method. We also discussed many other ways to analyze and segment an image. The main object is to find a suitable segmented result to compress.

The basic image compression algorithm in JPEG standard was introduced. We modified its DCT algorithm, quantization algorithm, and coding algorithm in order to use them in our proposed methods. The next subject is how to record the boundaries of the image segments efficiently. We discussed some popular boundary descriptors and proposed two improved boundary descriptors. By the robust theory, we chose the modified Fourier descriptor to implement and encode in our experiment.

After segmenting the image and recording the boundaries, we compressed the image segments. We modified the DCT which can only transform an $N\times N$ block so it can transform any arbitrary-shape image segment. The modified arbitrary-shape DCT preserves the property that the preceding coefficients correspond to the low frequency components. Therefore we can encode them by the same way of the JPEG standard. In our experiment, the increased compression ratio can be up to twice more than that of the traditional JPEG compression method.

9.2 Future Work

Due to the effect of the new compression technique correlated to the characteristic of the images, we use only simple geometric image with black background in our experiments. To achieve a better compression ratio on various images using the proposed compression technique, we have to improve the effect of the image segmentation.

Although the compression ratio was significantly increased, we believe that it still has some room for improvement. In the boundary compression, the boundary segmentation still needs to be improved. The more accurate of the corner points are, the more efficient the boundary descriptor can be. In the arbitrary-shape image segment compression, we can find some other bases which can centralize the coefficients more or can be obtained without orthogonalization.

If we apply the compression technique to a color image rather than a gray level image, maybe we can design some method to record the three color component more efficiently due to the high correlation of the color values in an image segment. For example, we calculate the average color value of an image segment and then we translate each color value to the difference of the average color value.

REFERENCE

A. Digital Image Processing

[1] R. C. Gonzolez, R. E. Woods, *Digital Image Processing Second Edition*, Prentice Hall, New Jersey, 2002.

[2] R. C. Gonzolez, R. E. Woods, S. L. Eddins, *Digital Image Processing Using Matlab*, Prentice Hall, New Jersey, 2004.

[3] T. Acharya, A. K. Ray, *Image Processing Principles and Applications*, John Wiley & Sons, New Jersey.

[4] W. K. Pratt, *Digital Image Processing Third Edition*, John Wiley & Sons, Manhattan, 2002.

[5] R. M. Haralick and L. G. Shapiro, *Computer and robot vision volume I*, Addison-Wesley, New York, 1992.

B. Image Compression

[6] 酒井善則、吉田俊之 共著，白執善 編譯，“影像壓縮技術”，全華，2004。

[7] G. K. Wallace, “The JPEG Still Picture Compression Standard,” *Communications of the ACM*, vol. 34, issue 4, pp. 30-44, 1991.

[8] C. Cuturicu, “A note about the JPEG decoding algorithm,” available in http://www.opennet.ru/docs/formats/jpeg.txt, 1999.

[9] ITU-T Recommendation T.81, “Digital compression and coding of continuous-tone still images - Requirements and guidelines,” available in http://www.itu.int/rec/T-REC-T/e.

[10] The Independent JPEG Group, C source code of JPEG Encoder research 6b, 1998.

[11] B. E. Usevitch, “A Tutorial on Modern Lossy Wavelet Image Compression: Foundations of JPEG 2000,” *IEEE Signal Processing Magazine,* vol. 18, pp. 22-35, Sept. 2001.

C. Edge Detection

[12] J. J. Ding, S. C. Pei, J. D. Huang, G. C. Guo, Y. C. Lin, N. C. Shen, and Y. S. Zhang, “Short Response Hilbert Transform for Edge Detection,” CVGIP, 2007.

[13] E. Abdou and W. K. Pratt, "Quantitative Design and Evaluation of Enhancement/Thresholding Edge Detectors," *Proc. IEEE,* vol. 67, pp. 753-763, May 1979.

[14] E. Argyle, "Techniques for Edge Detection," *Proc. IEEE,* vol. 59, pp. 285-287, Feb. 1971.

[15] J. Canny, "Finding Edges and Lines in Images," Massachusetts Institute of Technology 1983.

[16] J. Canny, "A Computational Approach to Edge Detection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8, pp. 679-698, Nov. 1986.

[17] D. Demigny and T. Kamie, "A Discrete Expression of Canny's Criteria for Step Edge Detector Performances Evaluation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, pp. 1199-1211, Nov. 1997.

[18] L. Ding and A. Goshtasby, "On the Canny edge detector," *Pattern Recognition*, vol. 34, pp. 721-725, 2001.

[19] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York, 1973.

[20] W. Frei and C. Chen, "Fast Boundary Detection: A Generalization and a New Algorithm," *IEEE Trans. Computers*, vol. 26, 10, pp. 988-998, Oct. 1977.

[21] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," in *Pro-*

ceedings of The Fourth Alvey Vision Conference, Manchester, 1988, pp. 147-151.

[22] M. Hueckel, "An Operator Which Locates Edges in Digital Pictures," *J. Association for Computing Machinery,* vol. 18, pp. 113-125, Jan. 1971.

[23] R. Kirsch, "Computer Determination of the Constituent Structure of Biomedical Images," *Computers and Biomedical Research*, vol. 4, pp. 315-328, 1971.

[24] D. G. Macleod, "Comments on Techniques for Edge Detection," *Proc. IEEE,* vol. 60, p. 344, Mar. 1972.

[25] D. Marr and E. Hildrith, "Theory of Edge Detection," *Proc. Royal Society of London,* vol. B207, pp. 187-217, 1980.

[26] H. Moon, "Optimal Edge-Based Shape Detection," *IEEE Trans. Image Processing,* vol. 11, no 11, Nov. 2002.

[27] V. S. Nalwa and T. O. Binford, "On Detecting Edges," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 6, pp. 699-714, Nov. 1986.

[28] R. Nevatia and K. R. Babu, "Linear Feature Extraction and Description," *Computer Graphics and Image Processing,* vol. 13, pp. 257-269, Jul. 1980.

[29] P. Paplinski, "Directional Filtering in Edge Detection," *IEEE Trans. Image Processing,* vol. 7, pp. 611-615, Apr. 1998.

[30] M. S. Prewitt, "Object Enhancement and Extraction," in *Picture Processing and Psychopictorics,* B. S. Lipkin and A. Rosenfeld, Eds. New York: Academic Press, 1970.

[31] R. Rao and J. Ben-Arie, "Optimal Edge Detection Using Expansion Matching and Restoration," *IEEE Trans. Paitern Analysis and Machine Intelligence,* vol. 16, no. 12, Dec. 1994.

[32] G. Roberts, "Machine Perception of Three-Dimensional Solids," in *Optical and Electro- Optical Information Processing,* J. T. T. e. al., Ed. Cambridge, MA: MIT

Press, 1965, pp. 159-197.

[33] G. S. Robinson, "Color Edge Detection," *Proc. SPIE Symposium on Advances in Image Transmission Techniques,* vol. 87, Aug. 1976.

[34] G. S. Robinson, "Edge Detection by Compass Gradient Masks," *Computer Graphics and Image Processing,* vol. 6, pp. 492-501, Oct. 1977.

[35] L. Rosenthaler, F. Heitger, O. Kiibler, and R. v. d. Heydt, "Detection of general edges and key points," *Proc. 2nd European Conf: on Comp. Vis., Italy,* pp. 78-86, May 1992.

[36] V. Torre and T. A. Poggio, "On Edge Detection," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 8, pp. 147-163, Mar. 1986.

[37] D. Ziou and S. Tabbone, "Edge Detection Techniques An Overview," 1998.

D. Edge Detection Based on the Hilbert Transform

[38] W. Lohmann, D. Mendlovic, and Z. Zalevsky, "Fractional Hilbert transform," *Opt. Lett.,* vol. 21, pp. 281-283, Feb. 1996.

[39] M. Livadas and A. G. Constantinides, “Image Edge Detection and Segmentation Based on the Hilbert Transform,” *ICASSP*, vol. 2, pp.1152-1155, 1988.

[40] K. Kohlmann, “Corner Detection in Natural Images Based on the 2-D Hilbert Transform,” *Signal Processing*, vol. 48, no. 3, pp. 225-234, 1996.

[41] J. A. Davis, D. E. McNamara, and D. M. Cottrell, “Image Processing with the Radial Hilbert Transform: Theory and Experiments,” *Opt. Lett.*, vol. 25, no. 2, pp. 99-101, 2000

[42] S. C. Pei and J. J. Ding, “The Generalized Radial Hilbert Transform and Its Applications to 2-D Edge Detection (Any Direction or Specified Directions),” *ICASSP*, vol. 3, pp. 357-360, Apr. 2003.

[43] J. K. T. Eu and A. W. Lohmann, “Isotropic Hilbert Spatial Filtering“, *Opt. Commun.*, vol. 9, no. 3, pp. 257-262, Nov. 1973.

E. Segmentation

[44] R. M. Haralick and L. G. Shapiro, "Image Segmentation Techniques," *Computer Vision, Graphics, and Image Processing,* vol. 29, pp. 100-132, Jan. 1985.

[45] T. Kanade, "Region Segmentation: Signal vs. Semantics," *Computer Vision, Graphics, and lmage Processing,* vol. 13, Aug. 1980.

[46] E. M. Riseman and M. A. Arbib, "Computational Techniques in the Visual Segmentation of Static Scenes," *Computer Vision, Graphics, and lmage Processing,* vol. 6, pp. 221-276, Jun. 1977.

[47] S. W. Zucker, "Region Growing: Childhood and Adolescence," *Computer Vision, Graphics, and lmage Processing,* vol. 5, pp. 382-389, Sep. 1976.

[48] K. S. Fu and J. K. Mui, "A Survey on Image Segmentation," *Pattern Recognition,* vol. 13, pp. 3-16, 1981.

[49] N. R. Pal and S. K. Pal, "A Review on Image Segmentation Techniques," *Pattern Recognition,* vol. 26, pp. 1277-1294, 1993.

[50] J. S. Weska, "A Survey of Threshold Selection Techniques," *Computer Vision, Graphics, and Image Processing,* vol. 7, pp. 259-265, 1978.

[51] B. Sankur, A. T. Abak, and U. Baris, "Assessment of Thresholding Algorithms for Document Processing," *Proc. IEEE International Conference on Image Processing,* pp. 580-584, Oct. 1999.

[52] D. Comaniciu and P. Meer, “Mean Shift: A Robust Approach toward Feature Space Analysis,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 24, pp. 603-619, 2002.

[53] K. Fukunaga, L. Hostetler, “The estimation of the gradient of a density function, with applications in pattern recognition,” *IEEE Trans. Information Theory*, vol. 21, pp. 32-40, 1975.

[54] R. Unnikrishnan, C. Pantofaru, and M. Hebert, “Toward Objective Evaluation of Image Segmentation Algorithms,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, Jun. 2007.

F. Segmentation-Based Image Compression

[55] M. J. Biggar, O. J. Morris, and A. G. Constantinides, "Segmented-image coding: performance comparison with the discrete cosine transform," *IEEE Proceedings*, 1988.

[56] M. Kunt and M. Kocher, "Second-Generation Image-Coding," *Proceedings of the IEEE*, pp. 549-574, 1985.

[57] O.-J. Kwon and R. Chellappa, "Segmentation-Based Image Compression," *Optical Engineering,* vol. 32, pp. 1581-1587, Jul. 1993.

[58] F.-C. Leou and Y.-C. Chen, "A Contour-Based Image Coding Technique with Its Texture Information Reconstructed by Polyline Representation," *Signal Processing,* vol. 25, pp. 81-89, 1991.

[59] L. Shen and R. M. Rangayyan, "A Segmentation-Based Lossless Image Coding Method for High-Resolution Medical Image Compression," *IEEE Trans. on Medical Imaging,* vol. 16, no 3, pp. 301-307, Jun. 1997.

[60] J. Vaisey, "Image Compression with Variable Block Size Segmentation," *IEEE Trans. on Signal Processing,* vol. 40, no 8, pp. 2040-2060, Aug. 1992.

[61] M. Bi, S.-H. Ong, and Y.-H. Ang, “A Hybrid Shape-Adaptive Orthogonal Transform for Coding of Image Segments,” *IEEE trans. circuits syst. video technol*, Vol.

10, No. 8, Dec. 2000.

G. Boundary Description and Compression

[62] F. W. Meier, G. M. Schuster and A. K. Katsaggelos, “An Efficient Boundary Encoding Scheme Which Is Optimal In The Rate Distortion Sense,” *IEEE*, 1997.

[63] G. M. Schuster and A. K. Katsaggelos, “An Optimal Polygonal Boundary Encoding Scheme in the Rate Distortion Sense,” *IEEE Trans. on Image Processing*, vol. 7, no. 1, Jan. 1998.

[64] H. Wang, G. M. Schuster, A. K. Katsaggelos, and T. N. Pappas, “An Optimal Shape Encoding Scheme Using Skeleton Decomposition,” *IEEE*, 2002.

[65] M. S. Schmalz and G. X. Ritter, “Boundary Representation Techniques for Object-Based Image Compression,” Proceedings of SPIE, vol. 5208, 2004.

[66] T. A. El Doker and P. A. Mlsna, “Efficient Region Boundary Approximation Using Adaptive Smoothing and Second-Order B-Splines,” IEEE, 2002.

[67] A. J. Pinho, “Encoding of Closed Boundaries Using Transition Points,” IEEE, 1998.

[68] B. J. Mealy, “Region Boundary Generation and Compression,” IEEE, 2001.

[69] R. Tello, “Fourier Descriptors for Computer Graphics,” *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 25, no. 5, May 1995.

[70] H. Jia, M. Xie, “Improvement of Fourier Descriptor Using Spatial Normalization,” *IEEE Proceedings of ISCIT*, 2005.

[71] P.-C. Wang, K.-P. Lin, T.-S. Chen, and P.-T. Hung, “Sectional Contour Interpolation Using Fourier Descriptor,” *IEEE Proceedings of Medicine and Biology Society*, vol. 20, no. 2, 1998.

H. Arbitrary-Shape Region Compression

[72] W. K. Ng and Z. Lin, “A New Shape-Adaptive DCT for Coding of Arbitrarily Shaped Image Segments,” *ICASSP*, vol. 4, pp. 2115-2118, 2000.

[73] Shen, B. Zeng, and M. L. Liou, “Arbitrarily Shaped Transform Coding Based on A New Padding Technique,” *IEEE trans. circuits syst. video technol*, vol. 11, no. 1, Jan. 2001.

[74] B. Mi, C. W. Kuen, and Z. Z. Hang, “Discrete Cosine Transform on Irregular Shape for Image Coding,” *IEEE tencon*, 1993.

[75] S. F. Chang and D. G. Messerschmitt, “Transform Coding of Arbitrarily-Shaped Image Segments,” *Proceedings of the first ACM international conference on Multimedia*, pp. 83-90, Aug. 1993.

I. Other Technique

[76] J. Kasson and W. Plouffe, "An Analysis of Selected Computer Interchange Color Spaces," *ACM Trans. on Graphics*, vol. 11, no. 4, pp. 373-405, 1992.