COMPRESSION OF FINGERPRINTS BASED ON WAVELET PACKET DECOMPOSITION AND FRACTAL SINGULARITY MEASURES

By

Eric Jang

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

Thesis Advisor: W. Kinsner, Ph.D., P. Eng.

(xiii+117+A69+B7=) 206 pp

0-612-23354-5

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

COPYRIGHT PERMISSION

COMPRESSION OF FINGERPRINTS BASED ON WAVELET PACKET DECOMPOSITION AND FRACTAL SINGULARITY MEASURES

BY

ERIC JANG

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

Eric Jang © 1997

ABSTRACT

Fingerprint images are used not only for criminal records, but also for applications such as social welfare, pension benefits, identity cards, passports, motor vehicle licenses, voter registration, immigration, work permits, customs and other security systems. To meet these needs, fingerprints must be processed digitally. The size of fingerprint databases increases rapidly because more and more facilities rely on fingerprints. For example, there are 30,000 new fingerprint cards a day in the U.S.A. along. Consequently, the need to reduce the size of fingerprint storage has become very urgent.

A decompressed fingerprint image can be either classifed by *automatic fingerprint identification systems* (AFIS) or submitted in court as an evidence. The reconstructed image must keep the relations of the minutiae of a fingerprint for identification. *Lossless* compression techniques can reconstruct an image without any change, but with a poor compression ratio about 2:1. *Lossy* techniques can reach compression ratios of hundreds to one, but at an expense of a poorer quality of the reconstructed image.

This thesis develops two new techniques for grey-scale fingerprint image compression, *quadtree decomposition with multifractal analysis* (QDMA) and *wavelet packet with multifractal analysis* (WPMA). Considering the limitations of the *human vision system*, the QDMA and WPMA use a wavelet transform guided by a multifractal measure to obtain the best reconstructed image in terms of a higher *peak signal to noise ratio* (PSNR) at the lowest bit rate. The fingerprint images and the corresponding wavelet coefficients are considered to be an approximation of strange attractors and can be analyzed by their multifractality. Wavelets can not only provide the grouping of subband information and the highest compression for optimum bit allocation (quantization), but also an optimum synthesis (combination of subbands) by the inverse wavelet transform to achieve the highest image quality. Using the QDMA technique, the compression ratio can reach 13.95:1 with 28.06 dB PSNR, while the compression ratio of WPMA can exceed 17.7:1 with PSNR up to 28.57 dB. The QDMA and WPMA techniques can make the reconstructed image with good quality for identification or as an evidence in court cases.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. W. Kinsner, for providing this research topic and giving me his time and patience in the course of this thesis. I would also like to thank Dr. Kinsner for his improvements and editing of this thesis document.

I would like to thank Mrs. Suzane Henry for correcting the grammar in this thesis.

I would also like to acknowledge everyone in the Delta Research Group, past and present, including Ken Ferens, Armein Langi, Geoff Stacey, Warren Grieder, Tom Tessier, Rasekh Rifaat, Don Shaw, Fan Mo, Pradeepa Yahampath, Jizong Li. Their technical advice and friendship have been an invaluable contribution to this thesis.

We also thank Telecommunications Research Laboratories (TR Labs), Winnipeg, for help in mastering the fingerprint database on a CD-ROM.

This thesis is for my loving wife Amy, who gave her unconditional support and inspiration to this work. It is also dedicated to my beautiful daughters Ruby and Sophie.

Partial financial support of this work from the Natural Sciences and Engineering Research Council (NSERC) of Canada is also acknowledged.

TABLE OF CONTENTS

Abstract i
Acknowledgements ii
Table of Contents iii
List of Figures vi
List of Tables ix
List of Abbreviations x
List of Symbols xi

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 FINGERPRINT INTERPRETATION AND IMAGE COMPRESSION 4

2.1 Types of Fingerprints and Their Interpretation 5
2.2 A Model of Human Visual System 7
2.3 Data Compression 8
Compression Ratio 9
Compression Quality, PSNR 9
Information Measurement, Entropy 10
Data compression Techniques, Statistical Coding 11
Statistical Coding 11
2.4 Image Compression Techniques, Transform Coding 13
Quantization 15
Vector Quantization 19
Transform Coding 19
Linear Transformation 20
Characteristics of Transform Coefficients 21
Optimum Bit Allocation 26
Zonal Filters 38
2.5 Summary of Chapter 2 29

CHAPTER 3 WAVELET TRANSFORM AND PACKETS 30

3.1 Time-Frequency Decompositions 31
3.2 Wavelet Transform 32
3.3 Two Channel PR-QMF Bank 35
3.4 Wavelet from Filter Banks, DWT 38
Multiresolution Analysis 38
Dilation and Wavelet Equations 40
3.5 Wavelet Packets 42
3.6 Summary of Chapter 3 44

CHAPTER 4 FRACTAL AND MULTIFRACTAL DIMENSIONS ... 46

4.1 Fractals and Self-Similarity ... 47
Self-Similarity ... 47
Fractals ... 47
Hausdorff Dimension of a Fractal ... 51
4.2 Multifractals ... 52
Strange Attractor ... 53
Can an Image be Treated as a Strange Attractor ? ... 53
Multifractal Dimensions ... 58
Rényi Dimension ... 59
Mandelbrot Dimension ... 62
Mandelbrot Dimension of a Segmented Image ... 64
4.3 Review of Chapter 4 ... 66

CHAPTER 5 EXPERIMENTAL RESULTS AND DISCUSSION ... 67

5.1 Zonal Filtering ... 69
5.2 Direct Thresholding ... 71
5.3 Zerotree ... 75
5.4 Quadtree Decomposition with Multifractal Analysis (QDMA) ... 77
Quadtree Decomposition (QT) ... 78
Scalar Quantization ... 80
Optimal Bit Allocation (OBL) ... 80
Quantizer Design ... 85
Computing the Fractal Spectra of Wavelet Transform Coefficients ... 87
Derivation of αq and fq ... 87
Example ... 89
Experimental Results of using QDMA ... 92
5.5 Wavelet Packet with Multifractal Analysis (WPMA) ... 95
5.6 Alternative Technique ... 106
5.7 Review of Chapter 5 ... 107

CHAPTER 6 CONCLUSIONS AND RECOMMENDATIONS ... 108

6.1 Conclusions ... 108
6.2 Contributions ... 110
6.3 Recommendations ... 111

REFERENCES ... 112

APPENDIX A STRUCTURE OF PROGRAM AND SOURCE CODE IN C ... A-1

A.1 Structure Charts and Functional Description of QDMA and WPMA Applications ... A-1

A.2 Source Code of QDMA and WPMA in C A-5
A.3 Source Code of Zerotree Coding in C A-57

APPENDEX B MATLAB M-FILES B-1

B.1 MATLAB M-File for Discrete Wavelet Transform B-1
B.2 MATLAB M-File for Inverse Discrete Wavelet Transform B-3
B-3 MATLAB M-File for Zonal Filter B-6

LIST OF FIGURES

Fig. 2.1 A fingerprint card. 2
Fig. 2.2 Three major types of fingerprint: (a) arch, (b) loop, and (c)whorl. 6
Fig. 2.3 A gray-scale bar chart with 32 levels. 8
Fig. 2.4 Basic data compression block diagram. 8
Fig. 2.5 Huffman coding flowchart and example. 12
Fig. 2.6 Transform coding block diagram of a two dimensional image. 14
Fig. 2.7 Illustration of the deterministic nature of quantization error in zero-memory uniform quantization (step size $\Delta = 1$). 16
Fig. 2.8 Block diagram of transform coding (block quantization). 20
Fig. 2.9 (a) Geometrical interpretation for Eq. 2.29 and (b) new coordinate system. 23
Fig. 2.10 (a) A 256 sample scan line of a gray-scale image, (b) DCT of (a), (c) DWT of (a). 25
Fig. 2.11 Uniform quantization of coefficients θ_0 and θ_1 without overload. The identical step size Δ ensures identical error variances σ^2_{q0} and σ^2_{q1}. The differing numbers of quantization levels depends on the variance of the input data. 28
Fig. 2.12 Zonal filters for 2:1, 4:1, 8:1, 16:1 sample reduction. Bright areas are passbands, dark areas are stopbands. 29
Fig. 3.1 Time-Frequency plane showing resolution cells for STFT. 32
Fig. 3.2 Typical wavelet family in time and frequency domains. 33
Fig. 3.3 Time-frequency plane showing resolution cells for wavelet transform. 35
Fig. 3.4 Frequency responses of QMF. 36
Fig. 3.5 Two channel subband QMF bank. 37
Fig. 3.6 Multiresolution of filter banks. 40
Fig. 3.7 Mother wavelet of Daub4. 41
Fig. 3.8 The meaning of channel ABBB in (a) frequency decomposition of wavelet transform domain (wavelet packet) and (b) tree structured representation. 44
Fig. 4.1 Cantor set. 48
Fig. 4.2 Koch curve. 49
Fig. 4.3 Sierpinski gasket. 50
Fig. 4.4 Sierpinski carpet. 50
Fig. 4.5 r-mesh cubes, scale = $1/r$, $N_r = 19$. (After[Kins95]). 51
Fig. 4.6 Hénon strange attractor. (a) the entire attractor, (b)enlargement of portion shown by square in part (a), (c) another enlargement of squared portion in (b). 54
Fig. 4.7 The grayscale of each pixel of a digital image (scanned photograph) stands for the energy in a certain area (*e.g.*, a box) that is restricted by the resolution. 55
Fig. 4.8 Barnsley's fern image, generated by Eq. 4.5-4.8. The IFS begins from a starting image, (a) a square and the iterated images after (b) one

iteration, (c) two iterations, (d) five iteration, (e) 15 iterations and (e) 100 iterations. 57
Fig. 4.9 (a) Barnsley's fern image, (b) enlarged portion of the square in (a), (c) enlarged portion of the square in (b). In (c) we find that the image turns into points. 58
Fig. 4.10 Rényi dimension D_q of Hénon strange attractor. 62
Fig. 4.11 Mandelbrot dimension D_{MAN} of Hénon strange attractor. 64
Fig. 4.12 (a) Original *Lena* image, segmentation of (a) with grayscale range of (b) 2 to 63, (c) 64 to 127, (d)128 to 191, (e) 192 to 255, (f) Mandelbrot dimension D_{MAN} of *Lena* and its segmentations ((b) - (e)). 65
Fig. 5.1 (a) 512 x 512, 8 bpp fingerprint image, (b) 3D presentation of (a); note there is a wide white band at the bottom of (a), (c) DWT coefficients of (a) in 3D.. 68
Fig. 5.2 A basic image compression scheme by using zonal filter mask. 69
Fig. 5.3 Zonal filter masks, bright areas are passbands, dark areas are stopbands.. 69
Fig. 5.4 (a) Original image (512x 512, 8 bpp), reconstructed images of using (b) 4:1, (c) 8:1 and (c) 16:1 zonal masks in Fig. 5.3.. 70
Fig. 5.5 A basic image compression scheme by using thresholding WT coefficients.. 71
Fig. 5.6 Effects of % truncation on the (a) quality, (b) compression ratio for the images in Table 5.2. 73
Fig. 5.7 (a) original image (512x 512 8 bpp), reconstructed images (part of Table 5.2) of truncation (b) 69%, (c) 92%, (d) 94%, (e) 96% and (f) 99%. . . 74
Fig. 5.8 Zerotree (3 scales) parent-child dependencies of subbands. 75
Fig. 5.9 (a) Original image (512x 512 8 bpp), reconstructed images of various PSNR (b) 33.76 dB at 1.48 bpp, (c) 29.63 dB at 0.84 bpp and (c) 26.09 dB at 0.47 bpp. 76
Fig. 5.10 A basic image compression scheme by using QDMA. 78
Fig.5.11 Quadtree wavelet packet decomposition. 79
Fig. 5.12 Frequency support of QDMA subbands (L = 3). 80
Fig. 5.13 Uniform quantizer characteristic.. 85
Fig. 5.14 Multifractal spectra of quadtree decomposition wavelet packet (L = 1) coefficients. The corresponding fingerprint image is shown in Fig. 5.15 (a).. 90
Fig. 5.15 (a) Original image (8 bpp), reconstructed image using quadtree decomposition (b) DUL, DUR, DLL, DLR (PSNR: 48.54 dB), (c) DUL (PSNR 30.19 dB), (d) DUL, DLL (PSNR: 32.44 dB), (e) DUL, DUR (PSNR: 31.75 dB), (f) DUL, DLR (PSNR: 31.61). . . . 91
Fig. 5.16 Multifractal spectra of quadtree decomposition wavelet packet (L = 3) coefficients. The fingerprint image is shown in Fig. 5.17 (a). 92
Fig. 5.17 (a) Original image (8 bpp), reconstructed image using quadtree decomposition (b) L3 + L2 + L1 (PSNR: Inf dB), (c) L3 + L2 (PSNR 29.78 dB), (d) L3 - L3DLL + L2 + L1 (PSNR 28.92),

(e) L3 (PSNR 23.05), (f) L3 + L2 - L2DLR (PSNR 28.35). 93
Fig. 5.18 Original image, 512 X 512, 8bpp (above), reconstructed image, L3+L2-L2DLR, using OBL quantization and lossless entropy coding, compression ratio 13.95:1 (0.57 bpp) at PSNR 28.06 dB (below). 94
Fig. 5.19 A basic image compression scheme by using WPMA. 95
Fig. 5.20 Frequency support of WPMA subbands (L = 5).. 96
Fig. 5.21 Multifractal spectra wavelet packet (WP) coefficients for all bands and subband groups I, II, III and IV. The fingerprint image is shown in Fig. 5.17 (a).. 99
Fig. 5.22 Multifractal spectra of coefficients for subband groups I, I1, I2, I3 and I4. 99
Fig. 5.23 Multifractal spectra of coefficients for subband groups I2, I21, I22, I23 and I24. 100
Fig. 5.24 Multifractal spectra of coefficients for subband groups 19, 20, 21 and 22. 100
Fig. 5.25 Multifractal spectra of coefficients for subband groups 27, 28, 29 and 30. 101
Fig. 5.26 Multifractal spectra of coefficients for subband groups 31, 32, 33 and 34. 101
Fig. 5.27 Multifractal spectra of coefficients for subband groups I3, I31, I32, I33 and I34. 102
Fig. 5.28 Multifractal spectra of coefficients for subband groups 35, 36, 37 and 38. 102
Fig. 5.29 Multifractal spectra of coefficients for subband groups 39, 40, 41 and 42. 103
Fig. 5.30 Multifractal spectra of coefficients for subband groups 47, 48, 49 and 50. 130
Fig. 5.31 Frequency spectrum of WSQ wavelet packet, shaded subbands will not be used for inverse wavelet transform and calculated for OBL. . . 104
Fig. 5.32 Original image, 512 X 512, 8bpp (above), reconstructed image using WPFA, OBL quantizer and lossless entropy coding for the shading subbands shown in Fig. 5.31, compression ratio 17.7:1 (0.4519 bpp) at PSNR 28.57 dB (below). 105
Fig. 5.33.(a) Original image, 256 x 256, 8 bpp, reconstructed image using VQ of (a) with block size 2 x 2, (b) codebook size 256, compression ratio 3.24:1 (2.47 bpp) at PSNR 35.02 dB, (c) codebook size 128, compression ratio 3.6:1 (2.22 bpp) at PSNR 33.26 dB, (d) VQ the wavelet packet (WSQ) coefficients with block size 2 x 2 and codebook size 256, compression ratio 3.24:1 (2.47 bpp) at PSNR 28.07 dB.. 106

LIST OF TABLES

Table 2.1 Types of fingerprint. 6
Table 4.1 D_H of the Cantor set, Koch curve, Sierpinski gasket and Sierpinski carpet. 52
Table 5.1 Experimental results of using the zonal mask method.. 70
Table 5.2 Experimental results of using direct thresholding method. 72
Table 5.3 Experimental results of using zerotree method. 76
Table 5.4 Subband group definition for wavelet packet. 97
Table 5.5 The effect on of reconstructed quality for subband group I21, I23 and I24. 98

LIST OF ABBREVIATIONS

AFIS	automatic fingerprint identification system
bpp	bits per pixel
CR	compression ratio
DCT	discrete cosine transform
DFT	discrete Fourier transform
DLL	decomposition lower left
DLR	decomposition lower right
DST	discrete sine transform
DUL	decomposition upper left
DUR	decomposition upper right
DWT	discrete wavelet transform
FBI	Federal Bureau of Investigation
FFT	fast Fourier transform
FIR	finite impulse response
FT	Fourier transform
HVS	human vision system
IFS	iterated function system
IWT	inverse wavelet transform
LBG	Linde, Buzo and Gray
NIST	National Institute of Standards and Technology
OBL	optimal bit allocation
PDF	probability density function
PSNR	peak signal to noise ratio
QDMA	quadtree decomposition with multifractal analysis
QDWT	quadtree decomposition wavelet transform
QMF	quadrature mirror filter
QT	quadtree decomposition
SNR	signal to noise ratio
SQ	scalar quantization
STFT	short-time Fourier transform
SWT	symmetric wavelet transform
VQ	vector quantization
WP	wavelet packet
WT	wavelet transform
WPMA	wavelet packet with multifractal analysis
WSQ	wavelet scalar quantization
1-D	1 dimension
2-D	2 dimension
3-D	3 dimension

LIST OF SYMBOLS

Variable

a	Scalar parameter of WT
$\hat{a}$	Quantized value
A_k	Weight factor of scalar quantizer
b	Shift parameter of WT
C	Parameter of quantizer
D_C	Correction dimension
D_E	Euclidean dimension
D_F	Fractal dimension
D_H	Hausdorff dimension
D_I	Information dimension
D_{MAN}	Mandelbrot dimension
D_q	Rényi dimension
Er_q	Quantization error
$F(\omega)$	Fourier transform of $f(t)$
$g(t)$	Prototype function in STFT
$g(k)$	Lowpass QMF coefficient
$G(\omega)$	Fourier transform of $g(t)$ coefficient
$h(k)$	Lowpass QMF
H	Entropy
k	Number of bits per symbol in the original data
k	The kth transform coefficient, quantizer, quantized transform coefficient, or quantization step
L	The number of quantization steps
L	Decomposition level
Ii	The information of the ith symbol
$I(i, j)$	Pixel of an image at position (i, j)
M	Dimension of an image, horizontal or vertical
n	Number of bits in an encoded codeword
n	Number of discrete gray-levels of an image
N	Dimension of an image, horizontal or vertical
n_j	The frequency intersected by a fractal of the jth vel
Nr	Number of boxes required to cover an fractal
p_i	The probability of the ith symbol
$p(x)$	Probability density function (PDF)
p	Number of symbols in the original data
$p_k(i, j)$	Quantized index, maps from location (i, j)
q	Number of encoded codewords produced by the compressor
$Q(x)$	Quantization of x

Q_k	quantization width
R	The average bit of the transform coefficients
R	Redundancy
R	Real number
R_c	Compression ratio
R_k	The bit assignment for the *k*th transform coefficient
T	Threshold
$v(k)$	The *k*th DCT coefficient
V	Maximum range of x
V_i	Multiresolution subspace at level i
W_i	Multiresolution subspace at level i
$WP_n(t)$	Wavelet packet basis functions (scaling and wavelet functions)
x_i	The *i*th original pixel
x	A zero-mean random variable (input of a quantizer)
$\hat{x}_i$	The *i*th reconstructed data/pixel
$x(t)$	Amplitude of signal at time t
$y(t)$	A finite dicrete number to present $x(t)$
y	Output of $Q(x)$ (output of a quantizer)
Z_k	Width of zero bin
$\alpha(k)$	The parameters of DCT
$\alpha_q(\alpha)$	Hölder exponent
β	Loading factor, $\beta = 1/\gamma$
β_T	Bandwidth of $g(t)$
β_ω	Bandwidth of $G(\omega)$
Δ	Quantization step
Δf_i	Difference between subspaces V_i at level i
σ_j^2	The variance of *j*th quantizer
σ_q^2	Variance of quantization errors
σ_r^2	The reconstruction error variance
$\sigma_{\theta k}^2$	Variance of transform coefficient $\theta(k)$
σ_{qk}^2	The quantization error variance of the *k*th quantizer
σ_x^2	The variance of x's
$\theta(k)$	The *k*th transform coefficient
τ	Time shift
μ_k	Mean

$\psi(t)$	Mother wavelet
$\Psi(\omega)$	Fourier transform of $\psi(t)$
$*$	Complex conjugate
$\oplus$	The direct sum
$\lceil \circ \rceil$	Round number to the next larger integer
$\lfloor \circ \rfloor$	Round number to the next lower integer
$\backslash$	Difference operator

Matrix

$U_i (U)$	An $i \times i$ unitary matrix
$U_i^* (U^*)$	The conjugate of U_i
U^H	The *Hermitian* transpose of U
$V_{ij} (V)$	The decoded transform block with i rows by j columns
$X_{ij} (X)$	Original transform block with i rows by j columns
Y_{ij}	Reconstructed block with i rows by j columns
Θ	The transformed coefficients

CHAPTER 1

INTRODUCTION

Fingerprint images are used not only for criminal records, but also for applications including social welfare, pension benefits, identity cards, passports, motor vehicle licenses, voter registration, immigration, work permits, customs and other security systems. To meet these needs, fingerprints must be processed digitally. The size of fingerprint databases increases rapidly because more and more facilities rely on fingerprints.

Before the FBI Identification Division was established in 1924, 810,000 fingerprint cards were collected by the National Bureau of Criminal Investigation [Unit87]. By 1992 this collection had expanded into 25 million cards [HoPr92]. To store a fingerprint card in an electronic format, the FBI follows the image capture standard established by the National Institute of Standards and Technology (NIST). A fingerprint card requires 9.8 MB of memory space (39 square inches scanning size with a resolution of 500 dots per inch, dpi, and 8 bits per pixel, 8 bpp). To store 25 million fingerprint cards requires an astonishing 245,000 GB of memory space. The purpose of this thesis is to find a new and better image compression technique for fingerprints.

The quality requirement for a decompressed fingerprint image is higher than for other images (such as *Lena*) because the reconstructed image can be identified and submitted in court as a evidence. The relationship of the minutiae of a fingerprint is the key for identification. That means the reconstructed image must reproduce the minutiae clearly. The objective of this thesis is to maximize the compression ratio, while minimizing the redundancy of the decompressed image.

Considering the limitations of the *human visual system* (HVS), we believe that transform coding techniques based on entropy reduction can reach our goal of the best reconstructed quality with the highest compression ratio. There are many transform coding techniques. In this thesis, we focus on the wavelet coding technique because every function in a wavelet basis is a dilated and translated version of one prototype (mother) wavelet. That allows local transformation to be carried out more successfully than other techniques such as *discrete sine transform* (DST), *discrete cosine transform* (DCT) and *discrete Fourier transform* (DFT) [Kins91]. That is the reason why *wavelet scalar quantization* (WSQ), the FBI fingerprint compression standard, selects the wavelet transform instead of others.

In this thesis, we use the *iterated function system* (IFS) concept to explain that images (spatial signals) are to be analyzed from the point of view of strange attractors [Kins95]. The coefficients of the wavelet transform (a projection from R^2 to L^2) can also be analyzed as a strange attractor.

We say that $f(\alpha)$ is the fractal Mandelbrot dimension (D_{MAN}) of the α subset [Kins94]. The Hölder exponent α (singularity of a fractal) is analogous to the energy, while $f(\alpha)$ is analogous to the entropy as a function of energy [Kins95]. Since the wavelet transform coefficients can be analyzed as a strange attractor, we can use the multifractal spectrum ($f(\alpha)$) to analyze the entropy of the wavelet transform singularity (*i.e.*, Hölder exponent α of the wavelet transform coefficients).

Based on the fact that the larger the wavelet coefficient is, the more important it is for the quality of the reconstructed image, we can simply regard those small wavelet coefficients as redundancies. However, the Gibb's phenomenon [Anso93] [OpWY75] becomes more evident while increasing the compression ratio. The reason is that the same magnitude of the transform coefficients does not contribute equally to the quality of the reconstructed image.

In this thesis, we propose two new techniques, *quadtree decomposition with multifractal analysis* (QDMA) and *wavelet packet with multifractal analysis* (WPMA), which can solve the problem of determining the redundant subbands of wavelet decompositions. These new techniques can improve the accuracy of bit allocation and reduce the quantization error. Thus, the compression ratio can be increased without lowering the quality of decompressed images.

There are five experiments in this thesis. The first three experiments (zonal masks, direct thresholding and zerotree [Shap93]) are based on the idea that the larger the transform coefficients the more important for the reconstructed image they are. In the zerotree technique, the emphasis is on the importance of different subbands, *i.e.*, the parent-children relationship (explained in Ch. 5). But the parent-child relationship cannot tell which subband is more important than others. Applying the multifractal spectrum analysis to each subband, we can easily find the pruned subbands and achieve the best grouping of subbands for bit allocation and synthesis of the inverse wavelet transform.

There are several other lossless and lossy techniques for fingerprint image compression, including differential data compression (lossless) [SaAu92], B-spline function representation (lossy) [CGTL92], and the WSQ. The compression ratio varies from 1.5:1 (lossless) to 20:1 (WSQ). Our QDMA can reach a compression ratio (CR) of 13.95:1 (0.5734 bpp) at 28.06 peak signal to noise ratio, PSNR, while the WPMA can achieve CR of 17.7:1 (0.4519 bpp) with 28.57 dB PSNR.

This thesis is organized into six chapters. Chapter 2 provides a general introduction to fingerprint interpretation and data compression. Entropy coding, transform coding, bit allocation and quantization techniques are discussed in this chapter. Chapter 3 presents the theory of the wavelet transform. Chapter 4 describes the theory of fractals and multifractals. We develop an *iterated function system* (IFS) image to explain the greyscale images as strange attractors. With this approach, we can use the multifractal measures for wavelet transform coefficients. Chapter 5 presents experimental results from five experiments. The first three experiments: *zonal filtering*, *direct thresholding*, and *zerotree coding* use either spatial location oriented or magnitude oriented techniques to find the best coefficients. Unfortunately, these techniques cannot reach our goal, higher PSNR with lower bit-rate. We propose a new technique, to solve the problem. Two experiments (QDMA and WPMA) are used to apply the new technique. Finally, conclusions, contributions and recommendations are presented in Chapter 6

CHAPTER 2

FINGERPRINT INTERPRETATION AND IMAGE COMPRESSION

A fingerprint card (Fig. 2.1) records one of the biological properties of a person. The *a (A)* and *t (T)* in Fig. 2.1 stand for ***arch*** and ***tented arch*** respectively, and the capital letter is reserved for the index fingers [Chap41]. To file a fingerprint card efficiently, the classification of fingerprints is necessary. The dots, bifurcations, islands (hollow circles and ovals), points where ridges suddenly end, and extremely short ridges are the essential characteristics for fingerprint classification and evidence for courts [Unit78]. For fingerprint compression, the reconstructed image must reproduce these minutiae clearly.

Compression techniques for fingerprints can be classified as lossless or lossy. Lossless techniques remove the redundancy which can be added back, so the decompressed data and the original data are identical. The disadvantage of lossless techniques is the low compression ratio, usually in the range of 2:1; for example, the *differential data compression* method can achieve a compression ratio 1.5:1 [SaAn92]. Lossy techniques remove entropy from the data and can achieve higher compression ratio; for example, the *wavelet scalar quantization* (WSQ), the FBI standard for fingerprint image compression, uses entropy reduction and can achieve a 20:1 (average) compression ratio at the expense of the quality of decompressed images.

Considering the limitations of the *human visual system* (HVS) and the quality requirement for a decompressed fingerprint image, we believe that lossy (entropy reduction) techniques can reach our goal (the best reconstructed quality with the highest compression ratio). There are many lossy techniques [Kins91]. In this chapter, we focus on transform coding techniques.

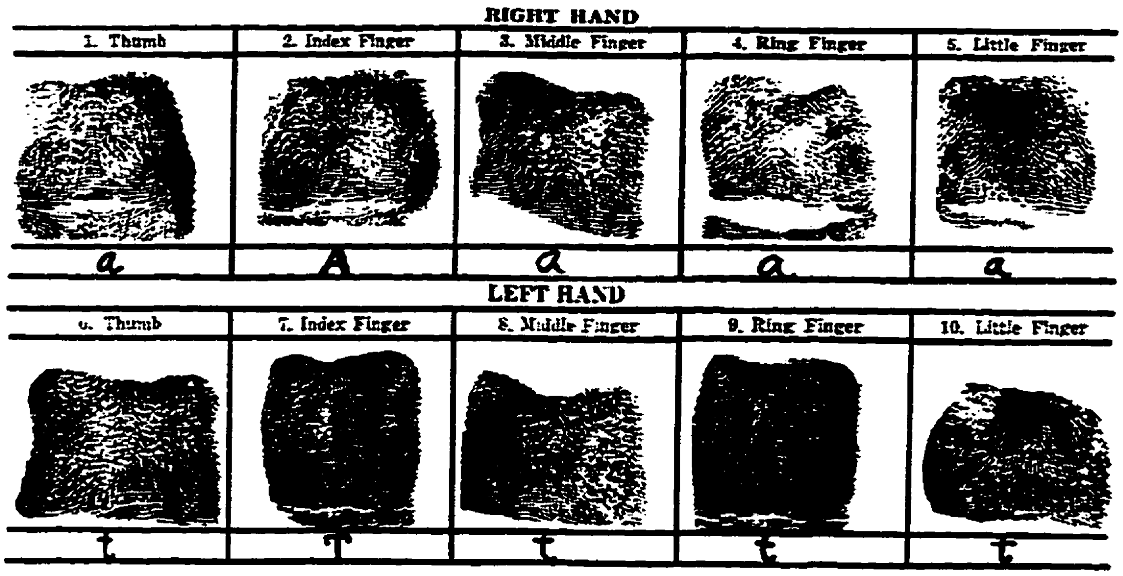


Fig. 2.1. A fingerprint card.

2.1 Types of Fingerprints and Their Interpretation

A fingerprint is the mark that the ridges of skin on fingers and thumbs leave on objects touched. Our fingerprints are perfectly formed seven months before we are born. The fingerprint is unique. In this world of approximately four and one half billion humans, each with ten fingerprints, there is not one fingerprint exactly like another. Even twins who look exactly alike have different fingerprints [Darr77].

Fingerprints may be resolved into the following three large general groups of types: *loop*, *arch* and *whorl* [Chap41]. Each group bears the same general characteristics or family resemblance. The types may be further divided into sub-groups by means of the smaller differences existing between the types in the same general group. Table 2.1 lists the groups and sub-groups.

Statistical data show that *loops* occur in about 65 percent of all fingerprints, *whorls* in about 30 percent, and *arches* in the remaining 5 percent [Chap41].

Table 2.1. Types of fingerprint.

ARCH	LOOP	WHORL
Plain arch	Radial loop	Plain whorl
Tented arch	Ulnar loop	Central pocket loop
		Double loop
		Accidental whorl

We shall first review a few technical terms used in fingerprint work. Figure 2.2 is a simple sketch to show the character of the three fingerprint types. More fingerprint images are included in a database [JaKi97]. A pattern area is enclosed by type lines. The pattern area can have *cores*, *deltas*, and *ridges* which are used in classifying fingerprints. Type lines may be defined as the two innermost ridges which start parallel, diverge, and surround or tend to surround the pattern area. The thick lines A and B in Fig. 2.2 (b) are the type lines. *Delta* is the first ridge or part of a ridge nearest the point of divergence of the two type lines. The *core* is placed upon or within the innermost sufficient recurve. It is important to concern ourselves with the *core* of the *loop* type only. For *loops* and *whorls*, the pattern area is used for classification and identification. On the other hand, *arches* or *tented arches* have no no particular boundary to define the pattern area.

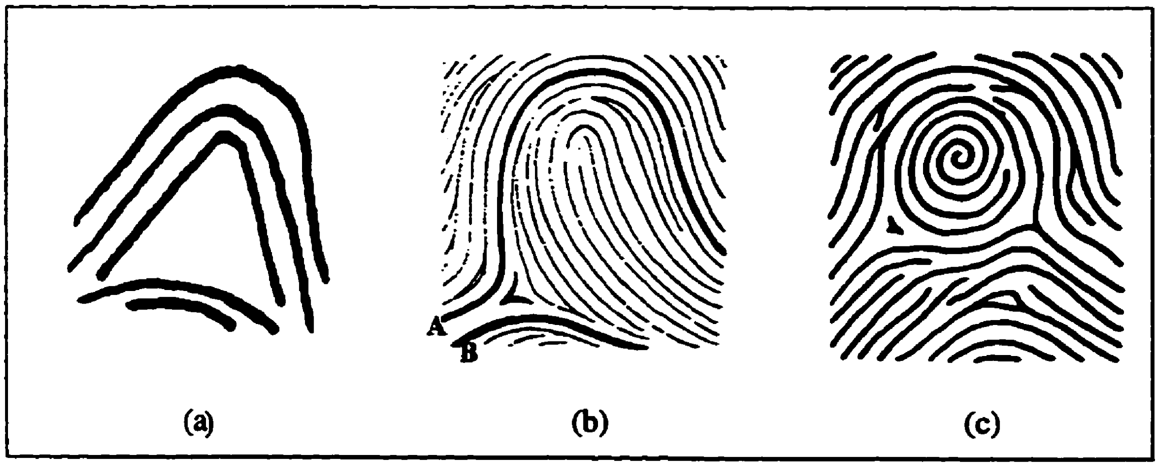


Fig. 2.2. Three major types of fingerprint: (a) arch, (b) loop, and (c) whorl. [after[Unit78]]

A *core* or sufficient recurve, a *delta*, and a *ridge count* across a looping ridge are the three essential features of a *loop* type. In the *whorl* type, at least two *deltas* are present with a recurve in front of each *delta*. The *arch* type has no pattern area. The *accidental whorl* is a pattern consisting of a combination of two different types of pattern, with the exception of the *plain arch*, which has two or more *deltas*, or a pattern which possesses some of the requirements for two or more different types, or a pattern which conforms to none of the above definitions.

2.2 A Model of the Human Visual System

The physical properties of the optical transmission pathway through the iris to the retina produce a low-pass spectral response which, when combined with the high-pass characteristic due to interconnection of the receptors gives an overall band-pass response. Associated with this spatial response is a logarithmic amplitude non-linearity due to adaptation to background luminance necessary for the eye to function over a wide range of average scene intensities. The fact that the eye has preferred regions of spatial frequency response and a non-linear amplitude response mechanism can be utilized to develop better coding algorithms.

Figure 2.3 shows a 32 gray-scale bar chart. Although the intensity of the stripes is constant, we actually perceive a brightness pattern that is strongly scalloped, especially near the boundaries. This phenomenon is called the *Mach band effect* [WoGo92]. Although there are 32 gray-scales in Fig. 2.3, the HVS cannot precisely distinguish them, especially near both ends of the chart. That is caused by the band-pass characteristic of the HVS.

The bandpass spatial frequency response of the eye has led to numerous attempts to improve coding efficiency by preferentially allocating bits to the frequency region (or to the corresponding transform coefficients) to which the eye is most sensitive. The transform coefficients corresponding to the most sensitive part of the HVS spatial response are preferentially weighted with respect to the others and so receive a higher bit allocation (i.e., more accurate quantization) than others [Clar96]. Such data encoded for human perception will be referred to as signals in this thesis.

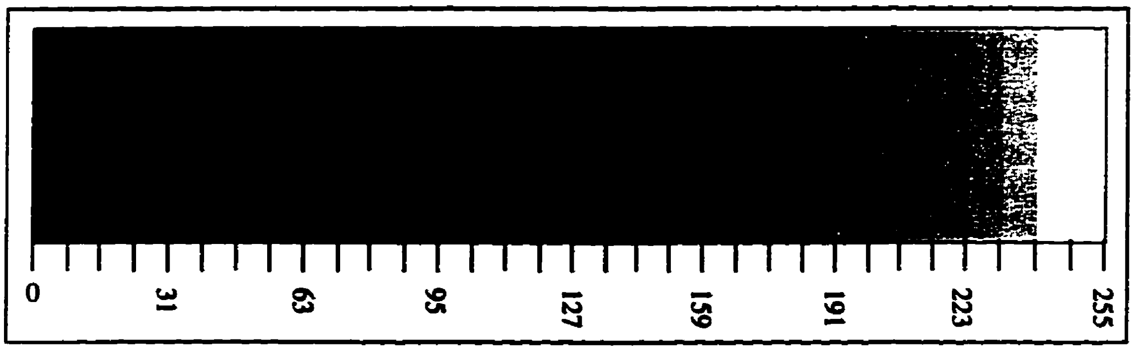


Fig. 2.3. A gray-scale bar chart with 32 levels.

2.3 Data Compression

As illustrated in Fig. 2.4, source data are operated upon according to a particular algorithm to produce compressed data. This compression of the original data is sometimes referred to as an encoding process, with the result that the compressed data is also called encoded data. Reversing the process, compressed data is decompressed to produce reconstructed data. Since this decompression process results in the decoding of the compressed data, the result is sometimes referred to as decoded data.

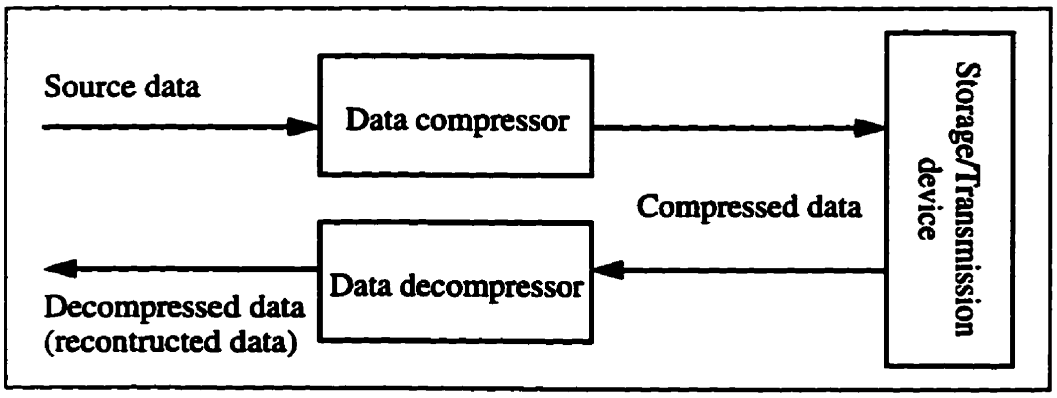


Fig. 2.4. Basic data compression block diagram.

Data compression techniques may be grouped into two classes: redundancy reduction and entropy reduction [Lync85]. A redundancy reduction operation removes, or at least reduces, the redundancy in such a way that it can be subsequently re-inserted into the data. Thus, redundancy reduction is always a reversible (lossless) process. An entropy reduction results in a reduction of information, since entropy is defined as the average information. The information lost can never be recovered, so an entropy reduction operation is an irreversible (lossy) process.

2.3.1 Compression Ratio

The degree of data reduction obtained as a result of the compression process is known as the compression ratio. This ratio measures the quantity of compressed data in comparison with the quantity of original data

$$R_c = \frac{pk}{qn} \tag{2.1}$$

where R_c is the compression ratio, p is the number of symbols in the original data, k is the number of bits per symbol in the original data, q is the number of encoded codewords produced by the compressor, and n is the number of bits in an encoded codeword [Kins91]. Thus, pk is the storage size of the original data and qn is the storage size of the compressed data. For example, $R_c = 4$ (compression ratio is 4:1) indicates a compression by 75%. From Eq. 2.1, it is obvious that the higher the compression ratio the more effective the compression technique employed.

2.3.2 Compression Quality, PSNR

Since reversible (lossless) compression techniques always perform a perfect reconstruction (decompressed data is the same as the original data), there is no distortion between the original and decompressed data. For most image compression applications, we use the *peak signal-to-noise ratio* (PSNR) parameter to measure the distortion of the reconstructed image. For a 256 gray scale image, the PSNR is defined as

$$PSNR = 10\log_{10}\frac{255^2}{MSE} \quad \text{[in dB]} \tag{2.2}$$

where *MSE* is the mean square error, the average of the energy of the difference between the original and the reconstructed image:

$$MSE = \frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N} \tag{2.3}$$

where N is the total number of pixels in the image, x_i and $\hat{x}_i$ are an original and a reconstructed pixel, respectively.

2.3.3 Information Measurement, Entropy

Information is defined in terms of a measure of uncertainty. The less likely a message, the larger its information, and the more likely a message, the smaller its information. The information content of the *i*th symbol of a discrete source (*i.e.*, a source with no dependence between successive symbols) is called the self-information [Lync85] and is given by

$$I_i = -\log_2 p_i \quad \text{[in bits]} \tag{2.4}$$

where p_i is the probability of the *i*th symbol and I_i is its information.

We will first consider the concept of entropy, which is a measure of uncertainty of a discrete random variable. For data (image) with n possible discrete symbols or gray levels with probability of occurrence p_1 to p_n, the average of all I_i ($1 \le i \le n$, defined in Eq. 2.4) of the data is defined as the entropy (H) of the data:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{2.5}$$

The entropy is used in information theory to denote the average number of bits required to present each symbol or gray level of the data (image).

2.3.4 Data Compression Techniques, Statistical Coding

There are many compression techniques [Kins91] either lossless (redundancy reduction) or lossy (entropy reduction), but in this thesis we will focus our attention upon two data compression methods in each group: statistical coding for redundancy reduction and transform coding for entropy reduction. The transform coding techniques will be discussed in Section 2.4 on image compression, transform coding.

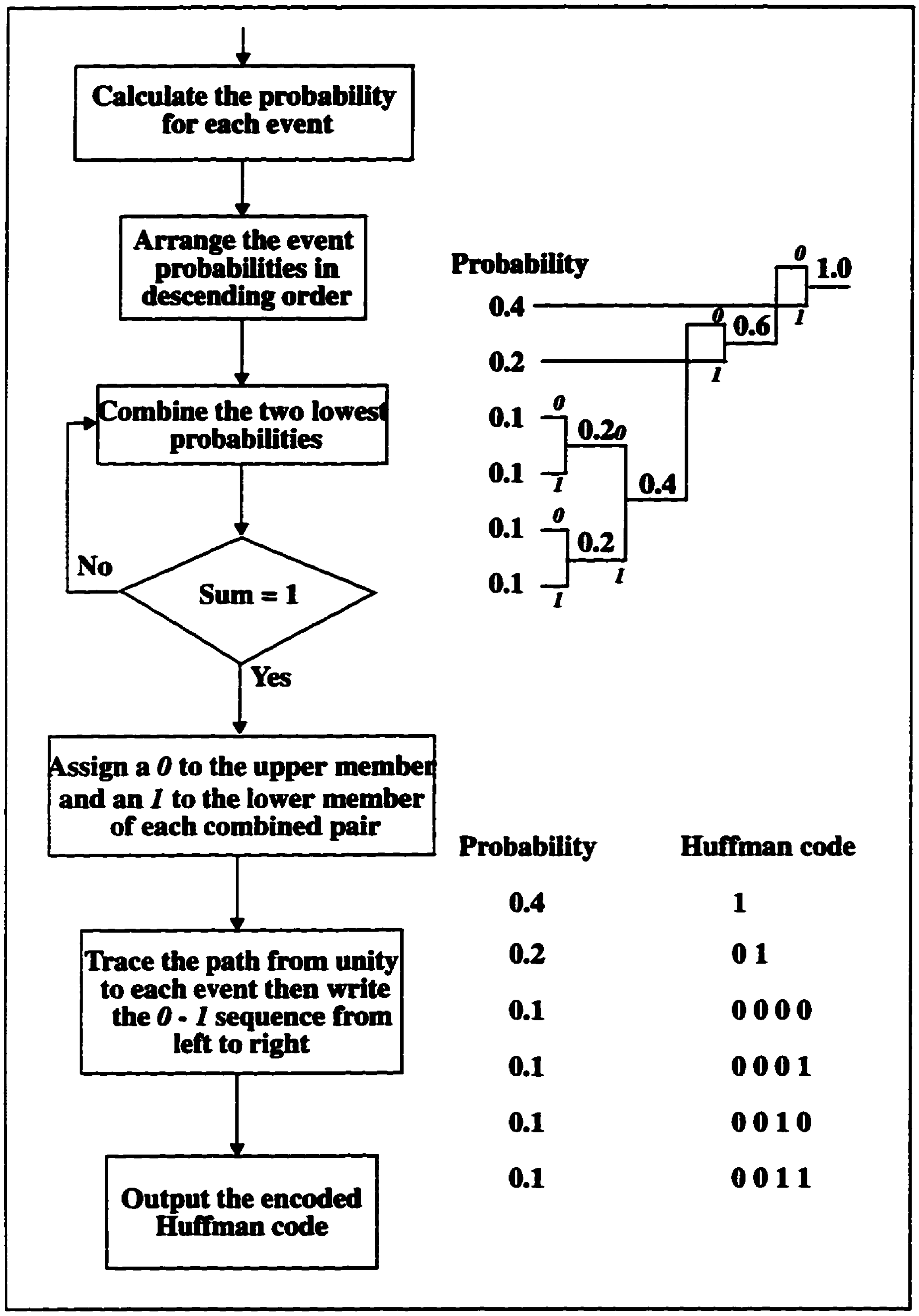
2.3.4.1 Statistical Coding

Statistical coding takes advantage of the probabilities of occurrence of each data, so that short codes can be used to present frequently occurring data, while longer codes are used to represent less frequently encountered data. The statistical coding process can be used to minimize the average code length of the encoded data. Since it needs $\lceil \log_2 n \rceil$ (an integer value greater than $\log_2 n$) bits to present an image with n gray scales, its redundancy is defined as

$$R = \log_2 n - H \qquad (2.6)$$

where H is the entropy of the image, defined in Eq. 2.5.

In this section we will use Huffman coding as an example to explain how statistical coding works and how it applies to data compression. Huffman coding is a procedure for encoding a statistically independent source in such a way as to yield the minimum average encoded word length. Huffman codes have a *prefix property*, which means that no short code group is duplicated at the beginning of a longer group. The prefix property of the Huffman code ensures that the code is uniquely decipherable. A coding procedure flowchart and an example are shown in Fig. 2.5. For a more complete description of this class of codes, see [Kins91].


Calculate the probability for each event
Arrange the event probabilities in descending order
Combine the two lowest probabilities
Sum = 1
No
Yes
Assign a *0* to the upper member and an *1* to the lower member of each combined pair
Trace the path from unity to each event then write the *0* - *1* sequence from left to right
Output the encoded Huffman code
Probability
0.4
0.2
0.1
0.1
0.1
0.1
0.2
0.2
0.4
0.6
1.0
0
1

Probability	Huffman code
0.4	1
0.2	0 1
0.1	0 0 0 0
0.1	0 0 0 1
0.1	0 0 1 0
0.1	0 0 1 1

Fig. 2.5. Huffman coding flowchart and example.

The average code length is computed by the formula:

$$L_{average} = \sum_i L_i p_i \tag{2.7}$$

where p_i is the probability of the *i*th symbol or level, and L_i is the *i*th encoded Huffman code length. The $L_{average}$ (Eq. 2.7) is found to be 2.4 bits for this Huffman code. The entropy (Eq. 2.5) is computed to be 2.3219 bits. The redundancy (Eq. 2.6) is calculated to be 0.263 bits. The efficiency is defined as:

$$Efficiency = \frac{H}{L_{average}} \times 100 \qquad \text{[in \%]} \tag{2.8}$$

and for this code is 96.75%.

2.4 Image Compression Techniques, Transform Coding

Transform coding involves a linear transformation in which the signal such as an image is projected to a transform space, a quantization which quantizes the transformed coefficients for transmission or storage, and an inverse transformation to get the reconstructed signal. It should be remembered that the transformation itself is not providing the compression, but rather it is mapping the signal into another domain in which compression can be accomplished more easily. Transform coding methods belong to lossy compression methods because of the quantization error. For gray-scale image compression due to the restriction of the HVS (see Sec. 2.2), we reduce the entropy during the quantization. The compressor deals with those quantized coefficients. In most cases, the compressor uses lossless compression methods to compress those quantized coefficients because the entropy has already been reduced in the quantization stage. A block diagram of how transform coding applies to compression is shown in Fig. 2.6.

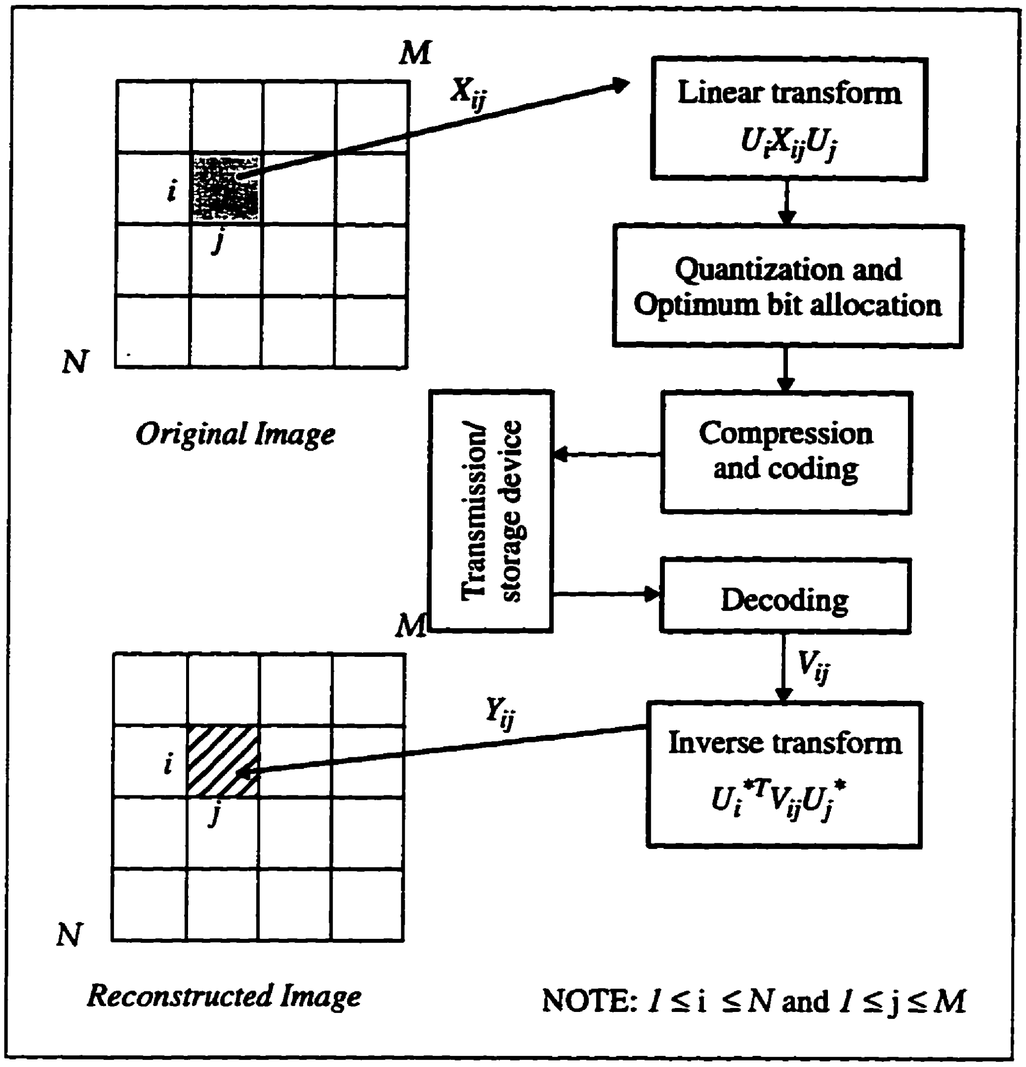


Fig. 2.6. Transform coding block diagram of a two dimensional image.

To make transform coding practical, a given image is divided into small rectangular blocks with *i* rows by *j* columns, and each block is transformed independently. The U_i is an $i \times i$ unitary matrix, U_i^* is the conjugate of U_i, V_{ij} is the decoded transform block, and the X_{ij}, Y_{ij} are the original and reconstructed blocks, respectively. For an $N \times M$ image divided into *NM/ij* blocks (Fig. 2.6), the main storage requirements for implementing the transform are reduced by a factor of *NM/ij*. The computational load is reduced by a factor of $\frac{\log_2 NM}{\log_2 ij}$ for a fast transform requiring $O(\log_2 N)$ operations to transform an $N \times 1$ vector. For a 512 × 512 image divided into 16 × 16 blocks, these factors are 1024 and 2.25, respectively. Although the operation count is not greatly reduced, the complexity of a hardware or software for implementing small-sized transforms is reduced significantly. However, smaller block sizes yield lower compression. Typically, a block size of 16 × 16 is used.

2.4.1 Quantization

Amplitude quantization is the procedure of transforming a given input signal amplitude $x(t)$ at time t into an amplitude $y(t)$ taken from a finite set of possible amplitudes. In this Section, we will assume the important simple situation of memoryless and instantaneous quantization, a procedure where the transformation at time k is not affected by earlier or later input samples. When dealing with memoryless quantizers, we will drop the time index, and use symbols such as x instead of $x(t)$. The mapping

$$y = Q(x) \tag{2.9}$$

is the quantizer characteristic, a staircase function, as shown in Fig. 2.7. There are four types of $Q(x)$: nonuniform midrise; uniform midrise; nonuniform midtread and uniform midtread. The difference between midrise and midtread type depends on whether zero is one of the output levels or not. For a uniform quantizer $Q(x)$, the quantization step (Δ) is defined as:

$$\Delta = x_{k+1} - x_k = y_{k+1} - y_k ; \quad k = 1, 2, ..., L\text{-}1 \tag{2.10}$$

where L is the number of quantization steps, the amplitudes y_k are called the representation levels or the reconstruction values, and the amplitudes x_k are called decision levels or transition levels.

Quantization error (Er_q) is defined as the difference between x and y. Apply Eq. 2.9 and we get

$$Er_q = x - y = x - Q(x) \tag{2.11}$$

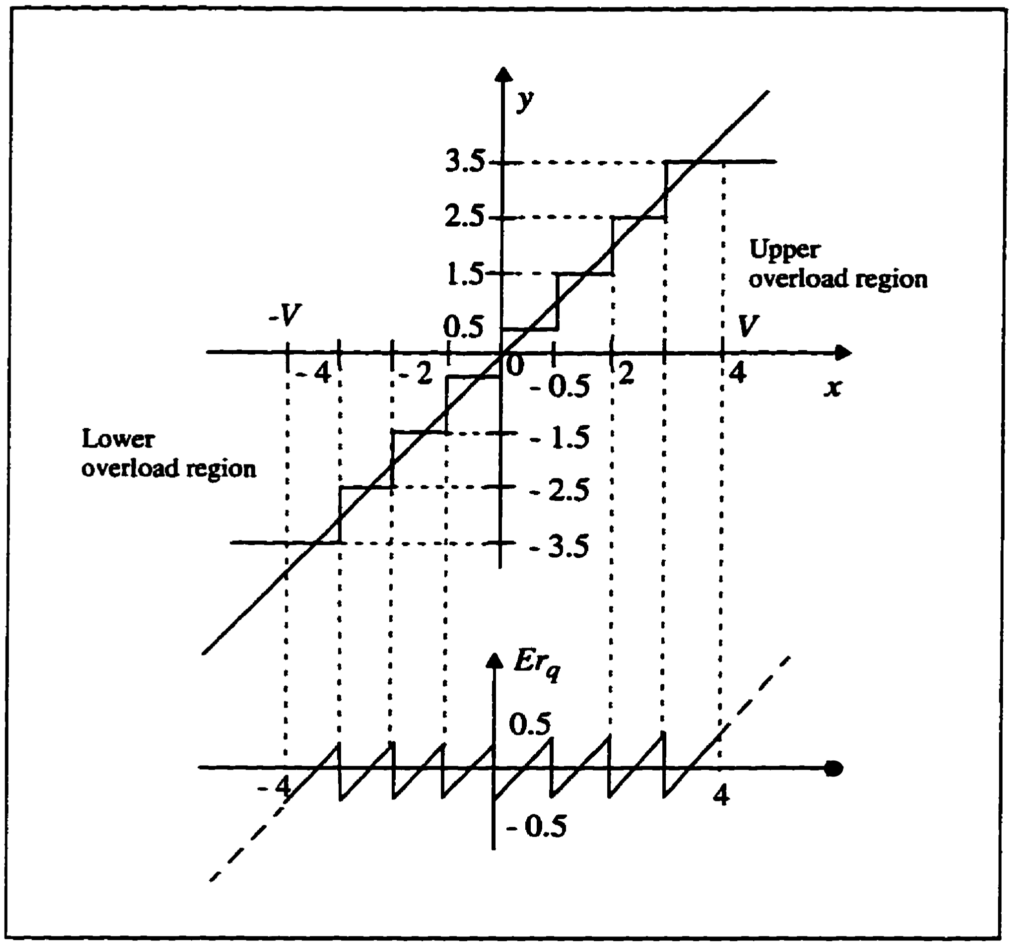


Fig. 2.7. Illustration of the deterministic nature of quantization error in zero-memory uniform quantization (step size $\Delta = 1$).

Let x, be a zero-mean random variable with probability density function (PDF) $p(x)$. The variance σ_x^2 is defined as:

$$\sigma_x^2 = E[X^2] = \int_{-\infty}^{\infty} x^2 p(x)\,dx \tag{2.12}$$

The quantization error (Eq. 2.11) is also a random variable and its variance σ_q^2 is defined as:

$$\sigma_q^2 = \sum_{k=1}^{L} \int_{x_k}^{x_{k+1}} (x-y_k)^2 p(x)\,dx \tag{2.13}$$

The quantization error variance is the most important quantity for comparing the performances of a quantizer. The signal-to-noise ratio can be written as:

$$SNR = 10\log_{10}\left(\frac{\sigma_x^2}{\sigma_q^2}\right) \quad \text{[in dB]} \tag{2.14}$$

A uniform quantizer is of interest because of its simplicity. The *p(x)* is constant within each input interval (*i.e.*, the input probability density is uniform over the quantizer's range). By giving equal intervals between the transition levels and the reconstruction levels, the uniform quantizer is also called a linear quantizer. For simplicity, we assume that the contribution of the overload region is negligible, so Eq. 2.13 becomes

$$\sigma_q^2 = \sum_{k=2}^{L-1} p(x_k) \left\{ \frac{[x-y_k]^3}{3} \right\} \Bigg|_{x_{k-1}}^{x_k} \tag{2.15}$$

From Eq. 2.10, we get $x_k = y_k + \frac{\Delta}{2}$ and $x_{k-1} = y_k - \frac{\Delta}{2}$. Substituting these values into Eq. 2.15 gives

$$\sigma_q^2 = \frac{1}{12} \sum_{k=2}^{L-1} p(x_k)\,\Delta^3 \tag{2.16}$$

However, since

$$\sum_{k=2}^{L-1} p(x_k)\,\Delta \approx 1 \tag{2.17}$$

substituting Eq. 2.17 into Eq. 2.16, the variance of quantization error will be:

$$\sigma_q^2 = \frac{\Delta^2}{12} \tag{2.18}$$

For the uniform quantizer, the PDF is

$$p(x) = \frac{1}{2V} \tag{2.19}$$

where $-V \leq x \leq V$. Substituting Eq. 2.19 into Eq. 2.12, we get

$$\sigma_x^2 = \frac{V^2}{3} \tag{2.20}$$

Since

$$\Delta = \frac{2V}{L}, \text{ for } L >> 2 \tag{2.21}$$

then substituting Eq. 2.18, Eq. 2.20 and Eq. 2.21 into Eq. 2.14, the SNR of uniform quantizer is

$$SNR = 10\log_{10} L^2 \quad \text{[in dB]} \tag{2.22}$$

If we use a binary code for L, then $L = 2^n$ and Eq. 2.22 becomes

$$SNR = 20n\log_{10} 2 \cong 6.02n \quad \text{[in dB]} \tag{2.23}$$

Equation 2.23 indicates that for every increasing bit of the uniform quantizer the SNR will be improved by 6 dB. Although Eq. 2.23 was derived by simplifying assumptions, this linear relationship between SNR and n, the quantization bits, indeed holds for other memoryless sources and quantizers. If the input signal PDF is uniform and a uniform quantizer is used, Eq. 2.23 predicts how many quantization bits are needed to achieve a certain SNR, or alternatively, for a given SNR value we can obtain the minimum number of quantization bits required by the quantizer.

2.4.1.1 Vector Quantization

Vector quantization (VQ) is one of the lossy compression techniques (Sec. 2.3). According to Shannon's rate-distortion theory, a better performance is always achievable in theory by coding vectors instead of scalars [Gray84]. Vector quantization groups a number of input data as a vector and quantizes the vector instead of one data at a time such as the scalar quantization (Sec. 2.4.1) [CORG93]. The vector quantizers codebook design is the core technique of VQ that dictates the performance of the vector quantizer (encoding speed and quantization error). In Nasrabadi and King's review article [NaKi88], many codebook design algorithms are described. However, the Lloyd [Lloy79], LBG (Linde, Buzo and Gray) [LiBG80] and Kohonen Map [Koho90] are the most used algorithms [XuKO93]. In chapter 5, we will show the result of applying the Lloyd algorithm to compress an image.

2.4.2 Transform Coding

Transform coding is also called block quantization [HuSh63]. A block of data (X) is unitarily transformed by a unitary matrix U, so that a large fraction of its total energy is packed in relatively few transform coefficients ($\theta_0, \ldots, \theta_{N-1}$), which are quantized independently ($Q_0, \ldots, Q_{N-1}$). The optimum quantizer is defined as the one that minimizes the mean square error of the reproduced data for a given number of total bits

[WiTa71][Wint72][ZeNo77]. The block diagram of transform coding is illustrated in Fig. 2.8.

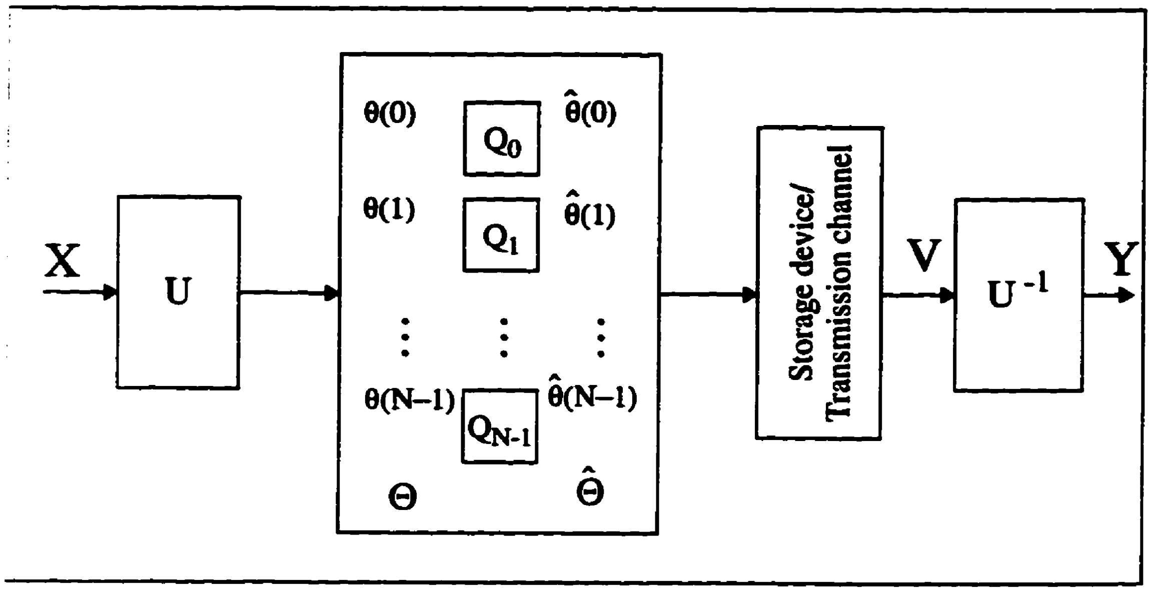


Fig. 2.8. Block diagram of transform coding (block quantization).

2.4.2.1 Linear Transformation

The forward and inverse transforms of a N × N image are shown in Eq. 2.25 and Eq. 2.26, respectively [Jain89]. To ensure that the mapping is a linear transformation and the inverse transformation exists, the transform matrix U must be a unitary matrix. If all elements of U are real then we can call U an orthogonal matrix [Gold91]. For a unitary matrix U, it satisfies Eq. 2.24 and its inverse matrix $U^{-1} = U^H$.

$$U^H U = U U^H \tag{2.24}$$

$$\Theta = U X U^T \tag{2.25}$$

$$X = U^H \Theta U^* \tag{2.26}$$

Note that X is the original N × N image; Θ is the matrix of transform coefficients; U is the transform matrix; U^* is the conjugate of U and U^H is the *Hermitian* transpose of U. The transformation functions as a projection in which the signal space is mapped into a transform space.

Equations 2.25 and 2.26 indicate that the transformation can be performed by first transforming each column of X and then transforming each row of the result to obtain the rows of Θ. The most popular linear transforms are the *discrete cosine transform* (DCT), Hadamard, *discrete Fourier transform* (DFT), *fast Fourier transform* (FFT) and *discrete wavelet transform* (DWT).

Since U is a unitary matrix, its column vectors are the orthogonal basis vectors. If the basis vectors are orthonormal, then the average sum of the variances of the transform coefficients is equal to the variance of the input data.

$$\frac{1}{N}\sum_{k=0}^{N-1}\sigma_{\theta k}^2 = \frac{1}{N}\sum_{k=0}^{N-1}E[\theta^2(k)] = \frac{1}{N}E[\Theta^T\Theta] = \frac{1}{N}E[X^TU^TUX]$$

$$= \frac{1}{N}E[X^TX] = \frac{1}{N}E[x^2(k)] = \frac{1}{N}\sum_{k=0}^{N-1}\sigma_{xk}^2 = \sigma_x^2 \tag{2.27}$$

where $\sigma_{\theta k}^2$ is the variance of transform coefficient $\theta(k)$, $k = 0, ..., N\text{-}1$. Please refer to Fig. 2.8 for the relationships between Θ, X, and U.

2.4.2.2 *Characteristics of Transform Coefficients*

We use a simple example to introduce the transform coding or block quantization. Let $\{x(n)\}$, $n = 0, 1$, be an input sequence with $x(0) = 2$; $x(1) = 3$, and a linear transform spanned by two column vectors $\boldsymbol{b}_0$ and $\boldsymbol{b}_1$, where

$$b_0 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \quad b_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix} \tag{2.28}$$

Apply to Eq. 2.25 in a one dimensional transform only. The transform coefficients $\theta(0) = \frac{5}{\sqrt{2}}$, $\theta(1) = \frac{-1}{\sqrt{2}}$ are calculated from

$$\begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{5}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \tag{2.29}$$

The transform coefficients $\frac{5}{\sqrt{2}}$ and $\frac{-1}{\sqrt{2}}$ can be expressed in the form

$$\theta(0) = \frac{5}{\sqrt{2}} = \frac{1}{\sqrt{2}}[x(0) + x(1)] \tag{2.30}$$

$$\theta(1) = \frac{-1}{\sqrt{2}} = \frac{1}{\sqrt{2}}[x(0) - x(1)] \tag{2.31}$$

Note $\theta(0)$ and $\theta(1)$ represent the sum and difference of the two input samples, and hence the power of low frequency and high frequency components in $\{x(n)\}$. For minimizing the mean squared error in the reconstruction, we expect that the coefficient $\theta(0)$ is more crucial than $\theta(1)$. This isolation of low and high frequency components is precisely the motivation for transform representation (Eqs. 2.30 and 2.31). The geometrical interpretations for Eq. 2.29 are shown in Fig. 2.9.

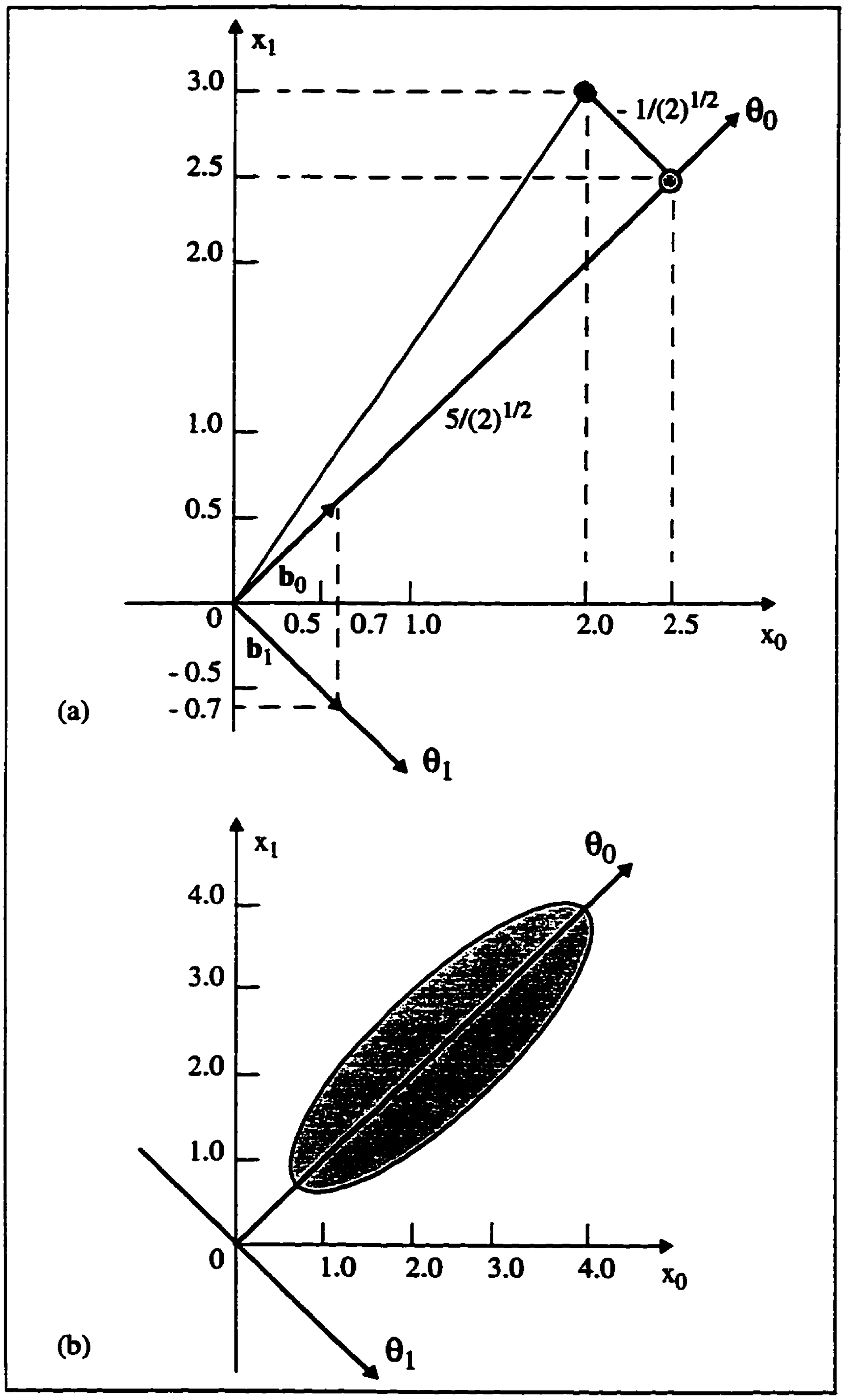


Fig. 2.9. (a) Geometrical interpretation for Eq. 2.29 and (b) new coordinate system.

Let us consider a population or ensemble of input sequences of length 2 ($\{x(n)\}, n = 0, 1$). Assume for illustration that each of these sequences exhibits a positive adjacent sample correlation as in Fig. 2.9(a). The members of such an ensemble of (x(0), x(1)) pairs will occupy the shaded ellipse-like region of Fig. 2.9(b). Points in this region have the property that their (x_0, x_1) coordinates are highly correlated, while the (θ_0, θ_1) coordinates are not. Given the θ_0 coordinate $\theta(0)$ of a sample point, one expects the $\theta(1)$ value to be smaller in magnitude.

Figure 2.10(a) shows a 256 sample scan line of an image (256 × 256). The magnitude of its DCT is shown in Fig. 2.10(b). From Figs. 2.10 (a) and (b), we can see clearly that the stationary highly correlated source (scan line) becomes low correlated transform coefficients after DCT. This result is caused by the characteristic of the unitary transform. Most unitary transforms have a tendency to pack a large fraction of the average energy of the image into relatively few components of the transform coefficients. Since the total energy is preserved, this means many of the transform coefficients will contain very little energy.

The one dimensional DCT of a sequence $\{u(t), 0 \le t \le N-1\}$ is defined as

$$v(k) = \alpha(k) \sum_{t=0}^{N-1} u(t) \cos\left[\frac{\pi(2t+1)k}{2N}\right], \quad 0 \le k \le N-1 \tag{2.32}$$

$$\alpha(0) = \sqrt{\frac{1}{N}} \tag{2.33}$$

$$\alpha(k) = \sqrt{\frac{2}{N}}, \quad 1 \le k \le N-1 \tag{2.34}$$

The inverse transform is given by

$$u(t) = \sum_{k=0}^{N-1} \alpha(k) v(k) \cos\left[\frac{\pi(2t+1)k}{2N}\right], \quad 0 \le t \le N-1 \tag{2.35}$$

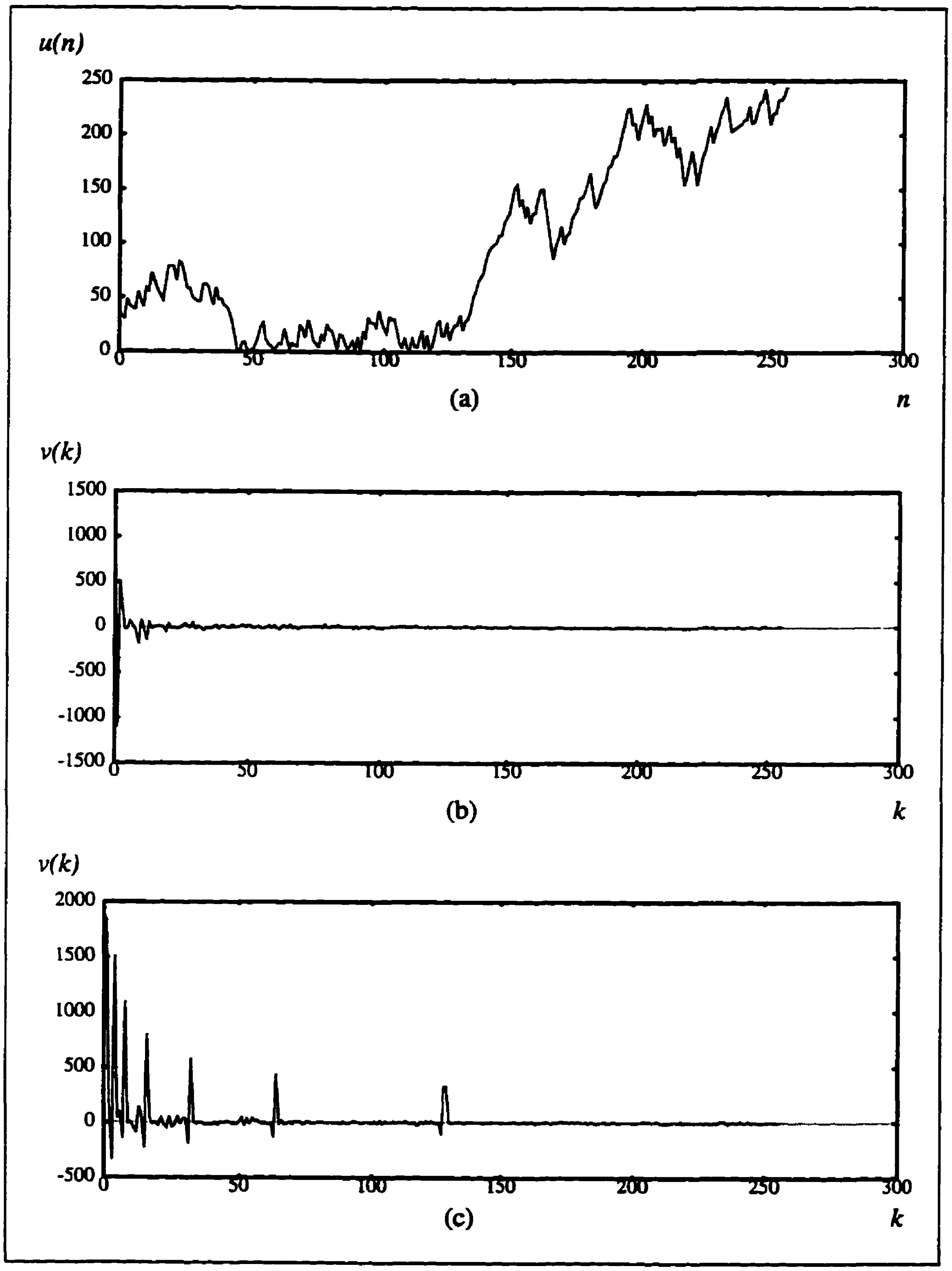


Fig. 2.10. (a) A 256-sample scan line of a gray-scale image, (b) DCT of (a), (c) DWT of (a).

2.4.2.3 Optimum Bit Allocation

For unitary (orthogonal) transforms, the reconstruction error variance for an input block is the sum of error variances in the quantization of individual transform coefficients,

$$\sigma_r^2 = \frac{1}{N}E[(\Theta - V)^T(\Theta - V)] = \frac{1}{N}E[Q^T Q] = \frac{1}{N}\sum_{k=0}^{N-1}\sigma_{qk}^2 = \sigma_q^2 \qquad (2.36)$$

where Θ is the column vector of the transform coefficients, V is the column vector of quantization reconstruction values (see Fig. 2.8), σ_r^2 is the reconstruction error variance and σ_q^2 is the quantization error variance. The optimum bit allocation for quantization of transform coefficients is used to minimize the σ_q^2.

We assume that x (input vector) has zero-mean and its elements are random variables. The average sum of variances $E[\theta^2(k)] = \sigma_k^2$ of the transform coefficients $\theta(k)$ equals the variance of the input σ_x^2 (defined in Eq. 2.27).

The R_k (bits/sample) is needed for coefficients $\theta(k)$ of variance $\sigma_{\theta k}^2$ if an average MSE σ_{qk}^2 is not to be exceeded. Values of $\sigma_{\theta k}^2$ are in general different and so are the individual rates R_k. In fact, the problem of optimizing a transform coding scheme can be stated as that of finding an orthogonal (unitary) matrix, and then of finding a distribution (allocation) of bits R_k such that the average coefficient error variance

$$\sigma_q^2 = \frac{1}{N}\sum_{k=0}^{N-1}\sigma_{qk}^2 \qquad (2.37)$$

is minimized with the constraint of a given average bit rate

$$R = \frac{1}{N}\sum_{k=0}^{N-1} R_k = constant \tag{2.38}$$

To minimize Eq. 2.37 and constrained by Eq. 2.38, the problem is solved in [HuSc63], and an optimum bit allocation is given by

$$R_k = R + \frac{1}{2}\log_2 \frac{\sigma_{\theta k}^2}{\left[\prod_{j=0}^{N-1} \sigma_j^2\right]^{\frac{1}{N}}} \tag{2.39}$$

where σ_j^2 is the variance of quantizer j ($j = 1, \ldots, N-1$).

The above result (Eq. 2.39) can be easily interpreted for the example of uniform quantizers operating without overload. Identical quantization error variances result if all quantizers have the same step size Δ, and hence the same $\sigma_q^2 = \frac{\Delta^2}{12}$ (Eq. 2.18). However, a coefficient with higher variance needs more levels than a coefficient of lower variance (as shown in Fig. 2.11), to match dynamic ranges of both the quantizer and the input. This is indeed guaranteed by the bit allocation result given by Eq. 2.39.

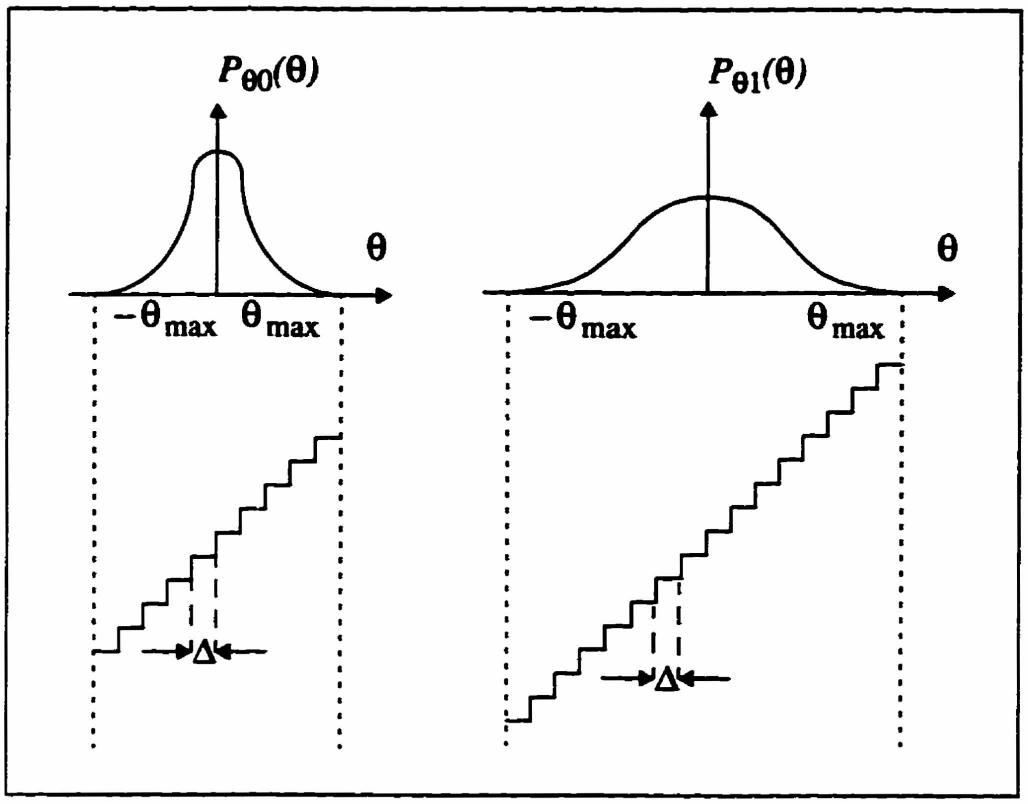


Fig. 2.11. Uniform quantization of coefficients θ_0 and θ_1 without overload. The identical step size Δ ensures identical error variances σ_{q0}^2 and σ_{q1}^2. The differing numbers of quantization levels depends on the variance of the input data.

2.4.3.4 Zonal Filters

Bit allocation as discussed above is a complex strategy, especially if it is adaptive. It involves quantizers with different numbers of levels. It also involves reassignment procedures. A simpler strategy is called zonal filtering. The image transform is filtered by a zonal mask (Fig. 2. 12) such that only a fraction of the transform coefficients are retained and the remaining ones are set to zero. The mask function can be described as

$a(k,l) = 1,$ $\qquad$ k,l located in the *zone*

$$= 0, \qquad \text{otherwise} \tag{2.40}$$

where $a(k,l)$ is the pixel of a two dimensional image, the *zone* is the area of the zonal mask that transmits the transform coefficients.

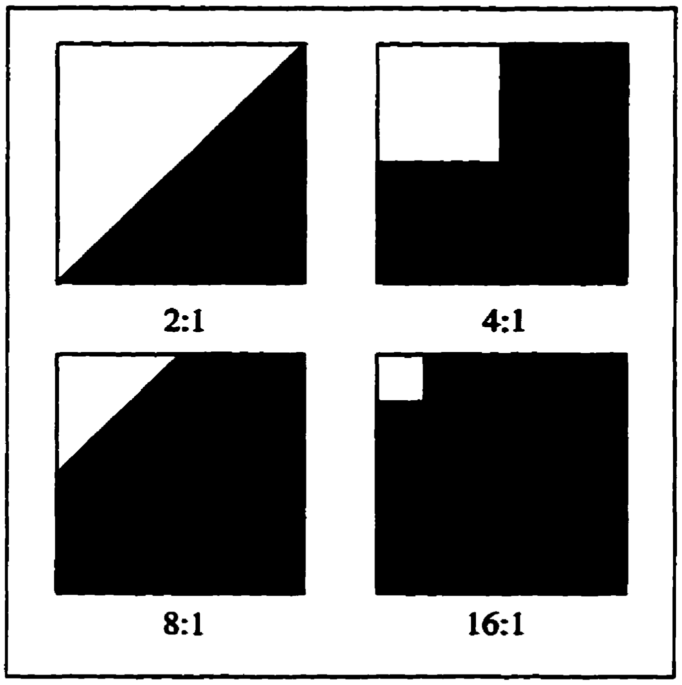


Fig. 2.12. Zonal filters for 2:1, 4:1, 8:1, 16:1 sample reduction. Bright areas are passbands, dark areas are stopbands.

2.5 Summary of Chapter 2

This chapter presents the requirement for better fingerprint compression techniques. It is more difficult to compress a fingerprint image than other images because the decompressed image can be used in court as evidence. Lossless and lossy compression techniques are discussed in general. We choose lossy compression techniques for two reasons: to achieve a higher compression ratio and to overcome the restriction of the *human visual system* (HVS). Transform coding and quantization are also discussed. In the next chapter, we shall describe the wavelet transform as the basis for fingerprint compression.

CHAPTER 3

WAVELET TRANSFORM AND PACKETS

Like sines and cosines in Fourier analysis, wavelets are used as basis functions in representing other functions. Once the wavelet (sometimes called the mother wavelet) $\psi(t)$ is fixed, one can make a basis of translations and dilations of the mother wavelet $\{ \psi\left(\frac{t-b}{a}\right), (a, b) \in R^{+} \times R \}$.

Why should we use wavelets instead of the traditional Fourier methods? There are some important differences between Fourier analysis and wavelets. Fourier basis functions are localized in frequency but not in time. Small frequency changes in the Fourier transform will produce changes everywhere in the time domain. Wavelets are local in both frequency/scale (via dilations) and in time (via translations). This localization is an advantage in many cases.

In particular, the *wavelet transform* (WT) is of interest for the analysis of non-stationary signals, because it provides an alternative to the classical *short-time Fourier transform* (STFT) or Gabor transform [RiVe91]. The basic difference is as follows: in contrast to STFT, which uses a single analysis window, the WT uses short windows at high frequencies and long windows at low frequencies.

Second, many classes of functions can be represented by wavelets in a more compact way. For example, functions with discontinuities and functions with sharp spikes usually take substantially fewer wavelet basis functions than sine-cosine basis functions to achieve a comparable approximation. This sparse coding makes wavelets excellent tools in data compression. For example, the FBI has standardized the use of wavelets in digital fingerprint image compression. The compression ratios are in the range of 20:1, and the difference between the original image and the decompressed one can be perceived only by an expert [ViMü93].

3.1 Time-Frequency Decompositions

Standard Fourier analysis decomposes a signal $f(t)$ into frequency ($f = \omega/2\pi$) and phase components and determines the relative strength of each component. It does not tell us when the signal exhibited the particular frequency. If the frequency content of a signal were to vary drastically from interval to interval as in a musical scale, the standard forward and inverse Fourier transform (FT) [AkHa92]

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-j\omega t} dt \tag{3.1}$$

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\, e^{j\omega t} d\omega \tag{3.2}$$

would sweep over the entire time axis and wash out any local anomalies (*e.g.*, bursts of high frequency) in the signal.

Under such conditions, Gabor (1946) resorted to the windowed, *short-time Fourier transform* (STFT), which moves a fixed-duration window over the time function and extracts the frequency content in that interval. This would be suitable for signals that are locally stationary, but globally nonstationary.

The STFT positions a window $g(t)$ at some point τ on the time axis, and calculates the Fourier transform of the signal within the extent or spread of the window. When the window $g(t)$ is Gaussian, the STFT is called a Gabor transform. The basis functions of this transform are generated by modulation and transformation of the window function,

$$F(\omega, \tau) = \int_{-\infty}^{\infty} f(t)\, g^{*}(t-\tau)\, e^{j\omega t} dt \tag{3.3}$$

where ω and τ are modulation and translation parameters, respectively, and the * denotes the complex conjugate.

The window function $g(t)$ is also called a prototype function, or sometimes, a mother function. As τ increases, the mother function simply translates in time, while keeping the spread of the window constant.

Similarly, as the modulation parameter ω increases, the transform simply translates in frequency, retaining a constant bandwidth. We see that each element β_T (bandwidth of $g(t)$) and β_ω (bandwidth of $G(\omega)$, Fourier transform of $g(t)$) of the resolution cell $\beta_T\beta_\omega$ is constant for any frequency ω and time shift τ as indicated by the rectangles of fixed area and shape in Fig. 3.1.

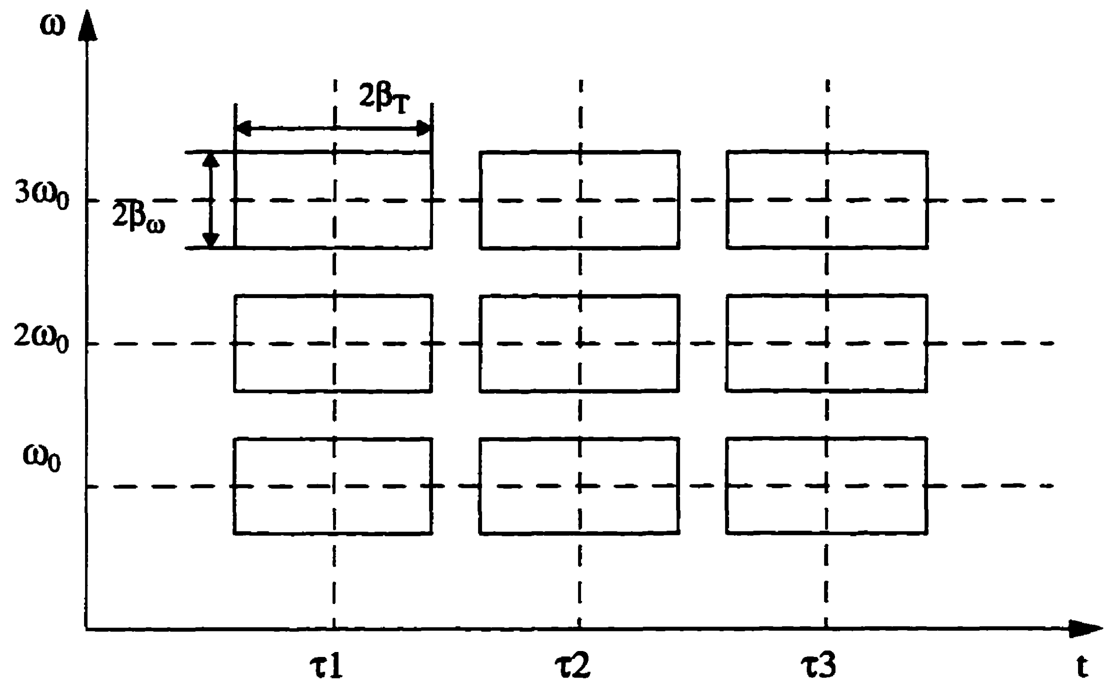


Fig. 3.1. Time-frequency plane showing resolution cells for STFT.

3.2 Wavelet Transform

Unlike the FT and STFT, the wavelet transform is founded on basis functions formed by dilation and translation of a prototype function $\psi(t)$. These basis functions are short-duration, high-frequency and long-duration, low-frequency functions. They are much better suited for representing short bursts of high-frequency signals or long-duration, slowly varying signals.

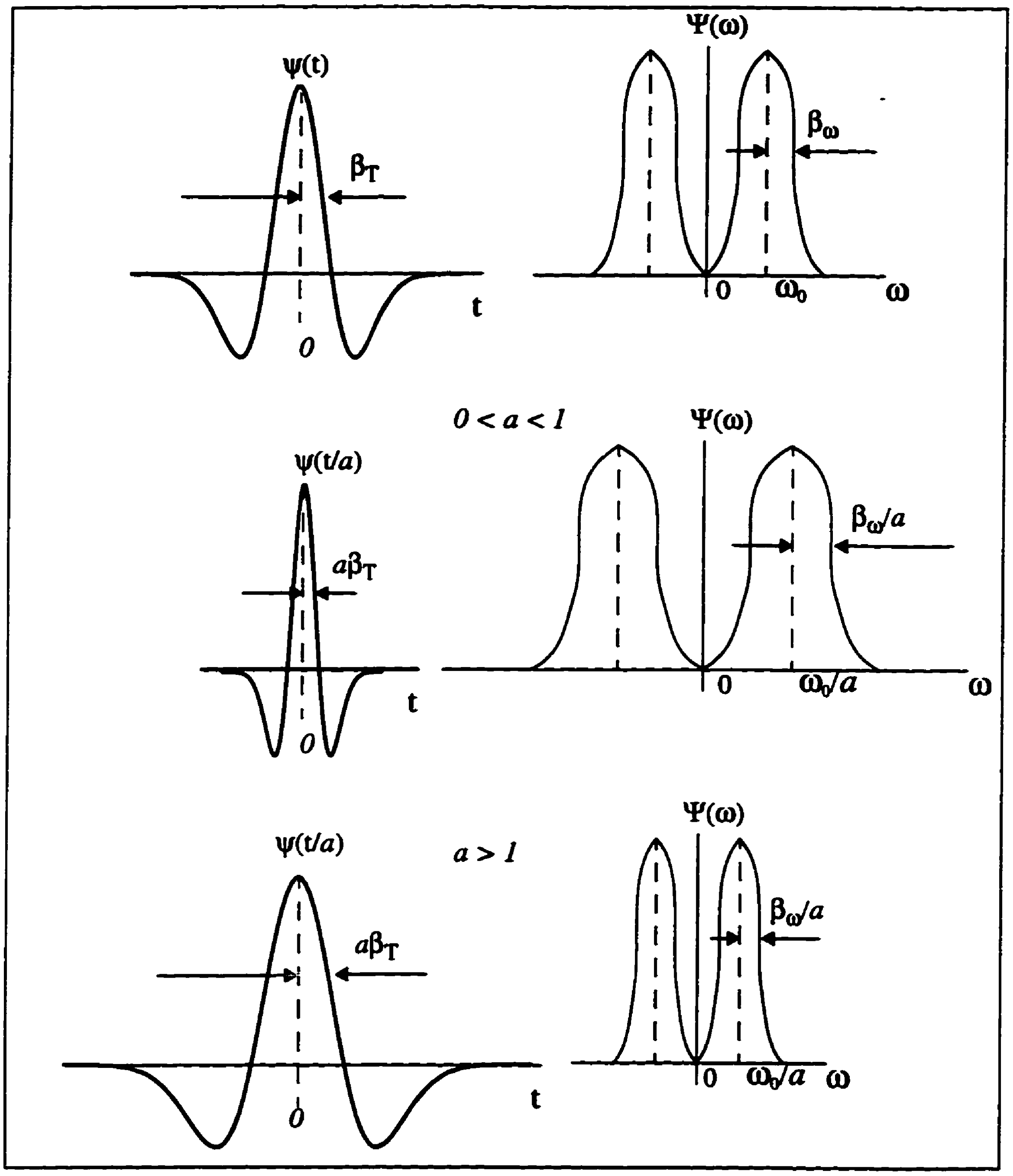


Fig. 3.2. Typical wavelet family in time and frequency domains.

This concept is suggested by the scaling property of Fourier transforms. If $\psi(t)$ and $\Psi(\omega)$ constitute a Fourier transform pair, then

$$\frac{1}{\sqrt{a}}\psi\left(\frac{t}{a}\right) \Leftrightarrow \sqrt{a}\Psi(a\omega) \tag{3.4}$$

where $a > 0$ is a continuous variable. Thus a contraction in one domain is accompanied by an expansion in the other, but in a nonuniform way over the time-frequency plane. The wavelet family is thus defined by scale and shift parameters a, b, as

$$\psi_{ab}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \tag{3.5}$$

and the wavelet transform is the inner product

$$W(a,b) = \int_{-\infty}^{\infty} \psi_{ab}(t) f^{*}(t)\, dt = \langle \psi_{ab}, f \rangle \tag{3.6}$$

where $a \in R^{+}$, $b \in R$, and $*$ denotes the complex conjugate.

For large a, the basis function becomes a stretched version of the prototype wavelet, that is, a low-frequency function, while for small a, this basis function is a contracted version of the wavelet function, which is a short-time duration, high-frequency function. Depending on the scaling parameter a, the wavelet function $\psi(t)$ dilates or contracts in time, causing the corresponding contraction or dilation in the frequency domain. Thus, the wavelet transform provides a flexible time-frequency resolution. Figure 3.3 displays the time-frequency plane showing resolution cells for the wavelet transform.

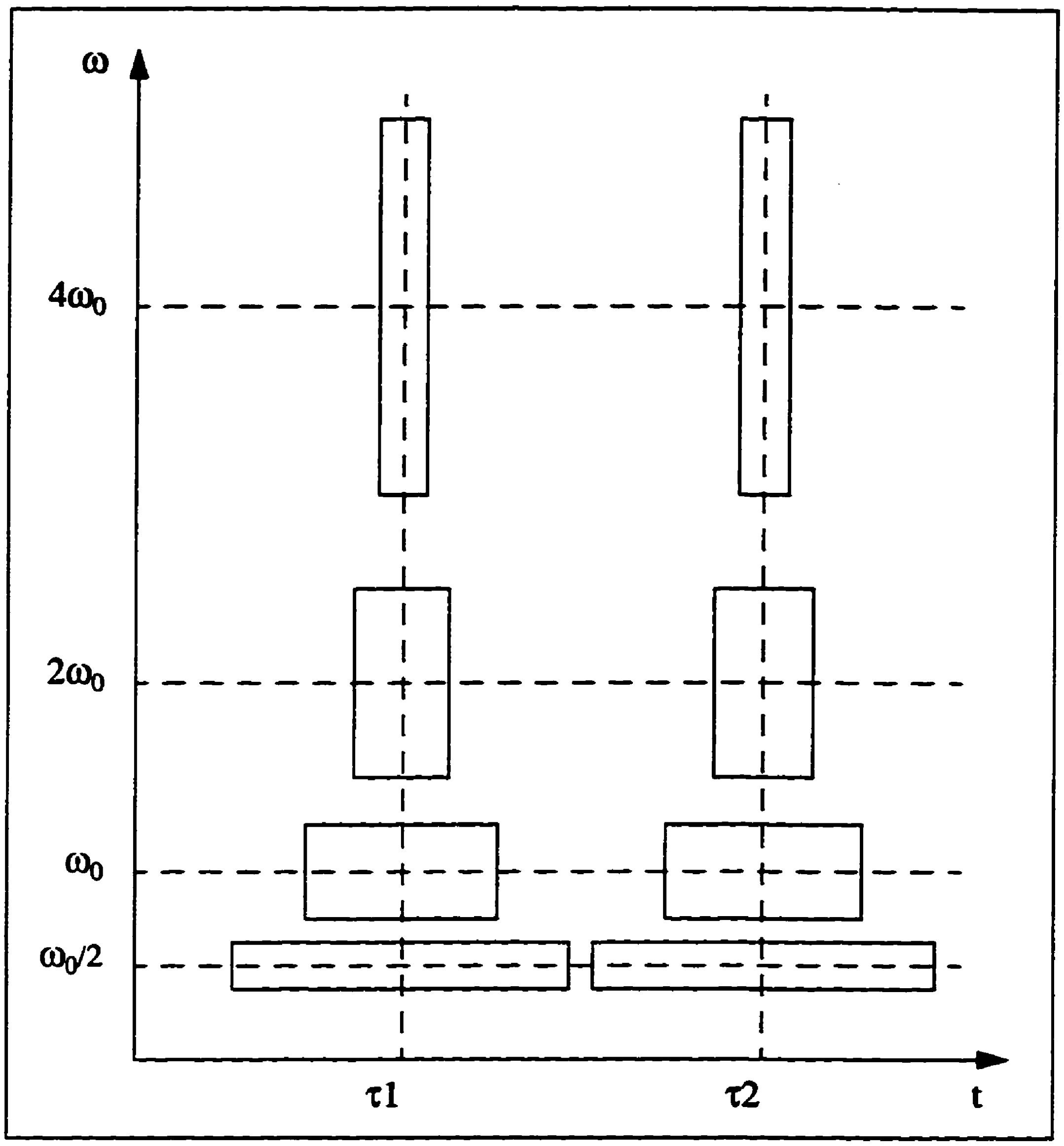


Fig. 3.3. Time-frequency plane showing resolution cells for wavelet transform.

3.3 Two Channel QMF Bank

Let $h(k)$, $0 \leq k < N$ be some *finite-impulse response* (FIR) lowpass filter with real coefficients. The mirror filter is defined as:

$$g(k) = (-1)^k h(k) \tag{3.7}$$

or, equivalently, in the Z-transform domain,

$$G(z) = H(-z)$$

or

$$G\left(e^{j\omega}\right) = H\left(e^{j(\omega-\pi)}\right) \tag{3.8}$$

Using the substitution $\omega \rightarrow \frac{\pi}{2} - \omega$, and noting that the magnitude is an even function of ω leads to

$$\left|G\left(e^{j\left(\frac{\pi}{2}-\omega\right)}\right)\right| = \left|H\left(e^{j\left(\frac{\pi}{2}+\omega\right)}\right)\right| \tag{3.9}$$

Since this last form demonstrates the mirror image property of $H(\omega)$ and $G(\omega)$ about $\omega = \pi/2$, they are called *quadrature mirror filters*, or QMF [AkHa92] [RaVH96], and are shown in Fig. 3.4.

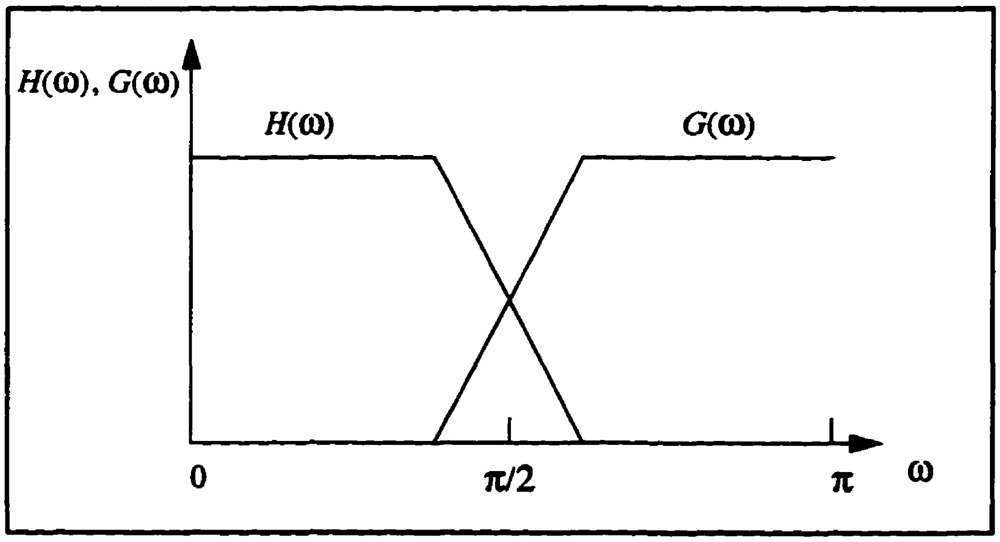


Fig. 3.4. Frequency responses of QMF.

The QMF filter bank structure is shown in Fig. 3.5. To guarantee the perfect reconstruction, *i.e.*, $\hat{x}(n) = x(n)$, the QMFs must satisfy some conditions.

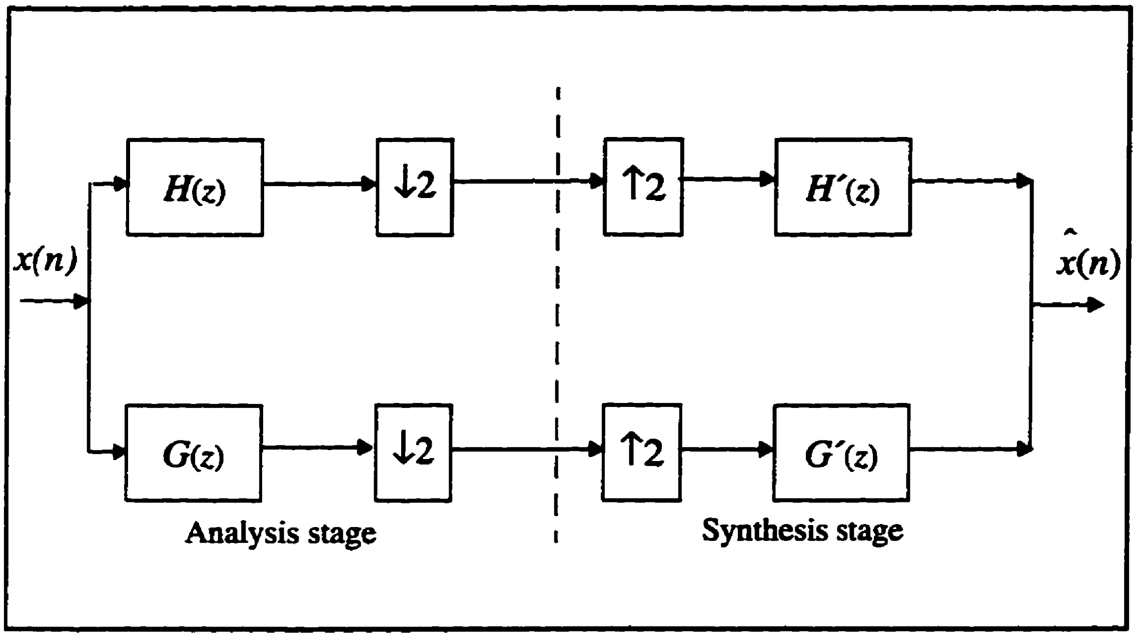


Fig. 3.5. Two channel QMF bank.

In summary of [AkHa92], the conditions for perfect reconstruction in the two-channel QMF are

$$(1)\ \frac{H'(z)}{G'(z)} = \frac{G(-z)}{H(-z)} \tag{3.10}$$

and this can be achieved by the following selection of a

$$H'(z) = -G(-z) \tag{3.11}$$

$$G'(z) = H(-z) \tag{3.12}$$

(2) $$\sum_{k=0}^{N-1} h(k)g(k) = 0 \tag{3.13}$$

This demonstrates that the impulse response $\{h(k)\}$ and $\{g(k)\}$ are orthogonal to each other. In the time domain, we have

$$g(k) = (-1)^{k+1}h(N-1-k) \tag{3.14}$$

$$h'(k) = h(N-1-k) \tag{3.15}$$

$$g'(k) = (-1)^{k}h(k) \tag{3.16}$$

3.4 Wavelets from Filter Banks, DWT

The objective of this section is to connect the wavelet transform (DWT) and QMFs. Let *h(k)* and *g(k)* be the filter cofficients of lowpass and highpass QMF respectively. The wavelet, $\psi(t)$, can be constructed from a scaling function. The scaling function $\phi(t)$ will be linked to the choice of a lowpass QMF, *h(k)*. The *g(k)* can be found by Eq. 3.14.

3.4.1 Multiresolution Analysis

A decomposition of the whole function space into individual subspaces following a multiple scales is know as multiresolution [StDS95]. In multiresolution, there are two families of subspaces, V_i and W_i, $-\infty < i < \infty$. The spaces V_i are increasing as i increases. The spaces W_i are the differences between the V_i. In summary of [ErHJ96], the family of subspaces V_i must satisfy the following four requirements.

$$(1)\ V_i \subset V_{i+1} \text{ and } \bigcap V_i = \{0\} \text{ and } \overline{\bigcup V_i} = L^2 \tag{3.17}$$

$$(2)\ f(t) \in V_i \Leftrightarrow f(2t) \in V_{i+1} \tag{3.18}$$

$$(3)\ f(t) \in V_0 \Leftrightarrow f(t-k) \in V_0 \tag{3.19}$$

$$(4)\ V_0 \text{ has an orthonormal basis } \{\phi(t-k)\}. \tag{3.20}$$

where $-\infty < t < \infty$ and $k = 0, 1, 2, \ldots$ is the shift time. A function $f(t)$ in the whole space has a projection in each subspace. Those projections contain more and more of the full information in $f(t)$. The projection in V_i is called $f_i(t)$.

Now, we identify the second family of subspaces. W_i contains the new information $\Delta f_i(t) = f_{i+1}(t) - f_i(t)$. This is the detail at level i. From the viewpoint of individual functions,

$$f_i(t) + \Delta f_i(t) = f_{i+1}(t) \tag{3.21}$$

and from the viewpoint of the subspaces they lie in, these are

$$V_i \oplus W_i = V_{i+1} \tag{3.22}$$

where $\oplus$ denotes the direct sum. Here, W_i is the orthogonal complement of V_i, within the large space V_{i+1}. Each function in V_{i+1} is the sum of two orthogonal parts, f_i in V_i and Δf_i in W_i. When we start from V_0, and add up details (subspaces), then we have

$$V_0 \oplus W_0 \oplus W_1 \oplus \ldots \oplus W_i = V_{i+1} \tag{3.23}$$

For the functions in these subspaces, this equation is simply

$$f(t) + \Delta f_0(t) + \Delta f_1(t) + \ldots + \Delta f_i(t) = \Delta f_{i+1}(t) \tag{3.24}$$

The construction of wavelets has succeeded by finding the V_i, through the scaling function $\phi(t)$. Then the wavelet spaces W_i are the differences between V_{i+1} and V_i. Similarly for filter banks, we design a QMF lowpass filter *h(k)*, then the highpass filter *g(k)* can be found by Eq. 3.14. To maintain this analogy between continuous and discrete wavelet transforms, Fig. 3.6 shows the filter banks. At each step, the highpass filter produces the new detail Δf_i in W_i.

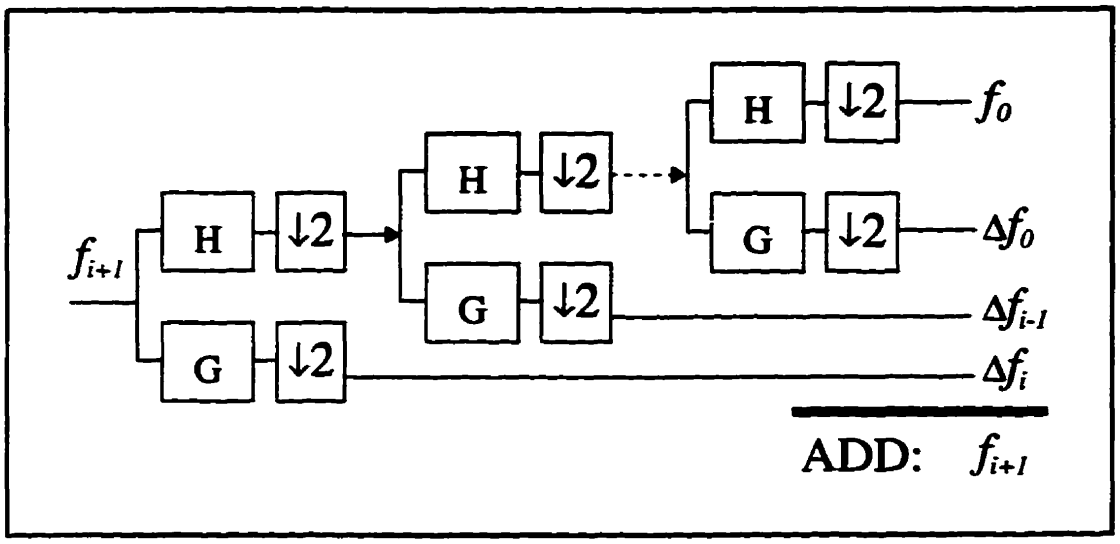


Fig. 3.6. Multiresolution of filter banks.

3.4.2 Dilation and Wavelet Equations

Summarizing [ErHJ96], the dilation equation

$$\phi(t) = \sqrt{2}\sum_k h(k)\phi(2t-k) \tag{3.25}$$

is a direct consequence of $V_0 \subset V_1$. There will be a finite set of coefficients $h(0), \ldots, h(N)$ when the function $\phi(t)$ is supported on $[0, N]$. From the orthogonality of the basis $\{\phi(t-k)\}$, we have the following characteristics of the coefficients $h(k)$:

Unit vector: $\sum |h(k)|^2 = 1$

Double-shift: $\sum h(k)\, h(k-m) = 0$ for $m \neq 0$

Let us consider the Daubechies 4 wavelet, Daub4, as shown in Fig. 3.7 [PTVF92].

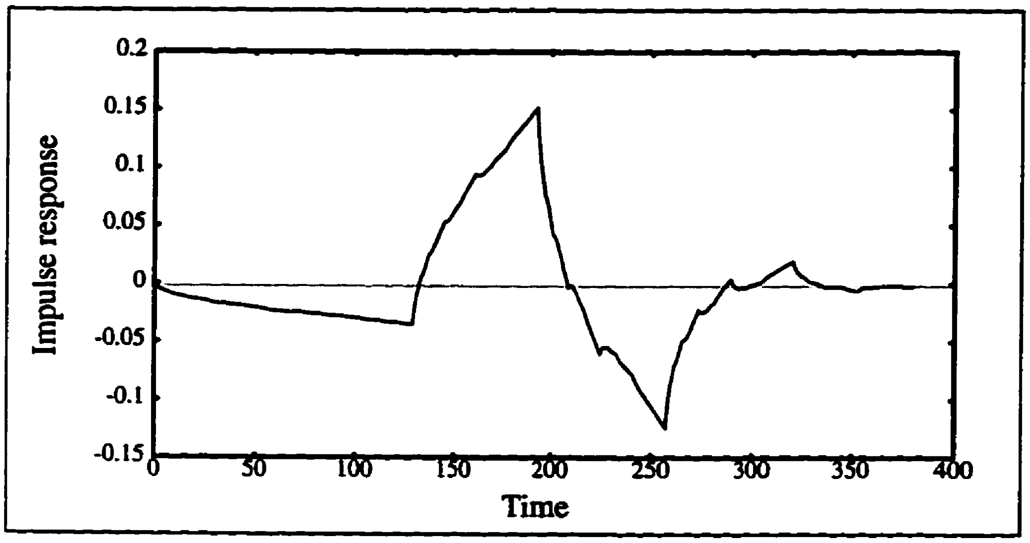


Fig. 3.7. Mother wavelet of Daub4.

The response $h(k)$ has four coefficients

$h(0), h(1), h(2), h(3) = 1 + \sqrt{3}, 3 + \sqrt{3}, 3 - \sqrt{3}, 1 - \sqrt{3}$ times $\sqrt{2}/8$.

Their sum equals $\sqrt{2}$ and their sum of squares is unity. They are orthogonal to their double shifts, because $h(0)h(2) + h(1)h(3) = 0$. From these coefficients, we construct $\phi(t)$ by solving the following dilation equation

$$\phi(t) = \sqrt{2} \sum_{k=0}^{3} h(k)\, \phi(2t-k) \tag{3.26}$$

The scaling functions $\phi(2^i t - k)$ are orthogonal at each scale separately. They are not orthogonal across scales. The function $\phi(t)$ in V_0 is also in V_1. So $\phi(t)$ is not orthogonal to the basis function $\phi(2t-k)$ in V_1. Orthogonality across scales comes from the wavelet subspaces W_i and their basis function $W_{ik}(t)$.

The translates of $\psi(t)$ span W_0. The translates of $\psi(2^i t)$ span W_i. Those spaces are orthogonal because $W_0 \subset V_i$ and $V_i \perp W_i$. The wavelet equation (Eq. 3.27) produces the wavelet directly from the scaling function:

$$\psi(t) = \sqrt{2}\sum g(k)\phi(2t-k) \tag{3.27}$$

The $g(k)$ can be obtained from a flip construction of $h(k)$,

$$g(k) = \sum_k (-1)^{N-k} h(N-k) \tag{3.28}$$

For example, from the Daubechies wavelet (Daub4), we start with the four *h*s. Then the flip construction (Eq. 3.28) gives the four gs (to normalize, multiply by $\sqrt{2}/8$):

$g(0), g(1), g(2), g(3) = 1 - \sqrt{3}, -(3 - \sqrt{3}), 3 + \sqrt{3}, -(1 + \sqrt{3})$.

Their sum is zero. Their sum of squares (normalized) is 1. They are orthogonal to their double shifts, because the *h*s are orthogonal to their double shifts. From these coefficients the wavelet equation gives the Daubechies 4 wavelet $\psi(t)$

$$\psi(t) = \sqrt{2}\sum_{k=0}^{3} g(k)\phi(2t-k) \tag{3.29}$$

3.5 Wavelet Packets

The pyramid structured wavelet transform decomposes a signal into a set of frequency channels that have narrower bandwidths in the lower frequency region. The transform is suitable for signals consisting primarily of smooth components, so that their information is concentrated in the low frequency regions. However, it may not be suitable for quasi-periodic signals such as speech signals whose dominant frequency channels are located in the middle frequency region. To analyze quasi-periodic signals, the concept of wavelet bases has been generalized to include a library of modulated waveform orthonormal bases, called wavelet packet bases [Wick94] or simply wavelet packets.

In summary of [ChKu93], the library of wavelet packet basis functions $\{WP_n\}_{n=0}^{\infty}$ can be generated by

$$WP_{2n}(t) = \sqrt{2}\sum_k h(k)\, WP_n(2t-k) \tag{3.30}$$

$$WP_{2n+1}(t) = \sqrt{2}\sum_k g(k)\, WP_n(2t-k) \tag{3.31}$$

where the function $WP_0(t)$ can be identified with the scaling function ϕ and $WP_1(t)$ with the mother wavelet ψ. Then, the library of wavelet packet bases can be defined to be the collection of orthonormal bases, composed of functions of the form $WP_n\left(2^l t-k\right)$, where $l, k, \in Z$, $n \in$ N. Each element of the library is determined by a subset of the indices: a scaling parameter l, a localization parameter k, and an oscillation parameter n. The tree-structured wavelet transform as shown in Fig. 3.8, provides an algorithmic approach to represent a function in terms of a certain wavelet packet basis.

The 2-D wavelet (or wavelet packet) basis functions can be expressed by the tensor product of 1-D wavelet (or wavelet packet) basis functions along the horizontal and vertical directions. The corresponding 2-D filter coefficients can be expressed as

$$\begin{aligned} h_{LL}(k,l) &= h(k)\,h(l), \\ h_{LH}(k,l) &= h(k)\,g(l), \\ h_{HL}(k,l) &= g(k)\,h(l), \\ h_{HH}(k,l) &= g(k)\,g(l) \end{aligned} \tag{3.32}$$

where the first and second subscripts denote the lowpass and highpass QMF characteristics in x and y directions, respectively.

The simplest case is based on tree structured filter banks, which generates wavelet packets [RaVH96].

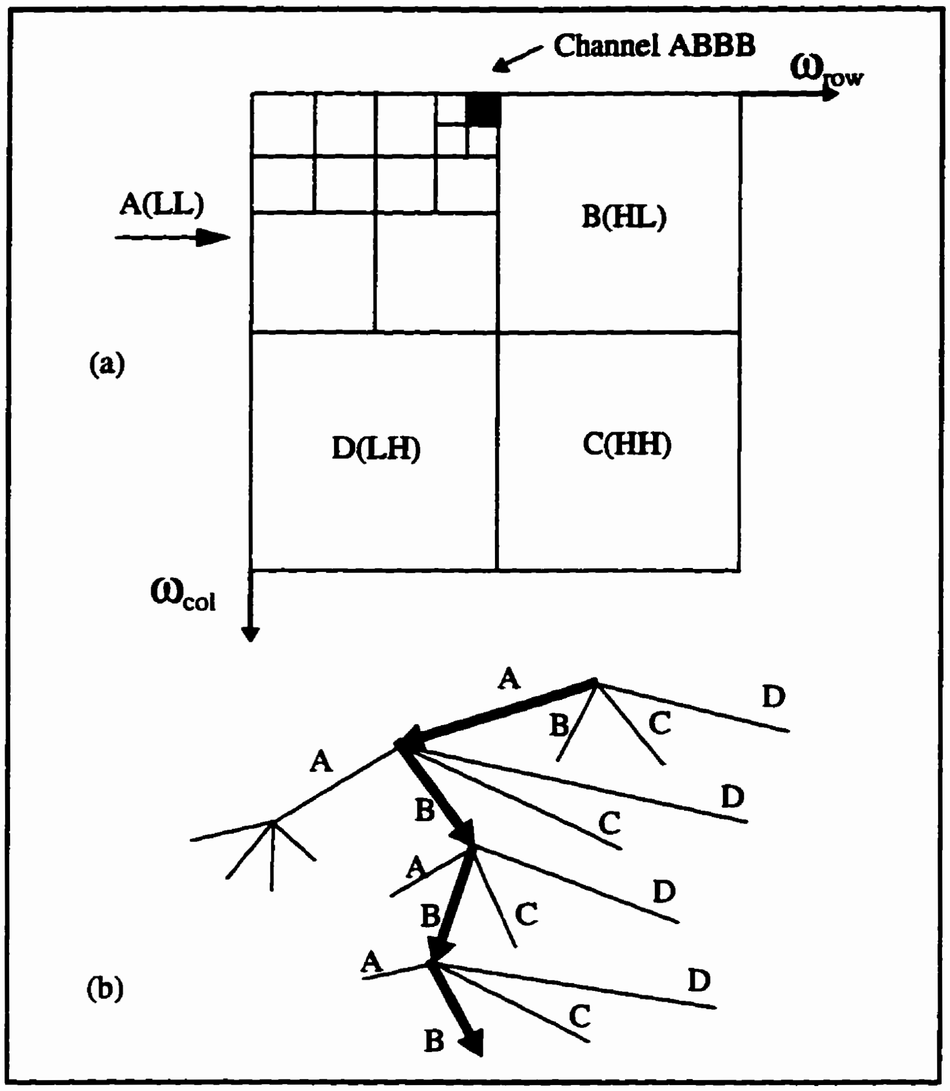


Fig. 3.8. The meaning of channel ABBB in (a) frequency decomposition of wavelet transform domain (wavelet packet) and (b) tree structured representation.

3.6 Summary of Chapter 3

In this chapter, we explained the reason for using the wavelet transform instead of the Fourier transform or the short-time Fourier transform. The dilation and translation characteristics of the wavelet transform are advantageous for analyzing a non-stationary signal, such as fingerprint images. The discrete wavelet transform (DWT) can be implemented by choosing proper quadrature mirror filters (QMFs). In section 3.3 and 3.4 we gave a summary of QMFs and the connection between wavelets and filters. The Daubechies wavelet (Daub4) that will be used in Chap. 5 was discussed as an example. A summary of a wavelet packet was given in Sec. 3.5. The wavelet packets can provide more decompositions than DWTs and will be used for our experiments in Chap. 5.

CHAPTER 4

FRACTAL AND MULTIFRACTAL DIMENSIONS

The concept of dimension has various mathematical connotations, the most common of which is known as the topological (or Euclidean) dimension D_E which is an integer. Thus point, line, plane (surface), and space (volume) have dimension *zero*, *one*, *two*, and *three*, respectively. The appearance of objects such as the Cantor set (Fig. 4.1) and Koch curve (Fig. 4.2) triggered a search for a better definition of dimension. As a result, the concept of dimension was further advanced from integer to fraction.

The fractal dimension D_F is usually a fraction. A fractal dimension can be interpreted simply as the "degree of meandering" (or roughness, or brokenness, or irregularity) of an object [Kins95]. This concept derived from the self-similarity property (power law), and is related to the logarithmic quotient between the change in the object. There are at least 19 different fractal dimensions [Kins95], but in this chapter we only focus on the Hausdorff (alias box counting, or capacity, or Kolmogorov) dimension (D_H), *Rényi* dimension (D_q) and Mandelbrot (alias multifractal) dimension (D_{MAN}) measurements.

We will often use the fractal dimensions either to measure the complexity of an object, or to assess the number of degrees of freedom of the underlying chaotic process, such as strange attractors (see Section 4.2). Strange attractors often do have a structure; they are self-similar (one scale) or self-affine (more than one scale). And they have fractal dimensions that hold important clues for our attempts to understand chaotic systems.

Recently, Michael Barnsley developed the *Iterated Function System* (IFS) [BaHu93] which can approximate many ordinary images by a superposition of the strange attractors of a limited number of fractal (affine) transformations and each transformation occurring with a given probability [Vrsc95][FaHV94]. With the IFS concept, we can treat images as strange attractors. Furthermore, the wavelet coefficients of an image can also be considered to be an approximation of a strange attractor [ErKi97] and can be analyzed by their multifractality.

4.1 Fractals and Self-Similarity

4.1.1 Self-Similarity

Some mathematical fractals, such as the Cantor set (Fig. 4.1) and the Sierpinski gasket (Fig. 4.3), have properties of both self-similarity and symmetry (mirror symmetry). To avoid any confusion between symmetry and self-similarity we first review their definition.

Symmetry itself is one of the most fundamental concepts of human thought. By symmetry we mean an invariance to change: something stays the same, in spite of some potentially significant alteration. Mirror symmetry; that is, invariance to flipping sides, is perhaps the most widely noticed symmetry. The most common type of invariance to changes in size is called self-similarity or, if more than one scale factor is involved, self-affinity.

Self-similarity comes in many different shapes and forms. Kinsner [Kins95], described different kinds of self-similarities such as continuous, discrete, deterministic, and probabilistic. A few cases of self-similarity are mathematically exact (Fig. 4.1 - 4.4); however, most instances in the real world are only approximately self-similar.

4.1.2 Fractals

There is no single definition of fractals. Mandelbrot defines it as follows: "a fractal is a set for which the Hausdorff-Besicovitch (fractal) dimension strictly exceeds the topological dimension". Fractals can also be called nondifferentiable sets [Kins95]. Robert L. Devaney defines it as "a subset in R^n which is self-similar and whose fractal dimension exceeds its topological dimension" [Kins94]. Hans Lauwerier defines it as "a geometrical figure that consists of an identical motif repeating itself on an ever-reduced scale" [Lauw91].

Now let us look at some mathematical fractals as examples. All of them have the properties of self-similarity and symmetry.

Georg Cantor thought up something that one could call the oldest fractal, *Cantor point-set* [Lauw91]. He started with the closed unit interval [0, 1], then wiped away the open middle third $\left(\frac{1}{3}, \frac{2}{3}\right)$ and repeated the process on the remaining two segments (N = 2) of length $r = \frac{1}{3}$ (see Fig. 4.1).

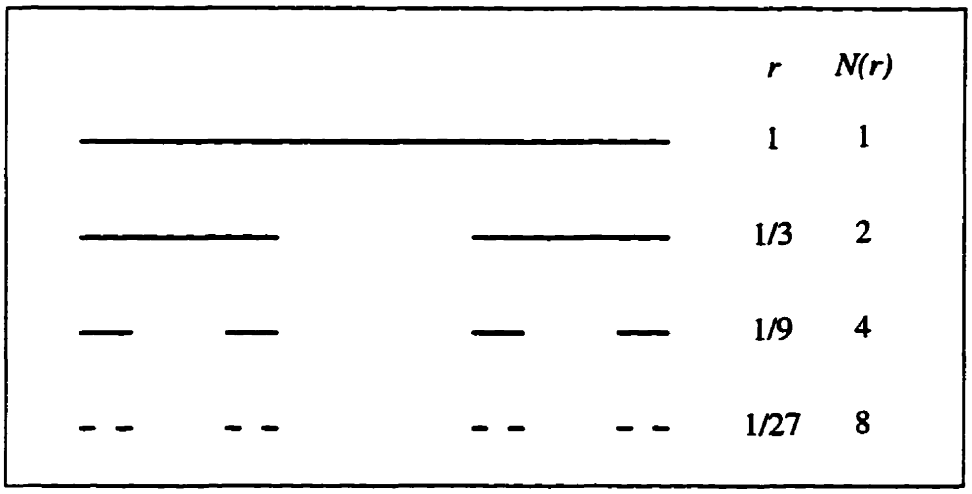


Fig. 4.1. Cantor set.

The *Koch curve*, illustrated in Fig. 4.2, is generated by the following iterated procedure. We start off with a line-segment, the base, and remove the middle third. Then, we fill the gap with the upright sides of an equilateral triangle. In this way, a bent line consisting of four equal line-segments, the so-called motif, results. In the next phase, each one of the four line-segments is taken as a base and replaced by the corresponding scaled-down motif.

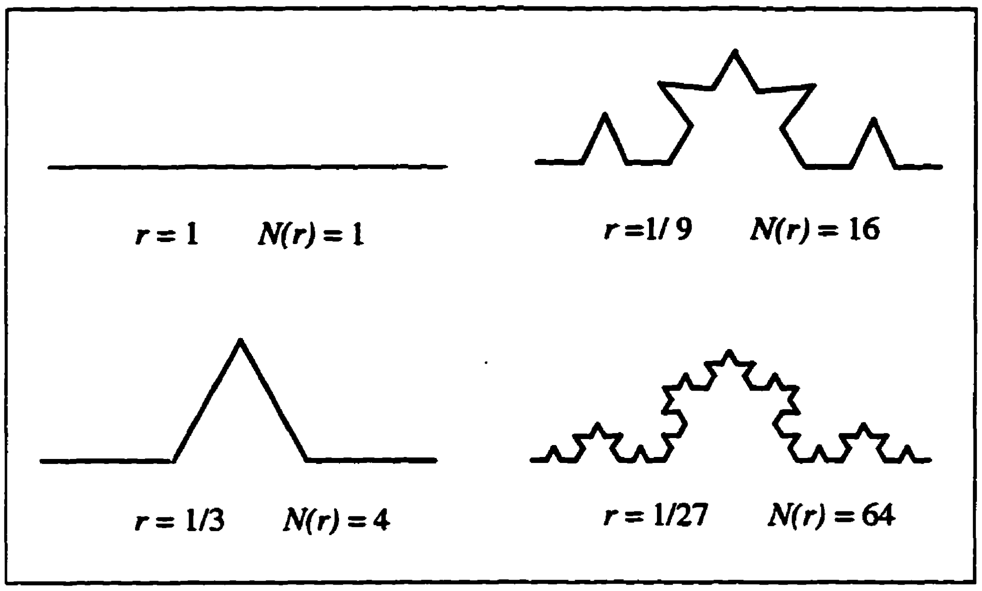


Fig. 4.2. Koch curve.

The *Sierpinski gasket* (Fig. 4.3) is obtained by starting with an equilateral triangle. Then, we divide this equilateral triangle into four smaller equilateral triangles, of which the middle one is removed. This way, a triangular hole is produced. With the three remaining solid equilateral triangles we proceed in just the same way, so that three smaller triangular holes appear. In Figure 4.3, we have carried out just one more step, but the process can be repeated indefinitely.

The generator of the *Sierpinski carpet* is the unit square with the central square of side length $\frac{1}{3}$ deleted. In the next iteration, the central squares of side length $\frac{1}{9}$ are removed from the eight remaining squares of side length $\frac{1}{3}$. Infinite iteration produces the *Sierpinski carpet*, approximated by the dark area in Fig. 4.4.

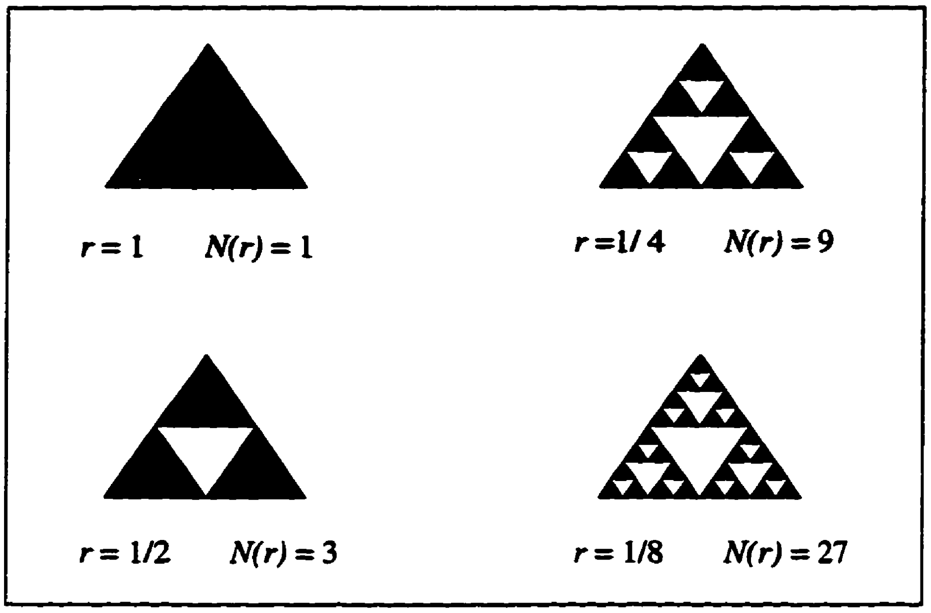


Fig. 4.3. Sierpinski gasket.

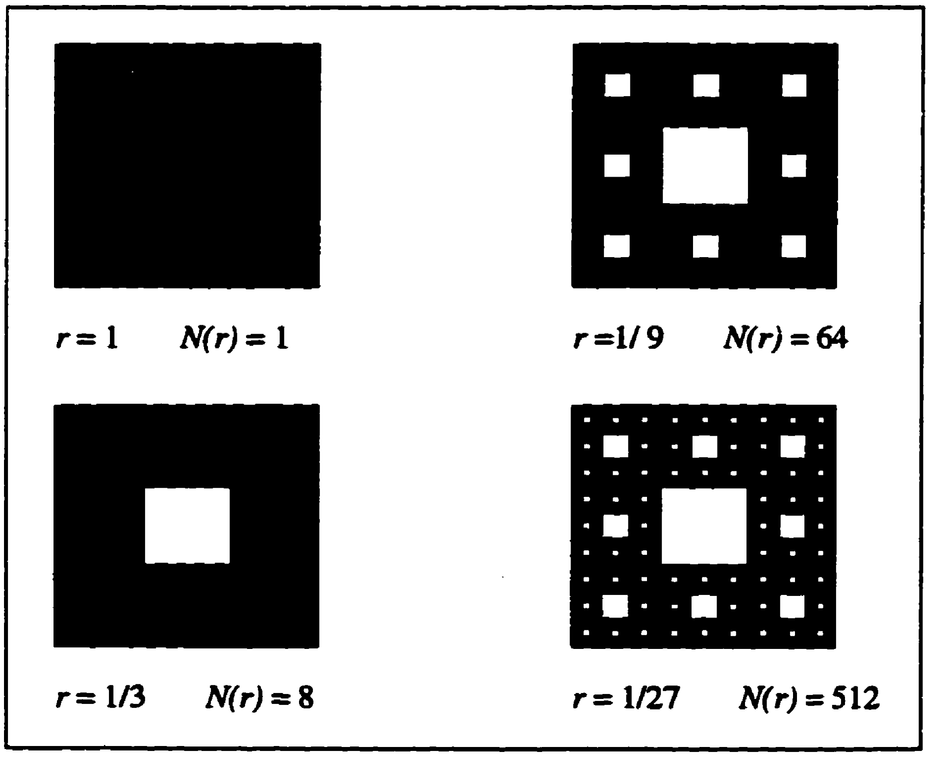


Fig. 4.4. Sierpinski carpet.

4.1.3 Hausdorff Dimension of a Fractal

When Mandelbrot introduced the concept of a fractal in 1977, he also introduced the term fractal dimension, a concept of dimension he based on a definition given by Hausdorff in 1919 [Kins95].

We use a concept of a neighbourhood, which could be a small region on any shape (often called a *ball* or *volume element* or *vel* for short [Kins95]), centred on a point on, or in the vicinity of, the fractal. Note, the neighbourhood in 1D could be a line segment. We start the measurement with a box (can be any shape) of size r_k and count the number $N_k(r_k)$ of boxes required to cover the object (fractal). Figure 4.5 illustrates the measurement.

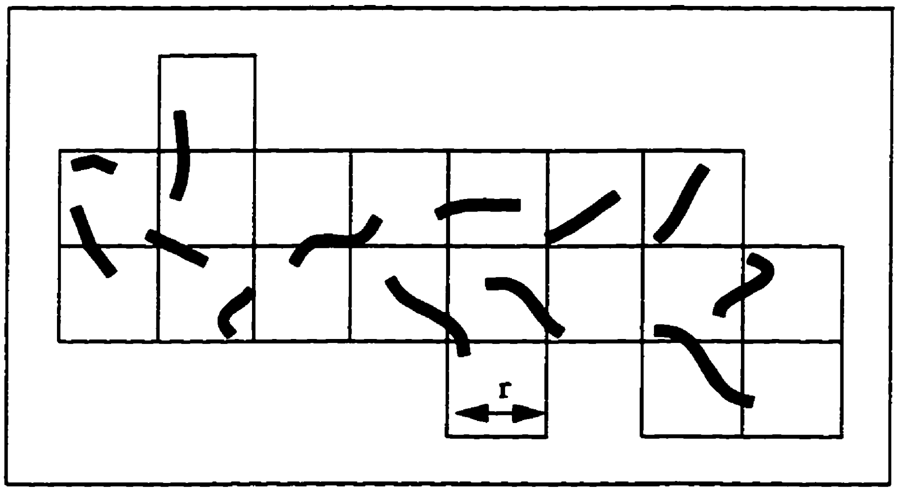


Fig. 4.5. r-mesh cubes, scale = $1/r$, $N_r = 19$. (After [Kins95])

Repeat the measure with an ever-reducing size r_k where r_k is the kth measure size. If there is a power-law relation between r_k and $N_k(r_k)$ as shown below

$$N_k(r_k) = \left(\frac{1}{r_k}\right)^{D_H} \tag{4.1}$$

then the Hausdorff dimension is defined as

$$D_H = \lim_{r \to 0} \frac{\log N_k(r_k)}{\log\left(\frac{1}{r_k}\right)} \tag{4.2}$$

Table 4.1 lists more Hausdorff dimensions (D_H) calculated for different mathematical fractals.

Table 4.1. D_H of the Cantor set, Koch curve, Sierpinski gasket and Sierpinski carpet.

Fractals	Hausdorff Dimension (D_H)
Cantor set (Fig. 4.4)	$D_H = \frac{\log 2}{\log 3} \approx 0.63$
Koch curve (Fig. 4.5)	$D_H = \frac{\log 4}{\log 3} \approx 1.26$
Sierpinski gasket (Fig. 4.6)	$D_H = \frac{\log 3}{\log 2} \approx 1.58$
Sierpinski carpet (Fig. 4.7)	$D_H = \frac{\log 8}{\log 3} \approx 1.89$

4.2 Multifractals

Fractals have greatly increased our ability to describe nature. There are many phenomena in various sciences that require a generalization of the fractal concept to include intricate structures with more than one scaling exponent. Many of these matters are in fact characterized by an entire spectrum of components, of which the Hausdorff dimension (D_H) is only one. The generalized fractals used to deal with these cases are called multifractals.

Many strange attractors of nonlinear dynamic systems are multifractals. In this section we take the Hénon strange attractor as an example and then introduce some multifractal dimensions which can analyze these self-affine objects (multifractals).

4.2.1 Strange Attractor

Strange attractors are among the most important realizations of multifractals. An attractor of an iteration $x_{n+1} = f(x_n)$ is a single point or an indecomposable bounded set of points to which starting values x_0 from the attractor's "basin of attraction" converge as $n \to \infty$. A strange attractor is an attractor for which the iterates x_n depend sensitively on the initial x_0; that is, arbitrarily close initial values will become macroscopically separated for a sufficiently large n. Poincaré sections of continuous strange attractors or strange attractors resulting from maps are fractal dusts with a fractal dimension smaller than the Euclidean dimension of the embedding space [Schr91][GrOY87].

Michael Hénon introduced a two-dimensional, two-parameter map which is well known as the Hénon strange attractor. The Hénon strange attractor is generated by the following equations:

$$x_{i+1} = y_i + 1 - ax_i \tag{4.3}$$

$$y_{i+1} = bx_i \tag{4.4}$$

For $a = 1.4$ and $b = 0.3$, the Hénon strange attractor after 10,000 iterations is shown in Fig. 4.6(a). Figure 4.6(b) and (c), show successive enlargements of the small square in the preceding figure. When the scale is changed (zoom in) we can observe a Cantor-set like structure. This suggests that we may regard the attractor in Fig. 4.6(c), for example, as being essentially a Cantor-set of approximately parallel curves.

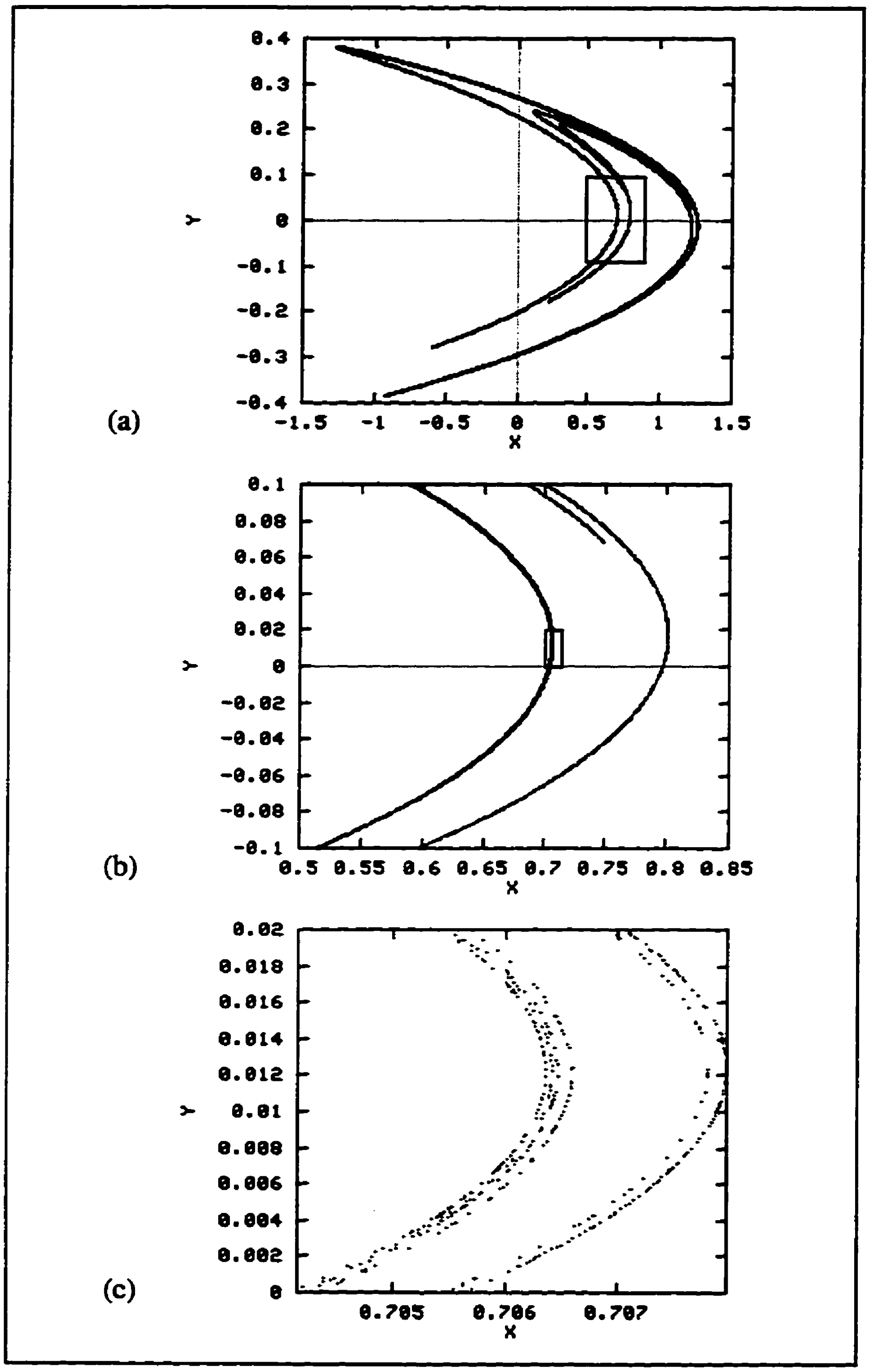


Fig. 4.6. Hénon strange attractor. (a) the entire attractor, (b)enlargement of portion shown by square in part (a), (c) another enlargement of squared portion in (b).

4.2.2 Can an Image be Treated as a Strange Attractor?

By a real world image, we mean an idealized entity that does not really exist, just as a mathematical line or triangle does not exit [BaHu93]. The real world images capture intuitive feelings that we have about the visual world. These feelings relate to the way we interact visually with the physical world; we can look closely at one thing, and from far away at another. Images feel as though they are analog entities that belong to an infinitely divisible world. These feelings are not exactly correct; they do not fit precisely with reality, but they allow our brains to cope with the complexity of what they perceive. These feelings are true more or less, over a range of scales, and to an approximation and they contain valid information about the nature of the physical world.

Moreover, a real world image cannot be defined as a physical object, such as an actual photograph, or the actual pattern of light that falls upon the retina of the eye. We usually perceive photographs (analog or digital) as static objects. That means the resolution (scale) is bounded, as shown in Fig. 4.7. But this point of view tells us little about the evolution or generation of a given growing structure [PeJs92].

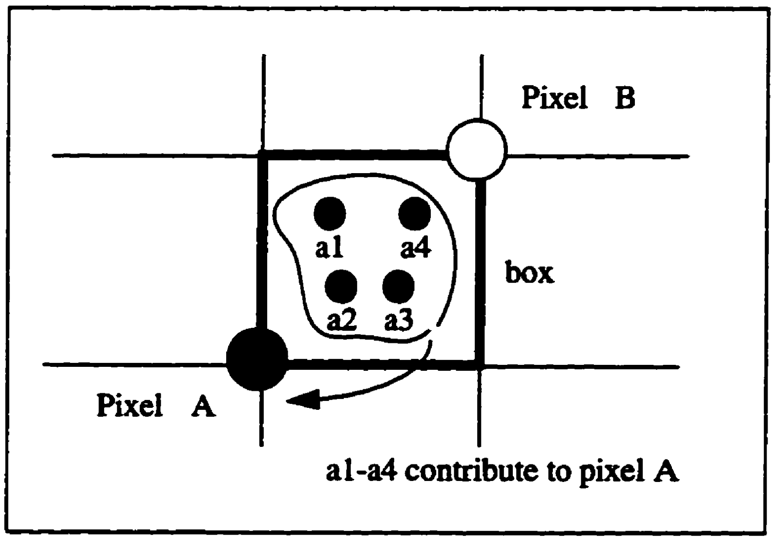


Fig. 4.7. The grayscale of each pixel of a digital image (scanned photograph) stands for the energy in a certain area (*e.g.*, a box) that is restricted by the resolution.

IFS fractals can be used to approximate real world images; they have the property that they are themselves models for real world images, and, at the same time, they are fully defined by finite strings of zeros and ones. Figure 4.8 shows the *Barnsley's fern* that is generated by the following *fractal transformations* (FT) [FaHV94]

$$f_1\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}0 & 0\\0 & 0.16\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}0\\0\end{bmatrix} \tag{4.5}$$

$$f_2\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}0.85 & 0.04\\-0.04 & 0.85\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}0\\0.16\end{bmatrix} \tag{4.6}$$

$$f_3\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}0.2 & -0.26\\0.23 & 0.22\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}0\\0.16\end{bmatrix} \tag{4.7}$$

$$f_4\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}-0.15 & 0.28\\0.26 & 0.24\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}0\\0.44\end{bmatrix} \tag{4.8}$$

with the probability distribution $p_1 = 0.01$, $p_2 = 0.85$, $p_3 = 0.07$ and $p_4 = 0.07$.

Figure 4.9 illustrates the successive enlargements of the Barnsley's fern image. When we scale up (zoom in), we see that the enlarged image turns to Cantor-set like dust (Fig. 4.9(c)). Compared to the enlarged Hénon strange attractor as shown in Fig. 4.6(c), we find that they have the same following properties, (x', y') (next position) is non-predictable, two different pixels never overlap (i.e., $(x'_1, y'_1) \neq (x'_2, y'_2)$ if $(x_1, y_1) \neq (x_2, y_2)$), both are sensitive to initial conditions. Actually, Barnsley's fern is a strange attractor and we can present it as a grayscale image, as shown in Fig. 4.9(a).

To answer the question "Can an image be treated as a strange attractor?" our answer is positive. Barnsley showed that most ordinary images can be approximated by a superposition of several strange attractors generated by IFS. That is also the concept he uses for the fractal compression of images.

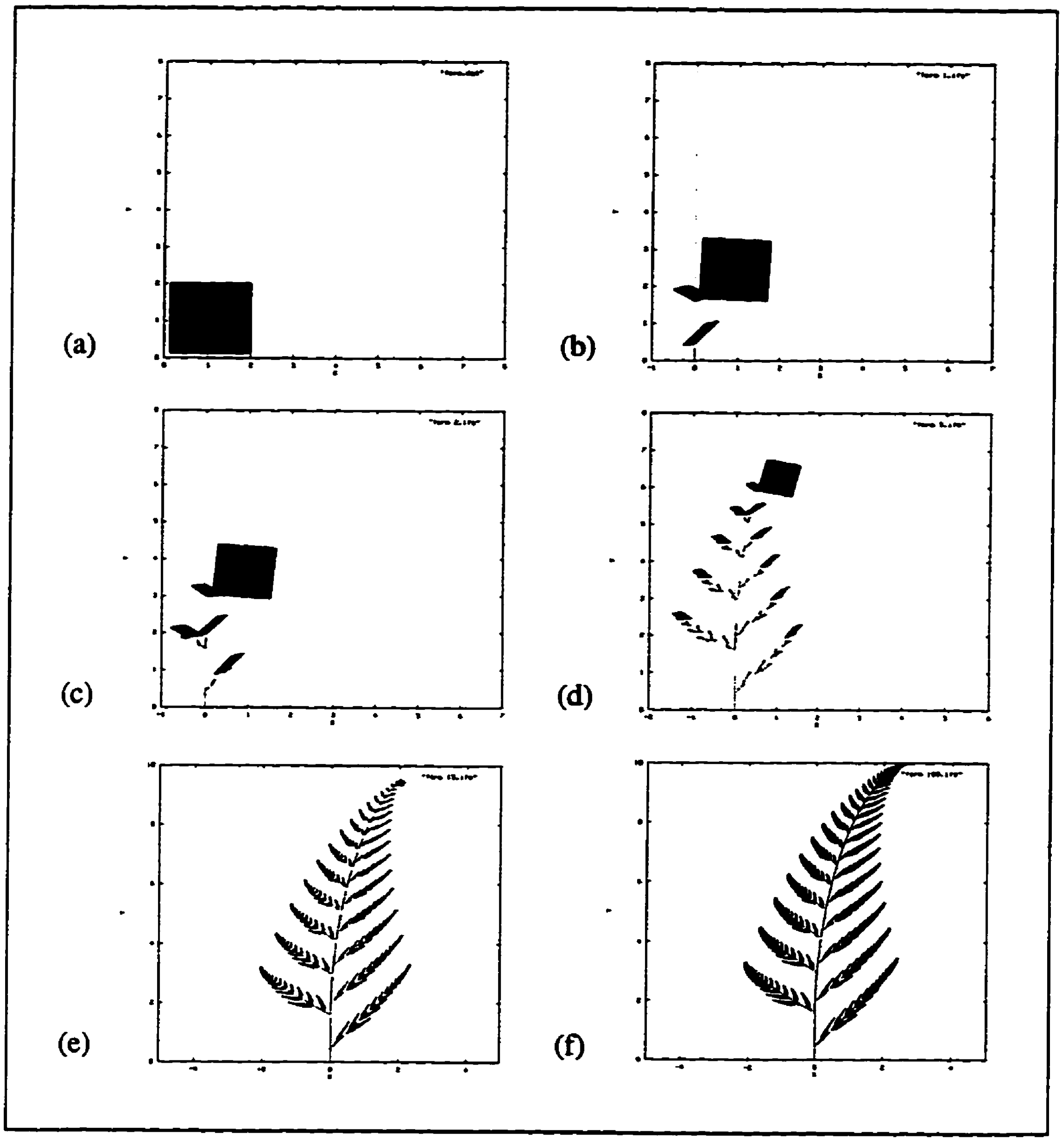


Fig. 4.8. Barnsley's fern image, generated by Eq. 4.5-4.8. The IFS begins from a starting image, (a) a square and the iterated images after (b) one iteration, (c) two iterations, (d) five iterations, (e) 15 iterations and (e) 100 iterations.

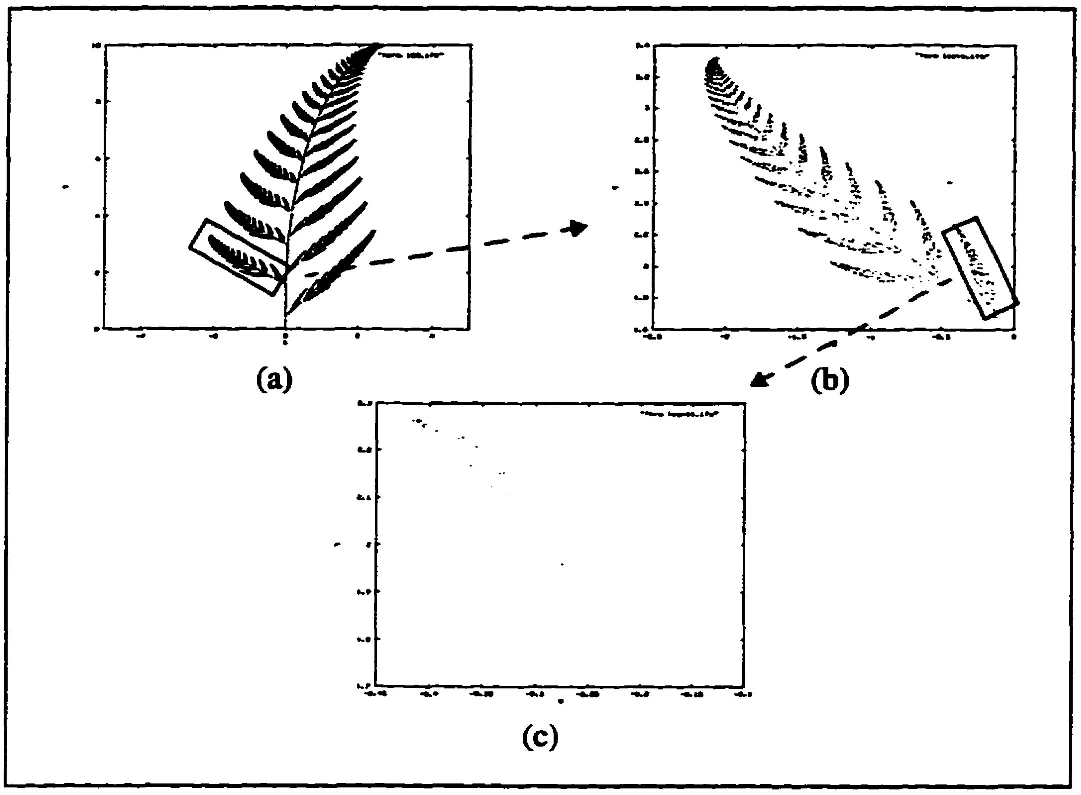


Fig. 4.9. (a) Barnsley's fern image, (b) enlarged portion of the square in (a), (c) enlarged portion of the square in (b). In (c) we find that the image turns into points.

4.2.3 Multifractal Dimensions

As we already know, self-similarity and self-affinity are indicators of fractal structures. This means that the mathematical fractals could be scaled indefinitely towards either of the two limits: 0 and ∞. However, morphological self-similarity may not imply self-similarity in other measures such as entropy, spectrum, or variance [Kins95].

The morphological-based dimensions, such as D_H, can be used if the distribution of a measure (such as probability) is uniform (*i.e.*, if the fractal is homogeneous) or the information about the distribution is not available. For those inhomogeneous fractals (such as the Hénon strange attractor) we have to use entropy-based dimensions. This class of dimensions includes: information dimension, correlation dimension, Rényi generalized entropy dimension, and multifractal (Mandelbrot) dimension. In this section we only focus on the last two dimensions because the information and correlation dimensions are a special case ($q = 1$ for information and $q = 2$ for correlation) of the Rényi dimension.

4.2.3.1 Rényi Dimension

In 1955 the mathematician A. Rény attempted to generalize the concept of entropy of a probability distribution, H_q (Eq. 4.11). First let us consider a covering of N_r vels, each with diameter r. Assume that the *j*th vel is intersected by the stochastic fractal (such as Hénon strange attractor) with a frequency n_j. We can then define the probability (relative frequency) of *j*th vel as:

$$p_j := \lim_{N_T \to \infty} \frac{n_j}{N_T} \tag{4.9}$$

where the total number of points in all the vels is

$$N_T = \sum_{j=1}^{N_r} n_j \tag{4.10}$$

The Rény entropy is defined as:

$$H_q = \frac{1}{q-1} \log \sum_{j=1}^{N_r} p_j^q \tag{4.11}$$

where q is called the moment order. Note that, q is not necessarily an integer. If we assume the following power-law relation

$$\frac{1}{q-1} \sum_{j=1}^{N_r} p_j^q \sim r^{D_q} \tag{4.12}$$

then the Rényi dimension is

$$D_q := \lim_{r \to 0} \frac{1}{q-1} \frac{\log \sum_{j=1}^{N_r} \left(p_j^q\right)}{\log (r)} \tag{4.13}$$

or

$$D_q := \lim_{k \to \infty} \frac{1}{q-1} \frac{\log \sum_{j=1}^{N_k} \left(p_j^q\right)}{\log (r_k)} \tag{4.14}$$

For $q = 0$, the Rényi dimension becomes the similarity dimension, D_S and Hausdorff dimension, D_H. The following equation shows this relation.

$$D_{q=0} := \lim_{k \to \infty} \frac{1}{0-1} \frac{\log \sum_{j=1}^{N_k} \left(p_j^0\right)}{\log (r_k)} = \lim_{k \to \infty} \frac{\log N_k}{\log\left(\frac{1}{r_k}\right)} = D_H \tag{4.15}$$

Similarly, we find that D_q becomes the information dimension, D_I for $q = 1$ and the correlation dimension, D_C for $q = 2$. The magnitude of D_H, D_I, and D_C satisfy the following relation:

$$D_C \le D_I \le D_H \tag{4.16}$$

For $q \to \infty$, only the highest probability p_{max} in the sum in equation 4.14 counts. Hence the Rényi dimension D_∞ $(q \to \infty)$ becomes

$$D_{\infty} = \lim_{k \to \infty} \lim_{q \to \infty} \frac{1}{q-1} \frac{\log \sum_{j=1}^{N_k} p_j^q}{\log (r_k)} = \lim_{k \to \infty} \frac{\log p_{max}}{\log (r_k)} \tag{4.17}$$

Conversely, for $q \to -\infty$ the smallest probability p_{min}, controls the sum. Thus, the Rényi dimension $D_{-\infty}$ $(q \to -\infty)$ becomes

$$D_{-\infty} = \lim_{k \to \infty} \lim_{q \to -\infty} \frac{1}{q-1} \frac{\log \sum_{j=1}^{N_k} p_j^q}{\log (r_k)} = \lim_{k \to \infty} \frac{\log p_{min}}{\log (r_k)} \tag{4.18}$$

In general, for any two dimensions with different q,

$$D_q \geq D_{q'} \qquad \text{for} \qquad q < q' \tag{4.19}$$

Thus, D_q is a monotonically nonincreasing function of q (see Fig. 4.10). One exception is a strictly self-similar fractal, such as Sierpinski gasket, where the D_q has the same value in the entire range $-\infty \leq q \leq \infty$. For a given object, the spread of the curve is an indicator of its multifractality.

For a given strange attractor F *(e.g.,* Hénon strange attractor*)*, if there is more than one corresponding Rényi dimension D_q for a range of q, then we can say that the attractor F is described by multiple fractals F_q. Since q may be noninteger, a continuous spectrum of fractals may be required to describe the structure of F.

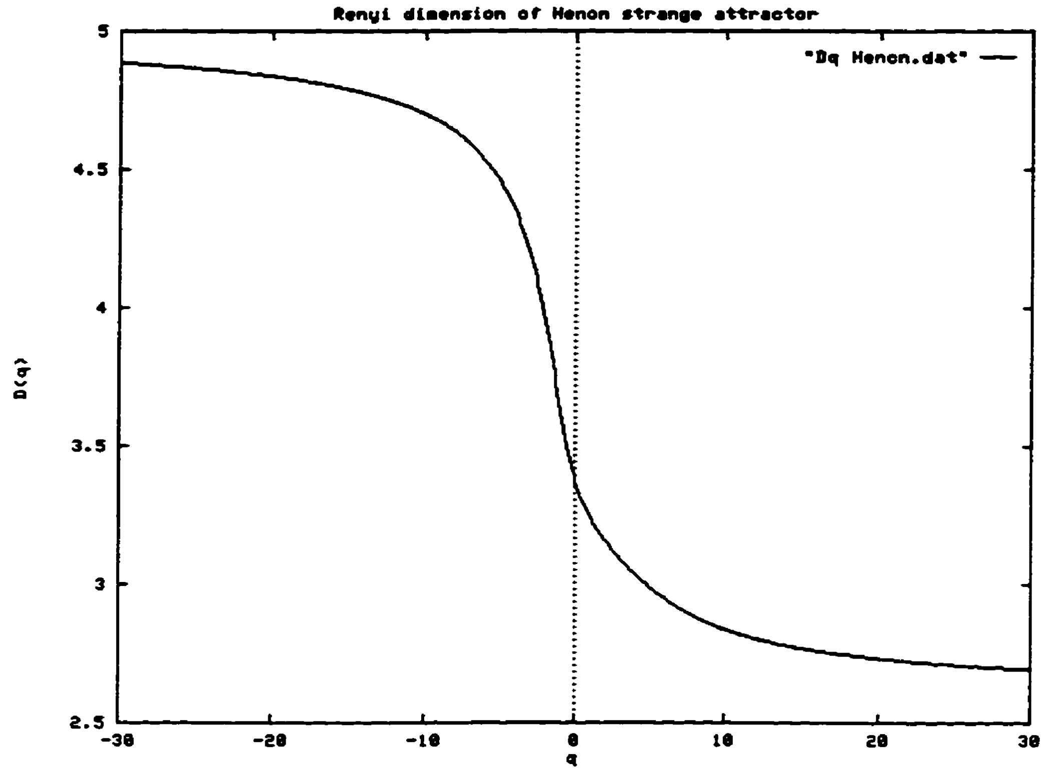
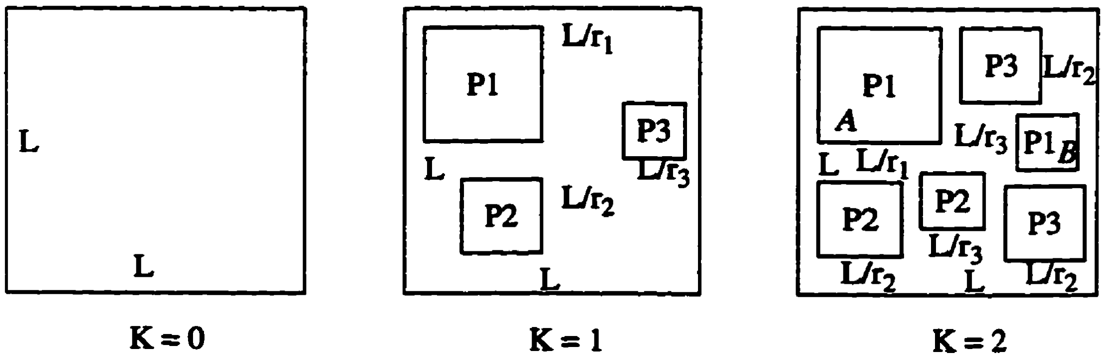


Fig. 4.10. Rényi dimension D_q of Hénon strange attractor.

4.2.3.2 Mandelbrot Dimension

Consider a recursive process generating a nonuniform fractal (*i.e.*, rescaled regions of different sizes r_j) with inhomogeneous measures (*i.e.*, regions with different probabilities P_j at each rescaled region). An example of such a process is shown below.



For a nonuniform fractal with inhomogeneous distribution, the local relationship is

$$p_j(r_j) \sim r_j^{\alpha_j} \quad \text{for} \quad \alpha_j = [-\infty,\infty] \tag{4.20}$$

where α_j is a noninteger which depends on the selected region of the measure. For K = 2 we find that there are two vels with the same probability $P1$ but of different size, r_1 and r_3, respectively. If we suppose $P1 = 0.5$, $r_1 = 0.5$ and $r_3 = 0.125$, then from Eq. 4.16 we have
$\alpha_1 = 1$ and $\alpha_3 = 3$. Since r_3 is smaller than r_1 and both regions have the same probability, we can say the density in region B is higher than A and that implies $\alpha_3 > \alpha_1$. Thus, the exponent α corresponds to the strength of the local singularity of the measure, and is called the Hölder exponent.

In addition, we can consider how many vels have the same α. In general, the number of vels with a specific α has the following power-law relation:

$$N_\alpha(r) \sim \frac{1}{r^{f(\alpha)}} \tag{4.21}$$

where $f(\alpha)$ is the fractal Mandelbrot dimension, D_{MAN}, of the α subset [Kins95]. The exponent α is analogous to the energy, while $f(\alpha)$ is analogous to the entropy as a function of energy.

The Hölder exponent α can be obtained by taking the derivative with respect to the Rényi exponent q

$$\alpha_q := \frac{d}{dq}[(q-1)D_q] \tag{4.22}$$

and $f(\alpha) \equiv f_q$ is obtained from

$$f_q := q\alpha_q - (q-1)D_q \tag{4.23}$$

Figure 4.11 shows the Mandelbrot dimension D_{MAN} of the Hénon strange attractor. At its maximum, $f(\alpha)$ is equal to the Hausdorff dimension and also to the 0*th* order Rényi dimension D_0.

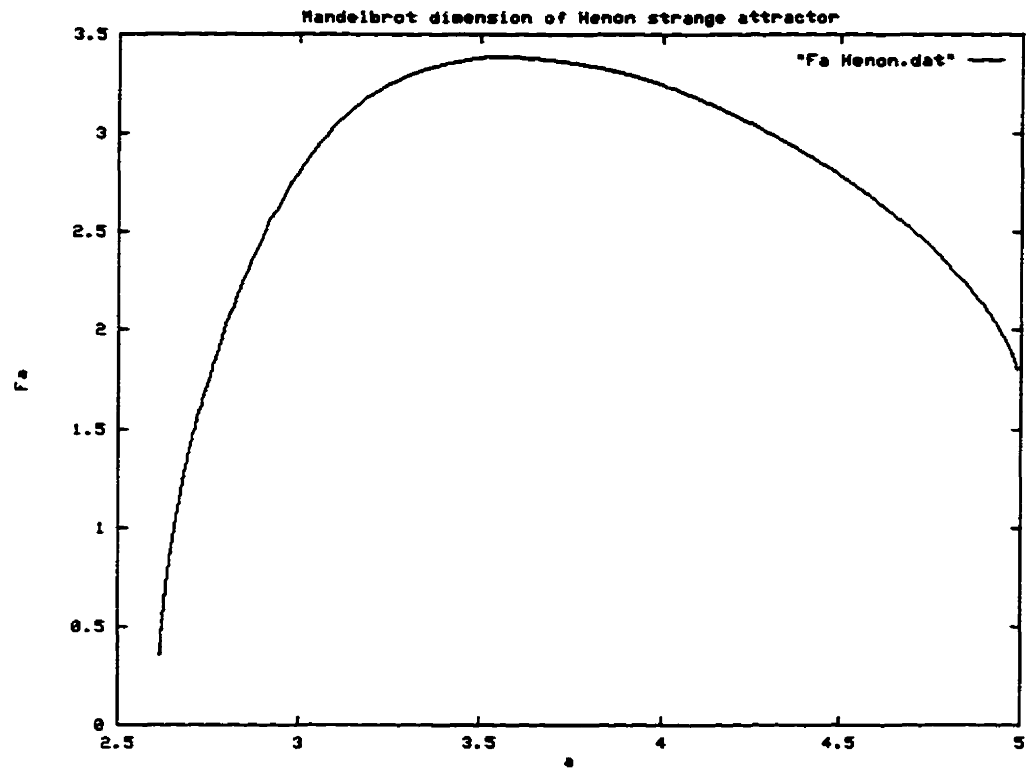


Fig. 4.11. Mandelbrot dimension D_{MAN} of Hénon strange attractor.

4.2.3.3 Mandelbrot Dimension of a Segmented Image

An image can be treated approximately as a strange attractor (see Section 4.2.2). A segmentation of an image is part of the image and can be analyzed by its multifractality. The relationship between the image and its segmentation still holds in their corresponding multifractal spectra. Since in the definition of Mandelbrot dimension (Eq. 4.23), the exponent α is analogous to the energy, while $f(\alpha)$ is analogous to the entropy as a function of energy. Now, we use *Lena* and its four segmentations (each segmentation occupies a quarter of the histogram of *Lena*) as an example to explain the relationship among segmentations in the multifractal spectra.

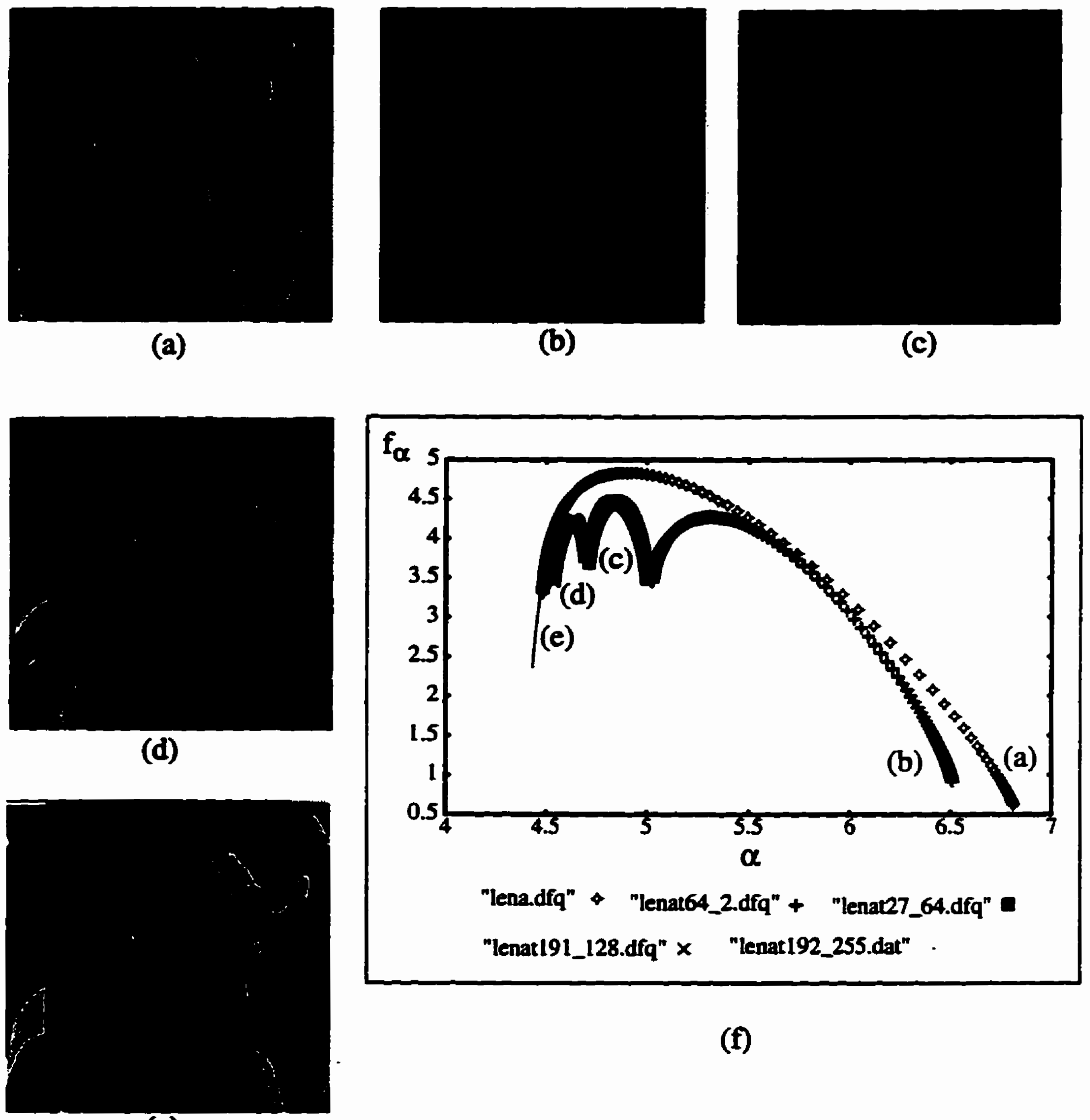


Fig. 4.12. (a) Original *Lena* image, segmentation of (a) with grayscale range of (b) 2 to 63, (c) 64 to 127, (d)128 to 191, (e) 192 to 255, (f) Mandelbrot dimension D_{MAN} of *Lena* and its segmentations ((b) - (e)).

Figure 4.12 shows that the α_{min} reflects p_{max} and α_{max} reflects p_{min}. This is an important property for multifractal spectra analysis. In chapter 5 we will use this property to analyze the multifractal spectra of wavelet coefficients.

4.3 Review of Chapter 4

In this chapter, we reviewed the concept of a fractal, multifractal and fractal dimensions. We selected several mathematical fractals, such as the Cantor set, Koch curve, and Sierpinski gasket, as examples to describe how to generate a fractal. The self-similarity and power law which are the essential parts of fractals were also discussed. The Hausdorff dimension can be used to measure those fractals mentioned above. Multifractals such as the Hénon strange attractor must be characterized by multifractal measures such as the Rényi dimension (D_q) and Mandelbrot dimension (D_{Man}). By applying the IFS, we showed that an image can be treated as a strange attractor and can be analyzed by the Mandelbrot dimension. We also gave an example of using D_{Man} to measure the segmentations of an image. In the next chapter we will apply (D_{Man}) to wavelet transform coefficients of fingerprint images and use the multifractality for image compression.

CHAPTER 5

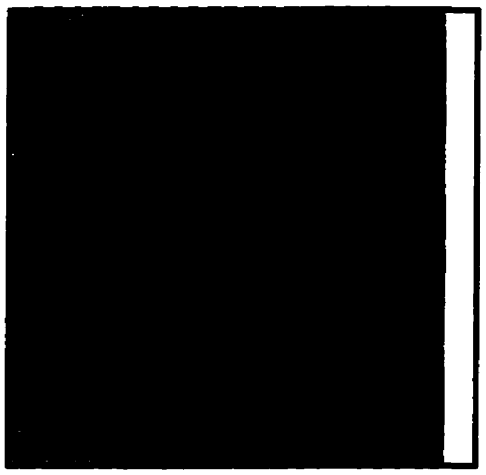
EXPERIMENTAL RESULTS AND DISCUSSION

Since our *human vision system* (HVS) can distinguish only a limited resolution and perceives an image as a nonlinear function of intensities, it appears that information (entropy) reduction techniques are the best candidates for fingerprint image compression. The quality requirement for a reconstructed fingerprint image is higher than other images (such as *Lena*) because the clarity of the ridge and the sharpness of the edge must be maintained. The lossy (entropy reduction) compression technique used in this Chapter is transform coding. Although we have discussed the concept of transform coding techniques for image compression in Chapter 2 and explained the reason for choosing the wavelet transform instead of other linear transforms (such as DFT) in Chapter 3, we would like to emphasize its advantage again. Figure 5.1 (a) shows a two dimensional gray-scale fingerprint, while Fig. 5.1 (b) presents it in 3D, and shows its DWT coefficients in (c). From Fig. 5.1 (b), we can see that the energy (gray-scale) is uniformly spread if we ignore the white band at the bottom of the image. However, a large fraction of its total energy is packed into relatively few transform coefficients as shown in Fig. 5.1 (c).

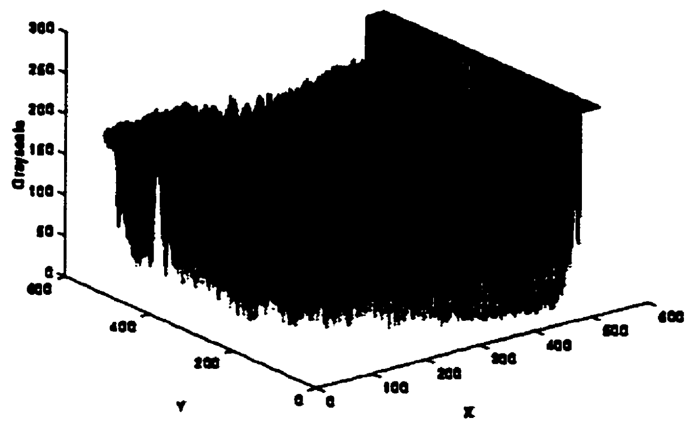
By using the property of the transformation (energy is presented by a few transform coefficients), we apply five different methods to compress fingerprint images. The purpose of utilizing so many different methods is to learn the advantages and disadvantages of each method and to try to develop a better solution. We propose two new techniques, *quadtree decomposition with multifractal analysis* (QDMA) and *wavelet packet with multifractal analysis* (WPMA), using the Mandelbrot singularity measure to analyze each subband of a wavelet packet that can help us to decide which subbands can be ignored for storage or transmission and which can be used for the *optimal bit allocation* (OBL) algorithm [BrBH93].

We have improved the *wavelet scalar quantization* method (WSQ, the FBI's gray-scale fingerprint image compression standard) [FeBI93] by adding the new technique, WPMA, described above. An alternative technique, *vector quantization* (VQ), is applied to fulfill the quantization techniques. A summary is also given at the end of this Chapter.

(a)



(b)



(c)

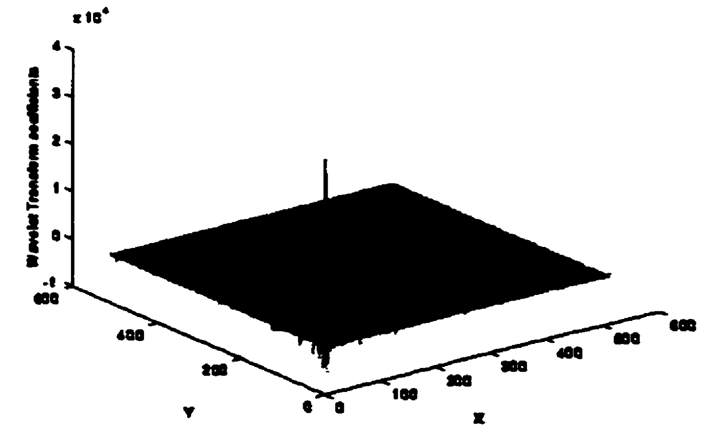


Fig. 5.1. (a) 512 x 512, 8 bpp fingerprint image, (b) 3D presentation of (a); note there is a wide white band at the bottom of (a), (c) DWT coefficients of (a) in 3D.

5.1 Zonal Filtering

Based on the concept that the larger the DWT coefficients the more important for reconstruction quality, the energy is concentrated in the corner as shown in Fig. 5.1 (c). The reason that we use a zonal filter (mask) to eliminate those coefficients outside the mask is that usually they are smaller than those inside the mask. The block diagram is shown in Fig. 5.2. The geometrical shape of the zonal filter mask is illustrated in Fig. 5.3. The experimental result is listed in Table 5.1 and Fig. 5.4 presents the reconstructed images.

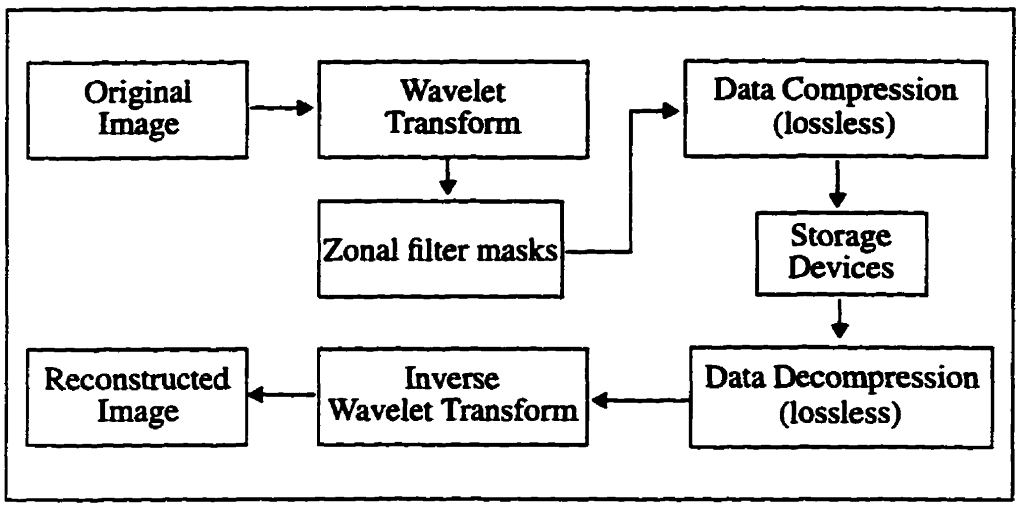


Fig. 5.2. A basic image compression scheme by using zonal filter mask.

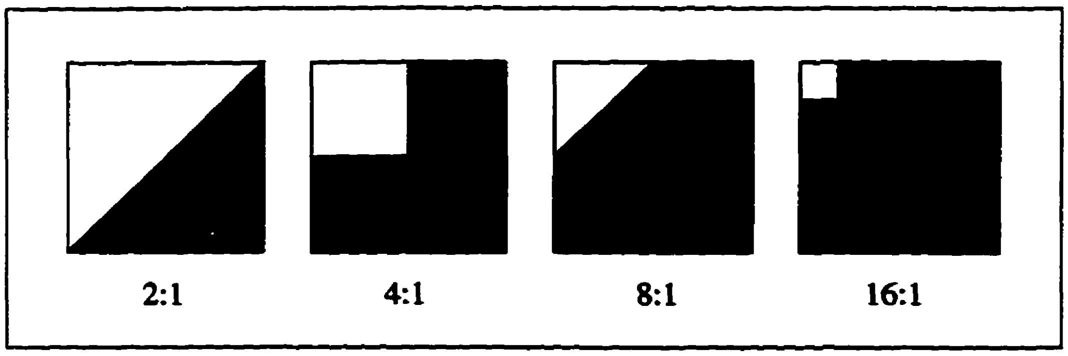
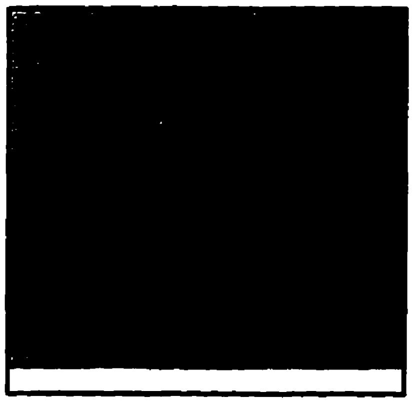
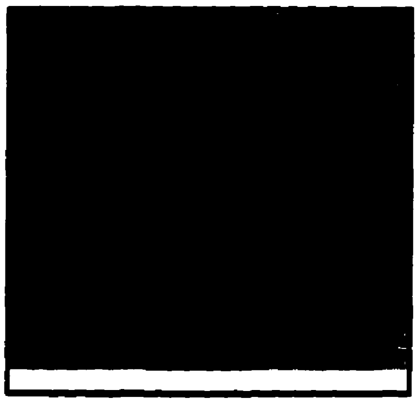


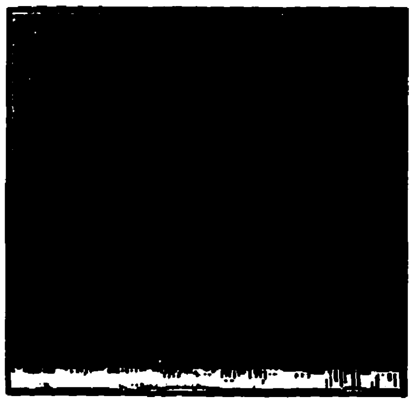
Fig. 5.3. Zonal filter masks, bright areas are passbands, dark areas are stopbands.

Table 5.1. Experimental results of using the zonal mask method.

Reconstructed images using various zonal masks	Compression ratio	PSNR
2 : 1	1.59 : 1 (5.03 bpp)	24.71 dB
4 : 1	2.83 : 1 (2.82 bpp)	26.13 dB
8 : 1	5.06 : 1 (1.58 bpp)	19.56 dB
16 : 1	9.53 : 1 (0.84 bpp)	18.19 dB


(a)


(b)


(c)

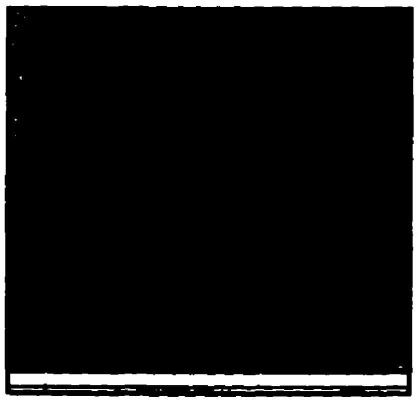

(d)

Fig. 5.4. (a) Original image (512x 512 8 bpp), reconstructed images of using (b) 4:1, (c) 8:1 and (c) 16:1 zonal masks in Fig. 5.3.

When we observe the reconstructed images in Fig. 5.4 we find that the quality of those using a diagonal shape zonal mask is poorer than those using a retangular shape. The reason is that the diagonal shape depresses the y direction (vertical) DWT coefficients more than the x direction (horizontal). That is why we can see the noise in the y direction in Fig. 5.4 (c). Because we are not using a *symmetric wavelet transform* (SWT) [Bris94] the boundary noise becomes significant when we eliminate all transform coefficients ouside the zonal mask. Of course the quality of the reconstructed images will get worse if we make the mask smaller.

5.2 Direct Thresholding

In wavelet decomposition the filter H is an averaging filter while its mirror counterpart G produces details. The wavelet coefficients correspond to details. When the details are small, they might be omitted without substantially affecting the "general picture" [ViMü93]. Thus the idea of direct thresholding wavelet coefficients is a way of removing unimportant details which are considered to be noise.

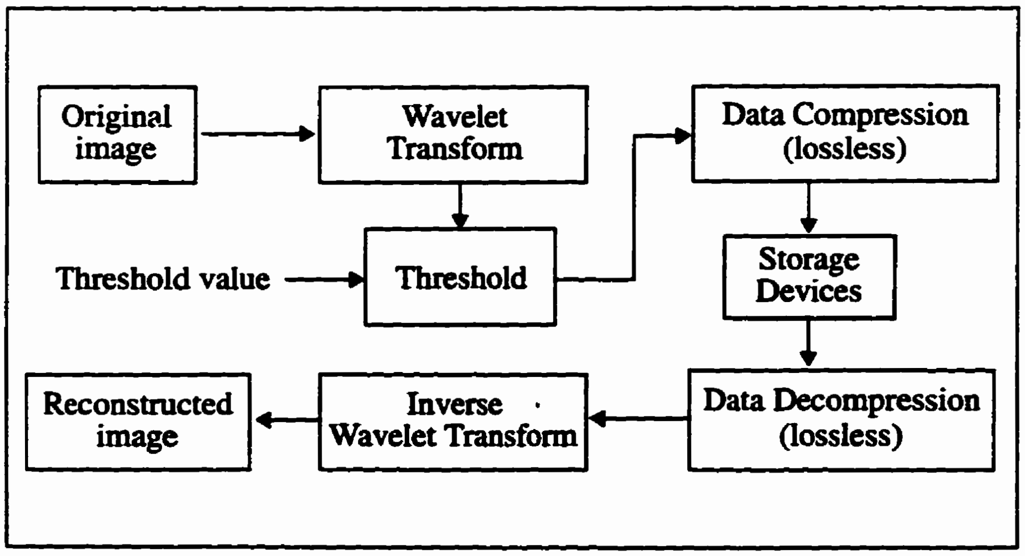


Fig. 5.5. A basic image compression scheme by using thresholding WT coefficients.

The absolute values of all wavelet coefficients are compared to a fixed threshold *T*. If the magnitude of the coefficient is less than *T*, the coefficient is replaced by zero

$$d_{jk}' = 0, \quad \text{if } d_{jk} < T$$

$$d_{jk}' = d_{jk}, \quad \text{if } d_{jk} > T \tag{5.1}$$

Figure 5.5 shows the block diagram and Table 5.2 lists the corresponding experimental results.

Table 5.2. Experimental results of using direct thresholding method.

Threshold value (*T*)	Truncation ratio (%)	PSNR (dB)	Compression ratio
1	22.52	50.2	1.24 : 1 (6.41 bpp)
2	37.21	48.6	1.39 : 1 (5.75 bpp)
3	47.87	46.3	1.55 : 1 (5.14 bpp)
4	55.47	44.3	1.72 : 1 (4.63 bpp)
5	61.15	42.6	1.89 : 1 (4.21 bpp)
7	69.14	40.1	2.22 : 1 (3.59 bpp)
10	76.69	37.4	2.72 : 1 (2.93 bpp)
11	78.45	36.8	2.89 : 1 (2.76 bpp)
13	81.36	35.6	3.23 : 1 (2.47 bpp)
15	83.62	34.6	3.58 : 1 (2.23 bpp)
30	91.99	30.3	6.35 : 1 (1.25 bpp)
35	93.34	29.3	7.41 : 1 (1.08 bpp)
40	94.31	28.6	8.45 : 1 (0.94 bpp)
45	95.08	28.1	9.58 : 1 (0.83 bpp)
55	96.21	26.9	11.75 : 1 (0.68 bpp)
60	96.63	26.5	12.92 : 1 (0.62 bpp)
90	98.13	24.6	20.42 : 1 (0.39 bpp)
120	98.88	23.3	28.98 : 1 (0.27 bpp)

In Fig. 5.6, we find that the compression ratio increases abruptly when the truncation is over 90 %. Unfortunately the quality of the reconstructed image (as shown in Fig. 5.7) becomes unacceptable (the ridges are smeared) when the truncation is over 90 %. The maximum compression ratio achievable by this method is in the range of 6 : 1 at 30 dB PSNR.

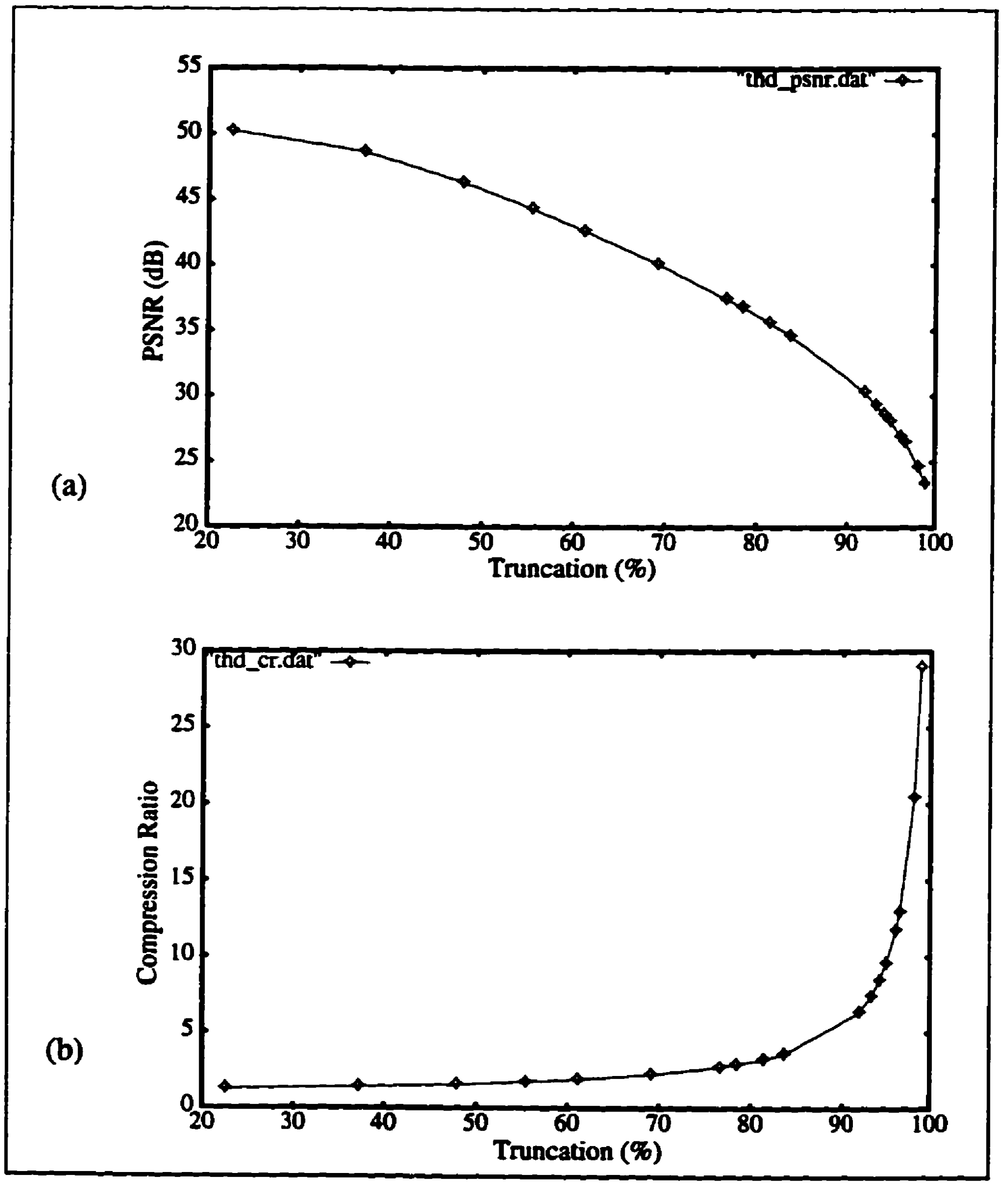


Fig. 5.6. Effects of % truncation on the (a) quality, (b) compression ratio for the images in Table 5.2.

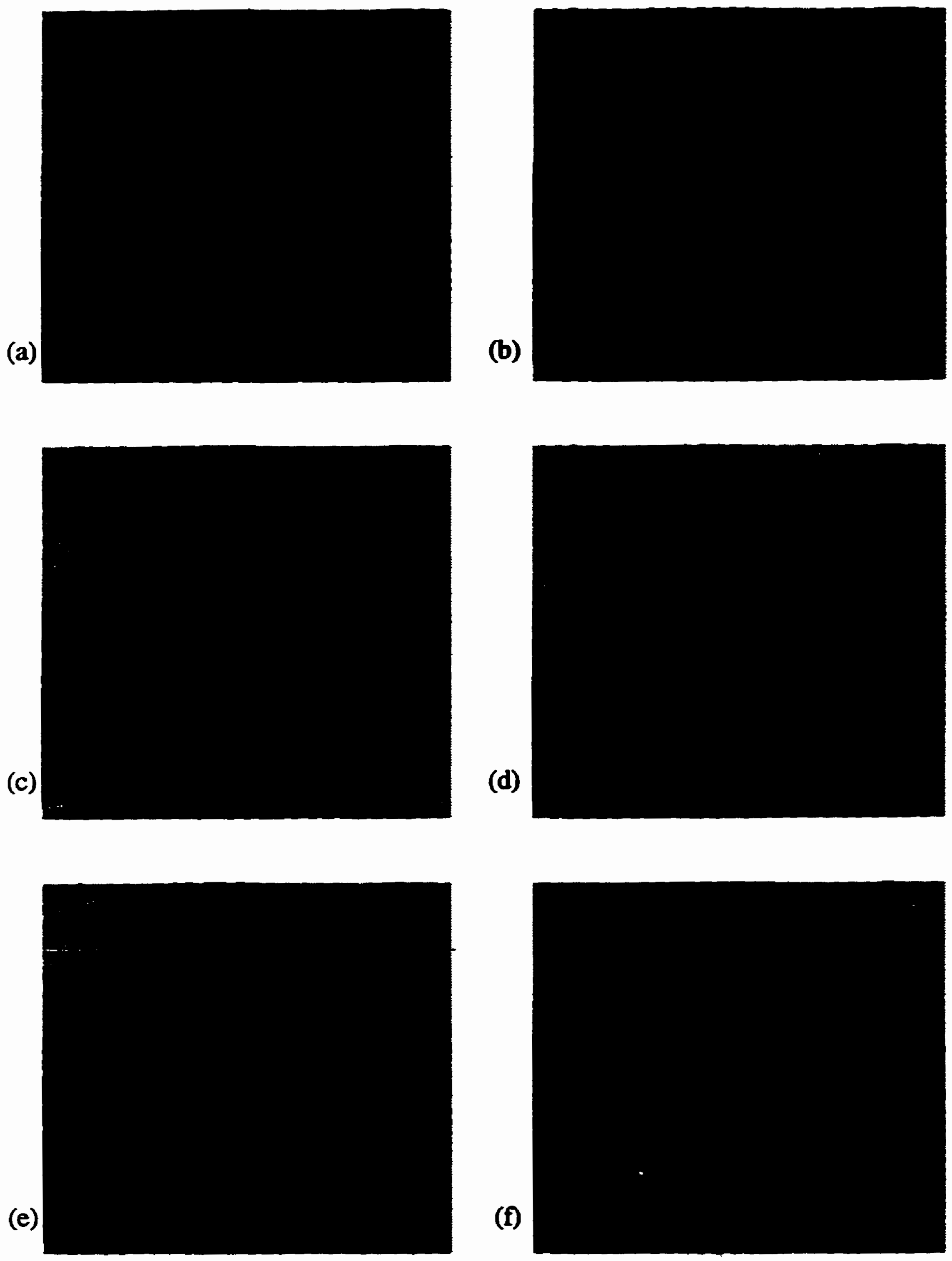


Fig. 5.7. (a) original image (512x 512 8 bpp), reconstructed images (part of Table 5.2) of truncation (b) 69%, (c) 92% , (d) 94%, (e) 96% and (f) 99%.

5.3 Zerotree

Zerotree is a data structure for compressing wavelet coefficients [Shap93]. A wavelet coefficient x is said to be insignificant with respect to a given threshold T if $|x| < T$. The *zerotree* is based on the hypothesis that if a wavelet coefficient at a coarse scale (which is called the *parent*) is insignificant with respect to a given threshold T, then all wavelet coefficients of the same orientation in the same spatial location at finer scales (which are called *descendants*) are likely to be insignificant with respect to T. Note that in particular, the larger coefficients are deemed more important than smaller coefficients regardless of their scale. For a QMF-pyramid subband decomposition, the parent-child dependencies are shown in Fig. 5.8. Table 5.3 lists the experimental result and the reconstructed images are shown in Fig. 5.9.

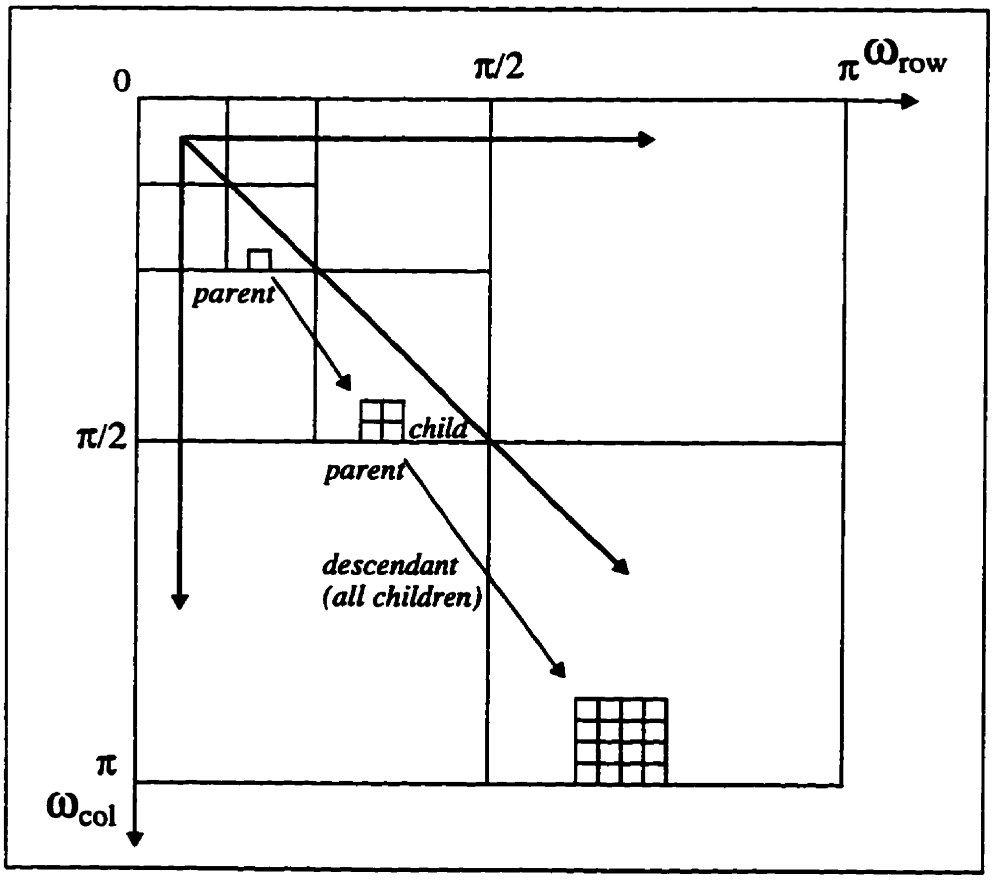
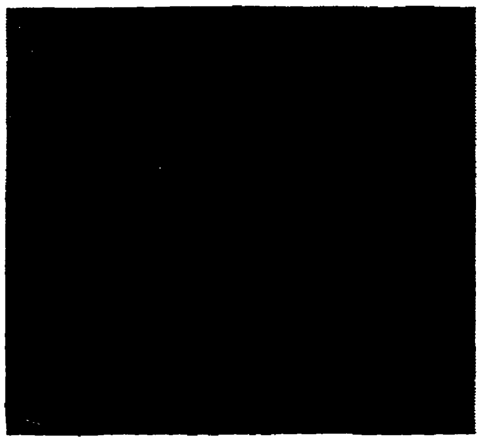


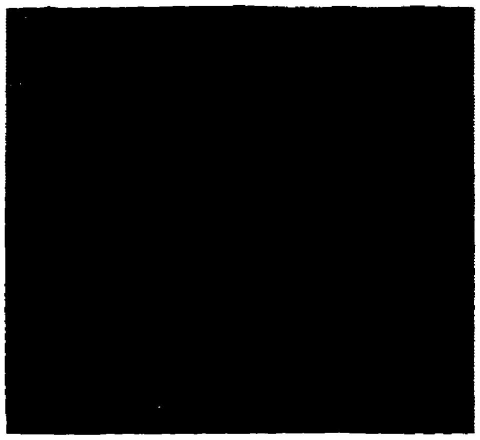
Fig. 5.8. Zerotree (3 scales) parent-child dependencies of subbands.

Table 5.3. Experimental results of using zerotree method.

Threshold (*T*)	PSNR (dB)	Compression ratio
16	33.76	5.37 : 1 (1.48 bpp)
32	29.63	9.52 : 1 (0.84 bpp)
64	26.09	17.08 : 1 (0.47 bpp)

(a) 

(b)

(c) 

(d)

Fig. 5.9. (a) Original image (512x 512 8 bpp), reconstructed images of various PSNR (b) 33.76 dB at 1.48 bpp , (c) 29.63 dB at 0.84 bpp and (c) 26.09 dB at 0.47 bpp.

In Table. 5.3 we find the compression ratio increases quickly when the threshold value T is over 64. However, the quality of the reconstructed images (as shown in Fig. 5.9) is getting unacceptable because the ridges are smeared. The maximum compression ratio in our experiments is in the range of 8 : 1 at 30 dB PSNR.

Shapiro reported that the compression ratio can reach 64 : 1 (0.125 bpp) with PSNR of 30.23 dB for *Lena* (512 x 512, 8bpp) [Shap93]. The advantage is gained by applying quantization. In our experiment the significant wavelet coefficients are compressed directly using the *GZIP* utility instead of quantization and then compression. Another interesting fact shown in [Shap93] is that a different image of a "girl" at the same compression ratio has a different reconstructed quality. For instance, with the same compression ratio 64 : 1 (0.125 bpp), the PSNR drops from 30.23 for *Lena* to 24.03 dB for *Barbara* (512 x 512, 8 bpp). Since fingerprints are special images the quality standard is higher than for others, such as *Lena*, *Barbara*. The idea of threshold from *parent* (i.e., the lowest frequency) seems against the transformation character (i.e., the energy is presented by these coefficients) and we will show it in the following two experiments (Section 5.4, 5.5).

5.4 Quadtree Decomposition with Multifractal Analysis (QDMA)

The QDMA method applies the same wavelet decomposition as zerotree but it can choose arbitrary decomposition levels (scales). The wavelet coefficients are sent to the *multifractal analysis block* where the Mandelbrot dimension (D_{MAN}) is calculated. By using the Mandelbrot singularity to measure each subband we can determine which subbands are more important than others. The selected subbands will be quantized, using the same optimal bit allocation (OBL) and quantizer algorithm as defined in the FBI specification [BrBH93] [HoPr92] [BrBr93]. The quantized coefficients will be compressed losslessly. We use the *GZIP* utility to compress and uncompress the quantized coefficients. The decoded quantized coefficients will be used for the inverse wavelet transform from which the reconstructed image is generated. The block diagram of QDMA is shown in Fig. 5.10.

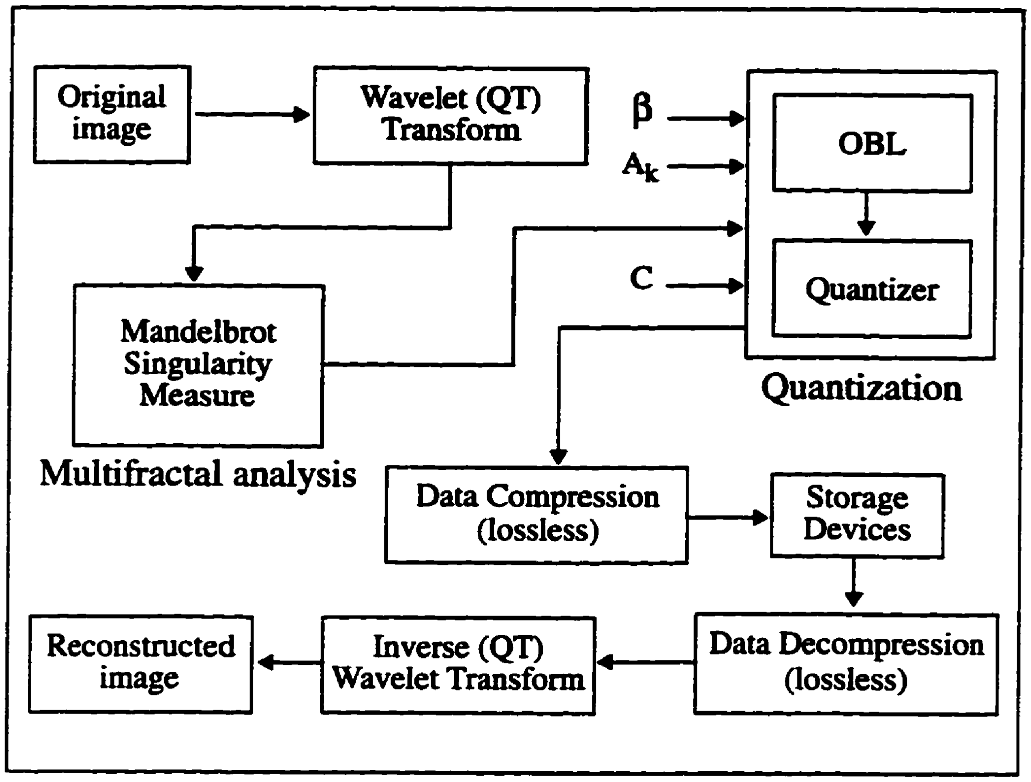


Fig. 5.10. A basic image compression scheme by using QDMA.

5.4.1 Quadtree Decomposition (QT)

Before an image, $I(i, j)$, is decomposed by using DWT, it is first normalized according to the following formula:

$$I'(m, n) = \frac{I(i, j) - \mu}{AR} \tag{5.2}$$

where μ is the image mean and

$$AR = \frac{1}{128} max(I_{max} - \mu, \mu - I_{min}) \tag{5.3}$$

where I_{min} and I_{max} are the minimum and maximum pixel values in the image $I(i,j)$, respectively. The main effect of this normalization is to give the lowest frequency of decomposed subbands a mean of approximately zero.

Quadtree decomposition (QT) is a type of of wavelet packet [Wick94]. Wavelet packets are particular linear combinations or superpositions of wavelets. They form bases which retain many of the orthogonality, smoothness, and localization properties of their parent wavelets. The choice of a decomposition topology corresponds to any pruned subtree of the original tree, i.e., any subtree sharing the same root as the original tree.

Let H, G be a conjugate pair of QMF filters from an orthogonal set. There are two others, QMFs H′ and G′, for which H*H′ + G*G′ = 1. We use the Daubechies wavelets (Daub4) in our wavelet packets. The decomposition in tree structure and frequency are shown in Fig. 5.11 and 5.12 respectively. The reason for using the quadtree decomposition is its simplicity.

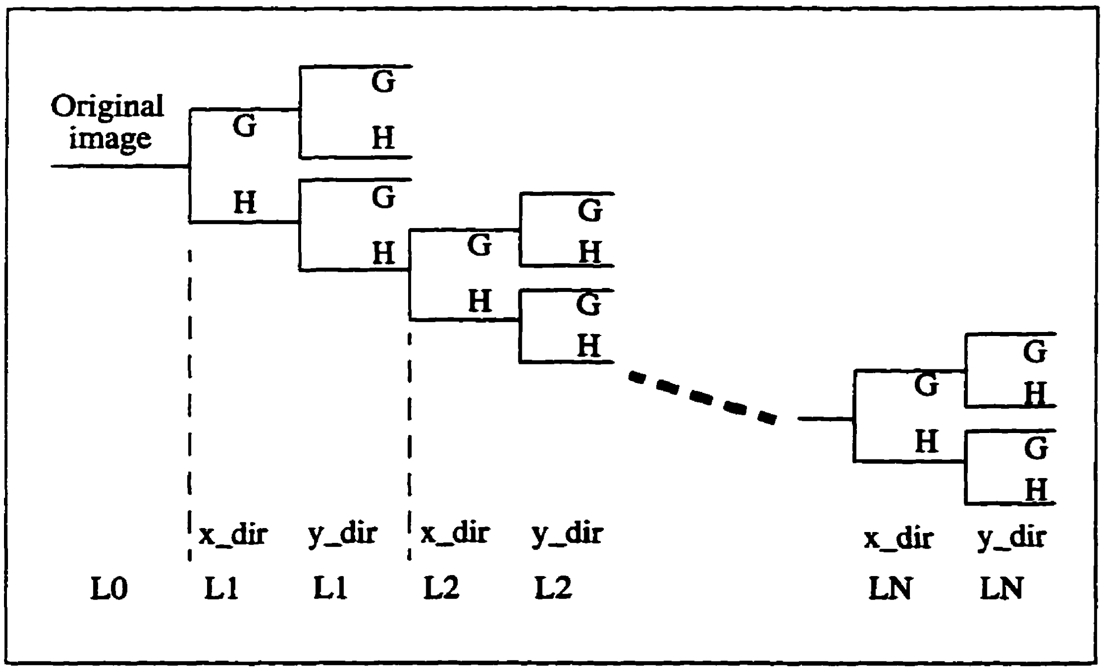


Fig. 5.11. Quadtree wavelet packet decomposition.

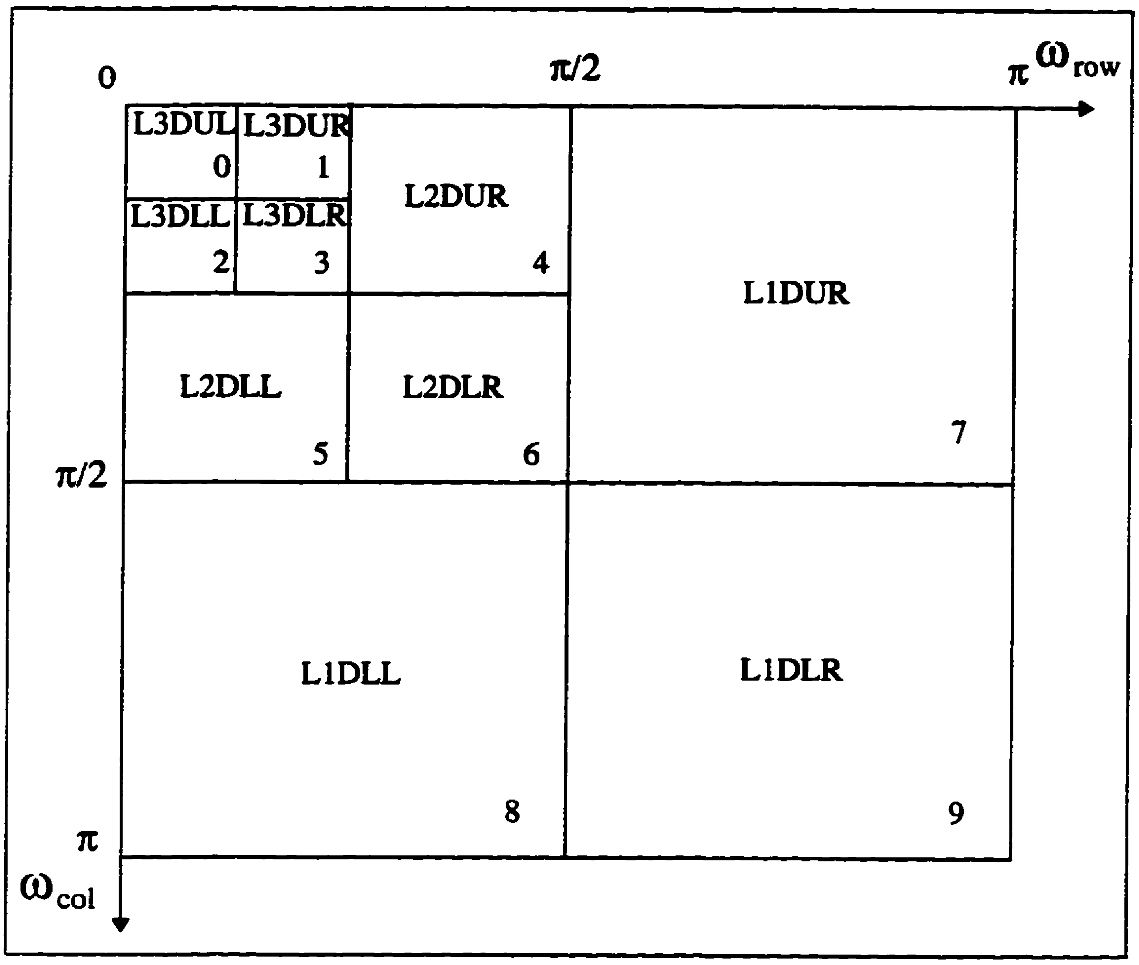


Fig. 5.12. Frequency support of QDMA subbands (L = 3).

5.4.2 Scalar Quantization

The Scalar Quantization (SQ) includes two parts, bit allocation and quantization. In Chapter 2 we have discussed their theories and algorithms. Here, we will introduce the method used by WSQ (a FBI fingerprint compression standard).

5.4.2.1 Optimal Bit Allocation (OBL)

Since raw fingerprint image data has a bandwidth of 8 bits per pixel (bpp), a bit rate of r bpp for the compressed data corresponds to a compression ratio of $8/r$. For the k^{th}

quadtree decomposition wavelet transform (QDWT) subband, let r_k denote the subband bit rate, μ_k the subband mean, and σ_k the subband variance. m_k denotes the downsample factor, which is defined to be the ratio of image size to subband size. For the quadtree decomposition as shown in Fig. 5.12, all downsample factors are powers of 4; e.g., $m_0 = 64$ and $m_9 = 4$.

In this case the bit allocation routine (shown at the end of this section) determines that a certain subband contains so little information that it should be discarded (i.e., subband will not be transmitted). To keep track of the non-discarded subbands, let K denote the set of all transmitted subbands (e.g., for Fig. 5.12, $K \subset \{0, 1, \ldots, 9\}$). The fraction of transmitted QDWT coefficients will be denoted by S, where

$$S = \sum_{k \in K} \frac{1}{m_k} \tag{5.4}$$

The targeted overall lossy bit rate, r, can be expressed as

$$r = \sum_{k \in K} \frac{r_k}{m_k} \tag{5.5}$$

To relate bit rate to quantizer bin widths (discussed in Section 5.4.3.2), we need to assume that the data being quantized lies in some interval of finite extent. Accordingly, the assumption is made that the quantization bins cover the interval $[\mu_k - \gamma\sigma_k, \mu_k - \gamma\sigma_k]$; i.e., that

$$Q_k = \frac{2\gamma\sigma_k}{L_k} \tag{5.6}$$

where L_k is the number of bins in the quantizer, and the loading factor, γ, is a parameter that specifies the number of standard deviations of data that are being coded. For our experiments, the value $\gamma = 2.5$ is used.

Note, while a poor choice of loading factor will not result in overload distortion, as is the case with other quantization strategies, it will affect the extent to which the lossy bit rate constraint, r, models the actual observed compression ratio. We also assume that the average transmission bit rate for subband k is

$$r_k = \log_2 L_k \quad \text{bits/sample} \tag{5.7}$$

This rate (Eq. 5.7) actually models the worst case scenario in which indices occur with equal probabilities and are coded with equal numbers of bits.

Now use the above model to determine q by applying the bit rate constraint to the bin widths. We get

$$Q_k = \frac{1}{q} Q'_k \tag{5.8a}$$

where Q'_k is independent of q

$$Q'_k = \frac{10}{\log_e\left(\sigma_k^2\right)} \tag{5.8b}$$

Substituting Eqs. 5.6 and 5.7 into 5.5 gives

$$\begin{aligned} r &= \sum_{k \in K} \frac{1}{m_k} \log_2 \frac{2\gamma\sigma_k q}{Q'_k} \\ &= \sum_{k \in K} \frac{1}{m_k}\left(\log_2 \frac{\sigma_k}{Q'_k} + \log_2 2\gamma q \right) \\ &= \sum_{k \in K} \frac{1}{m_k} \log_2 \frac{\sigma_k}{Q'_k} + S \log_2 2\gamma q \\ &= \sum_{k \in K} \log_2 \left(\frac{\sigma_k}{Q'_k} \right)^{\frac{1}{m_k}} + S \log_2 2\gamma q \end{aligned}$$

$$= \log_2\left(\prod_{k\in K}\left(\frac{\sigma_k}{Q_k^r}\right)^{\frac{1}{m_k}}\right) + S\log_2 2\gamma q \qquad (5.9a)$$

rearrange the above equation and we get

$$S\log_2 2\gamma q = r - \log_2\left(\prod_{k\in K}\left(\frac{\sigma_k}{Q_k^r}\right)^{\frac{1}{m_k}}\right)$$

$$\log_2 2\gamma q = \frac{r}{S} + \log_2\left(\prod_{k\in K}\left(\frac{\sigma_k}{Q_k^r}\right)^{\frac{1}{m_k}}\right)^{\frac{-1}{S}} \qquad (5.9b)$$

Remove the log for the above equation,

$$2\gamma q = 2^{\frac{r}{S}}\left(\log_2\left(\prod_{k\in K}\left(\frac{\sigma_k}{Q_k^r}\right)^{\frac{1}{m_k}}\right)^{\frac{-1}{S}}\right)$$

$$q = \frac{1}{\gamma}2^{\frac{r}{S}-1}\left(\log_2\left(\prod_{k\in K}\left(\frac{\sigma_k}{Q_k^r}\right)^{\frac{1}{m_k}}\right)^{\frac{-1}{S}}\right)$$

$$= \beta 2^{\frac{r}{S}-1}\left(\log_2\left(\prod_{k\in K}\left(\frac{\sigma_k}{Q_k^r}\right)^{\frac{1}{m_k}}\right)^{\frac{-1}{S}}\right) \qquad (5.10)$$

where $\beta = \frac{1}{\gamma}$ is the loading fraction.

There are two cases that require special attention. First, when $\log_e\left(\sigma_k^2\right)$ approaches zero we have $Q_k \to \infty$, which corresponds to $r_k \to 0$, so we discard any sub-band whose variance is too small, i.e., $\log_e\left(\sigma_k^2\right) \le 0$. Second, if $Q_k \ge 2\gamma\sigma_k$ then Eq. 5.6

and Eq. 5.7 imply that $r_k \le 0$. Since this distorts Eq. 5.5 by imposing the bit rate constraint, we use an iterative procedure to determine q which is shown below.

An Iterative Procedure for Computing Bin Widths

```
1. Initialization, j = 0:
      (a) If $\log_e\left(\sigma_k^2\right) \le 0$ then set $Q_k = 0$.
      (b) $K^{(0)} = \{k | \log_e\left(\sigma_k^2\right) > 0\} \subset \{0, 1, \ldots, 9\}$.
2. Iterate on Eq. 5.10 to calculate q:
      (a) $S^{(j)} = \sum_{k \in K^{(j)}} \frac{1}{m_k}$.
      (b) $q^{(j)} = \beta 2^{\frac{r}{S^{(j)}} - 1} \left( \log_2 \left( \prod_{k \in K^{(j)}} \left( \frac{\sigma_k}{Q'_k} \right)^{\frac{1}{m_k}} \right)^{\frac{1}{S^{(j)}}} \right)^{-1}$
3. Exclude bands that would theoretically have negative bit rates:
      (a) $\Xi^{(j)} = \left\{ k \in K^{(j)} | \frac{Q'_k}{q^{(j)}} \ge 2\gamma\sigma_k \right\}$.
      (b)   If $\Xi^{(j)} \ne \phi$ then
                  $K^{(j+1)} = K^{(j)} \backslash \Xi^{(j)}$,
                  $j = j + 1$,
                  go to 2;
            else
                  $K = K^{(j)}$, the bands with positive bit rates,
                  $\Xi = K^{(0)} \backslash K$, the excluded bands,
                  $q = q^{(j)}$,
                  continue.
4. Calculate bin widths:
            $Q_k = \frac{Q'_k}{q}$ for all $k \in K^{(0)}$.
5. Exit
```

Note that \ denotes the set difference operator; i.e., $A \backslash B = A \cap B^C$

5.4.2.2 Quantizer Design

Figure 5.12 shows the frequency decomposition of QT, with 9 subbands for L = 3. Each subband is coded separately according to a scalar quantizer characteristic having uniform width bins with the exception of the zero bin, which is 20% wider. Each scalar quantizer (SQ) is defined by an encoding and a decoding relationship. The quantization encoder maps a floating-point wavelet transform coefficient, *a*, to an integer quantizer index, *p*, that indicates the quantizer bin in which *a* lies. The quantization decoder maps the index to a prototypical (quantized) real number, $\hat{a}$, representing all data values that lie within that bin. The quantizer characteristic map is shown in Fig. 5.13.

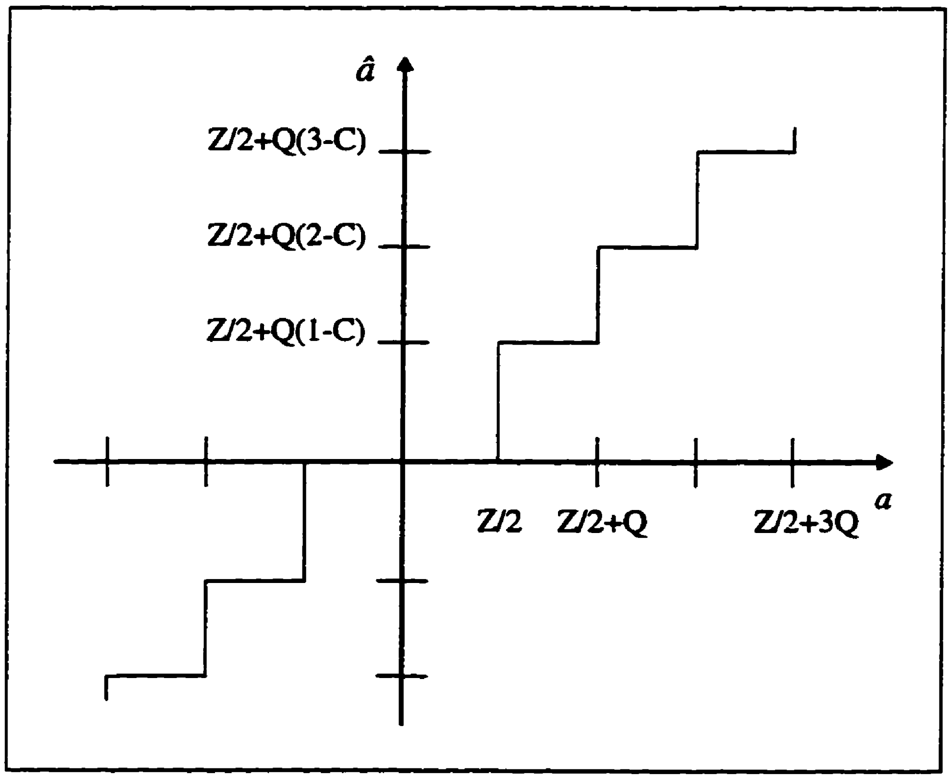


Fig. 5.13. Uniform quantizer characteristic.

Quantization encoding of the k^{th} two-dimensional QT subband, $a_k(i,j)$, is given by

$$
\begin{aligned}
p_k(i,j) &= \left\lfloor \frac{\left(a_k(i,j) - \frac{Z_k}{2}\right)}{Q_k} \right\rfloor + 1, & a_k(i,j) &> Z_k/2 \\
&= 0, & -Z_k/2 \le a_k(i,j) &\le Z_k/2 \\
&= \left\lceil \frac{\left(a_k(i,j) + \frac{Z_k}{2}\right)}{Q_k} \right\rceil - 1, & a_k(i,j) &< Z_k/2
\end{aligned}
\qquad (5.11)
$$

where Z_k is the width of the zero bin and Q_k is the width of the quantization bin. The $\lceil \circ \rceil$ and $\lfloor \circ \rfloor$ denote the ceiling and floor functions that round numbers to the next largest and next lowest integer, respectively. The quantized wavelet coefficients produced by the quantization decoder are given by

$$
\begin{aligned}
\hat{a}_k(i,j) &= (\, p_k(i,j) \;- C)Q_k + Z_k/2, & p_k(i,j) &> 0 \\
&= \;0, & p_k(i,j) &= 0 \\
&= (\, p_k(i,j) \;+ C)Q_k - Z_k/2, & p_k(i,j) &< 0
\end{aligned}
\qquad (5.12)
$$

where C is a parameter between 0 and 1 that determines the reconstructed values. Note that if $C = 1/2$ then the reconstructed value corresponding to each quantization bin would be the bin's midpoint. In our experiment the C is set to 0.44. The quantizer indices, $p_k(i,j)$, are transmitted losslessly. In our experiment we use *GZIP* (lossless) to compress and decompress the $p_k(i,j)$.

The formula for the quantization bin widths, Q_k, is defined as

$$Q_k = \frac{10}{qA_k \log_e\left(\sigma_k^2\right)} \tag{5.13}$$

where q is defined in Eq. 5.10 which determines the overall compression ratio. The constants A_k are empirically determined weights. A_k is set to 1 in our experiment.

5.4.3 Computing the Fractal Spectra of Wavelet Transform Coefficients

5.4.3.1 Derivation of α_q and f_q

We have already explained the Rényi dimension, D_q and Mandelbrot dimension, D_{MAN} in Chapter 4. The coefficients of the wavelet transform, the scalar of the projection (transformation), stand for the energy. Since there is no negative energy, we take the absolute value of the coefficients to calculate their fractal dimensions.

Recall, that the Rény dimension is defined as

$$D_q := \lim_{r \to 0} \frac{1}{q-1} \frac{\log \sum_{j=1}^{N_r} \left(P_j^q\right)}{\log(r)} \tag{5.14}$$

and the Hölder exponent, α, can be obtained by taking the derivative with respect to the Rény exponent q,

$$\alpha_q := \frac{d}{dq}[(q-1)D_q] \tag{5.15}$$

and $f(\alpha) \equiv f_q$ is obtained from:

$$f_q := q\alpha_q - (q-1)D_q \tag{5.16}$$

where $f(\alpha)$ is the fractal Mandelbrot dimension, D_{MAN}, of the α subset. It is difficult to calculate the derivative in Eq. 5.15. To make it more useful, Eq. 5.15 can be extended to

$$\alpha_q = \frac{1}{\lim\limits_{r \to 0} \log(r)} \frac{d}{dq} \log \sum_{j=1}^{N_r} \left(P_j^q \right) \tag{5.17}$$

If we consider some basic calculus theory

$$\frac{d}{du} \log_a u = \frac{1}{u} \log_a e \tag{5.18}$$

$$\frac{d}{du} a^u = \frac{a^u}{\log_a e} \tag{5.19}$$

and let x = $\sum_{j=1}^{N_r} \left(P_j^q \right)$ and y = $\log x$, then Eq. 5.17 becomes

$$\alpha_q = \frac{1}{\lim\limits_{r \to 0} \log(r)} \frac{dy}{dq} \tag{5.20}$$

Furthermore, from the chain rule, $\frac{dy}{dq} = \frac{dy}{dx} \cdot \frac{dx}{dq}$ we have

$$\frac{dy}{dq} = \frac{1}{x} \log_{10} e \cdot \frac{d}{dq} \left(\sum_{j=1}^{N_r} \left(P_j^q \right) \right) \tag{5.21}$$

Substituting $\frac{d}{dq} \left(\sum_{j=1}^{N_r} \left(P_j^q \right) \right) = \frac{d}{dq} \left(P_1^q + P_2^q + \ldots + P_{N_r}^q \right) = \frac{1}{\log_{10} e} \cdot \sum_{j=1}^{N_r} \left(P_j^q \right) \log P_j$ into Eq. 5.9 yields

$$\frac{dy}{dq} = \frac{1}{x} \cdot \sum_{j=1}^{N_r} \left(P_j^q \right) \log P_j \tag{5.22}$$

$$= \frac{\sum_{j=1}^{N_r} \left(P_j^q\right) \log P_j}{\sum_{j=1}^{N_r} \left(P_j^q\right)} = \frac{\sum_{j=1}^{N_r} \left(n_j^q\right) \log P_j}{\sum_{j=1}^{N_r} \left(n_j^q\right)} \tag{5.23}$$

where $P_j = \frac{n_j}{N_T}$. Substituting Eq. 5.23 into Eq. 5.17 gives

$$\alpha_q = \frac{1}{\lim\limits_{r \to 0} \log(r)} \frac{\sum_{j=1}^{N_r} \left(n_j^q\right) \log P_j}{\sum_{j=1}^{N_r} \left(n_j^q\right)} \tag{5.24}$$

Finally substituting Eqs. 5.12 and 5.2 into 5.4 yields

$$f_q = \frac{1}{\lim\limits_{r \to 0} \log(r)} \left(\frac{\sum_{j=1}^{N_r} \left(n_j^q\right) \log P_j^q}{\sum_{j=1}^{N_r} \left(n_j^q\right)} - \log \sum_{j=1}^{N_r} \left(P_j^q\right) \right) \tag{5.25}$$

The Hölder exponent, α, corresponding to the strength of the local singularity of the measure, is analogous to the energy, while $f(\alpha)$ is analogous to the entropy as a function of energy. Since we can regard fingerprints as a strange attractors (see Chapter 4) we can also treat their wavelet transform as a strange attractor and use the fractal singularity measure to analyze the wavelet coefficients.

5.4.3.2 Example

Let us now consider an example. Figure 5.14 shows the Mandelbrot multifractal spectra of a fingerprint image (Fig. 5.15(a)). If we synthesize all four subbands (DUL, DUR, DLL and DLR) using the inverse wavelet transform (IWT) then we can get a high quality (PSNR > 45 dB) reconstructed image. Although the subband DUL has the highest energy, its IWT produces a reconstructed image with poor quality (30.19 dB). To achieve a higher quality, we have to use more subbands in the IWT. Are the remaining three subbands (DUR, DLL and DLR) equal, in terms of the reconstructed image quality? Unfortunately there are differences. The problem of how to select the best subband (i.e., which one will affect the reconstructed quality most) is the motivation for this experiment. The Mandelbrot multifractal spectra provides the solution. The one with the widest coverage in the multifractal spectra is the candidate. From Fig. 5.14, we find that subband DLL has the widest energy coverage of the three subbands (DUR, DLL and DLR). That means if we synthesize subband DLL with DUL for IWT the reconstructed quality will be better than if we chose any of the others. Fig. 5.15 shows the experimental result.

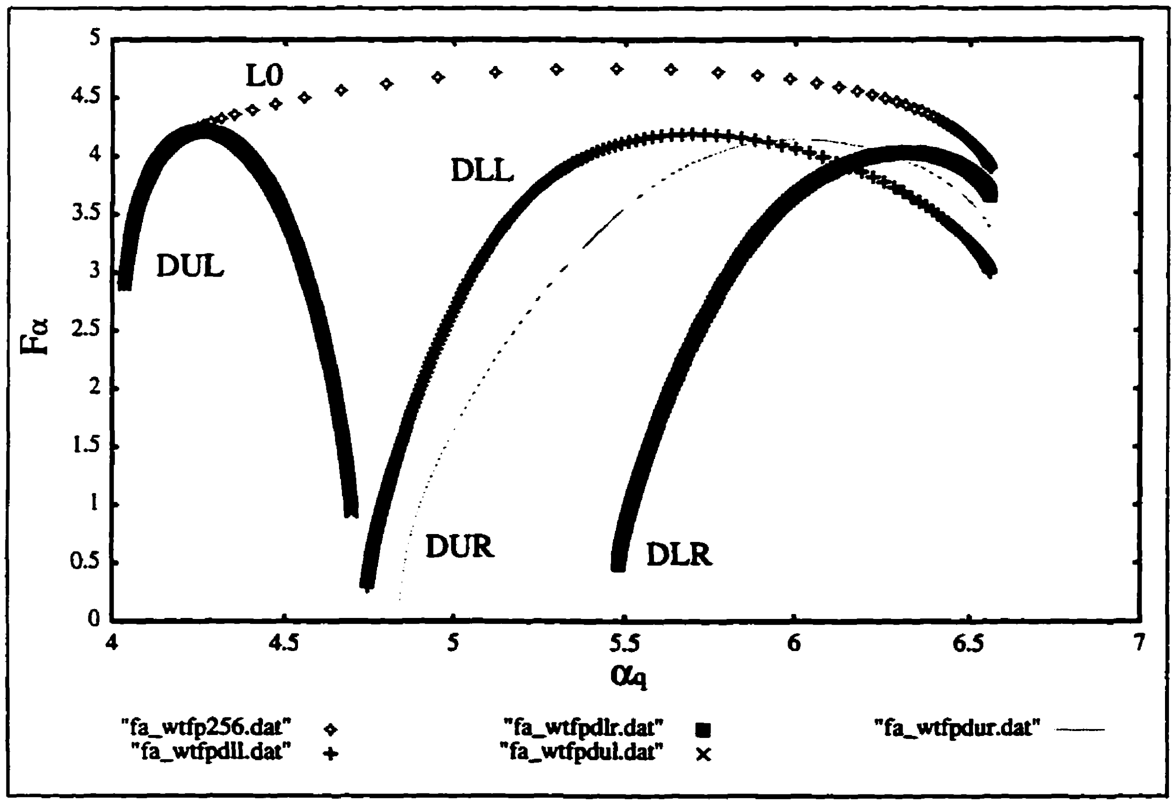
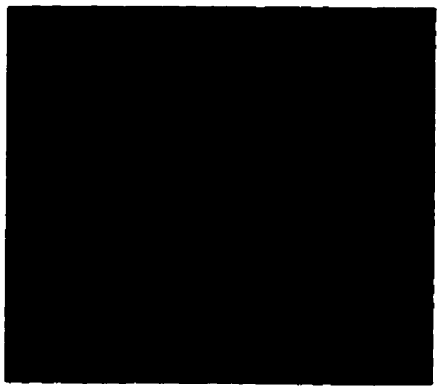
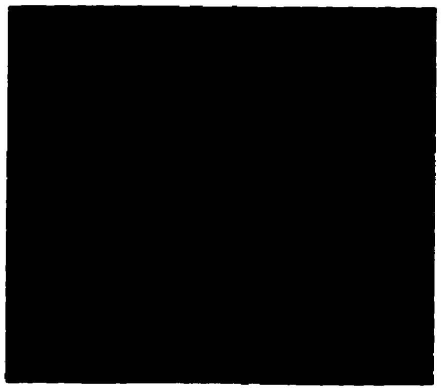
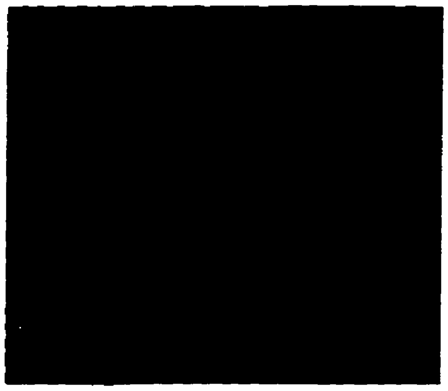


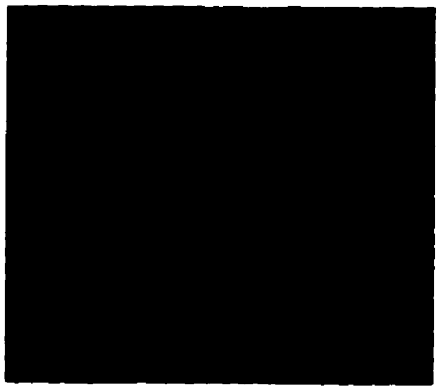
Fig. 5.14. Multifractal spectra of quadtree decomposition wavelet packet (L = 1) coefficients. The corresponding fingerprint image is shown in Fig. 5.15 (a).

(a)


(b)

(c)


(d)


(e)


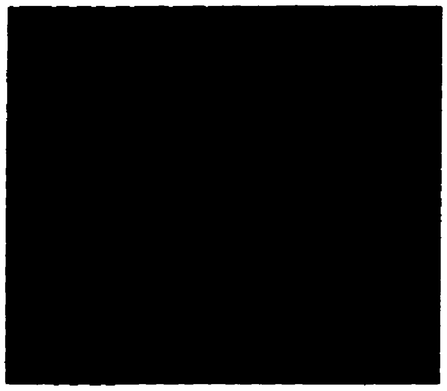
(f)


Fig. 5.15.(a) Original image (8 bpp), reconstructed image using quadtree decomposition (b) DUL, DUR, DLL, DLR (PSNR: 48.54 dB), (c) DUL (PSNR 30.19 dB), (d) DUL, DLL (PSNR: 32.44 dB), (e) DUL, DUR (PSNR: 31.75 dB), (f) DUL, DLR (PSNR: 31.61).

5.4.4 Experimental Result of using QDMA

In Fig. 5.16 we find that the energy of the wavelet coefficients can be divided into two segments, subband 0 in one group and the rest in another group (the number of subbands is defined in Fig. 5.12). Since subband 0 has the highest energy, it is the essential member of the synthesizer. To reach the higher compression ratio we have to add fewer subbands to the synthesizer. However, as more subbands are added, the quality of reconstructed image improves. The subband selection affects not only the compression ratio but also the accuracy of bit allocation.

We find that subbands L1DUR, L1DLL and L1DLR can be ignored completely because they are covered by subband L2DUR, L2DLL and L2DLR. Even subband L2DLR can be dropped. The experimental result is shown in Fig. 5.17.

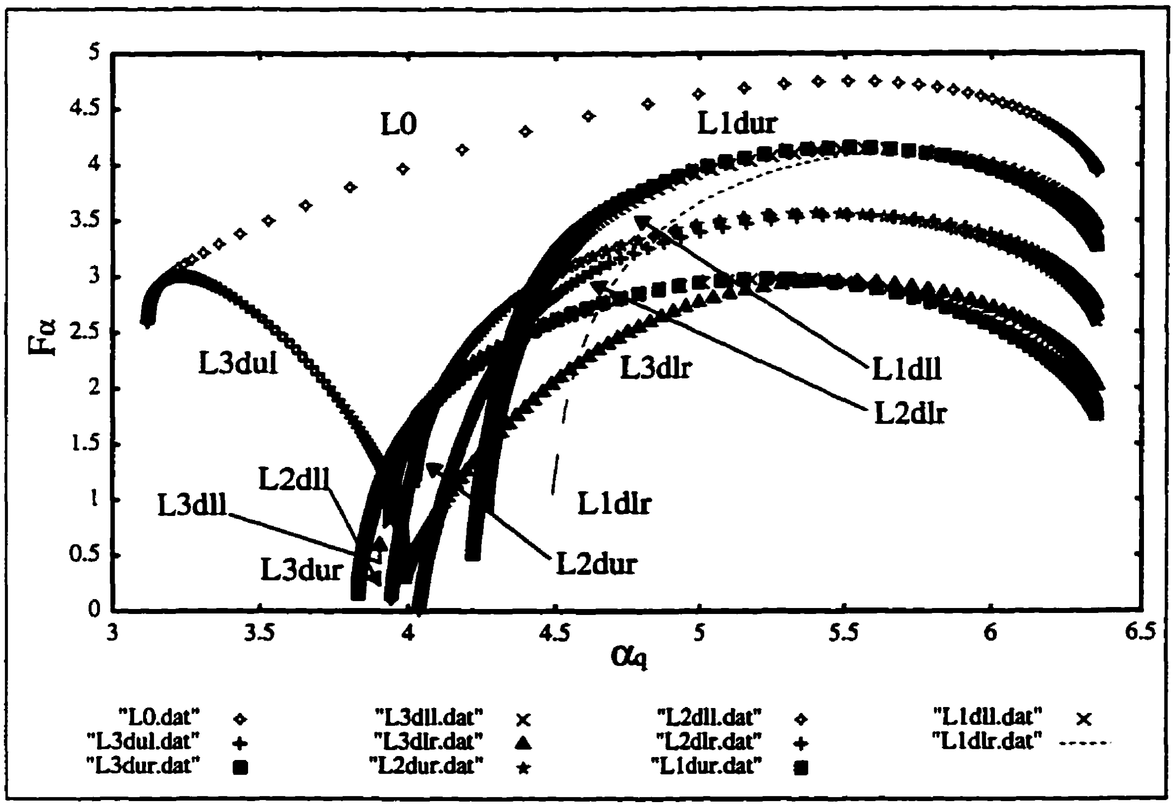


Fig. 5.16. Multifractal spectra of quadtree decomposition wavelet packet (L = 3) coefficients. The fingerprint image is shown in Fig. 5.17 (a).

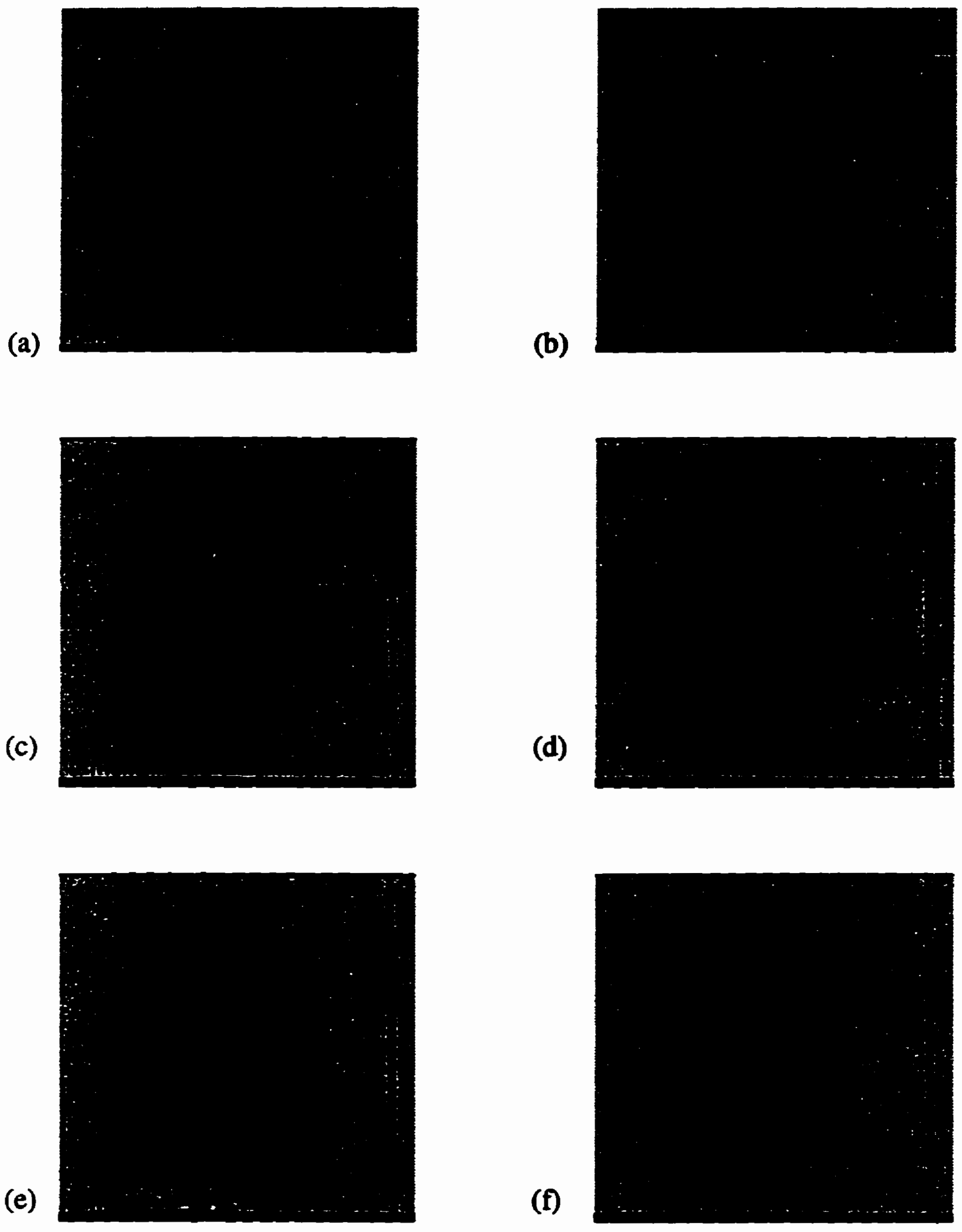


Fig. 5.17.(a) Original image (8 bpp), reconstructed image using quadtree decomposition
(b) L3 + L2 + L1 (PSNR: Inf dB), (c) L3 + L2 (PSNR 29.78 dB),
(d) L3 - L3DLL + L2 + L1 (PSNR 28.92), (e) L3 (PSNR 23.05),
(f) L3 + L2 - L2DLR (PSNR 28.06)

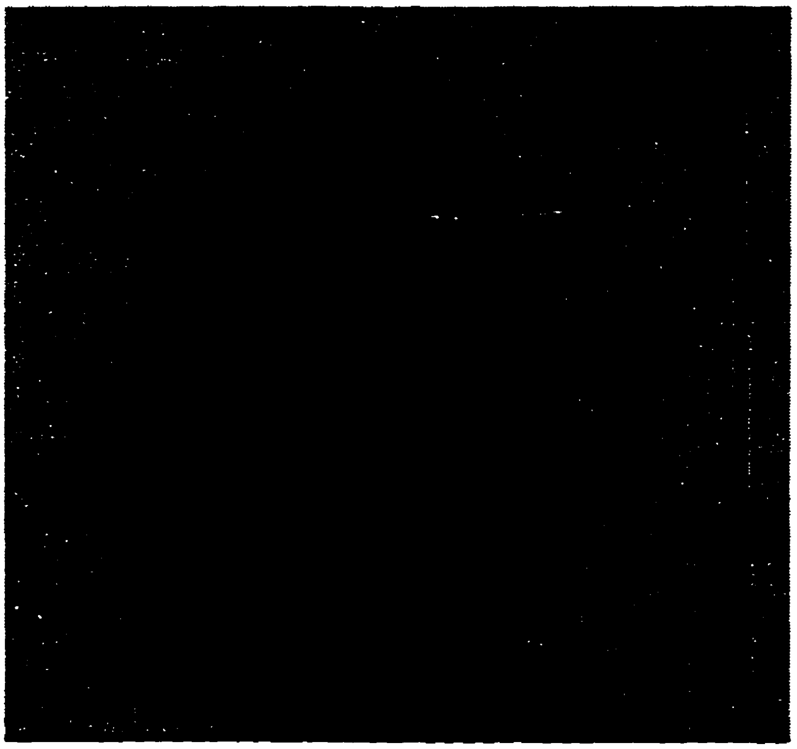


Fig. 5.18. Original image, 512 X 512, 8bpp (above), reconstructed image, L3+L2-L2DLR, using OBL quantization and lossless entropy coding, compression ratio 13.95:1 (0.57 bpp) at PSNR 28.06 dB (below).

The quality of reconstructed image (as shown in Fig. 5.18) is much better than the previous methods (such as zonal filter, direct thresholding and zerotree). The compression ratio can reach 13.95:1 (0.57 bpp) and the quality appears to be good enough for fingerprint identification or classification. To reach a higher compression ratio we have to use a more complicated wavelet packet instead of the quadtree decomposition, because in every decomposition the energy will concentrate to fewer coefficients and thus give us a better chance for subband selection. In the next section we will use the same wavelet packet as WSQ to prove our assumption.

5.5 Wavelet Packet with Multifractal Analysis (WPMA)

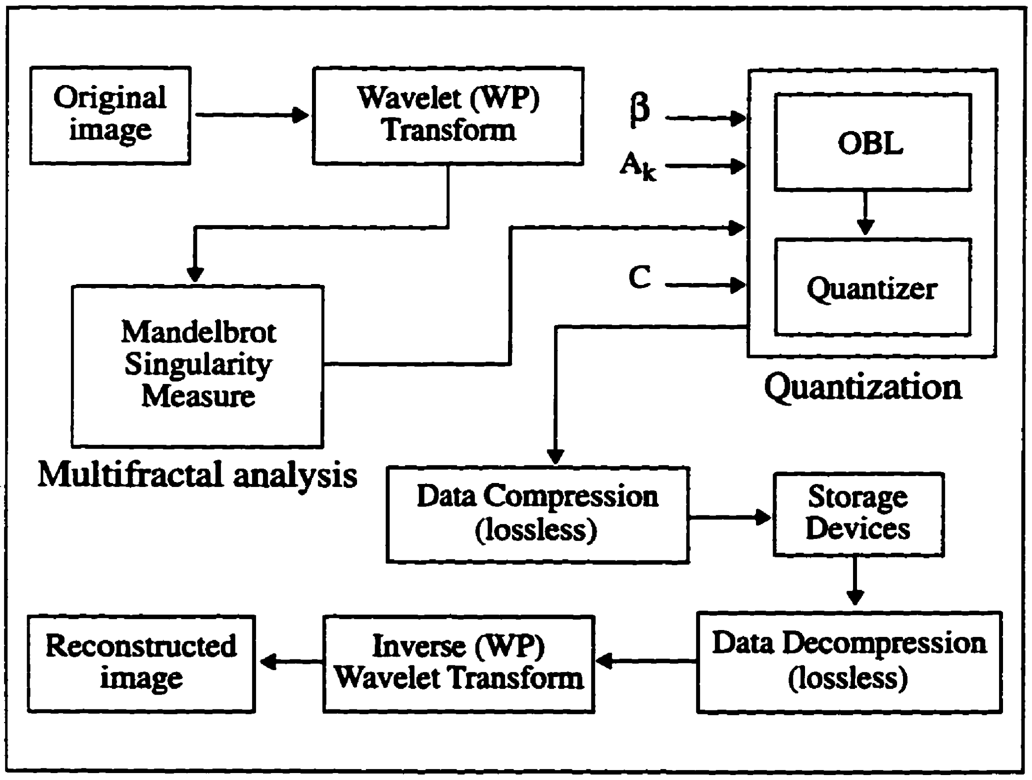


Fig. 5.19. A basic image compression scheme by using WPMA.

The WPMA method applies the same wavelet decomposition as WSQ. Both multifractal analysis and optimal bit allocation (OBL) blocks are the same as QDMA. The blockdiagram of WPMA is shown in Fig. 5.19.

We use the Daubechies wavelet (Daub4) in our wavelet packets. The decomposition in frequency is shown in Fig. 5.20.

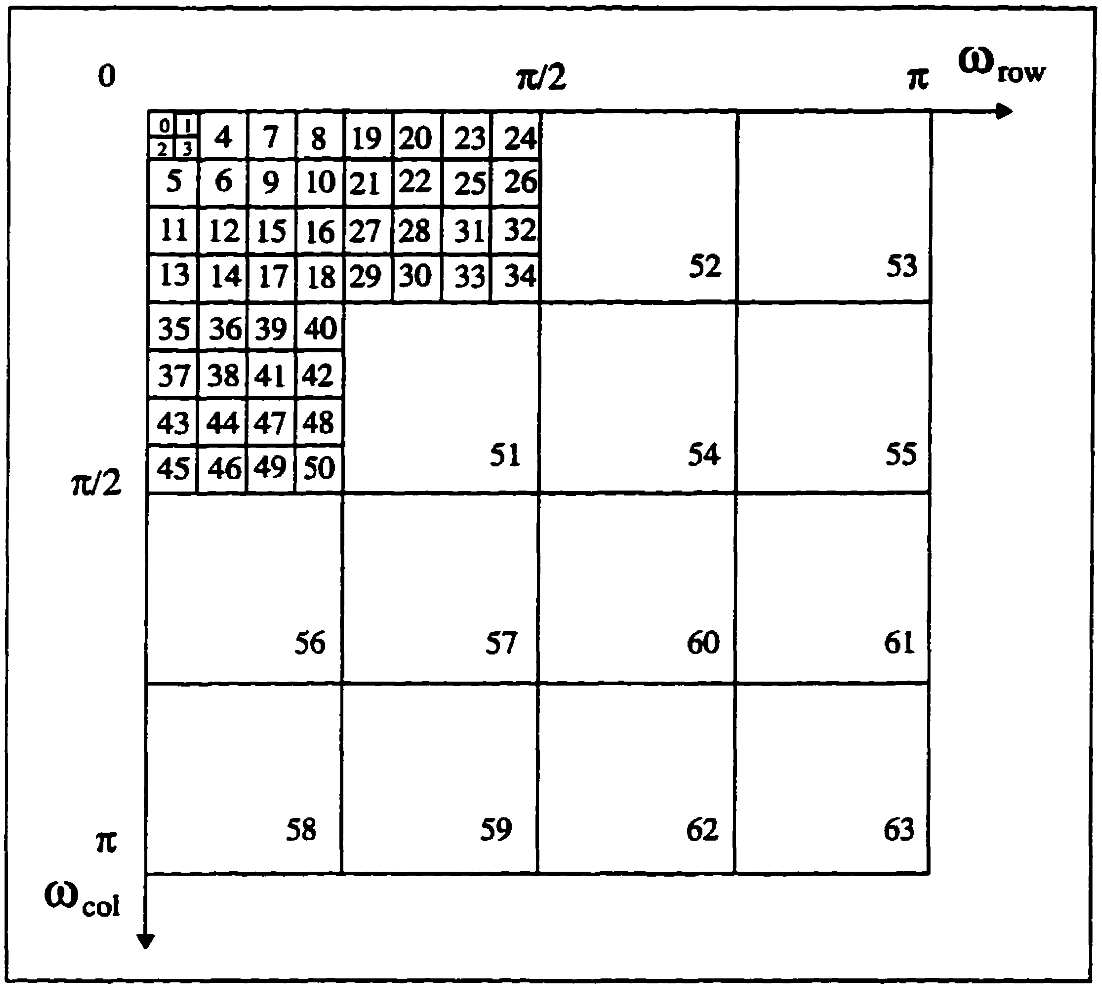


Fig. 5.20. Frequency support of WPMA subbands (L = 5).

The definition of subband groups (i.e., combination of subbands) is listed in Table 5.4. We will use the subband group definition in WPMA experiment discussion.

Table 5.4. Subband group definition for wavelet packet.

Group	Combined subbands
I	0 - 51
II	52 - 55
III	56 - 59
IV	60 - 63
I1	0 - 18
I2	18 - 34
I3	35 - 50
I4	51
I21	19 - 22
I22	23 - 26
I23	27 - 30
I24	31 - 34
I31	35 - 38
I32	39 - 42
I33	43 - 46
I34	47 - 50

The Mandelbrot multifractal spectra of wavelet coefficients for a fingerprint image (Fig. 5.15(a)) is shown in Fig. 5.21 to 5.30. In Fig. 5.21 we observe that the subband group I has the highest wavelet coefficient energy, so we have to keep it and consider removing the others (i.e., group II, III and IV). For further decomposition as shown in Fig. 5.22 we find that subband group I4 has the lowest coefficient energy in subband group I and we drop it. Since the natural frequencies of the ridges in fingerprint images are located in the portion of the frequency spectrum (Fig. 5.20) contained roughly in subbands

7 to 18 [BrBr93], and subbands 0 to 6 have the majority of wavelet coefficient energy, we reserve the whole subband group I1 for inverse wavelet transformation.

Now let us take a look at the multifractal spectra of group I2 which is shown in Fig. 5.23. The subband group I22 has the highest coefficient energy in subband group I2 and we keep it for the synthesizer. For the rest of the groups (i.e., group I21, I23 and I24) we make an experiment to see which subband group affects the quality of reconstructed image more than others. The result is shown in Table 5.5.

Table 5.5. The effect on of reconstructed quality for subband group I21, I23 and I24.

Subbands in synthesizer	PSNR (in dB)
I1, (I21, I22), I3	26.809394
I1, (I22, I23), I3	26.654160
I1, (I22, I24), I3	27.509226

It is clear from Table 5.5 that subband group I24 has the largest effect on reconstructed quality. Also, subband group I24 has the second highest coefficient energy as shown in Fig. 5.23. We consider removing one subband from group I24. Groups I21 and I23 are located in the low coefficient energy region and far apart from group I22, so we will remove two subbands from each of them.

The problem of which subbands can be removed from the synthesizer and not (or make it less) affect the quality of the reconstructed image is solved by the WPMA technique. From Fig. 5.24, we find subbands 19 and 21 can be removed. Similarly from Fig. 5.25 and 5.26, subbands 28, 29 and 32 can be removed from subband group I21 and I24 respectively.

The procedure for analyzing subband group I3 is similar. We keep subband group I33 for the synthesizer because it has the highest coefficient energy as shown in Fig. 5.27. There are two subbands that will be removed from each subband group I31 and I32 and one subband will be removed from group I34 by analysis through Fig. 5.28 to 5.30. The candidates for subband group I31 and I32 are subbands 35, 36 and 39, 40, respectively. The subband 49 will be removed from subband group I34.

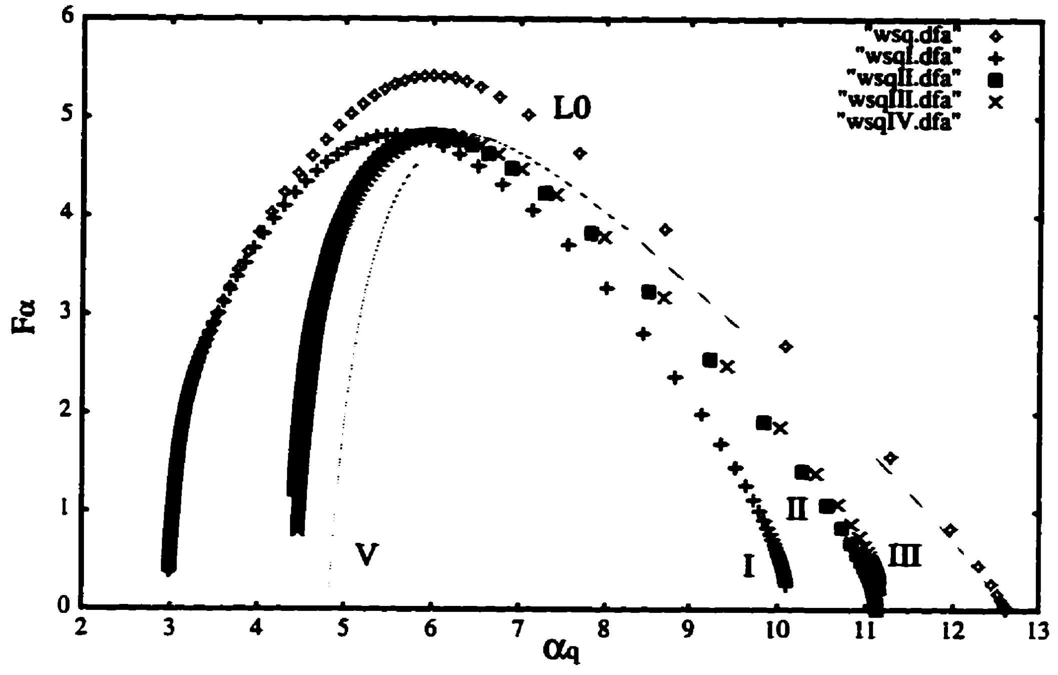


Fig. 5.21.Multifractal spectra wavelet packet (WP) coefficients for all bands and subband groups I, II, III and IV. The fingerprint image is shown in Fig. 5.17 (a).

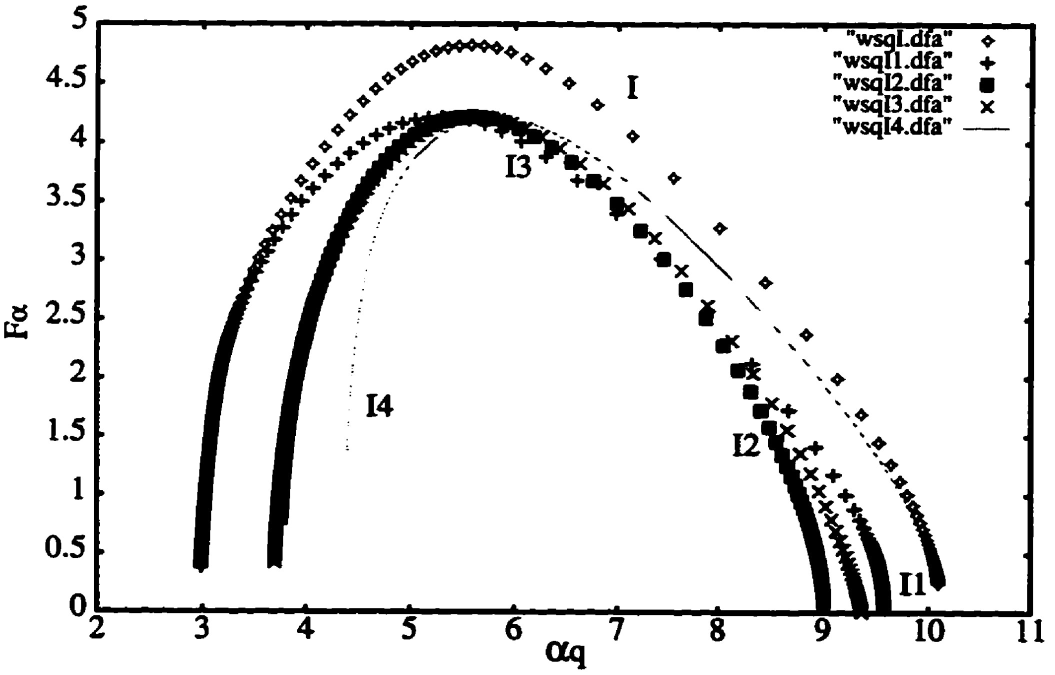


Fig. 5.22. Multifractal spectra of coefficients for subband groups I, I1, I2, I3 and I4.

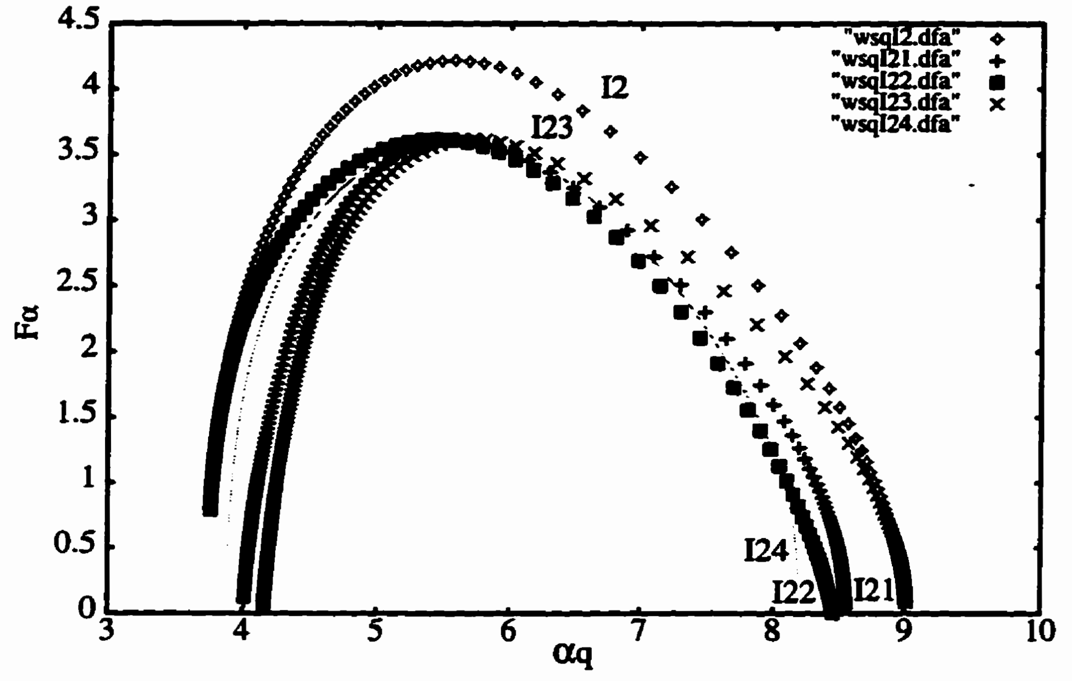


Fig. 5.23. Multifractal spectra of coefficients for subband groups I2, I21, I22, I23 and I24.

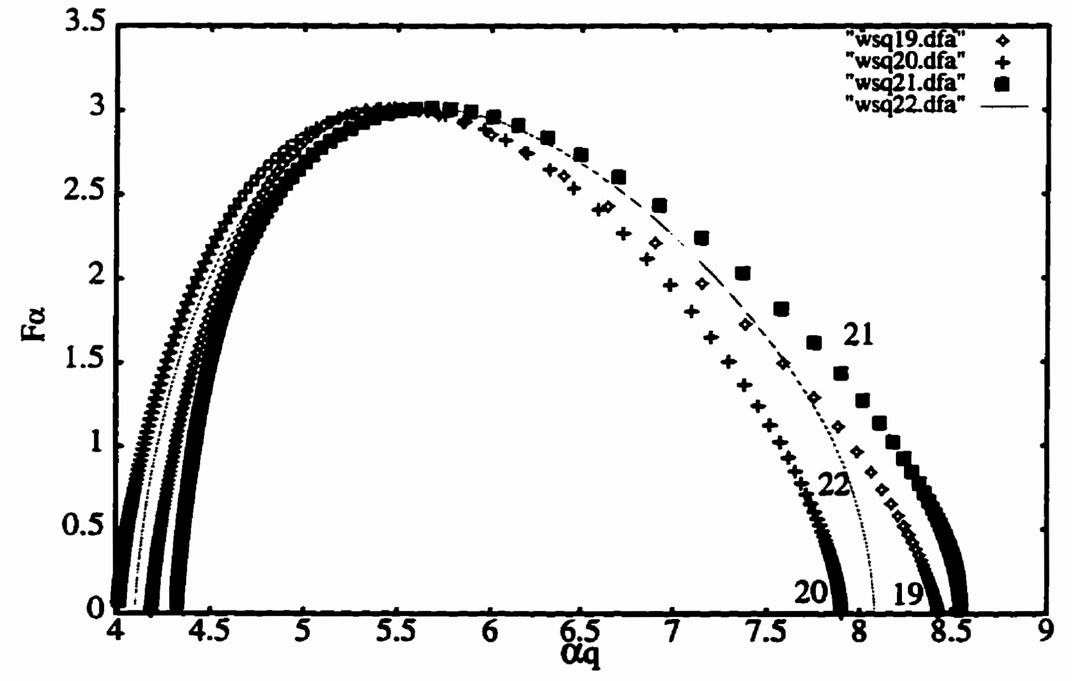


Fig. 5.24. Multifractal spectra of coefficients for subband groups 19, 20, 21 and 22.

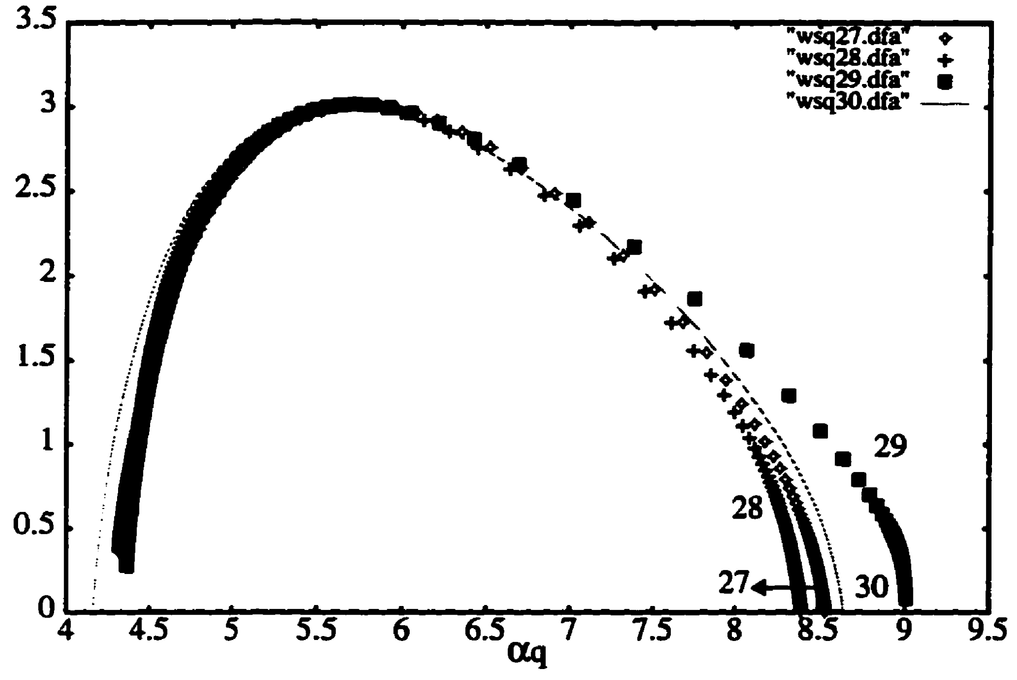


Fig. 5.25. Multifractal spectra of coefficients for subband groups 27, 28, 29 and 30.

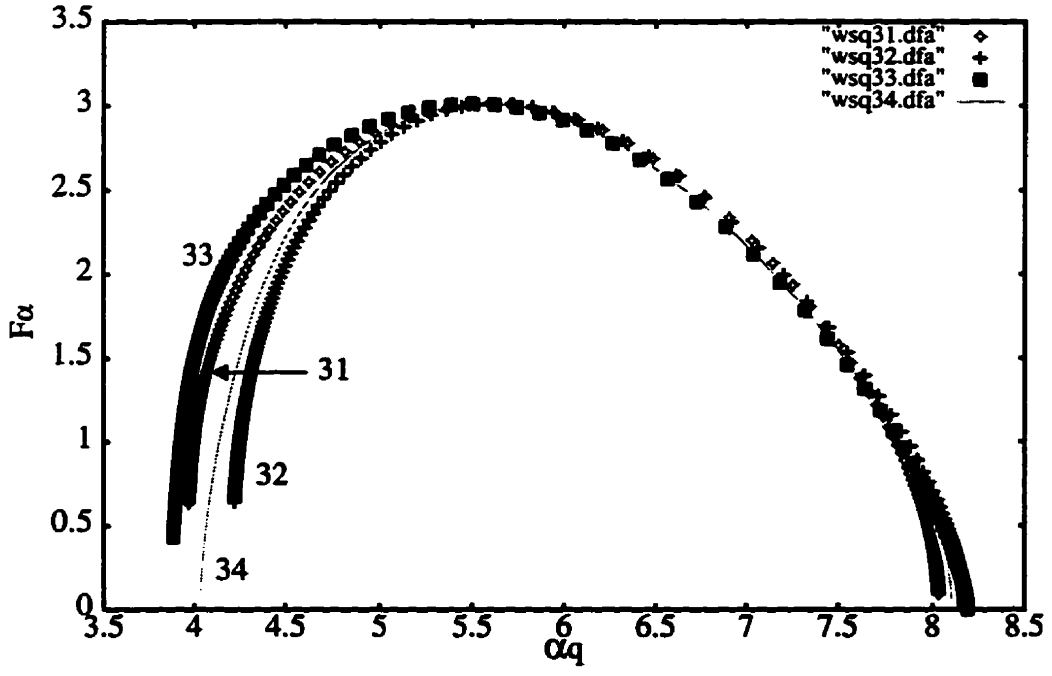


Fig. 5.26. Multifractal spectra of coefficients for subband groups 31, 32, 33 and 34.

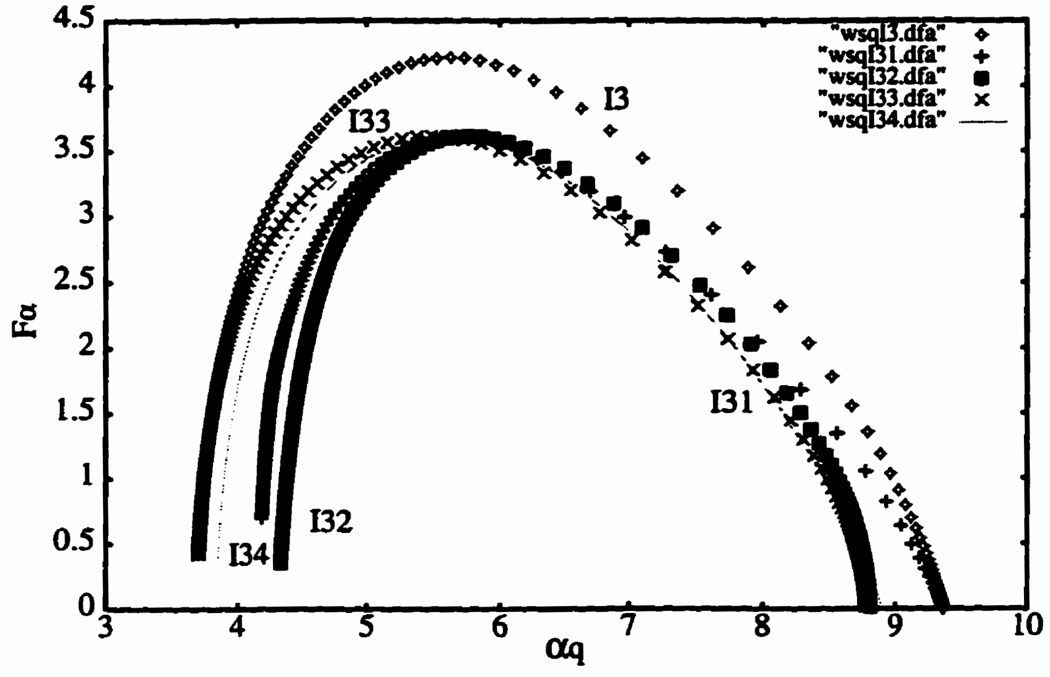


Fig. 5.27. Multifractal spectra of coefficients for subband groups I3, I31, I32, I33 and I34.

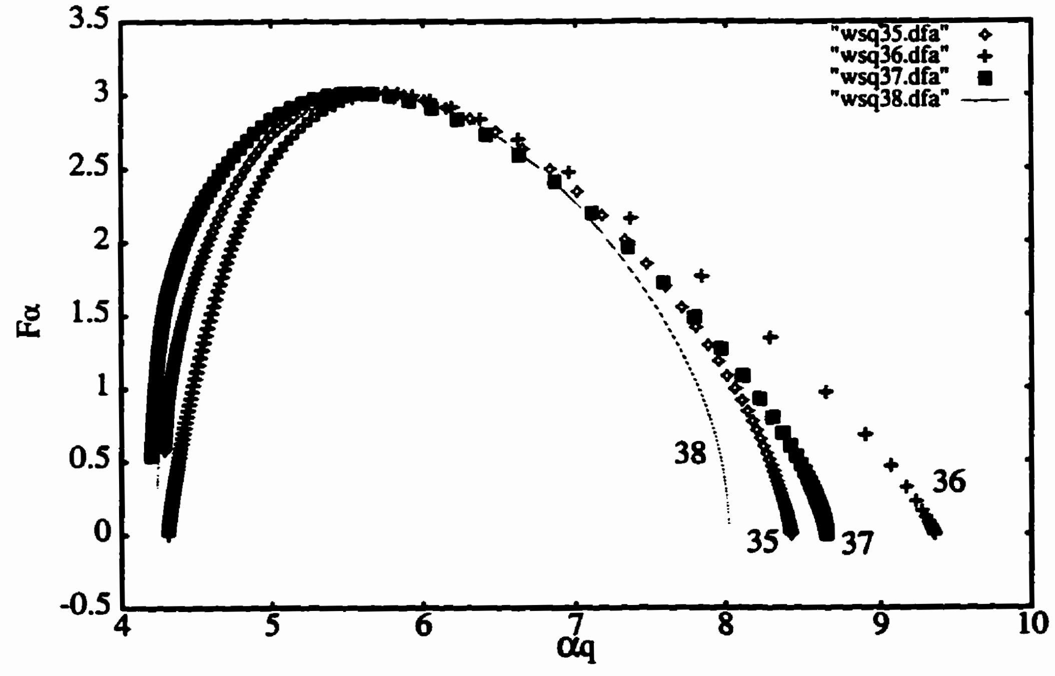


Fig. 5.28. Multifractal spectra of coefficients for subband groups 35, 36, 37 and 38.

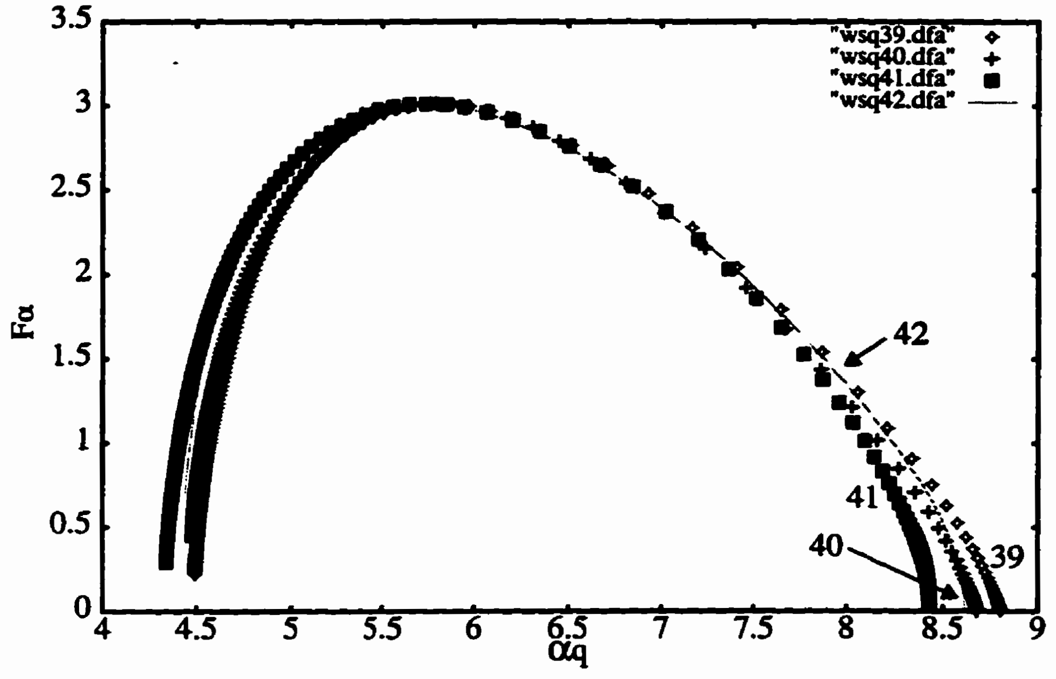


Fig. 5.29. Multifractal spectra of coefficients for subband groups 39, 40, 41 and 42.

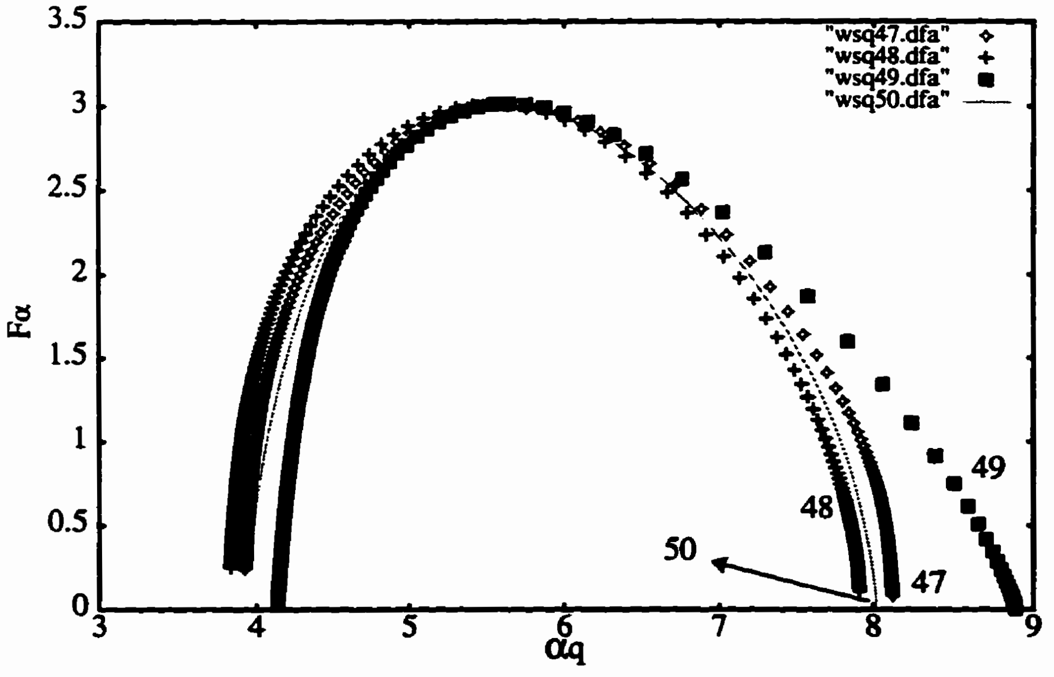


Fig. 5.30. Multifractal spectra of coefficients for subband groups 47, 48, 49 and 50.

Fig. 5.31 shows the result of multifractal spectra analysis for subbands 19 to 63 in WPMA wavelet packet decomposition. The shaded subband will not be included in the synthesizer (i.e., not be used for the inverse wavelet transform) and OBL. Without those subbands the OBL can calculate the assigned bits for each unshaded subband more precisely. The reconstructed image (as shown in Fig. 5.32) can reach a compression ratio of 17.7:1 (0.4519 bpp) at 28.57 dB PSNR.

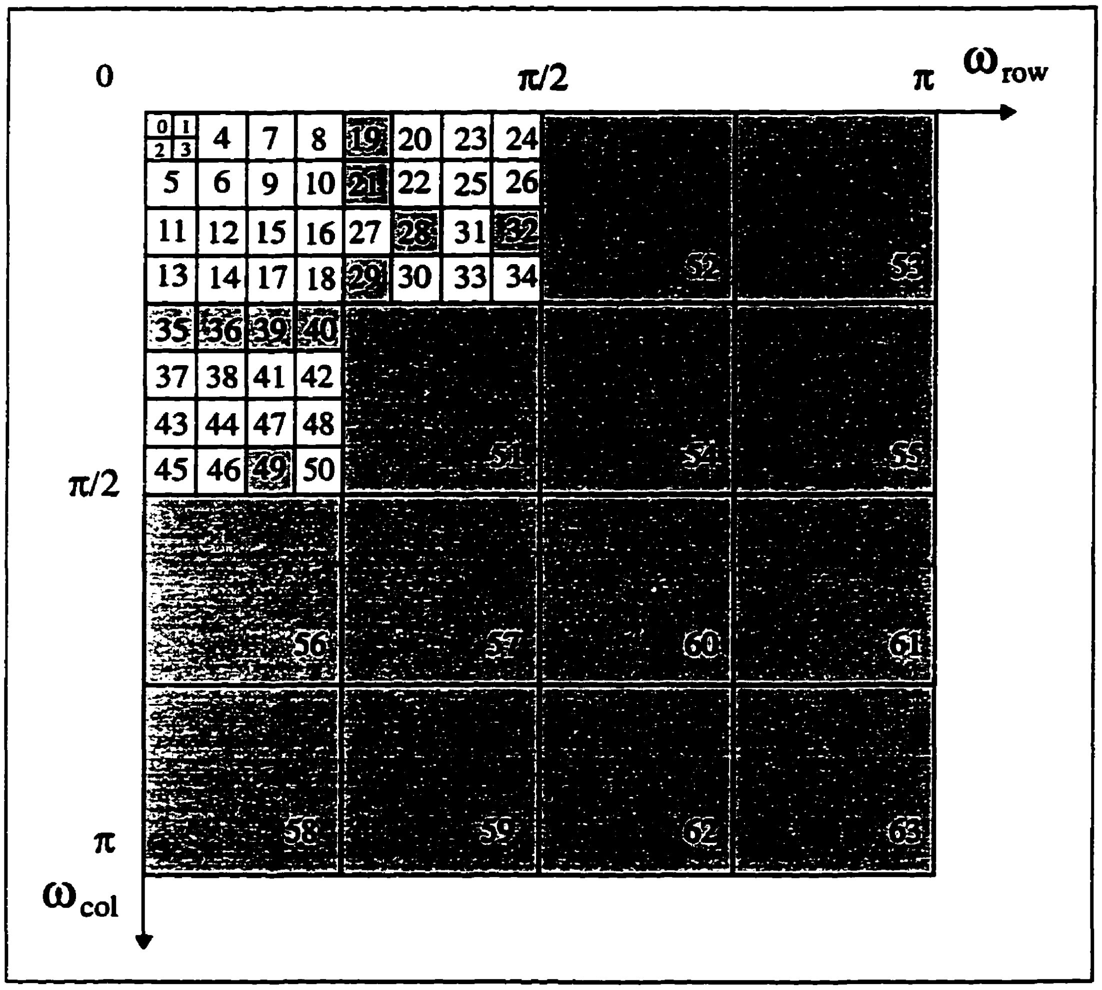


Fig. 5.31. Frequency spectrum of WPMA wavelet packet, shaded subbands will not be used for inverse wavelet transform and calculated for OBL.

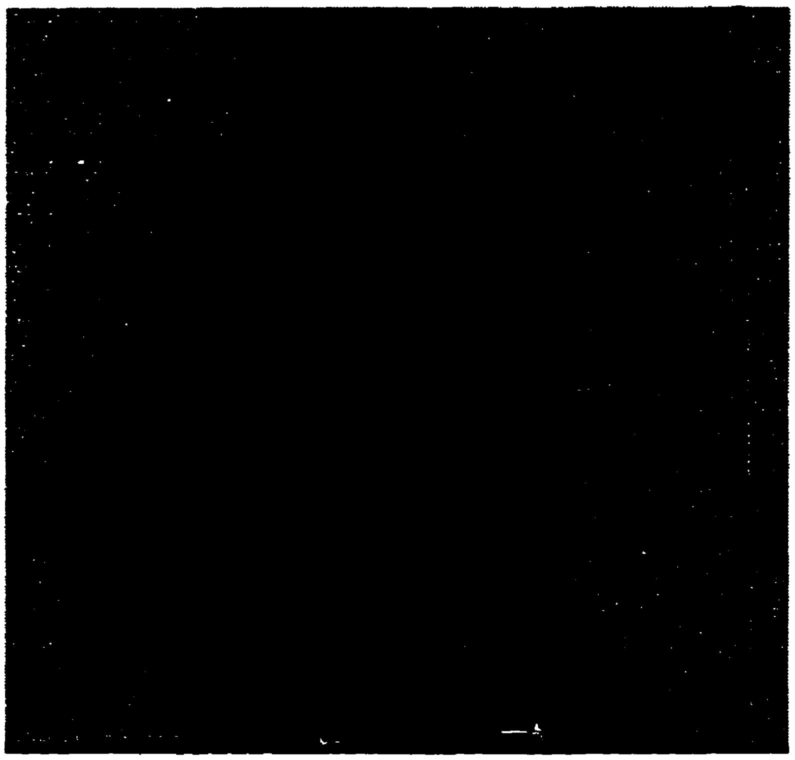


Fig. 5.32.Original image, 512 X 512, 8bpp (above), reconstructed image using WPMA, OBL quantizer and lossless entropy coding for the shading subbands shown in Fig. 5.31, compression ratio 17.7:1 (0.4519 bpp) at PSNR 28.57 dB (below).

5.6 Alternative Technique

There are many alternative techniques for general-purpose image compression. For example, vector quantization (VQ) algorithms have been used with much success for image compression [NaKi88]. According to Shannon's rate-distortion theory, a better performance is always achievable in theory by coding vectors instead of scalars [Gray84]. In this section, we apply the Lloyd [Lloy79] codebook design algorithm and the *full search equivalent encoding* algorithm [HBSH92] to compress *Lena* in the spacial domain and its corresponding wavelet packet (WPMA) coefficients. Figure 5.33 shows the experimental results.

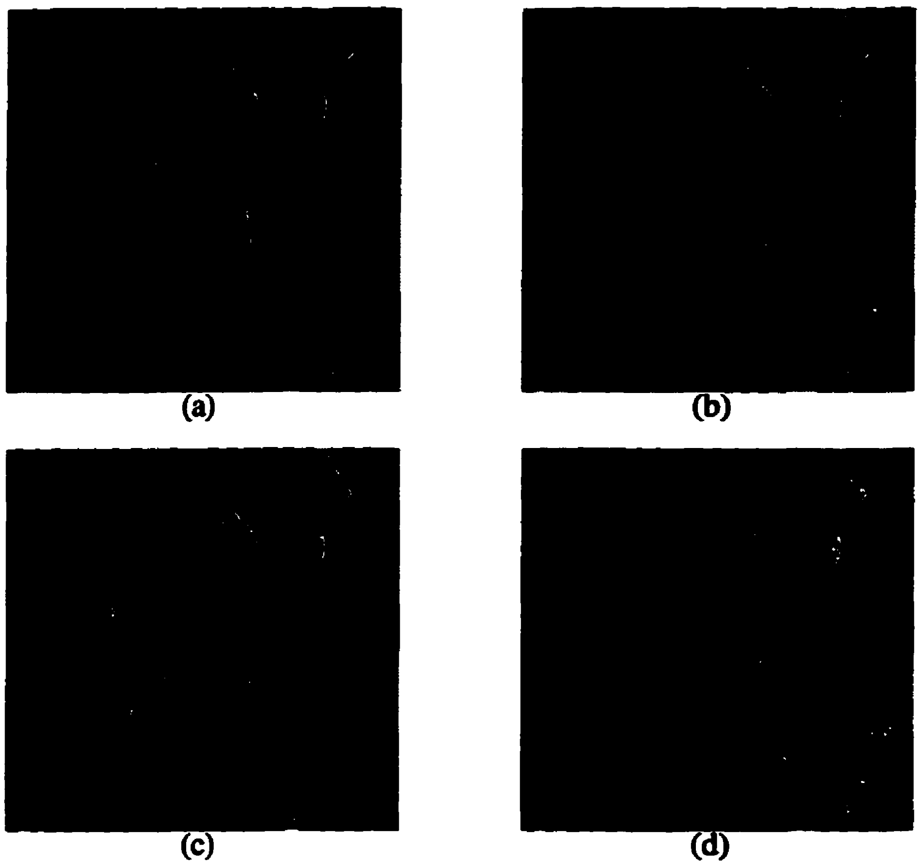


Fig. 5.33.(a) Original image, 256 x 256, 8 bpp, reconstructed image using VQ of (a) with block size 2 x 2, (b) codebook size 256, compression ratio 3.24:1 (2.47 bpp) at PSNR 35.02 dB, (c) codebook size 128, compression ratio 3.6:1 (2.22 bpp) at PSNR 33.26 dB, (d) VQ the wavelet packet (WPMA) coefficients with block size 2 x 2 and codebook size 256, compression ratio 3.24:1 (2.47 bpp) at PSNR 28.07 dB.

From Fig. 5.33, we find that the result of applying VQ to wavelet coefficients of an image is worse than applying it to a spatial image. The reason is that the variety of pixel values in spatial images (from 0 to 255) is smaller than its corresponding wavelet coefficient values (from negative one thousand to positive one thousand). However, the edges are kept well. This technique does not produce good results on images as detailed as fingerprints. Consequently, more elaborate techniques such as our QDMA and WPMA must be employed.

5.7 Review of Chapter 5

In this Chapter, we applied five different compression methods to compress a fingerprint image. Compression quality using the zonal mask method is the worst because of the boundary effect, especially for non-symmetrical images. The direct thresholding method is the easiest to use, but the Gibb's phenomenon [Anso93] restricts its maximum compression ratio. The zerotree algorithm is much more complicated than the previous methods; however, the compression ratio and quality are better than any of those methods. The zerotree method also has the Gibb's phenomenon, which becomes especially significant when increasing the threshold value. Our new QDMA and WPMA methods apply a multifractal spectrum analysis of wavelet coefficients by which we can eliminate the subbands for inverse wavelet transforms. The compression quality is dependent on the bin width of the quantizer. QDMA cannot reach a higher compression ratio because of the restriction produced by quadtree decomposition. To improve the QDMA we propose the WPMA method which also applies multifractal analysis of wavelet coefficients but uses a more complicated wavelet packet (as in WSQ). The WPMA method achieves the highest compression ratio and quality of all five methods. In addition, we applied the VQ technique for both the spatial image and its corresponding wavelet coefficients. Since the variety of the wavelet coefficients is larger than the variety of spatial images, the VQ technique applied for spatial images can achieve better performance than applied it to wavelet coefficients.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

The work described in this thesis was motivated by the need for new and better image compression techniques for fingerprints. The lossless techniques have the best reconstructed quality, but poor compression ratio (in the range of 2:1). On the other hand, lossy techniques have the limitation that the quality of the reconstructed image diminishes with increased compression ratio. The quality requirement of the decompressed fingerprint image is higher than others (such as *Lena*) because the reconstructed image must often be identified and submitted in court as evidence.

The objective of this thesis was to maximize the compression ratio, while minimizing the redundancy of the decompressed image. Since a fingerprint image is a non-stationary signal, we used the wavelet transform in our experiments.

In this thesis, we propose a new technique based on wavelets with multifractal analysis to determine the redundancies of the wavelet coefficients based not only on its magnitude but also on its fractal dimension. The Mandelbrot dimension (multifractal spectrum) is analogous to the entropy of energy (magnitude of wavelet transform coefficients).

Six different experiments were implemented in this thesis. The first two experiments investigated the idea that larger transform coefficients have a larger importance in the reconstructed image. From these experimental results, we found that the compression ratio varied from 3: 1 (zonal mask) to 6: 1 (direct thresholding) for an acceptable quality of the reconstructed image. The advantage of this technique is that it is easy to implement, but its disadvantage is that the Gibb's phenomenon becomes more evident when increasing the compression ratio. The reason is that the same scale of the transform coefficients does not contribute equally to the reconstructed image.

The third experiment involved the embedded zerotree wavelet algorithm (EZW) or zerotree technique for short. From this experiment, we can observe that, this technique is just a little bit better in terms of its compression ratio in the range of 8:1. The reason is that zerotree uses wavelet packet decomposition. As with the previous case, the zerotree method is not suitable for fingerprint image compression because we cannot arbitrarily reset the transform coefficients according to the threshold value, especially when we know that all the ridge information is reserved in subbands 7 to 18, and the majority of transform energy is kept in subbands 0 to 6 (the subband number is defined in Fig. 5.19).

From these three experiments, we learned that each subband (in frequency decomposition) has a different importance even though they have the same magnitude. The zerotree technique depends upon the importance of the parent-child relationship, but the parent-children relationship cannot tell which subband is more important than the others.

In the fourth (QDMA) and fifth (WPMA) experiments we applied a new technique based on wavelets with multifractal analysis. From multifractal spectra analysis, we can determine which subband can be ignored. We used quadtree wavelet decomposition in the fourth experiment. The compression ratio can reach 13.95:1 (0.5734 bpp) with 28.06 dB PSNR (Fig. 5.18).

We also observe that the use of the more complicated wavelet packet instead of the quadtree structure improves the quality of reconstructed images and the compression ratio. This is because, in every decomposition, the energy will concentrate on fewer coefficients and thus give us a better chance for subband selection. The fifth WPMA experiment proved the above assumption. The reconstructed image (as shown in Fig. 5.31) is almost indistinguishable from the original. The compression ratio can reach 17.7:1 (0.4519 bpp) with 28.57 dB PSNR and the quality is good enough for fingerprint identification or classification. From Fig. 5.18 and 5.31, we find that the reconstructed quality of WPMA is better than QDMA. Although the PSNR is only improved by 1.81% (from 28.06 to 28.57 dB), the compression ratio is increased 26.88% (from 13.95:1 to 17.7:1). Also, the Gibb's phenomenon of WPMA is smaller than in QDMA. This compares well to the WSQ (FBI fingerprint compression standard) which can reach 28.72 dB PSNR at a compression ratio 23.3:1. Since we only used *GZIP*, a run-length lossless compression technique, to compress the quantized wavelet coefficients in our experiments, our compression ratio can be further improved if we use better adaptive statistical coding [Kins91]. The final experiment used the vector quantization of a spatial image and its corresponding wavelet coefficient compression. Since the variety of wavelet coefficients is much larger than the variety of the pixel values of the spatial image, the VQ technique is not the best choice for wavelet coefficient compression.

6.2 Contributions

We believe that this thesis and the work described have provided the following contributions:

a. A study of fingerprint interpretation (see Sec. 2.1).

b. A study of the human vision system (see Sec. 2.2).

c. A study of data compression (lossless) techniques (see Sec. 2.3).

d. A study of data compression (lossy) techniques, transform coding (see Sec. 2.4.2).

e. A study of vector quantization (see Secs. 2.4.1.1 and 5.6).

f. A study of bit allocation and quantization (see Secs. 2.4.2.3 and 5.4.2).

g. A study of wavelet transform and wavelet packets (Chapter 3).

h. Software implementation of DWT (Appendix A and Appendix B).

i. A study of fractals and self-similarity including the Hausdorff dimension measure (see Sec. 4.1).

j. A study of multifractals including strange attractors, IFS and the multifractal dimension measures (see Sec. 4.2).

k. Software implementation of IFS, Hénon strange attractors, Rényi dimension and Mandelbrot dimension (Appendix A).

l. Software implementation of direct thresholding wavelet coefficients, zonal masks, zerotree compression, wavelet packets (WSQ) (Appendix A).

m. A study of singularity analysis of subband wavelet coefficients in a multifractal spectrum (Chapter 5).

n. Software implementation of QDMA and WPMA (Appendix A).

6.3 Recommendations

Based on the work of this thesis, we recommend the following:

a. Development of an entropy coding algorithm (lossless) to compress the quantized wavelet coefficients.

b. Development of a more formal theory of OBL and quantization. It is believed that this theory could help in the determination of the loading factors β and C.

c. Development of an algorithm to find the optimum wavelet packet by using multifractal spectrum analysis.

REFERENCES

[AkHa92] Ali N. Akansu and Richard A. Haddad. *Multiresolution Signal Decomposition: Transforms, Subbands, and Wavelets.* San Diego, CA. Academic Press, 1992, 355 pp. {ISBN 0-12-047140-X}

[Anso93] Louisa F. Anson, "Fractal image compression," *Byte*, pp. 195-202, Oct. 1993.

[BaHu93] Michael F. Barnsley and Lyman P. Hurd. *Fractal Image Compression.* Wellesley, MA. AK Peters, 1993, pp. 1-218. {TA1632.B353 1993; ISBN 1-56881-000-8}

[BrBH93] J. N. Bradley, C. M. Brislawn, and T. Hopper, "The FBI wavelet/scale quantization standard for gray-scale fingerprint image compression." in *Visual Info. Process. II*, Vol. 1961 of *Proc. SPIE*, (Orlando, FL), pp. 293-304, Apr. 1993.

[BrBr93] J. N. Bradley and C. M. Brislawn, "Proposed first-generation WSQ bit allocation procedure." *Proc. Symp. Criminal Justice Info. Services Trch.*, (Gaithersbug, MD), pp. C11-C17, Sept. 1993.

[Bris94] C. M. Brislawn, "Classification of nonexpensive symmetric extension transforms for multirate filter banks." Tech. Rep. LA-UR-94-1747, Los Alamos National Laboratories, May 1994.

[CGTL92] Michael M. S. Chong, Robert K. L. Gay, H. N. Tan and J. Liu, "Automatic representation of fingerprints for data compression by B-Spline functions," *Pattern Recognition*, vol. 25, no. 10, pp. 1199-1210, 1992.

[Chap41] Charles E. Chapel. *Fingerprinting: A manual of Identification.* Coward, NY. McCann, 1941, 299 pp.

[ChKu93] Tianhorng Chang and C. C. Jay Kuo, "Texture analysis and classification with tree-structured wavelet transform," *IEEE Trans. on Image Processing*, vol. 2, no. 4, pp. 429-441, Oct. 1993.

[Clar96] R.J. Clarke. *Digital Compression of Still Images and Video.* New York, NY. Academic Press, 1996, 453 pp. {ISBN 0-12-175720-X}

[CORG93] P. C. Cosman, K. L. Oehler, Eve A. Riskin, and R. M. Gray, "Using Vector Quantization for Image Processing." *Proceedings. of IEEE,* 81(9), pp. 1326-1341, Sep. 1993.

[Darr77] Jim Darrach. *Fingerprinting: A Science at your Fingertips.* Agincourt, Canada: The book society of Canada, 1977, 30 pp. {ISBN 0-7725-5117-0}

[Daub92] Ingrid Daubechies. *Ten Lectures on Wavelets.* Philadephia, PN. Capital City Press, 1992, 351 pp. {ISBN 0-89871-274-2}

[ErHJ96] G. Erlebacher, M. Y. Hussaini, and L. M. Jameson. *Wavelets: Theory and Application.* New York, NY. Oxford University Press, 1996, 520 pp. {ISBN 0-19-509423-9}

[FaHV94] M. Farge, J. C. R. Hunt, and J. C. Vassilicos. *Wavelets, Fractals and Fourier Transforms.* New York, NY. Oxford University Press, 1994, 403 pp. {QA403.3.W39; ISBN 0-19-853647-X}

[FeBI93] Federal Bureau of Investigation. *WSQ Gray-Scale Fingerprint Image Compression Specification,* IAFIS-IC-0110v2 (rev. 2.0) , Feb. 1993.

[GoWo92] R.C. Gonzalez and R.E. Woods. *Digital Image Processing.* New York, NY. Addison-Wesley, 1992, 703 pp.

[Gray84] R. M. Gray, "Vector quantization," *IEEE ASSP Magazine,* pp. 4-29, Apr. 1984.

[GrOY87] C. Grebogi, E. Ott, and J. A. Yorke, "Chaos, strange attractors and fractal basin boundaries in nonlinear dynamics." *Phys. Rev. A,* Vol. 37, No. 5, pp. 1711-1724, 1987.

[HBSH92] C. M. Huang, Q. Bi, G. S. Stiles, and R. W. Harris, "Fast full search equivalent ncoding algorithms for image compression using vector quantization," *IEEE Trans. on Image Processing,* vol. 1, no. 3, pp. 413-416, July 1992.

[Held91] G. Held. *Data Compression: Techniques and Applications, Hardware and Software Consideration.* West Sussex, England. John Wiley & Sons, 1991, 405 pp.

[HoPr92] Tom Hopper and Fred Preston, "Compression of grey-scale fingerprint images," *IEEE Comput. Soc. Conf.; DCC '92 Data Compression Conference*, pp. 309-318, 1992.

[HuSh63] J.J.Y. Huang and P.M. Schultheiss, "Block quantization of correlated gaussian random variables," *IEEE Trans. on Comm. systems*, CS-11, no. 3, pp. 289-296, Jan. 1963.

[Jain89] Anil K. Jain. *Fundamentals of Digital Image Processing.* Toronto, ONT. Prentice-Hall, 1989, 560 pp. {ISBN 0-13-336165-9}

[JaKi97] E. Jang and W. Kinsner, "A fingerprint database," *Technical Report,* DEL97-1, Dept. Electrical & Computer Engineering, University of Manitoba, July, 1997. (A CD-ROM containing the database is available on request.)

[Kins91] W. Kinsner, "Review of data compression methods, including Shannon-Fano, Huffman, arithmetic, Storer, Lempel-Ziv-Welch, fractal, neural network, and wavelet algorithms," *Technical Report,* DEL91-1, Dept. Electrical & Computer Engineering, University of Manitoba, 157 pp., Janurary, 1991.

[Kins94] W. Kinsner, "Fractal dimensions: Morphological, entropy, spectrum, and variance classes." *Technical Report*, DEL94-4, Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, Manitoba, Canada, May 1994, pp.140.

[Kins95] W. Kinsner. *Fractal and Chaos Engineering*, 24.721 Course Notes, University of Manitoba, 1995.

[Koho90] T. Kohonen, "The self-organizing map," *Proc. of IEEE*, vol. 78, pp. 1464-1480, 1990.

[Lauw91] Hans Lauwerier. *Fractals: Endlessly repeated geometrical figures.* Princeton, NJ. Princeton University Press, 1991, 205 pp. {ISBN 0-691-08551-X}

[LiBG80] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. on Commun.*, vol. COM-28, pp. 84-95, July 1979.

[Lloy79] S. P. Lloyd, "Asymptotically optimal block quantization," *IEEE Trans. on Inform. Theory*, vol. IT-25, pp. 373-380, Jan. 1980.

[Lync85] T. J. Lynch. *Data Compression: Techniques and Applications.* New York, NY. Van Nostrand Reinhold, 1985, 333 pp. {QA76.9. D33 L96; ISBN 0-534-03418-7}

[NaKi88] N. M. Nasrabadi and R. A. King, "Image coding using vector quantization: A review," *IEEE Trans. on Commun.*, vol. 36, pp. 957-971, Aug. 1988.

[Nugg84] Jayant Nuggehally. *Digital Coding of Waveforms.* Toronto, ONT. Prentice-Hall, 1984, 580 pp. {ISBN 0-13-211913-7-01}

[OpWY75]Alan V. Oppenheim, Alan S. Willsky, and Ian T. Young. *Signals and Systems.* Englewood Cliffs, NJ. Prentice-Hall, 1975, 782 pp. {ISBN 0-13-211913-7-01}

[PeJS92] Heinz-Otto Peitgen, Hartmut Jürgens, and Dietmar Saupe. *Chaos and Fractals: New Frontiers of Science.* New York, NY. Springer-Verlag, 1992, 894 pp. {TA1632.P45 1992; ISBN 0-387-97903-4}

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The art of Scientific Computing.* New York, NY. Cambridge University Press, 1992 pp. 591-606. (2nd edition) {ISBN 0-521-43108-5}

[RaVH96] Kannan Ramchandran, Martin Vetterli, and Cormac Herley, "Wavelets, subband coding and best bases," *IEEE Processing*, vol. 84, no. 4, pp. 541-560, Apr. 1996.

[RiVe91] Olivier Rioul and Martin Vetterli, "Wavelets and signal processing," *IEEE SP Magazine*, pp. 14-38, Oct. 1991.

[SaAn92] Khalid Sayood and Karen Andson, "A differential lossless image compression," *IEEE Trans. on Signal Processing*, vol. 40, no. 1, pp. 236-241, Jan. 1992.

[Schr91] Manfred Schroeder. *Fractal, Chaos, Power Laws: Minutes from an Infinite Paradise.* New York, NY. W.H. Freeman and Company, 1993, pp. 1-141. {ISBN 0-7167-2136-8}

[Shap93] Jerome M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans on Signal Processing*, vol. 41, n0. 12, pp. 3445-3462, 1993.

[StDS95] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin, "Wavelets for computer graphics: A primer, Part 2." *IEEE Computer Graphics and Application*, pp. 75-84, July 1995.

[TaWi71] Manfred Tasto and Paul A. Wintz, "Image coding by adaptive block quantization," *IEEE Trans on Communication Technology*, vol. Com-19, no. 6, pp. 402-418, Dec. 1971.

[Unit78] United States Department of Justice, Federal Bureau of Investigation. *The Science of Fingerprints: Classification and Uses.* Washington, DC: U.S. Government Printing Office, 1978, 209 pp. {Stock No. 027-001-00017-3}

[Vaid90] P. P. Vaidyanathan, "Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial," *IEEE Processing*, vol. 78, no. 1, pp. 56-93, Jan. 1990.

[ViMü93] Brani Vidakovic and Peter Müller, "Wavelets for kids: A tutorial introduction," pp. 1-24, 1993. (Available from ftp at: *ftp://ftp.isds.duke.edu/pub/Users/brani/papers/wav4kids.ps.Z*)

[Vrsc95] Edward R. Vrscay. *A Hitchhiker's Guide to Fractal-based Function Approximation and Image Compression*. Math Ties, Faculty of Mathematics, University of Waterloo, Canada, 1995, 20 pp. (Available from the web at: *http://links.uwaterloo.ca*)

[Wick94] Mladen V. Wickerhauser. *Adapted Wavelet Analysis from Theory to Software*. Wellesley, MA. AK Peters, 1994, 473 pp. (ISBN 1-56881-041-5)

[Wint72] Paul A. Wintz, "Transform picture coding," *Proc. IEEE*, Vol. 60, No. 7, pp. 809-820, July 1972.

[XuKO93] Lei Xu, Adam Krzyzak, and Erkki Oja, "Rival penalized competitive learning for clustering analysis, RBF net, and curve detection," *IEEE Trans. on Neural Networks*, vol. 4, no. 4, pp. 636-649, July 1993.

[ZeNo77] Rainner Zelinski and Peter Noll, "Adaptive transform coding of speech signals," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-25, no. 4, pp. 299-309, Aug. 1977.

APPENDIX A

STRUCTURE OF PROGRAM AND SOURCE CODE IN C

A.1 Structure Charts and Functional Description of QDMA and WPMA Applications

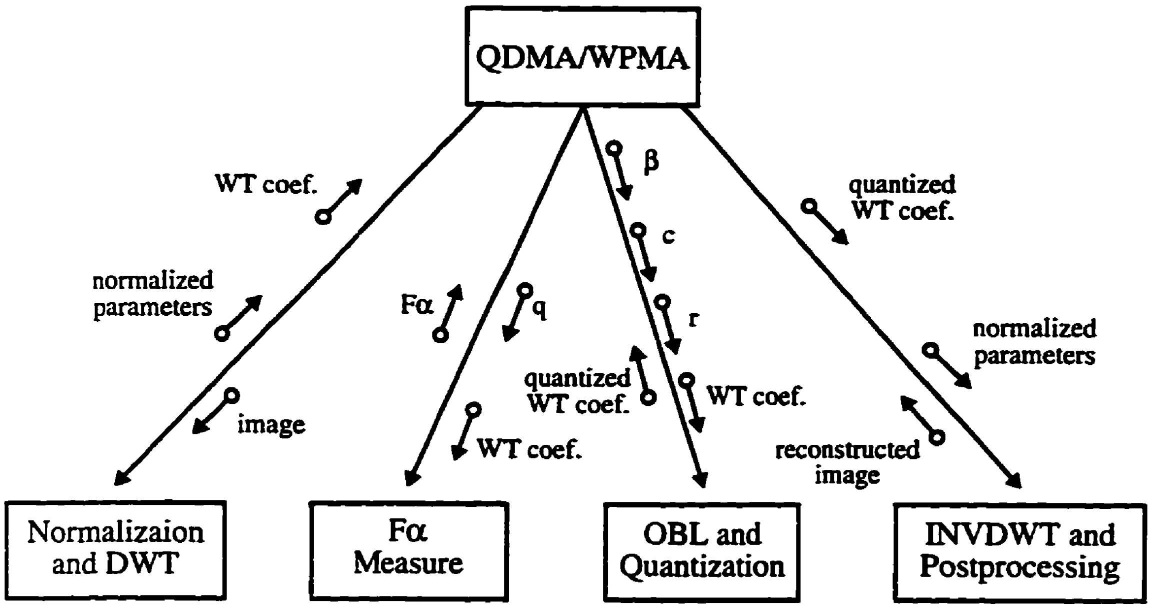


Fig. A.1. Structure chart for QDMA and WPMA applications.

Normalization and DWT

Modules: PRE_QTF, WP_QTF (QDMA) or WSQF (WPMA).

Inputs: Sun raster file (.ras).

Output: Wavelet transform coefficients (in floating format) and normalization parameters, M and R (defined in Section 5.4).

Description: The PRE_QTF module normalizes the input image. The theory is explained in Sec. 5.4. The module DWT can be WP_QTF for QDMA technique or WSQF for WPMA technique. We use DAUB4 wavelet for the transformation.

Fα Measure

Modules: FA_ADV.

Inputs: Wavelet transform coefficients (floating format).
Output: α and Fα (in ASCII format) which can be used for GNUPLOT.
Description: Calculate the Mandelbrot dimension (Fα).

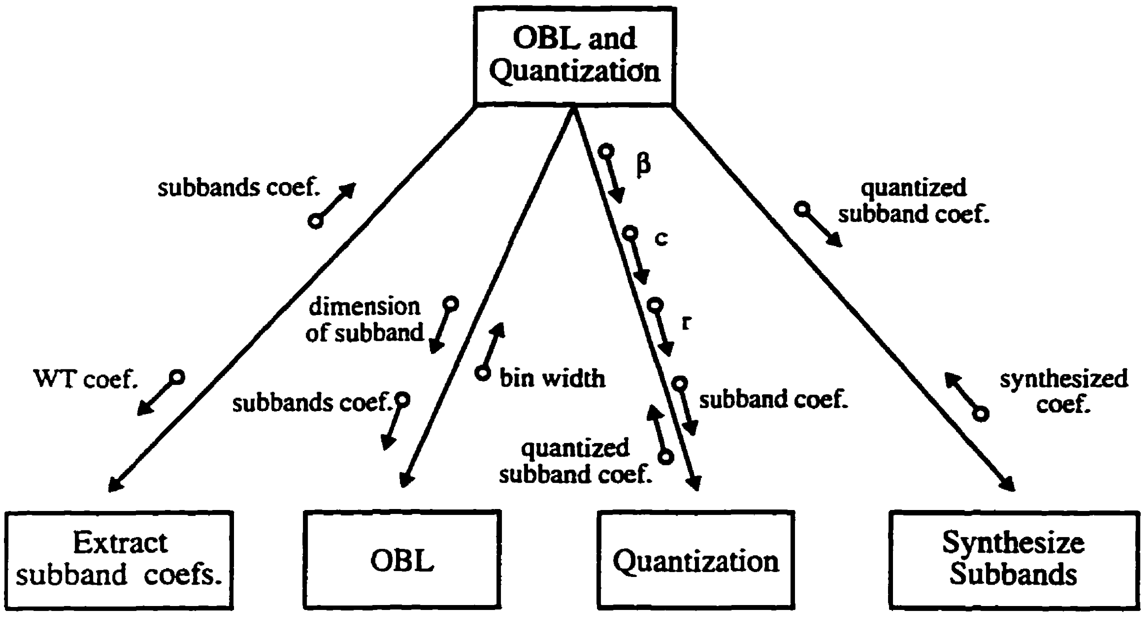


Fig. A.2. Structure chart for OBL and Quantization function and its subordinates.

Extract Subband Coefficients

Modules: SEP_WPQTF.

Inputs: Wavelet transform coefficients (floating format).
Output: Four quarters of the original coefficients.
Description: This function is to separate the original wavelet coefficients into four quarters. Each quarter with an 1/4 size of original. The four quarters are named dul, dur, dll and dlr follow the order of top-down and left-right.

OBL

Modules: OBL_1.

Inputs: Wavelet transform subband coefficients (floating format).
Output: bin width (floating format).
Description: This function is to calculate the variance of each subband. The OBL is defined according to the variance and the decomposition level.

Quantization

Modules: OBL_2.

Inputs: Wavelet transform subband coefficients (floating format), bin width, quantization parameters (C, r, and β).
Output: Quantized subband coefficients.
Description: This function is to quantize the wavelet subband coefficients according to the bin width. The quantization parameters are used for calculating the bin width of each subband and loading factors for dequantization

Synthesize Subbands

Modules: SYN_F.

Inputs: Quantized wavelet transform subband coefficients (short integer format).
Output: processed transform coefficients (floating format).
Description: This function is to synthesize the all processed subbands for inverse wavelet transform.

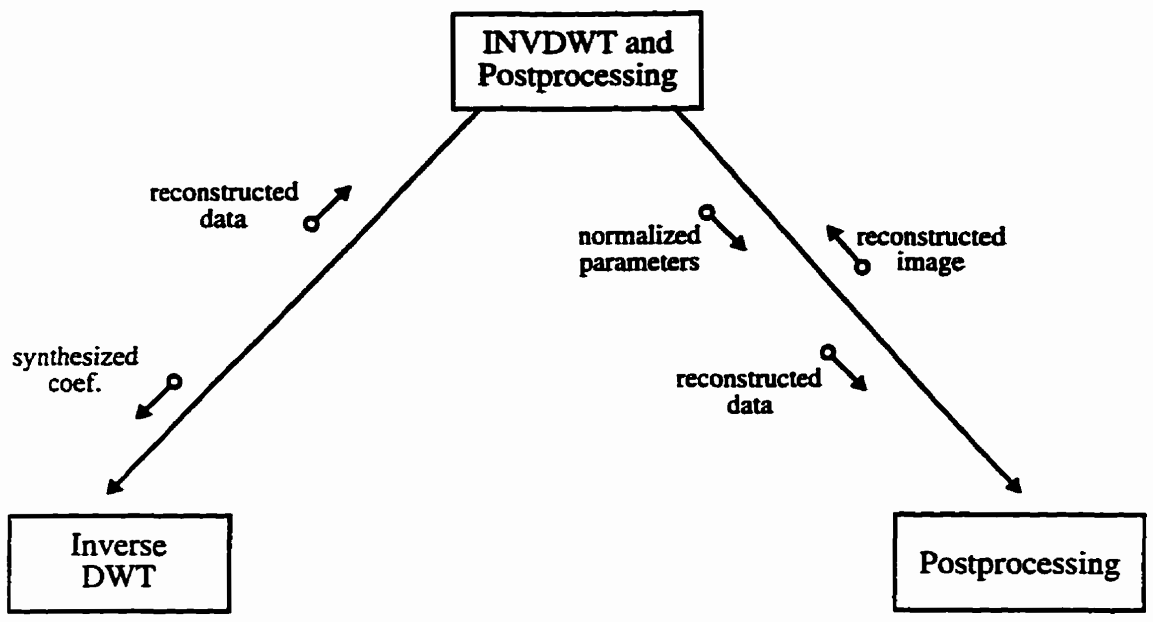


Fig. A.3. Structure chart for INVDWT and Postprocessing function and its subordinates.

INVDWT

Modules: INV_QTF (QMA) or INV_WSQF (WPMA).

Inputs: Synthesized coefficients (floating format).
Output: Reconstructed data.
Description: This function is to calculate the inverse wavelet transform of input data.

Postprocessing

Modules: PRO_QTF.

Inputs: Reconstructed data (floating format) and normalized parameters (M, R).
Output: Reconstructed image (SUN raster format).
Description: This function is to recover the reconstructed data and generate the reconstructed image in SUN raster file format.

A.2 Source Code of WMA in C

```
/****************************************************************************/
/* Name       : PRE_QTF.C */
/* Input      : .ras */
/* Output     : .nas (header[800] + data (float) */
/* Procedure  : Preprocessing the .ras file. Normalization */
/* Data       : 22/Feb/97 */
/* Version    : 1.0 */
/* Designer   : Eric Jang Tel:2476992, Email: jang@ee.umanitoba.ca */
/****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MaxDim 512
#define MaxDimSq 262144 /* 512 X 512 */

main()
{
  FILE *in, *out;

  unsigned char tmp_uc, header[800];
  unsigned long int f_size, k;
  char infile[70], outfile[70], conti;
  int i, h_size, v_size;
  float buf[MaxDimSq], max, min, M, R, a1, a2, tmp_f;
  double tmp_df;

  printf("Normalization ver. 1.0, 22/Feb/1997\n\n");
  printf("Enter input file name(float, path\\XXX.ras): \n");
  scanf("%s",infile);
  strcpy(outfile, infile);
  conti = '.';
   /* Search the input file name string to find the "." then replace the extention to "enc". */
  for (i = 0; i < 70; i++)
  {
    if (infile[i] == conti)
    {
      outfile[i+1] = 'n';
      break;
    }
  }

   /* Open input files. */
  if ((in = fopen(infile, "rb")) == NULL)
  {
    fprintf(stderr, "Cannot open input file.\n");
    exit (1);
  }
  /* Open output file. */
   if ((out = fopen(outfile, "w")) == NULL)
  {
    fprintf(stderr, "Cannot open output file.\n");
    exit (1);
  }

/* extract .ras file */
  fread(header, sizeof(unsigned char), 800, in);
  fwrite(header, sizeof(unsigned char), 800, out);
  h_size = (int)header[6]*256 + (int)header[7];
  v_size = (int)header[10]*256 + (int)header[11];
  f_size = h_size * v_size;
  if(h_size != v_size)
  {
    printf("Error, h_size != v_size.\n");
    exit (1);
  }
  else
    printf("h_size and v_size = %d\t f_size = %d\n", h_size, f_size);

  if(h_size > MaxDim)
  {
    printf("Error, dimension out of range(%d)\n", MaxDim);
    exit (1);
  }

  /* get ras file data and put it into buffer */
  for(k = 0; k < f_size; k ++)
  {
    fread(&tmp_uc, sizeof(unsigned char), 1, in);
    buf[k] = (float)tmp_uc;
  }

/* find maximum and minimum */
  max = -1000.0;
  min =  1000.0;
  tmp_df = 0;
  for(k = 0; k < f_size; k++)
  {
    tmp_df = tmp_df + (double)buf[k];
    if(buf[k] > max)
```

```
        max = buf[k];
      else if(buf[k] < min)
        min = buf[k];
    }
    M = (float)(tmp_df/(double)f_size);
    a1 = max - M;
    a2 = M - min;
    if(a1 >= a2)
      R = (1.0/128.0) * a1;
    else
      R = (1.0/128.0) * a2;

    printf("In file %s , max = %g  min = %g  mean = %g  R = %g\n", infile, max, min, M, R);
    for(k = 0; k < f_size; k++)
    {
      tmp_f = ((float)buf[k] - M)/R;
      fwrite(&tmp_f, sizeof(float), 1, out);  /* float format */
    }

    fclose(in);
    fclose(out);
}
```

```
/*****************************************************************/
/* Name        : WP_QTF.C */
/* Input       : .nas (header[800] + data[float) */
/* Output      : float, short int */
/* Procedure   : Wavelet Packet for Quard Tree */
/* Date        : 13/Nov/95, 17/Sep/96, 13/Oct, 20, 21, 22/Feb/97 */
/* Version     : 1.0, 2.0, 2.1, 3.0, 3.1, 3.2 */
/* Designer    : Eric Jang  Tel:(204)474-6991  jang@ee.umanitoba.ca */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define true 0
#define false 1
#define MaxDim 512
#define MaxDimSq 262144 /* 512 X 512 */

int dim;
float conv(int ptr1, int ptr2, double *ptr3, float *ptr4);
void d_sampling(int ptr5, int ptr6, int ptr7, int ptr8, int ptr9, float *ptr10);

main()
{
  FILE *in, *out, *out1, *out2;

  int i, h_size, v_size, px, py, dim_tmp, dim_org, tmp_i, level, si_en;
  short int tmp_si;
  char inf[70], co;
  float buf[MaxDimSq], tmp, max, min;
  double sum;
  unsigned char header[800], temp;
  unsigned long int f_size, sum_ui;

  printf("Wavelet Packet for Quard Tree ver. 3.2, 22/Feb/1997\n\n");
  printf("Enter original image file name(path\\XXX.nas, generating by pre.c): \n");
  scanf("%s",inf);
  printf("Please enter level(J <= 5):\n");
  scanf("%i",&level);
  printf("Output the WT_ZT coefficients in short int(Y -> yes, others -> no)?\n");
  co = getc(stdin);
  co = getc(stdin);
  if(co == 'y')
    si_en = true;
  else
    si_en = false;
```

```
/* Open input .ras file *in, get image data  */
 if((in = fopen(inf, "r")) == NULL)
{
   fprintf(stderr, "Cannot open input file \n");
   exit(1);
}
fread(header, sizeof(unsigned char), 800, in);
h_size = (int)header[6]*256 + (int)header[7];
v_size = (int)header[10]*256 + (int)header[11];
if(h_size != v_size)
{
   printf("Error, h_size != v_size.\n");
   exit (1);
}
else
   printf("h_size and v_size = %d\n", h_size);

if(h_size > MaxDim)
{
   printf("Error, dimension out of range(%d)\n", MaxDim);
   exit (1);
}
f_size = h_size * v_size;
dim = h_size;
dim_org = dim;

/* get ras file data and put it into buffer */
for(i = 0; i < f_size; i ++)
{
   fread(&tmp, sizeof(float), 1, in);
   buf[i] = tmp;
}
printf("W_packet for zerotree coding level 0 is ok !\n");
px = 0;
py = 0;
dim_tmp = dim;
dim_org = dim;
tmp_i = 1;
for(i = 1; i <= level; i++)
{
   d_sampling(px, py, dim_tmp, dim_tmp, dim_tmp, buf);
   printf("W_packet for zerotree coding level %d is ok !\n", i);
   tmp_i = tmp_i * 2;
   dim_tmp = dim/tmp_i;
}
```

```
/* Open output file *out, store WT result data */
if((out = fopen("wp_ztf.cof", "w")) == NULL)
{
  fprintf(stderr, "Cannot open output file \n");
  exit(1);
}

if((out1 = fopen("sum.dat", "w")) == NULL)
{
  fprintf(stderr, "Cannot open output file \n");
  exit(1);
}

if((out2 = fopen("wp_zti.cof", "w")) == NULL)
{
  fprintf(stderr, "Cannot open output file \n");
  exit(1);
}

/* find max, min and sum(base of probability) */
max    = -100.0; /* initialization */
min    =  100.0;
sum    =    0.0;
sum_ui =    0;
for(i = 0; i < f_size; i++)
{
  if(buf[i] > max)
    max = buf[i];
  else if(buf[i] < min)
    min = buf[i];

  sum_ui = sum_ui + (unsigned long int)(fabs((double)buf[i]));
  if(buf[i] < 0.0)
    sum    = sum    - (double)buf[i]; /* abs(buf[i]) */
  else
    sum    = sum    + (double)buf[i];
}

printf("max = %g\tmin = %g\n", max, min);
printf("sum of all coefficients = %lg   (int)%ld\n", sum, sum_ui);
/* store WT coefficients into file */
for(i = 0; i < f_size; i ++)
{
  if(si_en == true)
  {
      tmp_si = (short int)buf[i];
      fwrite(&tmp_si, sizeof(short int), 1, out2);
      sum = (double)sum_ui;
    }
    else
    {
      tmp = buf[i];
      fwrite(&tmp, sizeof(float), 1, out);
    }
  }

  fwrite(&sum, sizeof(double), 1, out1);
  fclose(in);
  fclose(out);
  fclose(out1);
  printf("finish !\n");
}

/*------------------------------------------------------------------------------*/
/* Array preperation and down sampling subroutine */
/* inputs       : starting location, dimension, data array */
/* return       : void */
/* procedure    : according to the starting position and dimension */
/*                to prepare the 1 dimension data array, down sampling*/
/*------------------------------------------------------------------------------*/
void d_sampling(int p_x, int p_y, int lgth, int lgthH, int lgthV, float *mat)
{
  int i, j, k, l, m, bndary;
  float data_ary[MaxDim];
  double h_flt[4], g_flt[4];

  h_flt[0] =  0.4829629131445341;
  h_flt[1] =  0.8365163037378079;
  h_flt[2] =  0.2241438680420134;
  h_flt[3] = -0.1294095225512604;

  g_flt[0] = -0.1294095225512604;
  g_flt[1] = -0.2241438680420134;
  g_flt[2] =  0.8365163037378079;
  g_flt[3] = -0.4829629131445341;

  /* x position(horizontal) */
  for(i = p_y; i < lgthV; i++)
  {
    k = 0;
    for(j = p_x; j < lgthH; j++)
```

```
        {
          data_ary[k] = mat[i*dim + j];
            k = k + 1;
        }
        l = 0;
        for(m = 0; m < lgth; m = m+2)
        {
          if(m == (lgth - 2))
            bndary = true;
          else
            bndary = false;

          mat[i*dim + p_x + lgth/2 + l] = conv(m, bndary, g_flt, data_ary);
          mat[i*dim + p_x + l] = conv(m, bndary, h_flt, data_ary);
          l = l + 1;
        }
      }

      /* y position(vertical) */
      for(i = p_x; i < lgthH; i++)
      {
        k = 0;
        for(j = p_y; j < lgthV; j++)
        {
          data_ary[k] = mat[j*dim + i];
            k = k + 1;
        }
        l = 0;
        for(m = 0; m < lgth; m = m+2)
        {
          if(m == (lgth - 2))
            bndary = true;
          else
          bndary = false;
          j = p_y;
          mat[(j+lgth/2+l)*dim + i] = conv(m, bndary, g_flt, data_ary);
          mat[(j+l)*dim + i] = conv(m, bndary, h_flt, data_ary);
          l = l + 1;
        }
      }
    }

    /*------------------------------------------------------------------------------*/
    /* 1 dimention convolution subroutine */
    /* inputs       : filter array, data array both are 1 dimension */
    /* return       : convoluted result(short int) */
```

```
/* procedure    : sum(Ai * Bi) */
/*------------------------------------------------------------------------------*/
float conv(int pos, int boundary, double *filter, float *data)
{
  int i;
  float result;
  double temp, sum;

/* pos       :starting position of the filter */
/* boundary :boundary flag */
  sum = 0.0;     /* initialization */

  for(i = 0; i < 4; i++)
  {
    if(boundary != true)
    {
      temp = filter[i] * (double)data[pos];
      pos = pos + 1;
      sum = sum + temp;
    }
    else if(i < 2)
    {
      temp = filter[i] * (double)data[pos];
      pos = pos + 1;
      sum = sum + temp;
    }
    else
    {
/*      pos = pos - 1;
      temp = filter[i] * (double)data[pos];    */
      temp = filter[i] * (double)data[i - 2];
      sum = sum + temp;
    }
  }
  result = (float)sum;
  return result;
}
```

```
/*****************************************************************/
/* Name       : WSQF.C */
/* Input      : .nas (header[800] + data(float) */
/* Output     : wsqf.cof (float) */
/* Procedure  : Wavelet Packet, WSQ floating */
/* Date       : 14/Oct/95, 31, 6/Nov, 8, 2/Mar/97 */
/* Version    : 1.0, 1.1, 1.2, 1.3, 2.0 */
/* Designer   : Eric Jang  Tel:(204)474-6991  jang@ee.umanitoba.ca */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define true 0
#define false 1
#define MaxDim 512
#define MaxDimSq 262144 /* 512 X 512 */

int dim;
float conv(int ptr1, int ptr2, double *ptr3, float *ptr4);
void d_sampling(int ptr5, int ptr6, int ptr7, int ptr8, int ptr9, float *ptr10);

main()
{
  FILE *in, *out;

  int i, h_size, v_size, px, py, dim_tmp, dim_org, tmp_i;
  char inf[70];
  float buf[MaxDimSq], tmp;
  unsigned char header[800], temp;
  unsigned long int f_size;

  printf("WSQ(FBI) ver. 2.0, 2/Mar/1997\n\n");
  printf("Enter normalized image file name(path\\XXX.nas): \n");
  scanf("%s",inf);
  /* Open input .nas file *in, get image data  */
  if((in = fopen(inf, "r")) == NULL)
  {
    fprintf(stderr, "Cannot open input file \n");
    exit(1);
  }
  fread(header, sizeof(unsigned char), 800, in);
  h_size = (int)header[6]*256 + (int)header[7];
  v_size = (int)header[10]*256 + (int)header[11];
  if(h_size != v_size)
  {
```

```
  printf("Error, h_size != v_size.\n");
  exit (1);
}
else
  printf("h_size and v_size = %d\n", h_size);

if(h_size > MaxDim)
{
  printf("Error, dimension out of range(%d)\n", MaxDim);
  exit (1);
}

f_size = h_size * v_size;
dim = h_size;
dim_org = dim;
/* get ras file data and put it into buffer */
for(i = 0; i < f_size; i ++)
{
  fread(&tmp, sizeof(float), 1, in);
  buf[i] = tmp;
}
printf("WSQ level 0 is ok !\n");

/* level 1 */
px = 0;
py = 0;
dim_tmp = dim;
dim_org = dim;
d_sampling(px, py, dim_tmp, dim_org, dim_org, buf);
printf("WSQ level 1 is ok !\n");
/* level 2 */
dim_tmp = dim/2;
dim_org = dim;
px = 0;
py = 0;
d_sampling(px, py, dim_tmp, dim_tmp, dim_tmp, buf);
px = dim_tmp;
py = 0;
d_sampling(px, py, dim_tmp, dim_org, dim_tmp, buf);
px = 0;
py = dim_tmp;
d_sampling(px, py, dim_tmp, dim_tmp, dim_org, buf);
px = dim_tmp;
py = dim_tmp;
d_sampling(px, py, dim_tmp, dim_org, dim_org, buf);
printf("WSQ level 2 is ok !\n");
```

```
/* level 3 */
dim_tmp = dim/4;
dim_org = dim/2;
px = 0;
py = 0;
d_sampling(px, py, dim_tmp, dim_tmp, dim_tmp, buf);
px = dim_tmp;
py = 0;
d_sampling(px, py, dim_tmp, dim_org, dim_tmp, buf);
px = 0;
py = dim_tmp;
d_sampling(px, py, dim_tmp, dim_tmp, dim_org, buf);
printf("WSQ level 3 is ok !\n");
/* level 4 */
/* part 4-1 */
  dim_tmp = dim/8;
  dim_org = dim/4;
  px = 0;
  py = 0;
  d_sampling(px, py, dim_tmp, dim_tmp, dim_tmp, buf);
  px = dim_tmp;
  py = 0;
  d_sampling(px, py, dim_tmp, dim_org, dim_tmp, buf);
  px = 0;
  py = dim_tmp;
  d_sampling(px, py, dim_tmp, dim_tmp, dim_org, buf);
  px = dim_tmp;
  py = dim_tmp;
  d_sampling(px, py, dim_tmp, dim_org, dim_org, buf);
/* part 4-2 */
  dim_tmp = dim/8;
  dim_org = dim/4;
  px = dim_org;
  py = 0;
  d_sampling(px, py, dim_tmp, px+dim_tmp, dim_tmp, buf);
  px = dim_org + dim_tmp;
  py = 0;
  d_sampling(px, py, dim_tmp, px+dim_tmp, dim_tmp, buf);
  px = dim_org;
  py = dim_tmp;
  d_sampling(px, py, dim_tmp, px+dim_tmp, py+dim_tmp, buf);
  px = dim_org + dim_tmp;
  py = dim_tmp;
  d_sampling(px, py, dim_tmp, px+dim_tmp, py+dim_tmp, buf);
/* part 4-3 */
  dim_tmp = dim/8;
```

```
    dim_org = dim/4;
    px = 0;
    py = dim_org;
    d_sampling(px, py, dim_tmp, dim_tmp, py+dim_tmp, buf);
    px = dim_tmp;
    py = dim_org;
    d_sampling(px, py, dim_tmp, px+dim_tmp, py+dim_tmp, buf);
    px = 0;
    py = dim_org + dim_tmp;
    d_sampling(px, py, dim_tmp, dim_tmp, py+dim_tmp, buf);
    px = dim_tmp;
    py = dim_org + dim_tmp;
    d_sampling(px, py, dim_tmp, px+dim_tmp, py+dim_tmp, buf);
    printf("WSQ level 4 is ok !\n");
  /* level 5 */
  dim_tmp = dim/16;
  dim_org = dim/8;
  px = 0;
  py = 0:
  d_sampling(px, py, dim_tmp, dim_tmp, dim_tmp, buf);
  printf("WSQ level 5 is ok !\n");

  /* Open output file *out, store WT result data */
  if((out = fopen("wsqf.cof", "w")) == NULL)
  {
    fprintf(stderr, "Cannot open output file \n");
    exit(1);
  }

  /* store WT coefficients into file */
  for(i = 0; i < f_size; i ++)
  {
    tmp = buf[i];
    fwrite(&tmp, sizeof(float), 1, out);
  }
  fclose(in);
  fclose(out);
  printf("finish !\n");
}

/*-------------------------------------------------------------------------------*/
/* Array preperation and down sampling subroutine */
/* inputs       : starting location, dimension, data array  */
/* return       : void */
/* procedure    : according to the starting position and dimension */
/*                to prepare the 1 dimension data array, down sampling*/
```

```
/*------------------------------------------------------------------------*/
void d_sampling(int p_x, int p_y, int lgth, int lgthH, int lgthV, float *mat)
{
  int i, j, k, l, m, bndary;
  float data_ary[MaxDim];
  double h_flt[4], g_flt[4];

  h_flt[0] =  0.4829629131445341;
  h_flt[1] =  0.8365163037378079;
  h_flt[2] =  0.2241438680420134;
  h_flt[3] = -0.1294095225512604;

  g_flt[0] = -0.1294095225512604;
  g_flt[1] = -0.2241438680420134;
  g_flt[2] =  0.8365163037378079;
  g_flt[3] = -0.4829629131445341;

  /* x position(horizontal) */
  for(i = p_y; i < lgthV; i++)
  {
    k = 0;
    for(j = p_x; j < lgthH; j++)
    {
      data_ary[k] = mat[i*dim + j];
      k = k + 1;
    }
    l = 0;
    for(m = 0; m < lgth; m = m+2)
    {
      if(m == (lgth - 2))
        bndary = true;
      else
        bndary = false;

    mat[i*dim + p_x + lgth/2 + l] = conv(m, bndary, g_flt, data_ary);
    mat[i*dim + p_x + l] = conv(m, bndary, h_flt, data_ary);
    l = l + 1;
    }
  }

  /* y position(vertical) */
  for(i = p_x; i < lgthH; i++)
  {
    k = 0;
    for(j = p_y; j < lgthV; j++)
    {
      data_ary[k] = mat[j*dim + i];
      k = k + 1;
    }
    l = 0;
    for(m = 0; m < lgth; m = m+2)
    {
      if(m == (lgth - 2))
        bndary = true;
      else
        bndary = false;
      j = p_y;
      mat[(j+lgth/2+l)*dim + i] = conv(m, bndary, g_flt, data_ary);
      mat[(j+l)*dim + i] = conv(m, bndary, h_flt, data_ary);
      l = l + 1;
    }
  }
}

/*-----------------------------------------------------------------------------------*/
/* 1 dimention convolution subroutine */
/* inputs       : filter array, data array both are 1 dimension */
/* return       : convoluted result(short int) */
/* procedure    : sum(Ai * Bi) */
/*-----------------------------------------------------------------------------------*/
float conv(int pos, int boundary, double *filter, float *data)
{
  int i;
  float result;
  double temp, sum;

/* pos: starting position of the filter */
/* boundary :boundary flag */
  sum = 0.0;    /* initialization */
  for(i = 0; i < 4; i++)
  {
    if(boundary != true)
    {
      temp = filter[i] * (double)data[pos];
      pos = pos + 1;
      sum = sum + temp;
    }
    else if(i < 2)
    {
      temp = filter[i] * (double)data[pos];
      pos = pos + 1;
      sum = sum + temp;
    }
    else
    {
/*      pos = pos - 1;
      temp = filter[i] * (double)data[pos];    */
      temp = filter[i] * (double)data[i - 2];
      sum = sum + temp;
    }
  }
  result = (float)sum;
  return result;
}
```

```
/******************************************************************************/
/* Name        : SP_WPZTF.C */
/* Procedure   :Seperate a N X N (N <= 512) data (float) into four N/4 X N/4 (s. int) blocks.  And
               set the output file name with ext. .dur; .dul; .dll; .dlr */
/* Data        : 4/May/95, 17/Sep/96, 13/Oct, 20/Feb/97 */
/* Version     : 1.0, 2.0, 2.1, 2.2 */
/* Designer    : Eric Jang Tel:2476992, Email: jang@ee.umanitoba.ca */
/******************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define true 0
#define false 1

void change_size(int *ptr1, char *ptr3);

main()
{
  int new_dim;
  char infile[70];

  printf("Please enter resized factor n < 8 (should be a power of 2):\n");
  printf("***        H_size == V_size for this version        ***\n");
  scanf("%i",&new_dim);
  printf("Enter file name(path\\**i.cof(float), generated by wp_zt.c): \n\n");
  scanf("%s",infile);
  change_size(&new_dim, infile);
  printf("\n\nCompleting!!!\n");
  return 0;
}

/*----------------------------------------------------------------------------*/
void change_size(int *ptr_m, char file[70])
{
  FILE *in, *out1, *out2, *out3, *out4;

  int i,j, k, org, n, m;
  char outfile1[70], outfile2[70], outfile3[70], outfile4[70], conti;
  int tmp_i;
  float tmp_f;

  m = *ptr_m;  /* Resized image dimension */
  strcpy(outfile1, file);
  strcpy(outfile2, file);
  strcpy(outfile3, file);
  strcpy(outfile4, file);
```

```
   conti = '.';

 /* Search the input file name string to find the "." and then replace the  */
 /* extention to "d**". */
   for (i = 0; i <= 70; i++)
   {
     if (file[i] == conti)
     {
       outfile1[i+1] = 'd';
       outfile1[i+2] = 'u';
       outfile1[i+3] = 'l';
       outfile2[i+1] = 'd';
       outfile2[i+2] = 'u';
       outfile2[i+3] = 'r';
       outfile3[i+1] = 'd';
       outfile3[i+2] = 'l';
       outfile3[i+3] = 'l';
       outfile4[i+1] = 'd';
       outfile4[i+2] = 'l';
       outfile4[i+3] = 'r';
       break;
     }
   }

 /* Open input file */
   if ((in = fopen(file, "r")) == NULL)
   {
     fprintf(stderr, "Cannot open input file.\n");
     exit (1);
   }

 /* Open output file */
   if ((out1 = fopen(outfile1, "w")) == NULL)
   {
     fprintf(stderr, "Cannot open output file.\n");
     exit (1);
   }
   if ((out2 = fopen(outfile2, "w")) == NULL)
   {
     fprintf(stderr, "Cannot open output file.\n");
     exit (1);
   }
   if ((out3 = fopen(outfile3, "w")) == NULL)
   {
     fprintf(stderr, "Cannot open output file.\n");
     exit (1);
```

```
  }
  if ((out4 = fopen(outfile4, "w")) == NULL)
  {
    fprintf(stderr, "Cannot open output file.\n");
    exit (1);
  }

  printf("Please enter original image dimension:\n");
  scanf("%i",&tmp_i);
  org = tmp_i; /* Original image dimension */
  n = org/m;
  printf("original dimension = %d\tresized dimension = %d\n", org, n);
  for(k = 0; k < org; k++ )
  {
    for(j = 0; j < org; j++ )
    {
      fread(&tmp_f, sizeof(float), 1, in);
      if((j < n) && (k < n))
        fwrite(&tmp_f, sizeof(float), 1, out1);
      else if((j >= n) && (k < n))
        fwrite(&tmp_f, sizeof(float), 1, out2);
      else if((j < n) && (k >= n))
        fwrite(&tmp_f, sizeof(float), 1, out3);
      else if((j >= n) && (k >= n))
        fwrite(&tmp_f, sizeof(float), 1, out4);
    }
  }
  fclose(in);
  fclose(out1);
  fclose(out2);
  fclose(out3);
  fclose(out4);
  printf("Finish output files !\n");
}
```

```
/**************************************************************************/
/* Name       : FA_ADV.C */
/* Input      : .ras, flt, int(short) */
/* Output     : ASCII (double) */
/* Procedure  :Calculate the Mandelbrot dimension Fa for WT coefficients generated by
               wp_qtf.c, wsqf.c or image .ras. This is a adaptive C program */
/* Note       : This program is a modified version from DQFA_ADV.C */
/* Information of DQFA_ADV.C */
/*             Date : 11 Dec/95, 12, 17/Sep/96, 9/Oct, 13, 14, 15, 18 */
/*             Version  : 1.0, 1.1, 2.0, 2.1, 2.2, 3.0, 3.1, 4.0 */
/* Version    : 1.0 */
/* Date       : 7/Mar/97 */
/* Designer   : Eric Jang  Tel:(204)474-6991  jang@ee.umanitoba.ca */
/**************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define true 0
#define false 1
#define MaxDim 512
#define MaxDimSQ 262144 /* 512 X 512 */

main()
{
  FILE  *in, *in1, *out1;

  int i, j, k, px, py, h_size, v_size, mode, box_cnt, q1, N, N1, N0;
  short int tmp_si;
  char inf[70], outfile[70], co;
  unsigned char header[800], tmp_uc;
  float data[MaxDim][MaxDim];
  float q_org, tmp_f;
  double prob, prob1, q, Iq[2001], Dq[2001], Aq[2000], Fq[2000], sum, sum1, tmp_df, tmp_df1,
              tmp_df2, Aq_tmp, h1;

  printf("Mandelbrot dimension FA_ADV ver. 1.0, 7/Mar/1997\n\n");
  printf("Please select input mode(1 -> ras, 2 ->flt, 3 -> int(short)\n");
  scanf("%i",&mode);
  printf("Enter input file name(path\\XXf/XXi.cof, .ras): \n");
  scanf("%s",inf);
  printf("Enter output file name(path\\XXX./dfq): \n");
  scanf("%s", outfile);
  printf("Please enter q(abs(q) <= 100, usually 30 is enough, q starts from negative value):\n");
  scanf("%g",&q_org);
```

```
/*  q_org = -30.0; */
q1 = abs((int)q_org);
N  = (q1 * 20) + 1;
N0 = (q1 * 10);
N1 = (q1 + 1) * 10;

/* Open input files */
if(mode != 1)  /* wt coefficients */
{
  if((in = fopen(inf, "r")) == NULL)
  {
    fprintf(stderr, "Cannot open input file \n");
    exit(1);
  }
  printf("Please enter h_size:\n");
  scanf("%i",&h_size);
  printf("Please enter v_size:\n");
  scanf("%i",&v_size);

  /* get input file data and put it into buffer */
  sum = 0.0;
  sum = 0.0;
  for(i = 0; i < v_size; i ++)
  {
    for(j = 0; j < h_size; j++)
    {
      if(mode == 2) /* flt format */
      {
        fread(&tmp_f, sizeof(float), 1, in);
        if(tmp_f < 0.0)
          tmp_f = 0.0 - tmp_f;
        data[i][j] = tmp_f;
        sum = sum + (double)tmp_f;
      }
      else
      {
        fread(&tmp_si, sizeof(short int), 1, in);
        if(tmp_si < 0.0)
          tmp_si = 0 - tmp_si;
        data[i][j] = (float)tmp_si;
        sum = sum + (double)tmp_si;
      }
    }
  }
  printf("finished loading wt coefficients into buffer and sum_si, sum_flt\n");
}
```

```
else /* .ras file */
{
  if((in = fopen(inf, "rb")) == NULL)
  {
    fprintf(stderr, "Cannot open input file(.ras)\n");
    exit(1);
  }
  fread(header, sizeof(unsigned char), 800, in);
  h_size = (int)header[6]*256  + (int)header[7];
  v_size = (int)header[10]*256 + (int)header[11];

  /* get input file data and put it into buffer */
  sum = 0.0;
  for(i = 0; i < v_size; i++)
  {
    for(j = 0; j < h_size; j++)
    {
      fread(&tmp_uc, sizeof(unsigned char), 1, in);
      data[i][j] = (float)tmp_uc;
      sum = sum + (double)tmp_uc;
    }
  }
  printf("finished loading .ras into buffer and sum_ras\n");
}

printf("\nload sum(base of probability) ? y --> yes, others(laocl calculation) --> no: \n\n");
co = getc(stdin);
co = getc(stdin);
if (co == 'y')
{
  if((in1 = fopen("sum.dat", "r")) == NULL)
  {
    fprintf(stderr, "Cannot open input file(sum.dat) \n");
    exit(1);
  }
  fread(&tmp_df, sizeof(double), 1, in1);
  sum1 = tmp_df;
  printf("finish loading sum.dat file\n");
  fclose(in1);
}
else
  sum1 = sum;

if((h_size > MaxDim) || (v_size > MaxDim))
{
  printf("Error, dimension out of range(%d)\n", MaxDim);
  exit (1);
}
else
  printf("h_size = %d\tv_size = %d\n", h_size, v_size);

printf("file size = %d\n", h_size*v_size);
printf("sum(summation of input data) = %lg\n", sum);
printf("sum1(base of probability) = %lg\n", sum1);
printf("q starts from %g\tN = %d\tN0 = %d\tN1 = %d\n", q_org, N, N0, N1);

/* Open output file *out, store q, Dq, *out1, store a, Fa */
if((out1 = fopen(outfile, "w")) == NULL)
{
  fprintf(stderr, "Cannot open output file 1\n");
  exit(1);
}

/* initialization */
box_cnt = 0;
h1   = 0.0;

for(py = 0; py < v_size; py++)
{
  for(px = 0; px < h_size; px++)
  {
    tmp_df = (double)data[py][px];
    if(tmp_df != 0.0)
    {
      box_cnt = box_cnt + 1;
      prob = tmp_df/sum;
      h1 = h1 + (prob * log10(1.0/prob));
    }
  }
}
printf("box count = %d\n", box_cnt);

/* initialization */
Aq_tmp = 0.0;
for(i = 0; i < N; i++)
{
  Iq[i] = 0.0;
  Aq[i] = 0.0;
  Fq[i] = 0.0;
}
```

```
/* Calculate Fq, Aq, from -q to +q */
for(py = 0; py < v_size; py++)
{
  for(px = 0; px < h_size; px++)
  {
    tmp_df = (double)data[py][px];
    if(tmp_df != 0.0)
    {
      prob = tmp_df/sum1;
      prob1 = 1.0/prob;
      Aq_tmp = Aq_tmp + log10(prob1);
    }

    q  = (double)q_org;
    for(i = 0; i < N0; i++)
    {
      if(tmp_df != 0.0)  /* Iq, q != 0, 1 */
      {
        tmp_df1 = pow(tmp_df, q);
        tmp_df2 = 1.0/tmp_df1;
        Iq[i] = Iq[i] + tmp_df1;
        Iq[(N-1)-i] = Iq[(N-1)-i] + tmp_df2;

        /* Calculate fq, from -q to +q, new technology */
        Aq[i] = Aq[i] + tmp_df1*log10(prob1);
        Aq[(N-1)-i] = Aq[(N-1)-i] + tmp_df2*log10(prob1);
        Fq[i] = Fq[i] + q*tmp_df1*log10(tmp_df);
        Fq[(N-1)-i] = Fq[(N-1)-i] - q*tmp_df2*log10(tmp_df);
      }
      q = q + 0.1;
    }
  }
}

q  = (double)q_org;
for(i = 0; i < N; i++)
{
  if(i != N0)
  {
    Aq[i] = Aq[i]/Iq[i];
    Fq[i] = log10(Iq[i]) - Fq[i]/Iq[i];
  }
  else /* q = 0 */
  {
    Aq[i] = Aq_tmp/(double)box_cnt;
    Fq[i] = log10((double)box_cnt);
    }
    fprintf(out1, "%g %g\n", (float)Aq[i], (float)Fq[i]);
    q = q + 0.1;
  }
  printf("finish Fq calculation\n");


  fclose(in);
  fclose(out1);
}
```

```
/***************************************************************************/
/* Name        : OBL_1.C */
/* Input       : DWT coefficients ( .cof/,dXX) */
/* Output      : ASCII (float) */
/* Procedure   : Optimal Bit Allocation, copy from FBI spec.. Step 1 -- get var and standard devi-
                 ation */
/* Data        : 20/Feb/97 */
/* Version     : 1.0 */
/* Designer    : Eric Jang Tel:2476992, Email: jang@ee.umanitoba.ca */
/***************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main()
{
  FILE *in;

  unsigned long int f_size, k;
  char infile[70];
  int org_dim;
  float tmp_f, max, min;
  double sum, mean, variance, sd;

  printf("Optimal Bit Allocation step I (FBI) ver. 1.0, 20/Feb/1997\n\n");
  printf("Enter input file name(path\\XXX.cof/dXX): \n");
  scanf("%s",infile);
  printf("Please enter input data dimension(should be a power of 2):\n");
  scanf("%i",&org_dim);
  f_size = org_dim * org_dim;
  printf("size of ""%s"" = %d", infile, f_size);

  /* Open input files. */
  if ((in = fopen(infile, "r")) == NULL)
  {
    fprintf(stderr, "Cannot open input file.\n");
    exit (1);
  }

  /* Find the max and min of the input file */
  max = -10000.0;
  min =  10000.0;
  for(k = 0; k < f_size; k++)
  {
    fread(&tmp_f, sizeof(float), 1, in);
    if(tmp_f > max)
```

```
      max = tmp_f;
    else if(tmp_f < min)
      min = tmp_f;
  }
  printf("max = %g\tmin = %g\n", max, min);

  /* Find the mean of the input file */
  rewind(in);
  sum = 0.0;
  for(k = 0; k < f_size; k++)
  {
    fread(&tmp_f, sizeof(float), 1, in);
    sum = sum + (double)tmp_f;
  }
  mean = sum/(double)f_size;
  printf("mean = %g\n", mean);

  /* Find the variation and standard deviation of the input file */
  rewind(in);
  sum = 0.0;
  for(k = 0; k < f_size; k++)
  {
    fread(&tmp_f, sizeof(float), 1, in);
    sum = sum + pow(((double)tmp_f - mean), 2.0);
  }
  variance = sum/(double)f_size;
  sd = sqrt(variance);
  printf("Variance = %g\n", variance);
  printf("Standard deviation = %g\n", sd);

  fclose(in);
}
```

```
/****************************************************************************/
/* Name        : OBA2.C */
/* Procedure   :  Optimal Bit Allocation, copy from FBI soecification.  Step 2 -- find q and Qk */
/* Data        : 20/Feb/97 */
/* Version     : 1.0 */
/* Designer    : Eric Jang Tel:2476992, Email: jang@ee.umanitoba.ca */
/****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main()
{
  FILE  *in;

  char infile[70];
  int i, N, cnt, conti, flag;
  float r, Beta, s, Var;
  double tmp_df, S, P, q, Qk, D[64][4];

  printf("Optimal Bit Allocation step II (FBI) ver. 1.0, 20/Feb/1997\n\n");
  printf("Please enter number of subband:\n");
  scanf("%i",&N);
  printf("Please enter Beta:\n");
  scanf("%g",&Beta);
  printf("Please enter r(average bit):\n");
  scanf("%g",&r);
  printf("Is L = 5 in the DWT(wp_zt) ?  (Yes -> 1, No -> others):\n");
  scanf("%d",&flag);
  printf("Enter input file name(path\\XXX.dat, var and m): \n");
  scanf("%s",infile);

   /* Open input files. */
  if ((in = fopen(infile, "r")) == NULL)
  {
    fprintf(stderr, "Cannot open input file.\n");
    exit (1);
  }

/* D[i][0] = Mk
  D[i][1] = Var
  D[i][2] = Qk
  D[i][3] = flag */

  for(i = 0; i < N; i++)
  {
    fscanf(in, "%g %g", &s, &Var);
    D[i][0] = (double)s;
    D[i][1] = (double)Var;
    D[i][3] = 1.0;

    if((flag == 1) && (i >= N-4))
      D[i][2] = 1.0;
    else
      D[i][2] = 10.0/log(D[i][1]);  /* Qk' */
  }

  conti = 1;
  printf("Finish initialization !\n");

  while( conti == 1)
  {
    S = 0.0;
    for(i = 0; i < N; i++)
    {
      if(D[i][3] != 0.0)
        S = S + 1.0/D[i][0];
    }

    P = 1.0;
    tmp_df = (double)Beta * pow(2.0, (((double)r/S) - 1.0));
    for(i = 0; i < N; i++)
    {
      if(D[i][3] != 0.0)
        P = P * pow(sqrt(D[i][1])/D[i][2], 1.0/D[i][0]);
    }
    q = tmp_df * (1.0/pow(P, 1.0/S));

    cnt = 0.0;
    for(i = 0; i < N; i++)
    {
      if((D[i][2]/q) >= (2.0 * (1.0/Beta) * sqrt(D[i][1])))
      {
        D[i][3] = 1.0;
        cnt = cnt + 1;
      }
      else
        D[i][3] = 0.0;
    }
    printf("cnt = %d\tq = %g\n", cnt, q);
    if(cnt == 0)
      conti = 0;
  }

  for(i = 0; i < N; i++)
  {
    Qk = D[i][2]/q;
    printf("Q[%d] = %g\n", i, Qk);
  }
  printf("Finish Optimal Bit Allocation II\n");

  fclose(in);
}
```

```
/*****************************************************************************/
/* Name        : ENC_DEC.C */
/* Procedure   :  Encoder/Decoder of OBL (FBI spec.) */
/* Data        : 20/Feb/97, 21, 22 */
/* Version     : 1.0, 1.1, 2.0 */
/* Designer    : Eric Jang Tel:2476992, Email: jang@ee.umanitoba.ca */
/*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main()
{
  FILE *in, *out, *out1;

  unsigned char tmp_uc;
  unsigned long int f_size, k;
  char infile[70], outfile[70], outfile1[70], conti;
  short int tmp_si;
  int org_dim, max, min, P[65536], i, tmp_i, cnt;
  float tmp_f, Q, C, Z, A[65536], mx, mn;
  double tmp_df;

  printf("Encoder/Decoder(FBI) ver. 2.0, 22/Feb/1997\n\n");
  printf("Enter input file name(float, path\\XXX.dXX): \n");
  scanf("%s",infile);
  printf("Please enter input data dimension(should be a power of 2):\n");
  scanf("%i",&org_dim);
  printf("Please enter bin width:\n");
  scanf("%g",&Q);
  printf("Please enter C:\n");
  scanf("%g",&C);

  Z = Q * 1.2;  /* zero bin width */
  f_size = org_dim * org_dim;
  printf("size of %s = %d\n", infile, f_size);
  strcpy(outfile, infile);
  strcpy(outfile1, infile);
  conti = '.';


  /* Search the input file name string to find the "." then replace the extention to "enc".  */
  for (i = 0; i < 70; i++)
  {
    if (infile[i] == conti)
    {
        outfile[i+1]  = 'e';
        outfile1[i+1] = 'r';
        break;
      }
    }

    /* Open input files. */
    if ((in = fopen(infile, "r")) == NULL)
    {
      fprintf(stderr, "Cannot open input file.\n");
      exit (1);
    }

    /* Open output file. */
    if ((out = fopen(outfile, "w")) == NULL)
    {
      fprintf(stderr, "Cannot open output file.\n");
      exit (1);
    }

    if ((out1 = fopen(outfile1, "w")) == NULL)
    {
      fprintf(stderr, "Cannot open output file.\n");
      exit (1);
    }


/* Encoder */
    cnt = 0;
    for(k = 0; k < f_size; k++)
    {
      fread(&tmp_f, sizeof(float), 1, in);
      if(tmp_f > Z/2.0)
      {
        tmp_df = (double)((tmp_f - Z/2.0)/Q);
        P[k] = (int)ceil(tmp_df) + 1;
      }
      else if((tmp_f < ((-1.0)*Z)/2.0))
      {
        tmp_df = (double)((tmp_f + Z/2.0)/Q);
        P[k] = (int)floor(tmp_df) - 1;
      }
      else
      {
        P[k] = 0;
        cnt = cnt + 1;
    }

  }
  printf( "There are %d truncated\n", cnt);

  max = -10000;
  min =  10000;
  for(k = 0; k < f_size; k++)
  {
    if(P[k] > max)
      max = P[k];
    else if(P[k] < min)
      min = P[k];
  }
  printf("In encoder, max = %d\tmin = %d\n", max, min);

  for(k = 0; k < f_size; k++)
  {
    tmp_si = (short int)P[k];
    tmp_i  = P[k];
    if((abs(max) <= 127) && (abs(min) <= 127))
    {
      if(P[k] > 0)
        tmp_uc = (unsigned char)P[k];
      else if(P[k] < 0)
        tmp_uc = (unsigned char)(abs(P[k]) + 128);
      else
        tmp_uc = (unsigned char)P[k];

      fwrite(&tmp_uc, sizeof(unsigned char), 1, out);  /* unsigned char format */
    }
    else if((abs(max) < 65536) && (abs(min) < 65536))
      fwrite(&tmp_si, sizeof(short int), 1, out);  /* Short int format */
    else
      fwrite(&tmp_i, sizeof(int), 1, out);  /* int format */
  }
  printf("Finish Encoder! \n");

/* Decoder */

  for(k = 0; k < f_size; k++)
  {
    if(P[k] > 0)
      A[k] = ((float)P[k] - C)*Q + Z/2.0;
    else if (P[k] < 0)
      A[k] = ((float)P[k] + C)*Q - Z/2.0;
    else
      A[k] = 0.0;

    tmp_f = A[k];
    fwrite(&tmp_f, sizeof(float), 1, out1);  /* float format */
  }

  mx = -10000.0;
  mn =  10000.0;
  for(k = 0; k < f_size; k++)
  {
    if(A[k] > mx)
      mx = A[k];
    else if(A[k] < mn)
      mn = A[k];
  }
  printf("In decoder, max = %g\tmin = %g\n", mx, mn);
  printf("Finish Decoder, fwrite with float! \n");

  fclose(in);
  fclose(out);
  fclose(out1);
}
```

```
/***************************************************************************/
/* Name         : SYN_F.C */
/* Input        : float     Q1 Q2
                            Q3 Q4 */
/* Output       : float */
/* Procedure    : Synthesize 4 quarter data(float format) */
/* Data         : 23/Oct/96 */
/* Version      : 1.0 */
/* Designer     : Eric Jang Tel:2476992, Email: jang@ee.umanitoba.ca */
/***************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main()
{
  FILE *in1, *in2, *in3, *in4, *out;

  int h, v, i, new_dim, org_dim;
  short int tmp_si;
  char infile1[70], infile2[70], infile3[70], infile4[70], outfile[70], conti;
  float tmp_f;

  printf("Synthesis 4 quarter data ver. 1.0, 21/Oct/1996\n\n");
  printf("Please enter input data dimension(should be a power of 2):\n");
  scanf("%i",&org_dim);
  printf("Enter name of input file(path\\pat#_***.dXX): \n");
  scanf("%s",infile1);
  printf("Enter name of input file(path\\pat#_***.dXX): \n");
  scanf("%s",infile2);
  printf("Enter name of input file(path\\pat#_***.dXX): \n");
  scanf("%s",infile3);
  printf("Enter name of input file(path\\pat#_***.dXX): \n");
  scanf("%s",infile4);
  strcpy(outfile, infile1);
  conti = '.';

  /* Search the input file name string to find the "." and then replace to extention to "bin". */
  for (i = 0; i <= 70; i++)
  {
    if (infile1[i] == conti)
    {
      outfile[i+1] = 's';
      outfile[i+2] = 'y';
      outfile[i+3] = 'n';
      break;
```

```
   }
 }

 /* Open input and output file */
 if ((in1 = fopen(infile1, "r")) == NULL)
 {
   fprintf(stderr, "Cannot open input file1.\n");
   exit (1);
 }
 if ((in2 = fopen(infile2, "r")) == NULL)
 {
   fprintf(stderr, "Cannot open input file2.\n");
   exit (1);
 }
 if ((in3 = fopen(infile3, "r")) == NULL)
 {
   fprintf(stderr, "Cannot open input file3.\n");
   exit (1);
 }
 if ((in4 = fopen(infile4, "r")) == NULL)
 {
   fprintf(stderr, "Cannot open input file4.\n");
   exit (1);
 }
 if ((out = fopen(outfile, "w")) == NULL)
 {
   fprintf(stderr, "Cannot open output file.\n");
   exit (1);
 }

 /*get the original 4 quarter data */
 new_dim = org_dim * 2;

 for(v = 0; v < new_dim; v ++)
 {
   for(h = 0; h < new_dim; h++)
   {
     if((h < org_dim) && (v < org_dim))  /* 1st quarter */
     {
       fread(&tmp_f, sizeof(float), 1, in1);
       fwrite(&tmp_f, sizeof(float), 1, out);
     }
     else if((h >= org_dim) && (v < org_dim))  /* 2nd quarter */
     {
       fread(&tmp_f, sizeof(float), 1, in2);
       fwrite(&tmp_f, sizeof(float), 1, out);
```

```
      }
      else if((h < org_dim) && (v >= org_dim))  /* 2nd quarter */
      {
        fread(&tmp_f, sizeof(float), 1, in3);
        fwrite(&tmp_f, sizeof(float), 1, out);
      }
      else
      {
        fread(&tmp_f, sizeof(float), 1, in4);
        fwrite(&tmp_f, sizeof(float), 1, out);
      }
    }
  }

  fclose(in1);
  fclose(in2);
  fclose(in3);
  fclose(in4);
  fclose(out);

  printf("finish !\n");
  return 0;
}
```

```
/*******************************************************************/
/* Name        : INV_QTF.C */
/* Input       : float */
/* Output      : dim(int), reconstructed data(float) */
/* Procedure   : INVerse Wavelet Packet for QMA */
/* Date          : 13/Nov/95, 22/Oct/96, 21/Feb/97 */
/* Version     : 1.0, 1.1, 1.2 */
/* Designer    : Eric Jang  Tel:(204)474-6991  jang@ee.umanitoba.ca */
/*******************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define true 0
#define false 1
#define MaxDim 512
#define MaxDimSq 262144 /* 512 X 512 */

int dim;
float conv(int ptr1, int ptr2, double *ptr3, float *ptr4);
void u_sampling(int ptr5, int ptr6, int ptr7, int ptr8, int ptr9, float *ptr10);

main()
{
  FILE *in, *out;

  int i, temp_i, h_size, v_size, px, py, dim_tmp, dim_org, level;
  char inf[70];
  short int tmp_si;
  float buf[MaxDimSq], temp_f;
  unsigned char header[800], temp;
  unsigned long int f_size;

  printf("INV Wavelet Packet for QMA(float) ver. 1.2, 21/Feb/1997\n\n");
  printf("Enter w_packet coefficients file name(path\\XXX.cof/syn): \n");
  scanf("%s",inf);
  printf("Please enter level(J <= 5):\n");
  scanf("%i",&level);

  /* Open input .cof file *in, get w_packet coefficients data */
  if((in = fopen(inf, "r")) == NULL)
  {
    fprintf(stderr, "Cannot open input file \n");
    exit(1);
  }
```

```
printf("Please enter dimension of input file:\n");
scanf("%i", &temp_i);
dim = temp_i;
printf("h_size and v_size = %d\n", dim);
f_size = dim * dim;

/* level 5 */
/* get .cof file data and put it into buffer */
for(i = 0; i < f_size; i ++)
{
  fread(&temp_f, sizeof(float), 1, in);
  buf[i] = temp_f;
}
printf("Reconstruction level %d is ok !\n", level);
temp_i = 1;
for(i = 1; i < level; i++)
{
  temp_i = temp_i * 2;
}

dim_tmp = dim/temp_i;
px = 0;
py = 0;
for( i = 0; i < level; i++)
{
  u_sampling(px, py, dim_tmp, dim_tmp, dim_tmp, buf);
  printf("Reconstruction level %d is ok !\n", level-(i+1));
  dim_tmp = dim_tmp * 2;
}

/* Open output file *out, store inv_WT result data */
if((out = fopen("recover.dat", "w")) == NULL)
{
  fprintf(stderr, "Cannot open output file \n");
  exit(1);
}
fwrite(&dim, sizeof(int), 1, out);

/* store WT coefficients into file */
for(i = 0; i < f_size; i ++)
{
  temp_f = buf[i]  ;
  fwrite(&temp_f, sizeof(float), 1, out);
}
fclose(in);
fclose(out);
```

```
  printf("finish !\n");
}

/*-------------------------------------------------------------------------------*/
/* Array preperation and up sampling subroutine                  */
/* inputs       : starting location, dimension, data array         */
/* return       : void                          */
/* procedure    : according to the starting position and dimension to prepare the 1 dimension data
                  array, up sampling  */
/*-------------------------------------------------------------------------------*/
void u_sampling(int p_x, int p_y, int lgth, int lgthH, int lgthV, float *mat)
{
  int i, j, k, l, m, bndary;
  float data_ary[MaxDim], data_aryH[MaxDim], data_aryG[MaxDim];
  double h_flt[4], g_flt[4];

  h_flt[0] = -0.1294095225512604;  /* C3 */
  h_flt[1] =  0.2241438680420134;  /* C2 */
  h_flt[2] =  0.8365163037378079;  /* C1 */
  h_flt[3] =  0.4829629131445341;  /* C0 */

  g_flt[0] = -0.4829629131445341;  /* -C0 */
  g_flt[1] =  0.8365163037378079;  /*  C1 */
  g_flt[2] = -0.2241438680420134;  /* -C2 */
  g_flt[3] = -0.1294095225512604;  /*  C3 */

  for(i = 0; i < lgth; i++)   /* initialization */
  {
    data_aryG[i] = 0.0;
    data_aryH[i] = 0.0;
  }

  /* synthesis y position(vertical) */
  for(i = p_x; i < lgthH; i++)
  {
    k = 0;
    for(j = p_y; j < lgthV; j++)
    {
      data_ary[k] = mat[j*dim + i];
      k = k + 1;
    }
    /* divide data_ary into 2 arrays then up_sampling */
    for(k = 0; k < lgth/2; k++)
    {
      data_aryG[k*2] = data_ary[lgth/2 + k];
      data_aryH[k*2] = data_ary[k];
  }

  l = 0;
  for(m = 0; m < lgth; m++)
  {
    if(m < 3)
    {
      bndary = m;
      j = p_y;
      mat[(j+l)*dim + i] = conv(lgth, bndary, g_flt, data_aryG) +
      conv(lgth, bndary, h_flt, data_aryH);
      l = l + 1;
    }
    else
    {
      bndary = 3;
      j = p_y;
      mat[(j+l)*dim + i] = conv(m-3, bndary, g_flt, data_aryG) +
      conv(m-3, bndary, h_flt, data_aryH);
      l = l + 1;
    }
  }
}

for(i = 0; i < lgth; i++)   /* initialization */
{
  data_aryG[i] = 0.0;
  data_aryH[i] = 0.0;
}

/* synthsis x position(horizontal) */
for(i = p_y; i < lgthV; i++)
{
  k = 0;
  for(j = p_x; j < lgthH; j++)
  {
    data_ary[k] = mat[i*dim + j];
    k = k + 1;
  }
  /* devide data_ary into 2 arrays then up_sampling */
  for(k = 0; k < lgth/2; k++)
  {
    data_aryG[k*2] = data_ary[lgth/2 + k];
    data_aryH[k*2] = data_ary[k];
  }
```

```
      l = 0;
      for(m = 0; m < lgth; m++)
      {
        if(m < 3)
        {
          bndary = m;
          mat[i*dim + p_x + l] = conv(lgth, bndary, g_flt, data_aryG) +
          conv(lgth, bndary, h_flt, data_aryH);
          l = l + 1;
        }
        else
        {
          bndary = 3;
          mat[i*dim + p_x + l] = conv(m-3, bndary, g_flt, data_aryG) +
          conv(m-3, bndary, h_flt, data_aryH);
          l = l + 1;
        }
      }
    }
  }


/*------------------------------------------------------------------------------*/
/* 1 dimention convolution subroutine */
/* inputs       : filter array, data array both are 1 dimension */
/* return       : convoluted result(short int) */
/* procedure   : sum(Ai * Bi) */
/*------------------------------------------------------------------------------*/
float conv(int pos, int boundary, double *flt, float *data)
{
  int i;
  float result;
  double temp, sum;

/* pos      :starting position of the filter */
/* boundary :boundary flag */
  sum = 0.0;    /* initialization */

  switch(boundary)
  {
    case 0:
        temp = (flt[0]*(double)data[pos-3])+(flt[1]*(double)data[pos-2]);
        sum = temp+(flt[2]*(double)data[pos-1])+(flt[3]*(double)data[0]);
        break;
    case 1:
        temp = (flt[0]*(double)data[pos-2])+(flt[1]*(double)data[pos-1]);
```

```
          sum = temp+(flt[2]*(double)data[0])+(flt[3]*(double)data[1]);
          break;
      case 2:
          temp = (flt[0]*(double)data[pos-1])+(flt[1]*(double)data[0]);
          sum = temp+(flt[2]*(double)data[1])+(flt[3]*(double)data[2]);
          break;
      case 3:
          for(i = 0; i < 4; i++)
          {
            temp = flt[i] * (double)data[pos];
            pos = pos + 1;
            sum = sum + temp;
          }
          break;
    }
    result = (float)sum;
    return result;
  }
```

```
/*****************************************************************/
/* Name       : INV_WSQF.C */
/* Input      : .cof (floating) */
/* Output     : recover.dat (dim(int) + data(float) */
/* Procedure  : INVerse WSQ (floating format) */
/* Date       : 6/Nov/95, 8, 2/Mar/97 */
/* Version    : 1.0, 1.1, 2.0 */
/* Designer   : Eric Jang  Tel:(204)474-6991  jang@ee.umanitoba.ca */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define true 0
#define false 1
#define MaxDim 512
#define MaxDimSq 262144 /* 512 X 512 */

int dim;
float conv(int ptr1, int ptr2, double *ptr3, float *ptr4);
void u_sampling(int ptr5, int ptr6, int ptr7, int ptr8, int ptr9, float *ptr10);

main()
{
  FILE *in, *out;

  int i, temp_i, h_size, v_size, px, py, dim_tmp, dim_org;
  char inf[70];
  float buf[MaxDimSq], temp_f;
  unsigned char header[800], temp;
  unsigned long int f_size;

  printf("INV WSQF(FBI) ver. 2.0, 2/Mar/1997\n\n");
  printf("Enter wsq coefficients file name(path\\XXX.cof, float): \n");
  scanf("%s",inf);

   /* Open input .cof file *in, get w_packet coefficients data */
   if((in = fopen(inf, "r")) == NULL)
  {
    fprintf(stderr, "Cannot open input file \n");
    exit(1);
  }

  printf("Please enter dimension of input file:\n");
  scanf("%i", &temp_i);
  dim = temp_i;
```

```
printf("h_size and v_size = %d\n", dim);
f_size = dim * dim;

/* level 3 */
/* get .cof file data and put it into buffer */
for(i = 0; i < f_size; i ++)
{
  fread(&temp_f, sizeof(float), 1, in);
  buf[i] = temp_f;
}
printf("Reconstruction level 5 is ok !\n");

/* level 4 */
dim_tmp = dim/16;
dim_org = dim/8;
px = 0;
py = 0;
u_sampling(px, py, dim_tmp, dim_tmp, dim_tmp, buf);
printf("Reconstruction level 4 is ok !\n");

/* level 3-1 */
  dim_tmp = dim/8;
  dim_org = dim/4;
  px = 0;
  py = 0;
  u_sampling(px, py, dim_tmp, dim_tmp, dim_tmp, buf);
  px = dim_tmp;
  py = 0;
  u_sampling(px, py, dim_tmp, dim_org, dim_tmp, buf);
  px = 0;
  py = dim_tmp;
  u_sampling(px, py, dim_tmp, dim_tmp, dim_org, buf);
  px = dim_tmp;
  py = dim_tmp;
  u_sampling(px, py, dim_tmp, dim_org, dim_org, buf);
/* level 3-2 */
  dim_tmp = dim/8;
  dim_org = dim/4;
  px = dim_org;
  py = 0;
  u_sampling(px, py, dim_tmp, px+dim_tmp, dim_tmp, buf);
  px = dim_org + dim_tmp;
  py = 0;
  u_sampling(px, py, dim_tmp, px+dim_tmp, dim_tmp, buf);
  px = dim_org;
  py = dim_tmp;
  u_sampling(px, py, dim_tmp, px+dim_tmp, py+dim_tmp, buf);
  px = dim_org+dim_tmp;
  py = dim_tmp;
  u_sampling(px, py, dim_tmp, px+dim_tmp, py+dim_tmp, buf);
/* level 3-3 */
  dim_tmp = dim/8;
  dim_org = dim/4;
  px = 0;
  py = dim_org;
  u_sampling(px, py, dim_tmp, dim_tmp, py+dim_tmp, buf);
  px = dim_tmp;
  py = dim_org;
  u_sampling(px, py, dim_tmp, px+dim_tmp, py+dim_tmp, buf);
  px = 0;
  py = dim_org + dim_tmp;
  u_sampling(px, py, dim_tmp, dim_tmp, py+dim_tmp, buf);
  px = dim_tmp;
  py = dim_org + dim_tmp;
  u_sampling(px, py, dim_tmp, px+dim_tmp, py+dim_tmp, buf);
printf("Reconstruction level 3 is ok !\n");

/* level 2 */
dim_tmp = dim/4;
dim_org = dim/2;
px = 0;
py = 0;
u_sampling(px, py, dim_tmp, dim_tmp, dim_tmp, buf);
px = dim_tmp;
py = 0;
u_sampling(px, py, dim_tmp, dim_org, dim_tmp, buf);
px = 0;
py = dim_tmp;
u_sampling(px, py, dim_tmp, dim_tmp, dim_org, buf);
printf("Reconstruction level 2 is ok !\n");

/* level 1 */
dim_tmp = dim/2;
px = 0;
py = 0;
u_sampling(px, py, dim_tmp, dim_tmp, dim_tmp, buf);
px = dim_tmp;
py = 0;
u_sampling(px, py, dim_tmp, dim, dim_tmp, buf);
px = 0;
py = dim_tmp;
```

```
  u_sampling(px, py, dim_tmp, dim_tmp, dim, buf);
  px = dim_tmp;
  py = dim_tmp;
  u_sampling(px, py, dim_tmp, dim, dim, buf);
  printf("Reconstruction level 1 is ok !\n");

  /* level 0 */
  px = 0;
  py = 0;
  dim_tmp = dim;
  u_sampling(px, py, dim_tmp, dim, dim, buf);
  printf("Reconstruction level 0 is ok !\n");

  /* Open output file *out, store inv_WT result data */
  if((out = fopen("recover.dat", "w")) == NULL)
  {
    fprintf(stderr, "Cannot open output file \n");
    exit(1);
  }
  fwrite(&dim, sizeof(int), 1, out);

  /* store WT coefficients into file */
  for(i = 0; i < f_size; i ++)
  {
    temp_f = buf[i]  ;
    fwrite(&temp_f, sizeof(float), 1, out);
  }

  fclose(in);
  fclose(out);
  printf("finish !\n");
}

/*---------------------------------------------------------------------------------*/
/* Array preperation and up sampling subroutine */
/* inputs       : starting location, dimension, data array */
/* return       : void */
/* procedure    : according to the starting position and dimension to prepare the 1 dimension data
                  array, up sampling */
/*---------------------------------------------------------------------------------*/
void u_sampling(int p_x, int p_y, int lgth, int lgthH, int lgthV, float *mat)
{
  int i, j, k, l, m, bndary;
  float data_ary[MaxDim], data_aryH[MaxDim], data_aryG[MaxDim];
  double h_flt[4], g_flt[4];
```

```
h_flt[0] = -0.1294095225512604;  /* C3 */
h_flt[1] =  0.2241438680420134;  /* C2 */
h_flt[2] =  0.8365163037378079;  /* C1 */
h_flt[3] =  0.4829629131445341;  /* C0 */

g_flt[0] = -0.4829629131445341;  /* -C0 */
g_flt[1] =  0.8365163037378079;  /*  C1 */
g_flt[2] = -0.2241438680420134;  /* -C2 */
g_flt[3] = -0.1294095225512604;  /*  C3 */

for(i = 0; i < lgth; i++)   /* initialization */
{
  data_aryG[i] = 0.0;
  data_aryH[i] = 0.0;
}

/* synthesis y position(vertical) */
for(i = p_x; i < lgthH; i++)
{
  k = 0;
  for(j = p_y; j < lgthV; j++)
  {
    data_ary[k] = mat[j*dim + i];
    k = k + 1;
  }
  /* devide data_ary into 2 arrays then up_sampling */
  for(k = 0; k < lgth/2; k++)
  {
    data_aryG[k*2] = data_ary[lgth/2 + k];
    data_aryH[k*2] = data_ary[k];
  }
  l = 0;
  for(m = 0; m < lgth; m++)
  {
    if(m < 3)
    {
      bndary = m;
      j = p_y;
      mat[(j+l)*dim + i] = conv(lgth, bndary, g_flt, data_aryG) +
      conv(lgth, bndary, h_flt, data_aryH);
      l = l + 1;
    }
    else
    {
      bndary = 3;
      j = p_y;
```

```
        mat[(j+l)*dim + i] = conv(m-3, bndary, g_flt, data_aryG) +
        conv(m-3, bndary, h_flt, data_aryH);
        l = l + 1;
      }
    }
  }

  for(i = 0; i < lgth; i++)  /* initialization */
  {
    data_aryG[i] = 0.0;
    data_aryH[i] = 0.0;
  }
  /* synthsize x position(horizontal) */
  for(i = p_y; i < lgthV; i++)
  {
    k = 0;
    for(j = p_x; j < lgthH; j++)
    {
      data_ary[k] = mat[i*dim + j];
      k = k + 1;
    }
    /* devide data_ary into 2 arrays then up_sampling */
    for(k = 0; k < lgth/2; k++)
    {
      data_aryG[k*2] = data_ary[lgth/2 + k];
      data_aryH[k*2] = data_ary[k];
    }
    l = 0;
    for(m = 0; m < lgth; m++)
    {
      if(m < 3)
      {
        bndary = m;
        mat[i*dim + p_x + l] = conv(lgth, bndary, g_flt, data_aryG) +
        conv(lgth, bndary, h_flt, data_aryH);
        l = l + 1;
      }
      else
      {
        bndary = 3;
        mat[i*dim + p_x + l] = conv(m-3, bndary, g_flt, data_aryG) +
        conv(m-3, bndary, h_flt, data_aryH);
        l = l + 1;
      }
    }
  }
```

```
}

/*------------------------------------------------------------------------*/
/* 1 dimention convolution subroutine */
/* inputs        : filter array, data array both are 1 dimension */
/* return        : convoluted result(short int) */
/* procedure   : sum(Ai * Bi) */
/*------------------------------------------------------------------------*/
float conv(int pos, int boundary, double *flt, float *data)
{
  int i;
  float result;
  double temp, sum;

/* pos      :starting position of the filter */
/* boundary :boundary flag */
  sum = 0.0;    /* initialization */
  switch(boundary)
  {
    case 0:
        temp = (flt[0]*(double)data[pos-3])+(flt[1]*(double)data[pos-2]);
        sum = temp+(flt[2]*(double)data[pos-1])+(flt[3]*(double)data[0]);
        break;
    case 1:
        temp = (flt[0]*(double)data[pos-2])+(flt[1]*(double)data[pos-1]);
        sum = temp+(flt[2]*(double)data[0])+(flt[3]*(double)data[1]);
        break;
    case 2:
        temp = (flt[0]*(double)data[pos-1])+(flt[1]*(double)data[0]);
        sum = temp+(flt[2]*(double)data[1])+(flt[3]*(double)data[2]);
        break;
    case 3:
        for(i = 0; i < 4; i++)
        {
          temp = flt[i] * (double)data[pos];
          pos = pos + 1;
          sum = sum + temp;
        }
        break;
  }
  result = (float)sum;
  return result;
}
```

```
/*****************************************************************************/
/* Name        : PRO_QTF.C */
/* Input       :  .dat (float) */
/* Output      :  .pat (float) */
/* Procedure   : POSTprocessing the .dat file(gen. by inv_wpqtf) */
/* Data        : 22/Feb/97 */
/* Version     : 1.0 */
/* Designer    : Eric Jang Tel:2476992, Email: jang@ee.umanitoba.ca */
/*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MaxDim 512
#define MaxDimSq 262144 /* 512 X 512 */

main()
{
  FILE *in, *out;

  unsigned long int f_size, k;
  char infile[70], outfile[70], conti;
  int i, h_size, v_size, dim, tmp;
  float M, R, tmp_f;

  printf("Postprocessing ver. 1.0, 22/Feb/1997\n\n");
  printf("Enter input file name(float, path\\recover.dat): \n");
  scanf("%s",infile);
  strcpy(outfile, infile);
  conti = '.';

  /* Search the input file name string to find the "." then replace the extention to "pXX". */
  for (i = 0; i < 70; i++)
  {
    if (infile[i] == conti)
    {
      outfile[i+1]  = 'p';
      break;
    }
  }

  /* Open input files */
  if ((in = fopen(infile, "r")) == NULL)
  {
    fprintf(stderr, "Cannot open input file.\n");
    exit (1);
  }
  /* Open output file. */
   if ((out = fopen(outfile, "w")) == NULL)
   {
     fprintf(stderr, "Cannot open output file.\n");
     exit (1);
   }

  fread(&tmp, sizeof(int), 1, in);
  fwrite(&tmp, sizeof(int), 1, out);
  dim = tmp;
  h_size = dim;
  v_size = dim;
  if(h_size > MaxDim)
  {
    printf("Error, dimension out of range(%d)\n", MaxDim);
    exit (1);
  }
  else
    printf("f_size = v_size := %d\n", dim);

  f_size = h_size * v_size;

  /* get dat file data and reconstruct it */
  printf("Please enter mean (M):\n");
  scanf("%g", &M);
  printf("Please enter R:\n");
  scanf("%g", &R);

  for(k = 0; k < f_size; k ++)
  {
    fread(&tmp_f, sizeof(float), 1, in);
    tmp_f = (tmp_f * R) + M;
    fwrite(&tmp_f, sizeof(float), 1, out);  /* float format */
  }

  printf("finish !\n");
  fclose(in);
  fclose(out);
}
```

A.3 Source Code of Zerotrees Coding in C

```
/*******************************************************************/
/* Name       : ZEROTREE.C -- Zerotree coding of DWT coefficients */
/* Procedure  : Please refer [Shap93]*/
/* Date       : 11/Nov/95, 14, 15 */
/* Version    : 1.0, 1.1, 1.2 */
/* Designer   : Eric Jang  Tel:(204)474-6991  jang@ee.umanitoba.ca  */
/*******************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define true 0
#define false 1
#define MaxDim 512
#define MaxDimSq 262144 /* 512 X 512 */

main()
{
  FILE *in, *out, *out1;

  int dim, sig, insig, k, m, p, iteration, div, det_r, level, p_sigP[512], n_sigP[512], lpr_en;
  int i, j, top, qtz[512], assign[512], t, tmpMID_i, dif_i, tmp_i, h_size, v_size, dim_tmp;
  short int tmp_si;
  char conti, co, inf[70], outf[70], outf1[70];
  float buf[MaxDimSq], max, min, tmp_f, out_f;
  double prob, info, entropy;
  unsigned long int f_size;

  printf("Zerotree coding of Wavelet coefficients ver. 1.2, 15/Nov/1995\n\n");
  printf("Enter original image file name(path\\XXX.cof): \n");
  scanf("%s",inf);
  printf("Please enter level(J <= 5):\n");
  scanf("%i",&level);
  printf("Do you want to print out filter matrix(y -> yes, others -> no) ? \n");
  co = getc(stdin);
  co = getc(stdin);
  if (co == 'y')
    lpr_en = true;
  else
    lpr_en = false;

  outf[0] = 's';
  outf[1] = 'i';
```

```
outf[2] = '_';
outf[3] = 'z';
outf[4] = 't';
outf[5] = '0';
outf[6] = '.';
outf[7] = 'c';
outf[8] = 'o';
outf[9] = 'f';

outf1[0] = 'f';
outf1[1] = 't';
outf1[2] = '_';
outf1[3] = 'z';
outf1[4] = 't';
outf1[5] = '0';
outf1[6] = '.';
outf1[7] = 'c';
outf1[8] = 'o';
outf1[9] = 'f';

conti = '.';
iteration = 8;

/* Open input .cof file *in, get image data */
if((in = fopen(inf, "r")) == NULL)
{
  fprintf(stderr, "Cannot open input file \n");
  exit(1);
}

fread(&tmp_i, sizeof(int), 1, in);
dim = tmp_i;
h_size = dim;
v_size = dim;
printf("h_size and v_size = %d\n", dim);
f_size = h_size * v_size;

/* get .cof file data and put it into buffer */
for(i = 0; i < f_size; i ++)
{
  fread(&tmp_f, sizeof(float), 1, in);
  buf[i] = tmp_f;
}
printf("Finish loading coefficients !\n");

/* find max and min of the input file */
```

```
max = 0.0;
min = 1000.0;
for(i = 0; i < f_size; i ++)
{
  if(buf[i] > max)
    max = buf[i];
  else if(buf[i] < min)
    min = buf[i];
}
printf("The input files max = %f, min = %f\n", max, min);

/* Initialization */
max = 0.0;
min = 1000.0;
tmp_i = 1;
for(i = 0; i < level; i++)
{
  tmp_i = tmp_i * 2;
}
dim_tmp = dim/tmp_i;
printf("subband dimension = %d, i.e. 1/%d of original dimension\n", dim_tmp, tmp_i);

/* Zerotree coding */
/* find subband max and min */
for(i = 0; i < dim_tmp; i++)
{
  for(j = 0; j < dim_tmp; j++)
  {
    tmp_f = buf[i*dim + j];
    if(tmp_f > max)
      max = tmp_f;
    else if(tmp_f < min)
      min = tmp_f;
  }
}
rewind(in);
fread(&tmp_i, sizeof(int), 1, in);  /* remove the header */
printf("subbands max = %f\tmin = %f\n", max, min);

if(abs((int)max) >= abs((int)min))
  tmp_i = abs((int)max);
else
  tmp_i = abs((int)min);
top = tmp_i;
printf("subband unsigned maximum value = %d\n", top);
tmp_i = 1;
```

```
  for(i = 0; i < 13; i++)  /* <2**12 = 8192 */
  {
    tmp_i = tmp_i * 2;
    if(tmp_i > top)
    {
      top = tmp_i;
      break;
    }
  }
  printf("maximum quantization value = %d\n", top);
/******************************************************************/
  /* T0 */
  det_r = 2;
  div = 4;
  t = top/2;  /* T(threshold) */
  tmpMID_i = (top + t)/2;
  dif_i = t/4;

  for(p = 0; p < iteration; p++)
  {
    printf("\n\n******************************************************************\n");
    printf("outf  = %s\noutf1 = %s\n", outf, outf1);

    /* Open output file *out, store WT result data(short int) */
    if((out = fopen(outf, "w")) == NULL)
    {
      fprintf(stderr, "Cannot open output file \n");
      exit(1);
    }

    /* Open output file *out1, store WT result data(float) */
    if((out1 = fopen(outf1, "w")) == NULL)
    {
      fprintf(stderr, "Cannot open output file \n");
      exit(1);
    }
    tmp_i = dim;
    fwrite(&tmp_i, sizeof(int), 1, out);
    fwrite(&tmp_i, sizeof(int), 1, out1);

    printf("threshold= %d\tmidpoint= %d\tdifference = %d\n", t, tmpMID_i, dif_i);

    for(i = 0; i < div; i++)
    {
      if(i == 0)  /* T0 */
      {
```

```
    qtz[0] = t;
    assign[0] = qtz[0] + dif_i;
    if(lpr_en == true)
      printf("qtz[%d]= %d, assign[%d]= %d\n", i, qtz[i], i, assign[i]);
  }
  else
  {
    qtz[i] = qtz[i-1] + dif_i*2;
    assign[i] = assign[i-1] + dif_i*2;
    if(lpr_en == true)
      printf("qtz[%d]= %d, assign[%d]= %d\n", i, qtz[i], i, assign[i]);
  }
}

/* thresholding */
prob = 0.0;
info = 0.0;
entropy = 0.0;
sig = 0;
insig = 0;
for(i = 0; i < det_r; i++)
{
  p_sigP[i] = 0;
  n_sigP[i] = 0;
}

for(i = 0; i < dim; i++)
{
  for(j = 0; j < dim; j++)
  {
    tmp_f = buf[i*dim + j];
    k = (int)tmp_f;
    tmp_i = abs(k);
    if(tmp_i < t)
    {
      tmp_si = 0;
      fwrite(&tmp_si, sizeof(short int), 1, out);
      out_f = (float)tmp_si;
      fwrite(&out_f, sizeof(float), 1, out1);
      insig = insig + 1;
    }
    else
    {
      for(m = 0; m < det_r; m++)
      {
        if((tmp_i >= qtz[m]) && (tmp_i < qtz[m+1]))
```

```
          {
            if(k > 0)
            {
              tmp_si = assign[m];
              p_sigP[m] = p_sigP[m] + 1;
            }
            else
            {
              tmp_si = -assign[m];
              n_sigP[m] = n_sigP[m] + 1;
            }
            fwrite(&tmp_si, sizeof(short int), 1, out);
            out_f = (float)tmp_si;
            fwrite(&out_f, sizeof(float), 1, out1);
            sig = sig + 1;
          }
        }
      }
    }
  }
  printf("significient cof = %d and insignificient cof = %d\n", sig, insig);
  prob = (double)(insig)/262144.0;
  info = (-1.0) * prob * ((log10(prob))/(log10(2.0)));
  printf("prob = %g\tinf = %g\n", prob, info);
  entropy = entropy + info;

  for(i = 0; i < det_r; i++)
  {
    if(p_sigP[i] != 0)
    {
      prob = (double)(p_sigP[i])/262144.0;
      info = (-1.0) * prob * ((log10(prob))/(log10(2.0)));
      entropy = entropy + info;
    }
    if(n_sigP[i] != 0)
    {
      prob = (double)(n_sigP[i])/262144.0;
      info = (-1.0) * prob * ((log10(prob))/(log10(2.0)));
      entropy = entropy + info;
    }
    if(lpr_en == true)
      printf("p_sigP[%d] = %d, n_sigP[%d] = %d\n", i, p_sigP[i], i, n_sigP[i]);
  }
  printf("entropy = %g\n", entropy);

  rewind(in);
```

```
      fread(&tmp_i, sizeof(int), 1, in);  /* remove the header */
      det_r = det_r*2 + 2;
      div = div * 2;
      top = t;
      t = t/2;
      tmpMID_i = (top + t)/2;
      dif_i = t/4;

      /* Search the input file name string to find the "." and then replace the  */
      /* extention to "cof". */
      for (i = 0; i < 70; i++)
      {
        if (outf[i] == conti)
        {
          outf[i+5]  = outf[i+4];
          outf1[i+5] = outf1[i+4];
          outf[i+4]  = outf[i+3];
          outf1[i+4] = outf1[i+3];
          outf[i+3]  = outf[i+2];
          outf1[i+3] = outf1[i+2];
          outf[i+2]  = outf[i+1];
          outf1[i+2] = outf1[i+1];
          outf[i+1]  = conti;
          outf1[i+1] = conti;
          outf[i]  = '0';
          outf1[i] = '0';
          break;
        }
      }

      fclose(out);
      fclose(out1);
    }

    fclose(in);
    printf("finish !\n");
  }
```

```
/*****************************************************************/
/* Name        : INVWP_ZERO.C -- INVerse Wavelet Packet for Zerotree coding */
/* Procedure   : Please refer to[Shap93] */
/* Date        : 13/Nov/95 */
/* Version     : 1.0 */
/* Designer    : Eric Jang  Tel:(204)474-6991  jang@ee.umanitoba.ca */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define true 0
#define false 1
#define MaxDim 512
#define MaxDimSq 262144 /* 512 X 512 */

int dim;
float conv(int ptr1, int ptr2, double *ptr3, float *ptr4);
void u_sampling(int ptr5, int ptr6, int ptr7, int ptr8, int ptr9, float *ptr10);

main()
{
  FILE *in, *out;

  int i, temp_i, h_size, v_size, px, py, dim_tmp, dim_org, level;
  char inf[70];
  float buf[MaxDimSq], temp_f;
  unsigned char header[800], temp;
  unsigned long int f_size;

  printf("INV Wavelet Packet for zerotree coding ver. 1.0, 13/Nov/1995\n\n");
  printf("Enter w_packet coefficients file name(path\\XXX.cof): \n");
  scanf("%s",inf);
  printf("Please enter level(J <= 5):\n");
  scanf("%i",&level);

  /* Open input .cof file *in, get w_packet coefficients data */
  if((in = fopen(inf, "r")) == NULL)
  {
    fprintf(stderr, "Cannot open input file \n");
    exit(1);
  }
  fread(&temp_i, sizeof(int), 1, in);
  dim = temp_i;
  printf("h_size and v_size = %d\n", dim);
  f_size = dim * dim;
  /* level 5 */
  /* get .cof file data and put it into buffer */
  for(i = 0; i < f_size; i ++)
  {
    fread(&temp_f, sizeof(float), 1, in);
    buf[i] = temp_f;
  }
  printf("Reconstruction level %d is ok !\n", level);
  temp_i = 1;
  for(i = 1; i < level; i++)
  {
    temp_i = temp_i * 2;
  }
  dim_tmp = dim/temp_i;
  px = 0;
  py = 0;
  for( i = 0; i < level; i++)
  {
    u_sampling(px, py, dim_tmp, dim_tmp, dim_tmp, buf);
    printf("Reconstruction level %d is ok !\n", level-(i+1));
    dim_tmp = dim_tmp * 2;
  }

  /* Open output file *out, store inv_WT result data */
  if((out = fopen("recover.dat", "w")) == NULL)
  {
    fprintf(stderr, "Cannot open output file \n");
    exit(1);
  }

  fwrite(&dim, sizeof(int), 1, out);

  /* store WT coefficients into file */
  for(i = 0; i < f_size; i ++)
  {
    temp_f = buf[i]  ;
    fwrite(&temp_f, sizeof(float), 1, out);
  }

  fclose(in);
  fclose(out);
  printf("finish !\n");
}

/*------------------------------------------------------------------------*/
```

```
/* Array preperation and up sampling subroutine */
/* inputs       : starting location, dimension, data array */
/* return       : void */
/* procedure    : according to the starting position and dimension  to prepare the 1 dimension data
                  array, up sampling */
/*--------------------------------------------------------------------------------*/
void u_sampling(int p_x, int p_y, int lgth, int lgthH, int lgthV, float *mat)
{
  int i, j, k, l, m, bndary;
  float data_ary[MaxDim], data_aryH[MaxDim], data_aryG[MaxDim];
  double h_flt[4], g_flt[4];

  h_flt[0] = -0.1294095225512604;  /* C3 */
  h_flt[1] =  0.2241438680420134;  /* C2 */
  h_flt[2] =  0.8365163037378079;  /* C1 */
  h_flt[3] =  0.4829629131445341;  /* C0 */

  g_flt[0] = -0.4829629131445341;  /* -C0 */
  g_flt[1] =  0.8365163037378079;  /*  C1 */
  g_flt[2] = -0.2241438680420134;  /* -C2 */
  g_flt[3] = -0.1294095225512604;  /*  C3 */

  for(i = 0; i < lgth; i++)   /* initialization */
  {
    data_aryG[i] = 0.0;
    data_aryH[i] = 0.0;
  }

  /* synthesis y position(vertical) */
  for(i = p_x; i < lgthH; i++)
  {
    k = 0;
    for(j = p_y; j < lgthV; j++)
    {
      data_ary[k] = mat[j*dim + i];
      k = k + 1;
    }
    /* divide data_ary into 2 arrays then up_sampling */
    for(k = 0; k < lgth/2; k++)
    {
      data_aryG[k*2] = data_ary[lgth/2 + k];
      data_aryH[k*2] = data_ary[k];
    }

    l = 0;
    for(m = 0; m < lgth; m++)
    {
      if(m < 3)
      {
        bndary = m;
        j = p_y;
        mat[(j+l)*dim + i] = conv(lgth, bndary, g_flt, data_aryG) +
        conv(lgth, bndary, h_flt, data_aryH);
        l = l + 1;
      }
      else
      {
        bndary = 3;
        j = p_y;
        mat[(j+l)*dim + i] = conv(m-3, bndary, g_flt, data_aryG) +
        conv(m-3, bndary, h_flt, data_aryH);
        l = l + 1;
      }
    }
  }

  for(i = 0; i < lgth; i++)   /* initialization */
  {
    data_aryG[i] = 0.0;
    data_aryH[i] = 0.0;
  }

  /* synthsis x position(horizontal) */
  for(i = p_y; i < lgthV; i++)
  {
    k = 0;
    for(j = p_x; j < lgthH; j++)
    {
      data_ary[k] = mat[i*dim + j];
                  k = k + 1;
    }
    /* devide data_ary into 2 arrays then up_sampling */
    for(k = 0; k < lgth/2; k++)
    {
      data_aryG[k*2] = data_ary[lgth/2 + k];
      data_aryH[k*2] = data_ary[k];
    }

    l = 0;
    for(m = 0; m < lgth; m++)
    {
      if(m < 3)
```

```
        {
          bndary = m;
          mat[i*dim + p_x + l] = conv(lgth, bndary, g_flt, data_aryG) +
          conv(lgth, bndary, h_flt, data_aryH);
          l = l + 1;
        }
        else
        {
          bndary = 3;
          mat[i*dim + p_x + l] = conv(m-3, bndary, g_flt, data_aryG) +
          conv(m-3, bndary, h_flt, data_aryH);
          l = l + 1;
        }
      }
    }
  }


/*--------------------------------------------------------------------------*/
/* 1 dimention convolution subroutine */
/* inputs       : filter array, data array both are 1 dimension */
/* return       : convoluted result(short int) */
/* procedure    : sum(Ai * Bi) */
/*--------------------------------------------------------------------------*/
float conv(int pos, int boundary, double *flt, float *data)
{
  int i;
  float result;
  double temp, sum;

/* pos      :starting position of the filter */
/* boundary :boundary flag */
  sum = 0.0;    /* initialization */

  switch(boundary)
  {
    case 0:
        temp = (flt[0]*(double)data[pos-3])+(flt[1]*(double)data[pos-2]);
        sum = temp+(flt[2]*(double)data[pos-1])+(flt[3]*(double)data[0]);
        break;
    case 1:
        temp = (flt[0]*(double)data[pos-2])+(flt[1]*(double)data[pos-1]);
        sum = temp+(flt[2]*(double)data[0])+(flt[3]*(double)data[1]);
        break;
    case 2:
        temp = (flt[0]*(double)data[pos-1])+(flt[1]*(double)data[0]);
        sum = temp+(flt[2]*(double)data[1])+(flt[3]*(double)data[2]);
        break;
    case 3:
        for(i = 0; i < 4; i++)
        {
          temp = flt[i] * (double)data[pos];
          pos = pos + 1;
          sum = sum + temp;
        }
        break;
  }
  result = (float)sum;
  return result;
}
```

APPENDIX B

MATLAB M-FILES

B.1 MATLAB M-File for Forward Discrete Wavelet Transform

```
G =  [-0.12940952255126O4 -0.2241438680420134  0.8365163037378079 -0.4829629131445341];
H =  [ 0.4829629131445341  0.8365163037378079  0.2241438680420134 -0.1294095225512604];
IH = [-0.1294095225512604  0.2241438680420134  0.8365163037378079  0.4829629131445341];
IG = [-0.4829629131445341  0.8365163037378079 -0.2241438680420134 -0.1294095225512604];

Inf =    input('Please enter input file (XXX.ras) name ', 's')
fid1 =   fopen (Inf, 'r')
fid2 =   fopen ('dwt_flt.dat', 'w')
Dim =    512;                                    % dimension of the input array

A =  [1: 1: Dim];
B =  [1: 1: Dim];
C =  fread(fid1, 800, 'uchar');
fwrite(fid2, C, 'uchar');                              % write back the header
C = fread(fid1, [Dim,Dim], 'uchar');
Data = C;

% *********************** Forward DWT ***********************
% **** X direction ****
DWT_X = zeros(Dim);
for k = 1 : 1 : Dim
      tmp = Data(k,:);                                 % extract the kth row
      N = Dim;
      while N >= 4
          tmp1 = tmp(1:2);
          tmp2 = [tmp tmp1];                          % append the 1st 2 data
          tmpG = conv(tmp2, G);                                      % HPF
          tmpH = conv(tmp2, H);                                      %LPF
          bufG = tmpG(4:2:N+3);                       % down sampling of HPF
          bufH = tmpH(4:2:N+3);                       % down sampling of LPF
          j = 1;
          N1 = N./2;
          for i = N1+1:1:N
              buf(i) = bufG(j);
```

```
            j = j + 1;
        end
        tmp = bufH;
        N = N1;
    end
    buf(1:2) = tmp;
    DWT_X(k, :) = buf;
end

% **** Y direction ****
DWT_Y = zeros(Dim);
Data = WT_X';
for k = 1 : 1 : Dim
    tmp = Data(k,:);                                        % extract the kth row
    N = Dim;
    while N >= 4
        tmp1 = tmp(1:2);
        tmp2 = [tmp tmp1];                                  % append the 1st 2 data
        tmpG = conv(tmp2, G);                                              % HPF
        tmpH = conv(tmp2, H);                                              % LPF
        bufG = tmpG(4:2:N+3);                               % down sampling of HPF
        bufH = tmpH(4:2:N+3);                               % down sampling of LPF
        j = 1;
        N1 = N./2;
        for i = N1+1:1:N
            buf(i) = bufG(j);
            j = j + 1;
        end
        tmp = bufH;
        N = N1;
    end
    buf(1:2) = tmp;
    DWT_Y(k, :) = buf;
end
%*************************************************************************
surf(A, B, DWT_Y);                                % print -dgif8 dwtf01_128.gif
fwrite(fid2, DWT_Y, 'float');
fclose(fid1);
fclose(fid2);
```

B.2 MATLAB M-File for Inverse Discrete Wavelet Transform

```
G =  [-0.1294095225512604 -0.2241438680420134  0.8365163037378079 -0.4829629131445341];
H =  [ 0.4829629131445341  0.8365163037378079  0.2241438680420134 -0.1294095225512604];
IH = [-0.1294095225512604  0.2241438680420134  0.8365163037378079  0.4829629131445341];
IG = [-0.4829629131445341  0.8365163037378079 -0.2241438680420134 -0.1294095225512604];

Inf =    input('Please enter input file name (XXX.dat, default: dwt_flt.dat) :  ', 's')
Outf =   input('Please enter output file name (XXX.ras) :   ', 's')
fid1 =   fopen (Inf, 'r')
fid2 =   fopen (Outf, 'w')
Dim =    512;                                          % dimension of the input array

A =  [1: 1: Dim];
B =  [1: 1: Dim];
C =  fread(fid1, 800, 'uchar');
fwrite(fid2, C, 'uchar');                                   % write back the header
C =  fread(fid1, [Dim,Dim], 'float');
Data1 = C;

%**************************Inverse DWT **********************
% **** Y direction ****
REC_Y = zeros(Dim);
for k = 1 : 1 : Dim
    buf = Data1(k,:);                                      % extract the kth row
    tmp = buf;
    M = Dim;                                                   % dimension
    N = 2;                                                  % initialization
    while N < M
        for i = 1:1:N.*2                                    % initialization
            LF(i) = 0;
        end
        for i= 1:1:N
            LF(2.*(i-1)+1) = tmp(i);                          %up sampling
        end
        tmp1 = LF((N.*2)-2 : N.*2);                     %copy the last 3 data
        tmp2 = [tmp1 LF];
        tmpH = conv(tmp2, IH);                                  %INV LPF
        LF   = tmpH(4 : 1 : (N.*2)+3);
        for i = 1:1:N.*2                                    %initialization
            HF(i) = 0;
        end
        for i = 1 : 1: N
            HF(2.*(i-1)+1) = buf(N+i);                        %up sampling
        end
```

```
            tmp1 = HF((N.*2)-2 : N.*2);
            tmp2 = [tmp1 HF];
            tmpG = conv(tmp2, IG);                                  %INV HPF
            HF   = tmpG(4 : 1 : (N.*2)+3);
            tmp = LF + HF;
            N = N.*2;
        end
        REC_Y(k, :) = tmp;
end

% **** X direction ****
REC_X = zeros(Dim);
Data = REC_Y';
for k = 1 : 1 : Dim
        buf = Data(k,:);                                  % extract the kth row
        tmp = buf;    M = Dim;                                    % dimension
        N = 2;                                               % initialization
        while N < M
            for i = 1:1:N.*2                                 % initialization
                LF(i) = 0;
            end
            for i= 1:1:N
                LF(2.*(i-1)+1) = tmp(i);                        % up sampling
            end
            tmp1 = LF((N.*2)-2 : N.*2);                 % copy the last 3 data
            tmp2 = [tmp1 LF];
            tmpH = conv(tmp2, IH);                                  %INV LPF
            LF   = tmpH(4 : 1 : (N.*2)+3);
            for i = 1:1:N.*2                                  %initialization
                HF(i) = 0;
            end
            for i = 1 : 1: N
                HF(2.*(i-1)+1) = buf(N+i);                       %up sampling
            end
            tmp1 = HF((N.*2)-2 : N.*2);
            tmp2 = [tmp1 HF];
            tmpG = conv(tmp2, IG);                                  %INV HPF
            HF   = tmpG(4 : 1 : (N.*2)+3);
            tmp = LF + HF;
            N = N.*2;
        end
        REC_X(k, :) = tmp;
end
%**************************************************************************
% F = REC_X';
fwrite(fid2, REC_X, 'uchar')
```

```
fclose(fid1);
fclose(fid2);
```

B.3 MATLAB M-File for Zonal Filter

```
Inf = input('Please enter input file name (XXX.dat, default: dwt_flt.dat) :  ', 's')
fid1 = fopen (Inf, 'r')
fid2 = fopen ('zonal1_2.dat', 'w')
fid3 = fopen ('zonal1_4.dat', 'w')
fid4 = fopen ('zonal1_8.dat', 'w')
fid5 = fopen ('zonal1_16.dat', 'w')
Dim = 512;                                                % dimension of the input array
C = fread(fid1, 800, 'uchar');
fwrite(fid2, C, 'uchar');                                    % write back the header
fwrite(fid3, C, 'uchar');
fwrite(fid4, C, 'uchar');
fwrite(fid5, C, 'uchar');
C = fread(fid1, [Dim,Dim], 'float');

%*************************** zonal filter 1/2 ***********************************
Empty = zeros(Dim);
Length = Dim;
for i = 1 : 1 : Dim
      Empty(i, 1:Length) = C(i, 1:Length);
      Length = Length - 1;
end
fwrite(fid2, Empty, 'float');

%*************************** zonal filter 1/4 ***********************************
Empty = zeros(Dim);
N = Dim./2;
Empty(1:N, 1:N) = C(1:N, 1:N);
fwrite(fid3, Empty, 'float');

%*************************** zonal filter 1/8 ***********************************
Empty = zeros(Dim);
Length = Dim./2;
for i = 1 : 1 : Dim
      Empty(i, 1:Length) = C(i, 1:Length);
      Length = Length - 1;
end
fwrite(fid4, Empty, 'float');

%*************************** zonal filter 1/16 **********************************
Empty = zeros(Dim);
N = Dim./4;
Empty(1:N, 1:N) = C(1:N, 1:N);
```

```
fwrite(fid5, Empty, 'float');

%*******************************************************************************
fclose(fid1);
fclose(fid2);
fclose(fid3);
fclose(fid4);
fclose(fid5);
```