SPIHT image coding: Analysis, Improvements and Applications

By

Jian Zhu

A thesis submitted for the Degree of

Doctor of Philosophy

Department of Engineering

University of Warwick

November 2002

Contents

List of Figures VI

List of Tables XI

Acknowledgements XIV

Declarations XV

Summary XVI

Abbreviations XVII

Chapter 1 Introduction 1

1.1 Requirements of image coding 1

1.1.1 Image and video compression 1

1.1.2 Lossless image coding 2

1.1.3 Lossy image coding 2

1.1.4 New features of image coding 3

1.2 Image coding methods 5

1.2.1 Statistical models 5

1.2.2 Physical models 8

1.3 Research objectives 8

Reference 10

Chapter 2 The Wavelet Transform 11

2.1 Two-channel filter banks 11

2.2 Biorthogonal filter banks 13

2.3 Perfect reconstruction FIR filter banks 15

2.4 Wavelet transform 17

2.4.1 Scaling function 17

2.4.2 Wavelet 18

2.4.3 Wavelets from filter banks 18

2.4.4 Filter banks from wavelets 22

2.4.5 Summary and example 23

2.5 Multi-resolution decomposition 23

2.6 Two-dimensional wavelet transform 25

2.7 Edge extension 27

2.8 Arrangements of wavelet coefficients 30

2.9 Summary 34

Reference 36

Chapter 3 Statistical Properties of Wavelet Coefficients 37

3.1 Distribution of all wavelet coefficients 40

3.2 Distribution of wavelet coefficients on LL 43

3.3 Distribution of wavelet coefficients on HL, LH and HH 46

3.4 Intra-subband correlation 54

3.5 Inter-subband correlation at same level 58

3.6 Inter-level correlation on HL, LH and HH 58

3.7 Summary 59

Reference 70

Chapter 4 SPIHT Image Coding 71

4.1 Organisation of wavelet coefficients 72

4.2 Successive approximation quantisation 75

4.3 Ordered bit-plane coding 76

4.4 Procedure of SPIHT coding 77

4.5 A simple example 83

4.6 Arithmetic coding 91
4.6.1 Arithmetic coding models 94
4.7 Summary 96
Reference 96
Chapter 5 Improvements to the SPIHT Algorithm 97
5.1 DC-level shifting 97
5.2 Introduce the virtual trees 98
5.3 Omit the predictable symbols 107
5.4 Offset the quantisation 109
5.5 Optimise the arithmetic coding 115
5.6 A simple example 117
5.7 Summary 126
Reference 127
Chapter 6 Pre-processing for SPIHT Coding 128
6.1 Speed up the judgement of significance of trees 128
6.2 Pre-processing for SPIHT 132
6.3 Summary 136
Chapter 7 Performance of SPIHT Image Coding 137
7.1 Implementation of the SPIHT algorithm 137
7.2 Performance of the SPIHT algorithm 141
7.3 Performance gain of the improvements to the SPIHT algorithm 146
7.3.1 DC-level shifting 146
7.3.2 Virtual trees 150
7.3.3 Omit predictable coding symbols 155
7.3.4 Quantisation offset 158

7.3.5 Pre-processing for lossy image coding 161

7.3.6 Arithmetic coding 164

7.3.7 Judgement of the significance of trees 167

7.3.8 Overall rate-distortion performance gain 171

7.4 Speed of the SPIHT algorithm 181

7.5 Summary 184

Reference 185

Chapter 8 Lossless image coding using the SPIHT algorithm 186

8.1 Reversible integer-to-integer wavelet transforms 186

8.2 Performance evaluation of the integer wavelet transforms for the SPIHT algorithm 189

8.3 Content-based lossless image coding using the SPIHT algorithm 193

8.4 Summary 202

Reference 203

Chapter 9 Discussion and Conclusions 204

9.1 Summary and conclusions 204

9.2 Future work 209

Reference 211

Bibliography 212

List of Figures

Figure 1.1 Neighbours used for prediction in CALIC....6

Figure 1.2 Transform coding system....6

Figure 2.1 Two-channel filter banks....11

Figure 2.2 Three-level octave decomposition....24

Figure 2.3 Three-level octave reconstruction....24

Figure 2.4 2D separable wavelet analysis....26

Figure 2.5 2D separable wavelet synthesis....26

Figure 2.6 Subbands of 3-scale wavelet transform....27

Figure 2.7 Periodic extension....28

Figure 2.8 Symmetric extension....29

Figure 2.9 Symmetric extension without overlapping....30

Figure 2.10 Down/up-sampling and edge extension in case of odd length input....31

Figure 2.11 Down/up-sampling and edge extension in case of even length input....32

Figure 2.12 Results of 3-scale wavelet transform, 50×37 image....33

Figure 2.13 Edge extension without overlapping in case of odd length input....34

Figure 3.1 Lena (512×512, 8bpp)....38

Figure 3.2 Subbands of 5-scale wavelet transform....39

Figure 3.3 Distribution of all wavelet coefficients of Lena (linear axis)....40

Figure 3.4 Distribution of all wavelet coefficients of Lena (log y-axis)....41

Figure 3.5 Distribution of all wavelet coefficients of Lena (non-linear x-axis)....42

Figure 3.6 Distribution of wavelet coefficients on LL of Lena....44

Figure 3.7 Distribution of all wavelet coefficients on HL, LH and HH of Lena....44

Figure 3.8 Distribution of wavelet coefficients on LL of Lena after DC-level shifting 45

Figure 3.9 Distribution of wavelet coefficients on level 4 of Lena 47

Figure 3.10 Distribution of wavelet coefficients of Lena in division ±2 48

Figure 3.11 Distribution of wavelet coefficients of Lena in division ±3 48

Figure 3.12 Distribution of wavelet coefficients of Lena in division ±4 49

Figure 3.13 Distribution of wavelet coefficients of Lena in division ±5 49

Figure 3.14 Distribution of wavelet coefficients of Lena in division ±5 50

Figure 3.15 Distribution of wavelet coefficients of Lena in division ±6 50

Figure 3.16 Distribution of wavelet coefficients of Lena in division ±7 51

Figure 3.17 Distribution of wavelet coefficients of Lena in division ±8 51

Figure 3.18 Distribution of wavelet coefficients of Lena in division ±9 52

Figure 3.19 Distribution of wavelet coefficients of Lena in division ±10 52

Figure 3.20 Intra-subband correlation of wavelet coefficients of Lena on HL4 55

Figure 3.21 Intra-subband correlation of wavelet coefficients of Lena on LH4 55

Figure 3.22 Intra-subband correlation of wavelet coefficients of Lena on HH4.......56

Figure 3.23 Intra-subband correlation of wavelet coefficients of Lena on HH0.......56

Figure 3.24 Barbara..60

Figure 3.25 Boats 61

Figure 3.26 Goldhill 62

Figure 3.27 Mandrill..63

Figure 3.28 Peppers..64

Figure 3.29 Zelda 65

Figure 3.30 Distribution of wavelet coefficients on LL 67

Figure 3.31 Distribution of all wavelet coefficients on HL, LH and HH..................67

Figure 4.1 SPIHT image coding system.........71

Figure 4.2 Correspondence of wavelet coefficients on different levels.........73

Figure 4.3 Value of wavelet coefficient in binary mode.........76

Figure 4.4 Bit-planes of wavelet coefficients.........77

Figure 4.5 Last rows and columns of HL0, LH0 and HH0.........78

Figure 4.6 Example of 2-scale wavelet transform of a 20×16 image.........84

Figure 4.7 Models for the probability of symbols in arithmetic coding.........92

Figure 4.8 Probability of symbol series in arithmetic coding.........93

Figure 5.1 Wavelet coefficients of 64×64 image after 4-scale wavelet transform..102

Figure 5.2 Merging of V_n to V_{n+1} for initial trees.........105

Figure 5.3 Partitioning of trees (including virtual trees).........106

Figure 5.4 Reconstructed Lena at 0.04 bpp.........113

Figure 5.5 Reconstructed Lena at 0.8 bpp.........114

Figure 5.6 Example of 2-scale wavelet transform of a 20×16 image.........118

Figure 6.1 Example of 2-scale wavelet transform of a 20×16 image.........131

Figure 6.2 M_{max} for the 20×16 example image.........132

Figure 6.3 M_{max} after pre-processing for the 20×16 example image.........135

Figure 7.1 Performance difference of the SPIHT algorithm between Said&Pearlman's and our implementation for Goldhill.........140

Figure 7.2 Performance comparison of SPIHT, JPEG and JPEG2000 image coding for Goldhill.........142

Figure 7.3 Performance comparison of SPIHT, JPEG and JPEG2000 image coding for Zelda.........142

Figure 7.4 Performance of the SPIHT algorithm under various levels of wavelet transform for Goldhill.........144

Figure 7.5 Performance of the SPIHT algorithm under various levels of wavelet transform for Zelda 145

Figure 7.6 Performance gain of various DC-level shifting schemes for Lena 148

Figure 7.7 Performance gain of DC-level shifting in the SPIHT algorithm for Lena 150

Figure 7.8 Performance gain of using the virtual trees in the SPIHT coding for Boat 152

Figure 7.9 Performance gain of using virtual trees in SPIHT coding for Boat 153

Figure 7.10 Performance gain of omitting the predictable symbols in SPIHT coding for Barbara 157

Figure 7.11 Performance gain of the quantisation offset in the SPIHT coding for Mandrill 160

Figure 7.12 Performance gain of pre-processing in the SPIHT algorithm for Peppers 163

Figure 7.13 Performance gain of the optimisation of the arithmetic coding in the SPIHT algorithm for Goldhill 166

Figure 7.14 Running time of SPIHT encoding for Goldhill 171

Figure 7.15 Performance of the SPIHT algorithm without arithmetic coding for Zelda 172

Figure 7.16 Performance gain of the improvements to the SPIHT algorithm without arithmetic coding for Zelda 173

Figure 7.17 Reconstructed Zelda, coded by the original SPIHT at 0.1 bpp (PSNR 26.1 dB) 174

Figure 7.18 Reconstructed Zelda, coded by the improved SPIHT at 0.1 bpp (PSNR 31.9 dB) 175

Figure 7.19 Performance of the SPIHT algorithm for Zelda........176

Figure 7.20 Performance gain of arithmetic coding and the improvements to the SPIHT algorithm using the 5-scale wavelet transform for Zelda........177

Figure 7.21 Performance gain of the improvements to the SPIHT algorithm without arithmetic coding for various test images........179

Figure 7.22 Speed of SPIHT encoding for Zelda........182

Figure 7.23 Speed of the SPIHT decoding for Zelda........182

Figure 8.1 CT image of head........191

Figure 8.2 CT image of liver........192

Figure 8.3 Histogram of CT head image........194

Figure 8.4 CT image of head: bone........195

Figure 8.5 CT image of head excluding bone........196

Figure 8.6 Context to model the arithmetic coding of shape........197

Figure 8.7 Direct neighbouring pixels........197

Figure 8.8 CT image of head: edge of bone........198

Figure 8.9 CT image of head: tissues........199

Figure 8.10 CT image of head: tissues (without blank margin)........201

List of Tables

Table 2.1 Filter coefficients of biorthogonal 9/7 wavelet transform........................23

Table 3.1 Mean value of the wavelet coefficients of Lena in each division 53

Table 3.2 Auto-correlation coefficients of maximum magnitude (excluding 1) of wavelet coefficients of Lena on level 1 ~ 4 57

Table 3.3 Inter-subband correlation of wavelet coefficients of Lena on HL, LH and HH of same level...58

Table 3.4 Inter-level correlation coefficients of wavelet coefficients of Lena..........59

Table 3.5 Mean value of the wavelet coefficients in each division 68

Table 3.6 Typical correlation coefficients...69

Table 4.1 Initial LIP for the example image...85

Table 4.2 Initial LIS for the example image...86

Table 4.3 Processing of the LIS in the third round 89

Table 4.4 Output bit-stream in the first three rounds of SPIHT encoding 91

Table 5.1 Wavelet coefficients on LL in the outermost divisions 99

Table 5.2 Number of wavelet coefficients (NWC) on HL, LH and HH in the outermost divisions (OMD) 100

Table 5.3 Average mean value of WCs in each division..110

Table 5.4 Encoded length of Lena after the scan of a list at each threshold in the original SPIHT 112

Table 5.5 Initial LIP for the example image...120

Table 5.6 Initial LIS for the example image...121

Table 5.7 Processing of the LIS in the third round 123

Table 5.8 Output bit-stream in the first three rounds of the improved SPIHT encoding........124

Table 5.9 Length of output bit-stream in the first three round of the original and the improved SPIHT encoding........125

Table 6.1 Maximum coding errors of wavelet coefficients during the scan of a list at threshold *T*........133

Table 7.1 Programs for the original SPIHT algorithm available on Internet........137

Table 7.2 Programs for the SPIHT coding........138

Table 7.3 Performance of the SPIHT algorithm without the arithmetic coding for Goldhill........139

Table 7.4 Average difference of PSNR for various test images........141

Table 7.5 Performance of SPIHT and EBCOT for Lena and Barbara........143

Table 7.6 Performance comparison for DC-level shifting (Lena)........147

Table 7.7 Performance comparison for DC-level shifting (various test images)........148

Table 7.8 Performance gain of DC-level shifting........149

Table 7.9 Performance of the SPIHT algorithm using the virtual trees for Boat........151

Table 7.10 Performance gain of using the virtual trees in the SPIHT coding for various test images........155

Table 7.11 Performance of the SPIHT algorithm omitting the predictable symbols for Barbara........156

Table 7.12 Performance gain of omitting the predictable symbols in the SPIHT coding for various test images........158

Table 7.13 Performance of the SPIHT algorithm with the quantisation offset for Mandrill........159

Table 7.14 Performance gain of the quantisation offset in the SPIHT coding for various test images........161

Table 7.15 Performance of the SPIHT algorithm with pre-processing for Peppers 162

Table 7.16 Performance gain of the pre-processing in the SPIHT algorithm for various test images........164

Table 7.17 Optimisation of arithmetic coding (Goldhill)........165

Table 7.18 Performance gain of the optimisation of arithmetic coding in the SPIHT algorithm for various test images........167

Table 7.19 Speed of the original SPIHT algorithm judging the significance of trees directly (Goldhill @ 1.0 bpp)........168

Table 7.20 Running time (seconds) of the original SPIHT encoding judging the significance of trees directly........170

Table 7.21 Running time (seconds) of the SPIHT encoding using the proposed scheme to judge the significance of trees........170

Table 7.22 Performance of the SPIHT algorithm without the arithmetic coding for Goldhill and Zelda........180

Table 7.23 Running time of the improved SPIHT encoding........181

Table 7.24 Speed of wavelet transform (Zelda)........184

Table 8.1 Compression ratio of lossless image coding using the improved SPIHT algorithm........190

Table 8.2 Comparison of the original (OSPIHT) and the improved SPIHT algorithm (ISPIHT) for lossless image coding........193

Table 8.3 Lossless coding rate (bpp) of CT head image using the SPIHT algorithm based on segmentation and margin cutting........201

Acknowledgements

My deepest appreciation and thanks go to my supervisor, Dr. Stuart S. Lawson, for his guidance, advice, help and patience over the last several years.

Many thanks to Dr. Sarah Wayte, Imaging Physicist in the Department of Clinical Physics and Bioengineering, University Hospitals Coventry and Warwickshire NHS Trust, for providing the medical image data which I worked on.

Finally I would like to thank Professor Malcolm McCrae and the Warwick Graduate School for the financial support.

Declarations

The work in this thesis has been discussed in the following papers:

S.S.Lawson and J.Zhu, 'Image compression using wavelets and JPEG-2000: A tutorial', IEE Electronics & Communication Engineering Journal, Vol.14, No.3, pp.112-121, June 2002.

J.Zhu and S.S.Lawson, 'Improvements to SPIHT for lossy image coding', The 8th IEEE International Conference on Electronics, Circuits and Systems (ICECS2001), Vol.3, pp.1363-6, 2-5 September 2001, Malta.

J.Zhu and S.S.Lawson, 'Pre-processing of SPIHT for lossy image coding', IEE Electronics Letters, Vol.37, No.11, pp.687-8, 24 May 2001.

L.Q.Xu, J.Zhu, F.Stentiford, 'Video summarization and semantics editing tools', Photonics West – International Symposia on Electronic Imaging (Electronic Storage and retrieval for media databases), Proceedings of SPIE, Vol.4315, No.25, 24-26 January 2001, San Jose, California.

J.Zhu and S.S.Lawson, 'The generic stochastic gradient adaptive algorithm', The fifth IMA (the institute of mathematics and its applications) International conference on Mathematics in Signal Processing: 18-20 December 2000, Warwick University.

J.Zhu and S.S.Lawson, 'Improvements of the SPIHT for Image Coding by Wavelet Transform', IEE Colloquium on Time-Scale and Time-Frequency Analysis and Applications, pp.24/1-5, 29 February 2000, London.

Summary

Image compression plays an important role in image storage and transmission. In the popular Internet applications and mobile communications, image coding is required to be not only efficient but also scalable. Recent wavelet techniques provide a way for efficient and scalable image coding. SPIHT (set partitioning in hierarchical trees) is such an algorithm based on wavelet transform.

This thesis analyses and improves the SPIHT algorithm. The preliminary part of the thesis investigates two-dimensional multi-resolution decomposition for image coding using the wavelet transform, which is reviewed and analysed systematically. The wavelet transform is implemented using filter banks, and the z-domain proofs are given for the key implementation steps. A scheme of wavelet transform for arbitrarily sized images is proposed.

The statistical properties of the wavelet coefficients (being the output of the wavelet transform) are explored for natural images. The energy in the transform domain is localised and highly concentrated on the low-resolution subband. The wavelet coefficients are DC-biased, and the gravity centre of most octave-segmented value sections (which are relevant to the binary bit-planes) is offset by approximately one eighth of the section range from the geometrical centre. The intra-subband correlation coefficients are the largest, followed by the inter-level correlation coefficients in the middle then the trivial inter-subband correlation coefficients on the same resolution level. The statistical properties reveal the success of the SPIHT algorithm, and lead to further improvements.

The subsequent parts of the thesis examine the SPIHT algorithm. The concepts of successive approximation quantisation and ordered bit-plane coding are highlighted. The procedure of SPIHT image coding is demonstrated with a simple example. A solution for arbitrarily sized images is proposed.

Seven measures are proposed to improve the SPIHT algorithm. Three DC-level shifting schemes are discussed, and the one subtracting the geometrical centre in the image domain is selected in the thesis. The virtual trees are introduced to hold more wavelet coefficients in each of the initial sets. A scheme is proposed to reduce the redundancy in the coding bit-stream by omitting the predictable symbols. The quantisation of wavelet coefficients is offset by one eighth from the geometrical centre. A pre-processing technique is proposed to speed up the judgement of the significance of trees, and a smoothing is imposed on the magnitude of the wavelet coefficients during the pre-processing for lossy image coding. The optimisation of arithmetic coding is also discussed.

Experimental results show that these improvements to SPIHT get a significant performance gain. The running time is reduced by up to a half. The PSNR (peak signal to noise ratio) is improved a lot at very low bit rates, up to 12 dB in the extreme case. Moderate improvements are also made at high bit rates.

The SPIHT algorithm is applied to lossless image coding. Various wavelet transforms are evaluated for lossless SPIHT image coding. Experimental results show that the interpolating transform (4, 4) and the S+P transform (2+2, 2) are the best for natural images among the transforms used, the interpolating transform (4, 2) is the best for CT images, and the bi-orthogonal transform (9, 7) is always the worst.

Content-based lossless coding of a CT head image is presented in the thesis, using segmentation and SPIHT. Although the performance gain is limited in the experiments, it shows the potential advantage of content-based image coding.

Abbreviations

The following abbreviations are used within this thesis:

1D	One-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional
3G	Third Generation
Bio97	Bi-orthogonal transform (9, 7)
bpp	Bits Per Pixel
bps	Bits Per Second
Kbps	Kilo Bits Per Second
Mbps	Million Bits Per Second
CALIC	Context-based Adaptive Lossless Image Coding
CR	Compression Ratio
CT	Computer Tomography
dB	Decibel
DC	Direct Current
DCT	Discrete Cosine Transform
DVD	Digital Video/Versatile Disc
DWA	Discrete Wavelet Analysis
DWS	Discrete Wavelet Synthesis
EBCOT	Embedded Block Coding with Optimised Truncation
EOB	End Of Bit-stream
EZW	Embedded Zero-tree Wavelet image coding

FIR	Finite Impulse Response
HDTV	High Definition Television
IEE	The Institution of Electrical Engineers
IEEE	The Institute of Electrical and Electronics Engineers
IPT22	Interpolating Transform (2, 2)
IPT24	Interpolating Transform (2, 4)
IPT42	Interpolating Transform (4, 2)
IPT44	Interpolating Transform (4, 4)
IPT62	Interpolating Transform (2, 2)
JPEG	Joint Photography Experts Group
LIP	List of Insignificant Pixels (wavelet coefficients)
LMS	Least Mean Square
LIS	List of Insignificant Sets
LSB	Least Significant Bit
LSP	List of Significant Pixels (wavelet coefficients)
MPEG	Moving Picture Experts Group
MSB	Most Significant Bit
MSE	Mean Square Error
PR	Perfect Reconstruction
PSNR	Peak Signal to Noise Ratio
SP222	S+P transform (2+2,2)
SPIE	The Society of Photo-Optical Instrumentation Engineers
SPIHT	Set Partitioning In Hierarchical Trees
UMTS	Universal Mobile Telecommunication System
VCD	Video Compact Disc

WC	Wavelet coefficient
WT	Wavelet transform

Chapter 1

Introduction

1.1 Requirements of image coding

1.1.1 Image and video compression

Image and video coding play an important role in many existing and emerging applications, such as digital camera, HDTV (high definition television), video storage (VCD, DVD, etc.), video conference, multimedia (games, etc.), internet image and video browsing, digital library, and so on.

The size of the digitised image and video are very large. For example, a colour image of 512×512 pixels with 24 bits per pixel, is about 786 kilobytes in size. A 90-minute film of 25 frames per second, 352×288 pixels per frame, and 24 bits per pixel for colour, is about 41 gigabytes in size. That does not include the voice. As can be seen, to store images and videos, huge storage space is needed.

It is often needed to transmit images and videos in applications. To browse the above image at home, through the Internet, using a modem with the communication speed of 28 kilobits per second, one need to wait for more than 4 minutes after the request is issued. To see the above film on line, the communication bandwidth needs to be about 60 megabits per second.

In reality, the storage capacity and communication bandwidth of the images and videos stated above is not affordable. Thanks to the coding techniques, especially compression in coding, we do not have to suffer all these. In other words, images and videos have to be compressed for storage and communication.

The discussion here is on image coding. Image is the basic element of a video. Although video coding is different from image coding, some image coding techniques are used in video coding. For example, intra-frame coding for video is just image coding, and the object-based video-coding standard MPEG-4 adopts image-coding technique directly for static texture coding [1].

1.1.2 Lossless image coding

An image can be compressed and reconstructed without loss. The reconstructed image after decoding is exactly the same as the original image before encoding.

High compression ratio (CR) is the main concern of lossless image coding. CR is the ratio of the coding rate of the original image (in bit/pixel or bpp) over the average coding rate of the encoded image:

$$\text{Compression Ratio} = \frac{\text{Coding rate of original image}}{\text{Average coding rate of encoded image}} \qquad \textbf{(1.1)}$$

As can be seen later, compression is very limited in lossless image coding.

Lossless image coding is often used in some special cases, such as medicine and astronautics. In some countries, the law imposes lossless coding for medical image compression. In astronautics, the photos from space are rare, so no one would like to lose any detail.

1.1.3 Lossy image coding

Normally an image is to be viewed by humans, no matter it is for storage or for communication. The image definition that human eyes can distinguish is limited. Human sight is not so sensitive to some extent with the minute details of an image.

In most cases, lossless image coding is not necessary, lossy image coding is acceptable if enough detail is retained.

For lossy image coding, the reconstructed image quality changes with the coding rate. The rate-distortion curve is the key to evaluate the performance of lossy image coding. The distortion is measured by the peak signal to noise ratio (PSNR). For 8-bits grey image, the maximum pixel value is 255, and PSNR is defined as follows:

$$PSNR = 10\log_{10}(\frac{255^2}{MSE})\ dB \tag{1.2}$$

Where MSE is the mean square error between the original and the reconstructed images. For an image whose size is $M \times N$, denoting $a_{i,j}$ the pixels of the original image and $b_{i,j}$ the pixels of the reconstructed image ($0 \leq i < M$ and $0 < j < M$), MSE is calculated as follows:

$$MSE = [\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(a_{i,j} - b_{i,j})^2]/(M \times N) \tag{1.3}$$

For both lossless and lossy image coding, easy implementation is very important. Computation complexity, speed of encoding and decoding, and memory usage are some of the concerned aspects.

1.1.4 New features of image coding

Internet and mobile phones are becoming more and more popular, and have a profound impact on our daily life. Recently, with the development and convergence of mobile communications and Internet, mobile multimedia services have become feasible and are in demand. Due to the limited available bandwidth and error-borne property of mobile communication channels, image and video transmission and delivery for mobile communications (and Internet as well) require efficient, scalable and error-resilient image and video coding.

For example, in the current second-generation mobile communication system GSM (global system for mobile communications) and its evolutions (e.g. GPRS – general packet radio service, and EDGE – enhanced data rates for GSM evolution), the available channel to a single user is 9.6 Kbps (kilo bits/second) in bandwidth, and up to 384 Kbps at most. In the emerging third generation mobile communication systems, such as UMTS (universal mobile telecommunication system), the available single channel will be 384 Kbps initially, and up to 2 Mbps (million bits/second) later. But in practice, the channel may need to be shared by several users for various services. Transmission of an image over such a channel takes time. It is desirable to build up the image progressively while the transmission is being carried on, then the user can terminate the transmission when the received image is clear enough and he/she does not want to wait any longer. On the other hand, the user may be moving. Unlike the wired channels (such as optical fibre or copper cable), the error rate of the mobile channel is relatively high due to the changing multi-path fading and other interference. The user may move from the covering region/cell of one base-station to that of another. The connecting channel of the user may need to change during communication (called handover). In case of bad coverage or lack of sufficient channels, the communication may break down. It is desired to build up the image based on the transmitted data. All these cases require a scalable and error resilient image coding. Of course, efficient image coding is the key above all.

Other new features are required for image coding in various applications, such as region-of-interest coding. For example, doctors may be interested in a particular region of a medical image for diagnosis, thus this region needs to be shown very clearly while a lower quality is acceptable for other regions. Similarly, for a portrait in a background, the viewers often tend to look at the face more than other regions, thus a clearer face gives better subjective feeling on overall image quality. It is desired to raise the definition for the region of interest in image coding.

1.2 Image coding methods

There are many approaches to image coding. Most of them can be put into two fundamental categories: image coding based on statistical models and physical models. They treat image data differently, but they are often combined and integrated in practical solutions.

1.2.1 Statistical models

Most image coding methods are based on statistical models. An image is taken as a set of data. Statistical characteristics of the data set are explored and exploited for compression. Compression can be carried out in image domain or transform domain. In image domain, prediction is often used. CALIC (Context-based Adaptive Lossless Image Coding) is one of the successful candidates for predictive coding [2]. The image pixels are coded in raster scan order (from left to right in a row, then down to the next row). Current coding pixel is predicted according to a few neighbours, as indicated in Figure 1.1. The prediction is adjusted according to the gradient in the neighbourhood, to reflect the classified horizontal or vertical edge type – sharp, normal, weak, or flat. The prediction error is coded using context-based arithmetic coding. The context used for arithmetic coding is the texture information of selected neighbours with respect to the predicted value, plus the quantised energy of previous prediction error. For each context, the expectation of prediction error is fed back to improve the prediction accuracy. Full details can be found in [2].

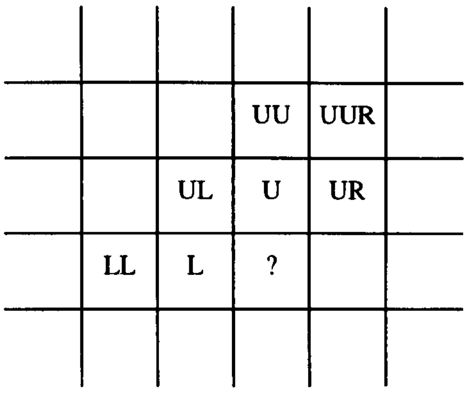


Figure 1.1 Neighbours used for prediction in CALIC

? = Pixel to be coded, U = Up, R = Right, L = Left

Here we concentrate on transform coding. Figure 1.2 shows the framework of a transform coding system. Image pixels are transformed to coefficients on subbands, then the coefficients are quantised and entropy encoded. On the decoding side, the reverse is done – the encoded bit-stream is entropy decoded and de-quantised (dequantisation here means mapping of codes to values of coefficients), and the resulting coefficients are inverse transformed to reconstruct the image.

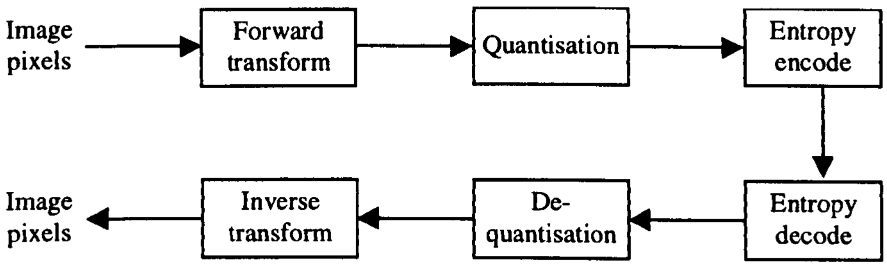


Figure 1.2 Transform coding system

Two transforms are widely used for transform coding: the discrete cosine transform (DCT) and the wavelet transform.

The DCT is orthogonal, and is used to de-correlate the image data so as to reduce its entropy. The DCT is adopted in JPEG – the international image coding standard [3].

In JPEG, an image is divided into blocks of 8×8 pixels. Then two-dimensional (2D) DCT is applied to the image blocks. The coefficients in the transform domain are quantised and entropy coded.

Recently the wavelet transform has shown its advantages in image coding. The advantages include not only the high rate-distortion performance, but also the scalability of coding rate and PSNR. Embedded zerotree wavelet coding (EZW) [4], set partitioning in hierarchical trees (SPIHT) [5], and embedded block coding with optimised truncation (EBCOT) [6] are three well-known representatives. They have two key concepts in common, as described in the following two paragraphs.

The first is successive approximation quantisation of wavelet coefficients. The wavelet coefficients are quantised using a gradually refined threshold other than a fixed value. Usually the quantisation threshold is reduced by half in each round of the coding. The coding is in fact a binary bit-plane coding. The wavelet coefficients are coded bit-plane by bit-plane, from the most significant bit (MSB) to the least significant bit (LSB).

Second, the three algorithms exploit in different ways the facts that neighbouring pixels of an image are often continuous in value and that the edges of objects in an image tend to cluster together. Neighbouring wavelet coefficients from a squared region are grouped together in a set for coding in the EZW and SPIHT algorithms. EBCOT uses context based models for arithmetic coding, where a wavelet coefficient is entropy coded according to the conditional probability under the condition of the latest known status of neighbouring wavelet coefficients.

EZW set up the foundations of the successful novel embedded wavelet image coding. SPIHT, as an improvement to EZW, is very efficient even without entropy coding – this is exclusive to SPIHT. EBCOT, the most recent one, is adopted as the basic encoding engine in the new international standard JPEG 2000 [7].

1.2.2 Physical models

Video coding based on physical models has been successful, for example, motion-compensated video coding in MPEG-1 and MPEG-2 [8], and object-based video coding in MPEG-4 [1]. They are based on the natural characteristics of the objects in the video, and the physical formation of the video signal (e.g. the projection of three-dimensional objects on a two-dimensional plane, and the effect of camera movement such as pan, zoom in/out, rotation, tilt, etc.).

The similar concept can be used in image coding. The objects can be separated from the background in the image. If the shape of the object is regular, and the colour of the object is consistent, the object can be encoded separately, to get the most compression.

Due to its complexity, image coding based on physical models has not been studied very much.

1.3 Research objectives

This research is on the SPIHT algorithm. The objective was to investigate various aspects of SPIHT image coding, such as the wavelet transform, organisation of wavelet coefficients, quantisation, and entropy coding. This included exploring statistical properties of wavelet coefficients of natural images, developing novel algorithms for image compression, and applying them in image coding applications.

In chapter 2, the wavelet transform for image coding is discussed. Chapter 3 explores the statistical properties of wavelet coefficients of natural images. Chapter 4 examines the SPIHT algorithm. Chapter 5 and 6 present some novel schemes to improve SPIHT, and chapter 7 is the numerical results of the improvements for lossy image coding. In Chapter 8, the SPIHT algorithm and the new improvements are applied to lossless image coding. Performance evaluation is done for various reversible integer-to-integer wavelet transforms. Chapter 8 also gives some of the attempts on image coding based on physical models – content-based SPIHT coding for CT head image. Chapter 9 concludes the thesis.

This research started before EBCOT was published and JPEG 2000 was established. Although EBCOT outperforms SPIHT in some aspects, SPIHT still has its exclusive advantage. Besides, as shall be seen, some of the research results for SPIHT are applicable to JPEG 2000 as well.

References

[1] T.Ebrahimi and C.Horne, 'MPEG-4 Natural Video Coding - An Overview', Signal Processing: Image Communication, Vol.15, No.4-5, pp.365-85, Elsevier Science, January 2000.

[2] X.Wu and N.Memon, 'Context-based, Adaptive, Lossless Image Codec', IEEE Transactions on Communications, Vol.45, No.4, pp.437-44, April 1997.

[3] W.B.Pennebaker and J.L.Mitchell. JPEG still image data compression standard. Van Nostrand Reinhold, New York, 1993.

[4] J.M.Shapiro, 'Embedded Image Coding Using Zerotrees of Wavelet Coefficients', IEEE Transactions on Signal Processing, Vol.41, No.12, pp.3445-62, December 1993.

[5] A.Said and W.A.Pearlman, 'A new, fast, and efficient image codec based on set partitioning in hierarchical trees', IEEE Transactions on Circuits and Systems for Video Technology, Vol.6, No.3, pp.243-50, June 1996.

[6] D.Taubman, 'High performance scalable image compression with EBCOT', IEEE Transactions on Image Processing, Vol.9, No.7, pp.1158-70, July 2000.

[7] A.Skodras, C.Christopoulos and T.Ebrahimi, 'The JPEG2000 Still Image Compression Standard', IEEE Signal Processing Magazine, Vol.18, No.5, pp.36-58, September 2001.

[8] J.L.Mitchell, et al. MPEG video compression standard. Van Nostrand Reinhold, New York, 1997.

Chapter 2

The Wavelet Transform

This chapter describes some of the properties of the discrete wavelet transform (DWT) that are pertinent to image compression. The wavelet analysis and synthesis are reviewed, along with multi-resolution transform, and from one-dimensional (1D) to two-dimensional (2D) transform. The relationship of the filter banks and the wavelet transform (WT) are explored, with emphasis on the biorthogonal case. Image edge extension for WT are discussed.

2.1 Two-channel filter banks

To understand 2D wavelet analysis and synthesis for image coding, we start with two-channel filter banks, as shown in Figure 2.1. Here we take a brief review, with emphasis on perfect reconstruction (PR) which leads to biorthogonal filter banks. More details can be found in [1] – [6].

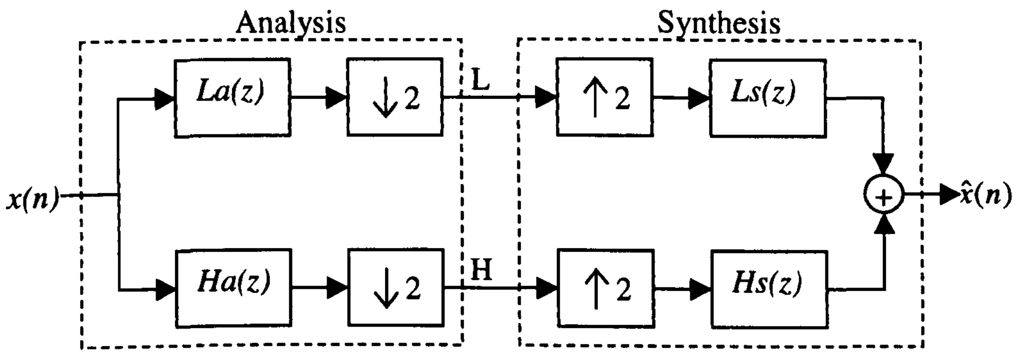


Figure 2.1 Two-channel filter banks

The analysis filter banks are inside the left dashed box of Figure 2.1, and the synthesis filter banks are inside the right box. The signal *x(n)* is filtered by lowpass and highpass analysis filters — *La(z)* and *Ha(z)*, and then down-sampled (decimated) by 2 (denoted by ↓2 in Figure 2.1). The analysis filter banks decompose *x(n)* and produces two subbands: the low (L) and high (H) subbands. The decomposed signals are

$$\begin{aligned} l(k) &= \sum_n la(2k-n)x(n) \\ h(k) &= \sum_n ha(2k-n)x(n) \end{aligned} \qquad (2.1)$$

To reconstruct *x(n)*, The L and H signals are up-sampled (stretched) by 2 (denoted by ↑2 in Figure 2.1), interpolated with zeros, and then filtered by lowpass and highpass synthesis filters — *Ls(z)* and *Hs(z)*. The results from the two channels are added up to the reconstructed signal:

$$\hat{x}(m) = \sum_k [ls(m-2k)l(k) + hs(m-2k)h(k)]$$

Insert (2.1), we get

$$\hat{x}(m) = \sum_n x(n) \sum_k [la(2k-n)ls(m-2k) + ha(2k-n)hs(m-2k)] \qquad (2.2)$$

We use the z-transform to analyse the filter banks. For a sequence *x(n)*, $X(z) = \sum_n x(n)z^{-n}$. *X(z)* becomes $[X(z^{1/2}) + X(-z^{1/2})]/2$ if down-sampled by 2, and becomes $X(z^2)$ if up-sampled by 2.

In Figure 2.1, the results of analysis are

$$\begin{aligned} L(z) &= [La(z^{1/2})X(z^{1/2}) + La(-z^{1/2})X(-z^{1/2})]/2 \\ H(z) &= [Ha(z^{1/2})X(z^{1/2}) + Ha(-z^{1/2})X(-z^{1/2})]/2 \end{aligned} \qquad (2.3)$$

The synthesis result is

$$\hat{X}(z) = Ls(z)L(z^2) + Hs(z)H(z^2)$$

Insert (2.3), we get

$$\hat{X}(z) = [La(z)Ls(z) + Ha(z)Hs(z)]X(z)/2 \\ + [La(-z)Ls(z) + Ha(-z)Hs(z)]X(-z)/2 \quad \textbf{(2.4)}$$

To get PR, for any $x(n)$, $\hat{x}(n) = x(n)$. For (2.4), this implies

$$La(z)Ls(z) + Ha(z)Hs(z) = 2 \quad \textbf{(2.5)}$$

$$La(-z)Ls(z) + Ha(-z)Hs(z) = 0 \quad \textbf{(2.6)}$$

(2.5) and (2.6) are necessary and sufficient conditions for PR. In the time domain, as can be seen from (2.2), they are equivalent to

$$\sum_k [la(2k-n)ls(m-2k) + ha(2k-n)hs(m-2k)] = \delta(m-n) \quad \textbf{(2.7)}$$

If we exchange $La(z)$ with $Ls(z)$ and $Ha(z)$ with $Hs(z)$, (2.5) – (2.6) or (2.7) still hold. In other words, $\{La(z), Ha(z)\}$ and $\{Ls(z), Hs(z)\}$ are interchangeable.

2.2 Biorthogonal filter banks

We shall see in this section that PR filter banks are biorthogonal. Stephane Mallat proved this in [1] through the Fourier transform. Here we present a detailed proof in z-domain.

From (2.5) and (2.6), $Ls(z)$ and $Hs(z)$ can be expressed by $La(z)$ and $Ha(z)$:

$$Ls(z) = -2 \cdot Ha(-z)/D(z) \quad \textbf{(2.8)}$$

$$Hs(z) = 2 \cdot La(-z)/D(z) \quad \textbf{(2.9)}$$

Where $D(z) = La(-z)Ha(z)-La(z)Ha(-z)$. It can be verified that $D(-z) = -D(z)$. From (2.8) and (2.9), we get, respectively,

$$Ha(z) = Ls(-z)D(z)/2 \quad \textbf{(2.10)}$$

$$La(z) = -Hs(-z)D(z)/2 \quad \textbf{(2.11)}$$

Insert (2.9) and (2.10) into (2.5), we get

$$La(z)Ls(z) + La(-z)Ls(-z) = 2 \quad \textbf{(2.12)}$$

Insert (2.8) and (2.11) into (2.5), we get

$$Ha(-z)Hs(-z) + Ha(z)Hs(z) = 2 \tag{2.13}$$

Rewrite (2.8) and (2.11):

$$Ha(-z) = -Ls(z)D(z)/2 \tag{2.14}$$

$$Hs(-z) = -2La(z)/D(z) \tag{2.15}$$

From (2.10) and (2.14), we have

$$Ls(z)Ha(z) + Ls(-z)Ha(-z) = 0 \tag{2.16}$$

From (2.9) and (2.15), we have

$$La(z)Hs(z) + La(-z)Hs(-z) = 0 \tag{2.17}$$

In time domain, (2.12) – (2.13) and (2.16) – (2.17) mean, respectively,

$$\sum_n ls(n)la(2k-n) = \delta(k) \tag{2.18}$$

$$\sum_n hs(n)ha(2k-n) = \delta(k) \tag{2.19}$$

$$\sum_n ls(n)ha(2k-n) = 0 \tag{2.20}$$

$$\sum_n hs(n)la(2k-n) = 0 \tag{2.21}$$

Denote $\overline{la}(n) = la(-n)$, and $\overline{ha}(n) = ha(-n)$. (2.18) – (2.21) become

$$\sum_n ls(n)\overline{la}(n-2k) = \delta(k) \tag{2.22}$$

$$\sum_n hs(n)\overline{ha}(n-2k) = \delta(k) \tag{2.23}$$

$$\sum_n ls(n)\overline{ha}(n-2k) = 0 \tag{2.24}$$

$$\sum_n hs(n)\overline{la}(n-2k) = 0 \tag{2.25}$$

The relations (2.22) – (2.25) are known as biorthogonality, and *La(z)*, *Ls(z)*, *Ha(z)* and *Hs(z)* are called biorthogonal filter banks. We see here that general PR filter banks are biorthogonal.

2.3 Perfect reconstruction FIR filter banks

As a special case of biorthogonal filter banks, PR FIR filter banks can have linear phase, which is often desired in practice. As we shall see, PR FIR filter banks are used throughout the thesis.

We explore the relations of filters for PR FIR filter banks. We follow similar steps as in [2], but not the same, and we give more details.

If we use FIR (finite impulse response) filters, *La(z)*, *Ls(z)*, *Ha(z)* and *Hs(z)* can be written as a product of a polynomial in z^{-1} with an integer power of z^{-1}. Notice that for every root of *La(-z)* (root for z^{-1}, same afterwards) there is an equal and opposite root of *La(z)*, and the roots of *Ha(-z)* are also equal and opposite to the roots of *Ha(z)*. So *La(-z)* and *Ha(-z)* have no zeros in common otherwise the PR condition of (2.5) could not always hold. It follows from (2.6) that $Ls(z) = 0$ whenever $Ha(-z) = 0$, and $Hs(z) = 0$ whenever $La(-z) = 0$. We can write *Ls(z)* in the form

$$Ls(z) = Ha(-z)p(z) \tag{2.26}$$

Where *p(z)* is also a product of a polynomial in z^{-1} with an integer power of z^{-1}. Substitute (2.26) into (2.6), we get

$$Hs(z) = -La(-z)p(z) \tag{2.27}$$

Similarly, *Ls(z)* and *Hs(z)* have no zeros in common because of (2.5). It follows from (2.6) that $La(-z) = 0$ whenever $Hs(z) = 0$, and $Ha(-z) = 0$ whenever $Ls(z) = 0$. Consequently

$$Ha(-z) = Ls(z)q(z) \tag{2.28}$$

$$La(-z) = -Hs(z)q(z) \tag{2.29}$$

Where $q(z)$ is again a product of a polynomial in z^{-1} with an integer power of z^{-1}. Substitute (2.28) into (2.26), we get

$$p(z)q(z) = 1 \tag{2.30}$$

Recall that $p(z)$ and $q(z)$ are both products of a polynomial in z^{-1} with an integer power of z^{-1}. The only possible solution for (2.30) is

$$p(z) = cz^{k} \tag{2.31}$$

$$q(z)=c^{-1}z^{-k} \tag{2.32}$$

Where c is a constant ($c\neq 0$) and k is an integer. Substitute (2.32) into (2.28), we get

$$Ha(z)=(-1)^{k}c^{-1}z^{-k}Ls(-z) \tag{2.33}$$

Substitute (2.31) into (2.27), we get

$$Hs(z) = -cz^{k}La(-z) \tag{2.34}$$

Insert (2.33) and (2.34) into (2.5), we get

$$La(z)Ls(z) - (-1)^{k}La(-z)Ls(-z) = 2 \tag{2.35}$$

For (2.35) to hold, k can only be odd, $k=2K+1$. Then (2.33) – (2.35) become

$$Ha(z)=-c^{-1}z^{-2K-1}Ls(-z) \tag{2.36}$$

$$Hs(z) = -cz^{2K+1}La(-z) \tag{2.37}$$

$$La(z)Ls(z) + La(-z)Ls(-z) = 2 \tag{2.38}$$

In time domain, (2.36) – (2.38) means

$$ha(n)=(-1)^{n}\cdot c^{-1}\cdot ls(n-2K-1) \tag{2.39}$$

$$hs(n)=(-1)^{n}\cdot c\cdot la(n+2K+1) \tag{2.40}$$

$$\sum_{n} ls(n)la(2k-n) = \delta(k) \tag{2.41}$$

As in section 2.2, denote $\overline{la}(n) = la(-n)$ and $\overline{ha}(n) = ha(-n)$. Choose $c=1$ and $K=-1$, (2.39) – (2.41) become

$$\overline{ha}(n) = (-1)^n ls(1-n) \tag{2.42}$$

$$hs(n) = (-1)^n \overline{la}(1-n) \tag{2.43}$$

$$\sum_n ls(n)\overline{la}(n-2k) = \delta(k) \tag{2.44}$$

The relation (2.42) to (2.44) are very important and well known for the PR FIR filter banks.

2.4 Wavelet transform

Now we explain the relations between the filter banks and the WT. First we build the WT from the filter banks, and then get the filter banks from the WT. For generality, the biorthogonal case is used for analysis. As we shall see later, the orthogonal WT is a special case of biorthogonal WT.

2.4.1 Scaling function

The dilation equations define the scaling functions $\varphi(t)$ and $\tilde{\varphi}(t)$:

$$\varphi(t) = \sqrt{2}\sum_n \overline{la}(n)\varphi(2t-n) \tag{2.45}$$

$$\tilde{\varphi}(t) = \sqrt{2}\sum_n ls(n)\tilde{\varphi}(2t-n) \tag{2.46}$$

$\varphi(t)$ and $\tilde{\varphi}(t)$ satisfy the biorthogonal relations:

$$\langle \varphi(t), \tilde{\varphi}(t-n) \rangle = \delta(n) \tag{2.47}$$

Where $\langle x(t), y(t) \rangle$ is the inner product of $x(t)$ and $y(t)$. $\langle x(t), y(t) \rangle = \int_t x(t)y(t)dt$.

Denote $\varphi_{j,k}(t) = 2^{j/2}\varphi(2^j t-k)$, and $\tilde{\varphi}_{j,k}(t) = 2^{j/2}\tilde{\varphi}(2^j t-k)$. For fixed j, $\varphi_{j,k}(t)$ spans $\mathbf{V}_j$, and $\tilde{\varphi}_{j,k}(t)$ spans $\tilde{\mathbf{V}}_j$. It can be seen that $\mathbf{V}_j \subset \mathbf{V}_{j+1}$, $\tilde{\mathbf{V}}_j \subset \tilde{\mathbf{V}}_{j+1}$, and

$$\varphi_{j,k}(t) = \sum_n \overline{la}(n)\varphi_{j+1,2k+n}(t) \tag{2.48}$$

$$\tilde{\varphi}_{j,k}(t) = \sum_n ls(n)\tilde{\varphi}_{j+1,2k+n}(t) \tag{2.49}$$

2.4.2 Wavelet

The wavelet equations define the wavelet $w(t)$ and $\tilde{w}(t)$:

$$w(t) = \sqrt{2}\sum_n \overline{ha}(n)\varphi(2t-n) \tag{2.50}$$

$$\tilde{w}(t) = \sqrt{2}\sum_n hs(n)\tilde{\varphi}(2t-n) \tag{2.51}$$

Denote $w_{j,k}(t) = 2^{j/2}w(2^jt\text{-}k)$, and $\tilde{w}_{j,k}(t) = 2^{j/2}\tilde{w}(2^jt\text{-}k)$. For fixed j, $w_{j,k}(t)$ spans $\mathbf{W}_\mathbf{j}$, and $\tilde{w}_{j,k}(t)$ spans $\tilde{\mathbf{W}}_\mathbf{j}$. It can be seen that $\mathbf{W}_\mathbf{j} \subset \mathbf{V}_{\mathbf{j+1}}$, $\tilde{\mathbf{W}}_\mathbf{j} \subset \tilde{\mathbf{V}}_{\mathbf{j+1}}$, and

$$w_{j,k}(t) = \sum_n \overline{ha}(n)\varphi_{j+1,2k+n}(t) \tag{2.52}$$

$$\tilde{w}_{j,k}(t) = \sum_n hs(n)\tilde{\varphi}_{j+1,2k+n}(t) \tag{2.53}$$

2.4.3 Wavelets from filter banks

Now suppose $La(z)$, $Ls(z)$, $Ha(z)$ and $Hs(z)$ are PR filter banks. (2.7) is satisfied. $\overline{La}(z)$, $Ls(z)$, $\overline{Ha}(z)$ and $Hs(z)$ are biorthogonal, as related by (2.22) to (2.25). We build the biorthogonal WT. The key is to prove the biorthogonal relations of $\varphi_{j,k}(t)$, $\tilde{\varphi}_{j,k}(t)$, $w_{j,k}(t)$ and $\tilde{w}_{j,k}(t)$:

$$\left\langle \varphi_{j,m}(t), \tilde{\varphi}_{j,n}(t) \right\rangle = \delta(m-n) \tag{2.54}$$

$$\left\langle w_{j,m}(t), \tilde{w}_{j,n}(t) \right\rangle = \delta(m-n) \tag{2.55}$$

$$\left\langle \varphi_{j,m}(t), \tilde{w}_{j,n}(t) \right\rangle = 0 \tag{2.56}$$

$$\left\langle \tilde{\varphi}_{j,m}(t), w_{j,n}(t) \right\rangle = 0 \tag{2.57}$$

Although a proof can be found in literature, we present our own proof here.

By definition, we have

$$\begin{aligned}\left\langle \varphi_{j,m}(t), \tilde{\varphi}_{j,n}(t) \right\rangle &= \int_t \varphi_{j,m}(t)\tilde{\varphi}_{j,n}(t)dt \\ &= \int_t 2^{j/2}\varphi(2^j t - m)\cdot 2^{j/2}\tilde{\varphi}(2^j t - n)dt\end{aligned}$$

Let $\tau = 2^j t - m$,

$$\left\langle \varphi_{j,m}(t), \tilde{\varphi}_{j,n}(t) \right\rangle = \int_\tau \varphi(\tau)\tilde{\varphi}(\tau + m - n)d\tau$$

Applying (2.47), we obtain (2.54).

Similarly, by definition and applying (2.52) and (2.53), we have

$$\begin{aligned}\left\langle w_{j,m}(t), \tilde{w}_{j,n}(t) \right\rangle &= \int_t w_{j,m}(t)\tilde{w}_{j,n}(t)dt \\ &= \int_t \sum_l \overline{ha}(l)\varphi_{j+1,2m+l}(t)\cdot \sum_k hs(k)\tilde{\varphi}_{j+1,2n+k}(t)dt \\ &= \sum_k \sum_l \overline{ha}(l)hs(k)\int_t \varphi_{j+1,2m+l}(t)\tilde{\varphi}_{j+1,2n+k}(t)dt\end{aligned}$$

Applying (2.54):

$$\begin{aligned}\left\langle w_{j,m}(t), \tilde{w}_{j,n}(t) \right\rangle &= \sum_k \sum_l \overline{ha}(l)hs(k)\delta(2m + l - 2n - k) \\ &= \sum_k hs(k)\overline{ha}(k - 2m + 2n)\end{aligned}$$

Applying (2.23), we obtain (2.55).

By definition and applying (2.48) and (2.53), we have

$$\begin{aligned}\left\langle \varphi_{j,m}(t), \tilde{w}_{j,n}(t) \right\rangle &= \int_t \varphi_{j,m}(t)\tilde{w}_{j,n}(t)dt \\ &= \int_t \sum_l \overline{la}(l)\varphi_{j+1,2m+l}(t)\cdot \sum_k hs(k)\tilde{\varphi}_{j+1,2n+k}(t)dt \\ &= \sum_k \sum_l hs(k)\overline{la}(l)\int_t \varphi_{j+1,2m+l}(t)\tilde{\varphi}_{j+1,2n+k}(t)dt\end{aligned}$$

Applying (2.54):

$$\begin{aligned}\left\langle \varphi_{j,m}(t), \tilde{w}_{j,n}(t) \right\rangle &= \sum_k \sum_l hs(k)\overline{la}(l)\delta(2m + l - 2n - k) \\ &= \sum_k hs(k)\overline{la}(k - 2m + 2n)\end{aligned}$$

Applying (2.25), we obtain (2.56).

By definition and applying (2.49) and (2.52), we have

$$\begin{aligned}\left\langle \tilde{\varphi}_{j,m}(t), w_{j,n}(t)\right\rangle &= \int_t \tilde{\varphi}_{j,m}(t) w_{j,n}(t)dt \\ &= \int_t \sum_k ls(k)\tilde{\varphi}_{j+1,2m+k}(t)\cdot \sum_l \overline{ha}(l)\varphi_{j+1,2n+l}(t)dt \\ &= \sum_k \sum_l ls(k)\overline{ha}(l)\int_t \varphi_{j+1,2n+l}(t)\tilde{\varphi}_{j+1,2m+k}(t)dt\end{aligned}$$

Applying (2.54):

$$\begin{aligned}\left\langle \tilde{\varphi}_{j,m}(t), w_{j,n}(t)\right\rangle &= \sum_k \sum_l ls(k)\overline{ha}(l)\delta(2n+l-2m-k) \\ &= \sum_k ls(k)\overline{ha}(k-2n+2m)\end{aligned}$$

Applying (2.24), we obtain (2.57). The proof is completed.

(2.56) says $\tilde{\mathbf{W}}_\mathbf{j} \perp \mathbf{V}_\mathbf{j}$, and (2.57) says $\mathbf{W}_\mathbf{j} \perp \tilde{\mathbf{V}}_\mathbf{j}$. Now we check the relationship of $\tilde{\mathbf{V}}_{\mathbf{j+1}}$, $\tilde{\mathbf{V}}_\mathbf{j}$ and $\tilde{\mathbf{W}}_\mathbf{j}$. Any *f(t)* $\in$ $\tilde{\mathbf{V}}_{\mathbf{j+1}}$ can be expressed as

$$f(t) = \sum_k a_{j+1,k}\tilde{\varphi}_{j+1,k}(t) \tag{2.58}$$

It follows from (2.58) and (2.54) that

$$\begin{aligned}\left\langle f(t), \varphi_{j+1,k}(t)\right\rangle &= \int_t [\sum_n a_{j+1,n}\tilde{\varphi}_{j+1,n}(t)]\varphi_{j+1,k}(t)dt \\ &= \sum_n a_{j+1,n}\int_t \varphi_{j+1,k}(t)\tilde{\varphi}_{j+1,n}(t)dt \\ &= \sum_n a_{j+1,n}\delta(k-n) \\ &= a_{j+1,k}\end{aligned}$$

That is

$$a_{j+1,k} = \left\langle f(t), \varphi_{j+1,k}(t)\right\rangle \tag{2.59}$$

Define

$$a_{j,k} = \left\langle f(t), \varphi_{j,k}(t)\right\rangle \tag{2.60}$$

$$b_{j,k} = \left\langle f(t), w_{j,k}(t)\right\rangle \tag{2.61}$$

Insert (2.58) and (2.48) into (2.60):

$$\begin{aligned} a_{j,k} &= \int_t [\sum_n a_{j+1,n}\tilde{\varphi}_{j+1,n}(t)]\cdot[\sum_m \overline{la}(m)\varphi_{j+1,2k+m}(t)]dt \\ &= \sum_m \sum_n a_{j+1,n}\overline{la}(m)\int_t \varphi_{j+1,2k+m}(t)\tilde{\varphi}_{j+1,n}(t)dt \end{aligned}$$

Applying (2.54):

$$\begin{aligned} a_{j,k} &= \sum_m \sum_n a_{j+1,n}\overline{la}(m)\delta(2k+m-n) \\ &= \sum_n \overline{la}(n-2k)a_{j+1,n} \end{aligned}$$

That is

$$a_{j,k} = \sum_n la(2k-n)a_{j+1,n} \tag{2.62}$$

Similarly, insert (2.58) and (2.52) into (2.61), and apply (2.54), we obtain:

$$b_{j,k} = \sum_n ha(2k-n)a_{j+1,n} \tag{2.63}$$

Now with (2.49), (2.53), (2.62) and (2.63), we have

$$\begin{aligned} &\sum_k a_{j,k}\tilde{\varphi}_{j,k}(t) + \sum_k b_{j,k}\tilde{w}_{j,k}(t) \\ &= \sum_k [\sum_n la(2k-n)a_{j+1,n}]\cdot[\sum_m ls(m)\tilde{\varphi}_{j+1,2k+m}(t)] \\ &+ \sum_k [\sum_n ha(2k-n)a_{j+1,n}]\cdot[\sum_m hs(m)\tilde{\varphi}_{j+1,2k+m}(t)] \end{aligned}$$

Let $2k+m = i$,

$$\begin{aligned} &\sum_k a_{j,k}\tilde{\varphi}_{j,k}(t) + \sum_k b_{j,k}\tilde{w}_{j,k}(t) \\ &= \sum_k [\sum_n la(2k-n)a_{j+1,n}]\cdot[\sum_i ls(i-2k)\tilde{\varphi}_{j+1,i}(t)] \\ &+ \sum_k [\sum_n ha(2k-n)a_{j+1,n}]\cdot[\sum_i hs(i-2k)\tilde{\varphi}_{j+1,i}(t)] \\ &= \sum_n \sum_i a_{j+1,n}\tilde{\varphi}_{j+1,i}\sum_k [la(2k-n)ls(i-2k) + ha(2k-n)hs(i-2k)] \end{aligned}$$

Applying (2.7),

$$\begin{aligned} &\sum_k a_{j,k}\tilde{\varphi}_{j,k}(t) + \sum_k b_{j,k}\tilde{w}_{j,k}(t) \\ &= \sum_n \sum_i a_{j+1,n}\tilde{\varphi}_{j+1,i}\delta(n-i) \\ &= \sum_n a_{j+1,n}\tilde{\varphi}_{j+1,n} \end{aligned}$$

Applying (2.58), we obtain

$$\sum_k a_{j,k}\tilde{\varphi}_{j,k}(t)+\sum_k b_{j,k}\tilde{w}_{j,k}(t)=f(t)$$

Or rewrite as

$$f(t)=\sum_k a_{j,k}\tilde{\varphi}_{j,k}(t)+\sum_k b_{j,k}\tilde{w}_{j,k}(t) \qquad \textbf{(2.64)}$$

(2.64) means that $\tilde{\mathbf{V}}_{\mathbf{j}} \oplus \tilde{\mathbf{W}}_{\mathbf{j}} \supseteq \tilde{\mathbf{V}}_{\mathbf{j+1}}$. Because $\tilde{\mathbf{V}}_{\mathbf{j}} \subset \tilde{\mathbf{V}}_{\mathbf{j+1}}$ and $\tilde{\mathbf{W}}_{\mathbf{j}} \subset \tilde{\mathbf{V}}_{\mathbf{j+1}}$, we have $\tilde{\mathbf{V}}_{\mathbf{j+1}} = \tilde{\mathbf{V}}_{\mathbf{j}} \oplus \tilde{\mathbf{W}}_{\mathbf{j}}$.

Similarly, we have $\mathbf{V}_{\mathbf{j+1}} = \mathbf{V}_{\mathbf{j}} \oplus \mathbf{W}_{\mathbf{j}}$.

According to the relations, we can decompose a function in a high level by the scaling function of the low level and the wavelet. The wavelet part is the details that represent the complement between the high resolution and the low resolution.

Repeating the relations for various j, we have

$$\tilde{\mathbf{V}}_{\mathrm{j}} = \tilde{\mathbf{W}}_{\mathrm{j-1}} \oplus \tilde{\mathbf{W}}_{\mathrm{j-2}} \oplus \ldots \oplus \tilde{\mathbf{W}}_{\mathrm{j-k+1}} \oplus \tilde{\mathbf{W}}_{\mathrm{j-k}} \oplus \tilde{\mathbf{V}}_{\mathrm{j-k}}$$

$$\mathbf{V}_{\mathrm{j}} = \mathbf{W}_{\mathrm{j-1}} \oplus \mathbf{W}_{\mathrm{j-2}} \oplus \ldots \oplus \mathbf{W}_{\mathrm{j-k+1}} \oplus \mathbf{W}_{\mathrm{j-k}} \oplus \mathbf{V}_{\mathrm{j-k}}$$

This suggests the multi-resolution decomposition, which is to be discussed in the next section.

A.Cohen, Ingrid Daubechies, and J.-C. Feauveau proved in [2] that in the FIR case, $w_{j,k}(t)$ constitute a frame in $L^2(R)$, and $w_{j,k}(t)$ and $\tilde{w}_{j,k}(t)$ constitute dual Riesz bases.

2.4.4 Filter banks from wavelets

Now inversely, we get the filter banks from the WT. We can easily get the fast WT, (2.62) and (2.63), from biorthogonal wavelets [1][5]. (2.62) means that filtered by *La(z)* and sub-sampled by 2, we get $a_{j,k}$ from $a_{j+1,k}$, while (2.63) means that filtered by *Ha(z)* and sub-sampled by 2, we get $b_{j,k}$ from $a_{j+1,k}$. That is to say, *La(z)* and *Ha(z)* constitute the two-channel analysis filter banks.

On the other hand, we can get (2.22) to (2.25) from (2.54) to (2.57) [3]. That is, $La(z)$, $Ls(z)$, $Ha(z)$ and $Hs(z)$ are biorthogonal. As can be verified, $Ls(z)$ and $Hs(z)$ constitute the two-channel synthesis filter banks with PR.

2.4.5 Summary and example

If we impose $\varphi(t) = \tilde{\varphi}(t)$ or $Ls(z) = \overline{La}(z)$, the biorthogonal WT defined by (2.45) to (2.53) becomes orthogonal WT, and the biorthogonal filter banks become orthogonal filter banks.

Now with the relations between the filter banks and the WT, we can choose filters from a wide range of wavelet families for subband decomposition.

As an example, Table 2.1 gives the filter coefficients of the biorthogonal 9/7 WT [2], which is very popular for image coding. It is symmetric.

Table 2.1 Filter coefficients of biorthogonal 9/7 wavelet transform

n	$la(n)/\sqrt{2}$	$ls(n)/\sqrt{2}$
0	0.602949018236	0.557543526229
±1	0.266864118443	0.295635881557
±2	-0.078223266529	-0.028771763114
±3	-0.016864118443	-0.045635881557
±4	0.026748757411	0

2.5 Multi-resolution decomposition

A signal can be decomposed into multi-levels or subbands using two-channel filter banks as a building block. Figure 2.2 is an example of a 3-level decomposition. The input signal is decomposed into two subbands, and the decomposition is iterated on the low subband output signal. This results in octave decomposition in the frequency domain, as shown on the scale at the bottom of Figure 2.2.

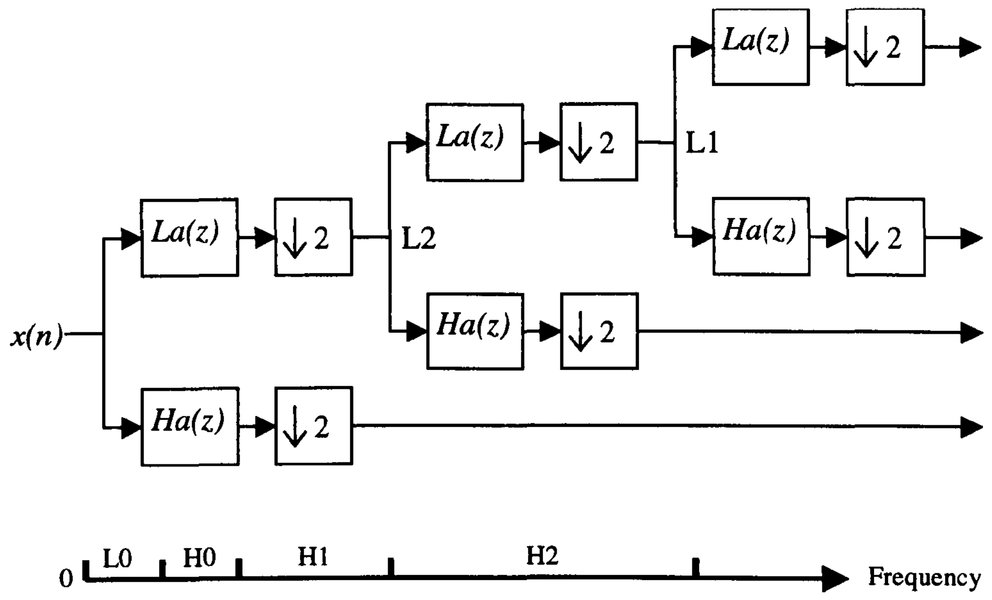


Figure 2.2 Three-level octave decomposition

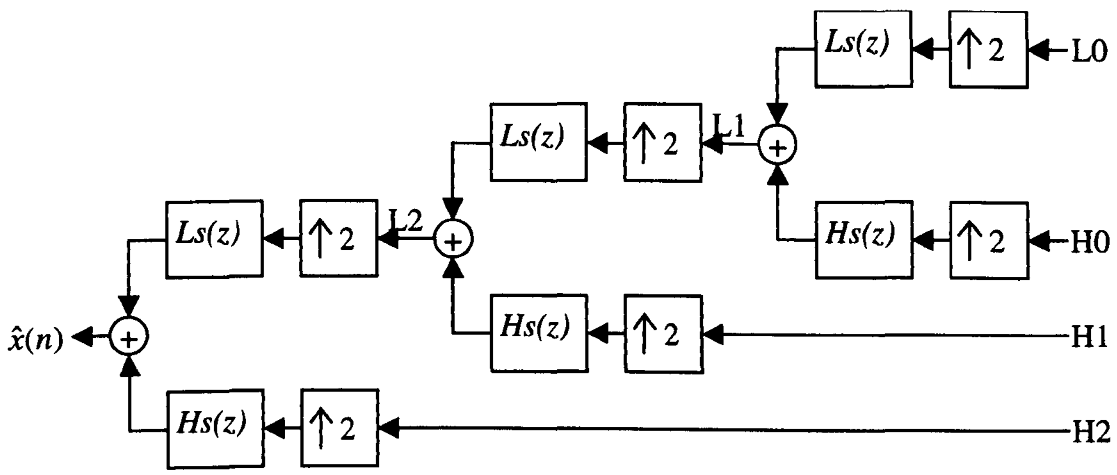


Figure 2.3 Three-level octave reconstruction

The octave decomposition is identical to multi-scale WT, as supported by the wavelet theory discussed in section 2.4. From the wavelet point of view, Figure 2.2 is three-scale WT. Take *x(n)* as $a_{3,n}$ in (2.59), which is an analysis coefficient of WT on $\tilde{\mathbf{V}}_3$, then L2, L1, and L0 are coefficients on $\tilde{\mathbf{V}}_2$, $\tilde{\mathbf{V}}_1$, and $\tilde{\mathbf{V}}_0$ respectively, and H2, H1, and H0 are coefficients on $\tilde{\mathbf{W}}_2$, $\tilde{\mathbf{W}}_1$, and $\tilde{\mathbf{W}}_0$ respectively.

Figure 2.3 does the opposite of Figure 2.2: three-level octave reconstruction. It is just the reverse of Figure 2.2, replacing the analysis filters with the synthesis filters and down-sampling with up-sampling.

The octave decomposition is widely used for multi-resolution decomposition.

2.6 Two-dimensional wavelet transform

The previous sections discussed the 1D WT. For 2D images, we need the 2D WT. We can extend the 1D WT to 2D WT simply by applying the 1D WT to the two spatial dimensions separately. In this case, the corresponding transfer function $B(z_1, z_2)$, where z_1 and z_2 relate to the two spatial dimensions, is separable, $B(z_1, z_2) = B_1(z_1)B_2(z_2)$. Non-separable $B(z_1, z_2)$ is much more complex mathematically. As a polynomial of two variables, factorisation is not possible in general. It is beyond our discussion.

Figure 2.4 shows the separable 2D wavelet analysis. First, 1D horizontal filter banks are used for each rows of the 2D image pixels, *x(m,n)*, resulting in the horizontal low (L) and high (H) subband. Then, 1D vertical filter banks are applied to each columns of L and H, resulting in four subbands: LL, LH, HL, and HH, where the first letter of the two- letter acronym denotes the horizontal L or H subband and the second denotes the vertical L or H subband. The same filter banks are used for horizontal and vertical filtering in Figure 2.4.

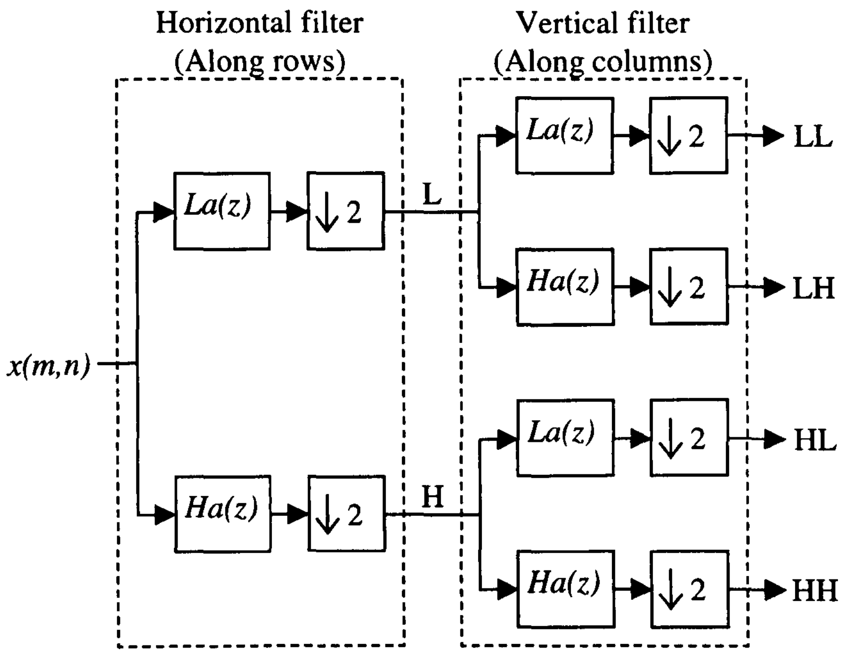


Figure 2.4 2D separable wavelet analysis

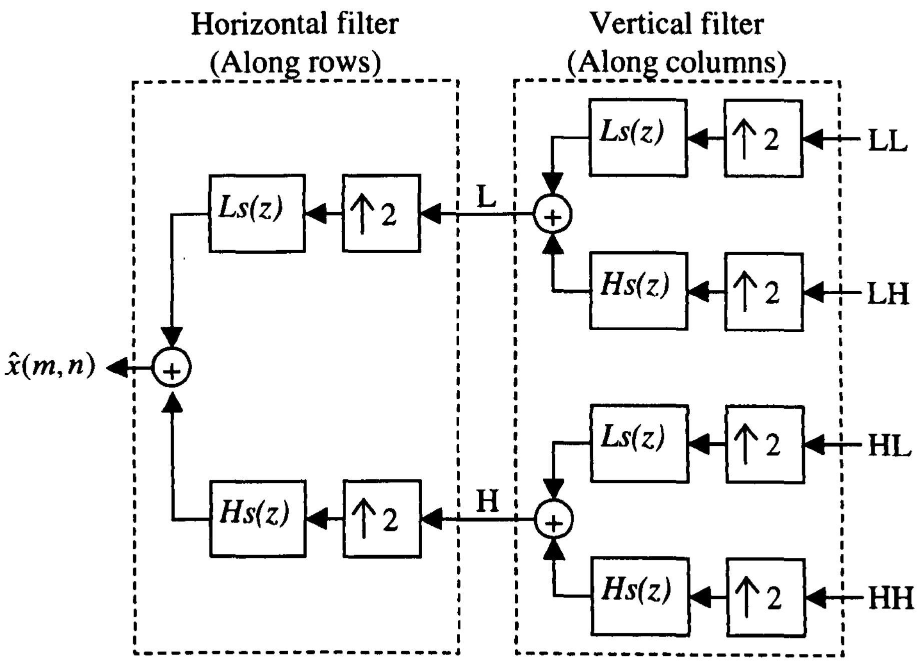


Figure 2.5 2D separable wavelet synthesis

Figure 2.5 shows the 2D separable wavelet synthesis. It is just the reverse of the 2D separable wavelet analysis, with synthesis filters for analysis filters and up-sampling for down-sampling.

For 2D multi-resolution decomposition, the 2D wavelet analysis in Figure 2.4 is repeated on LL. As the result, 3-scale 2D WT produces 10 subbands, as shown in Figure 2.6. We call the resulting data in the transform domain wavelet coefficients (WC).

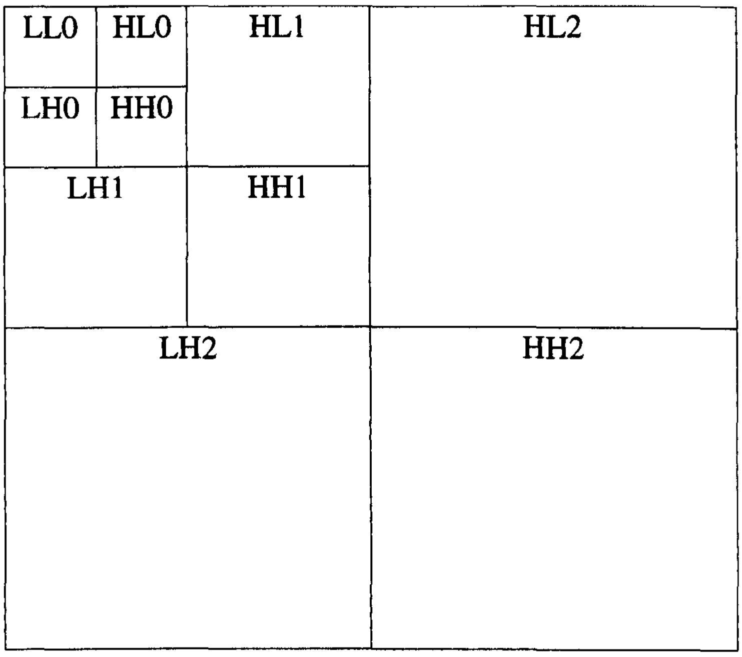


Figure 2.6 Subbands of 3-scale wavelet transform

2.7 Edge extension

When an input signal of length M is convolved with a FIR filter of length N, the valid output is of length $M + N - 1$, which is normally longer than the input. It is desired that the length of the output does not expand after the WT in image coding.

The solution is to take the finite length input as part of a periodic signal. We know that the output is periodic, and the period is the same as the input. We can keep only a period of the output without losing any information. Now we discuss the formation of the periodic signal.

The direct method is to repeat the input periodically, as shown in Figure 2.7. In practice, the signal is normally longer than the FIR filters, and we just need to repeat *M-1* samples of the input in order to calculate a whole period of the output, as indicated in Figure 2.7. So, it is in fact edge extension. We call this method periodic extension.

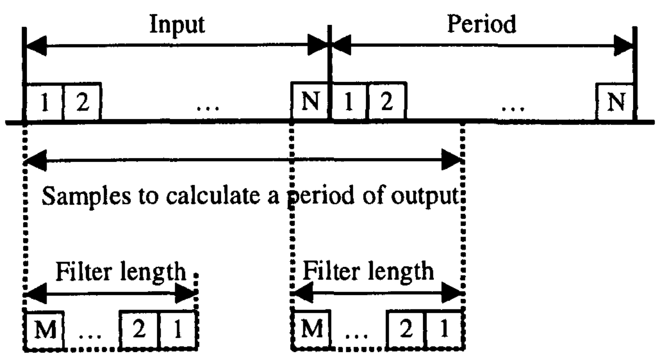


Figure 2.7 Periodic extension

Another method is symmetric periodic extension. We flip over the input and append to the original input (with one sample at the end overlapped), then repeat the new extended signal periodically (again with one sample at the end overlapped), as shown in Figure 2.8. The resulting signal is periodic, and is symmetric around the overlapped sample at both ends of the input. The output signal after convolution is also periodic, and can be symmetric or anti-symmetric if the filter used is symmetric or anti-symmetric (also known as linear phase in signal processing terminology). For linear phase PR real FIR filter banks, the filters can only be biorthogonal and of odd length (*M* = *2m* + *1*), except the trivial Haar filters [1]. In practice, we just need the m samples added at both ends of the input in order to calculate all the output samples that are not redundant. As in periodic extension, we call this method symmetric edge extension.

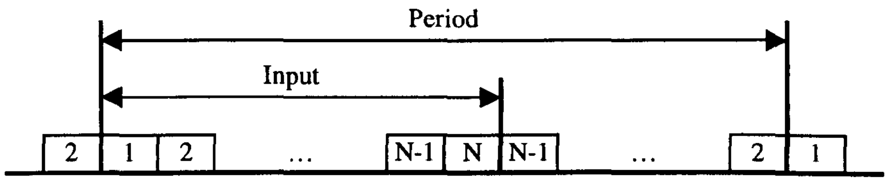


Figure 2.8 Symmetric extension

Notice that at the edge, the added samples in symmetric extension are from the neighbours, and those in periodic extension are from the other end of the input. In image coding, neighbouring image pixels tend to be similar (because they are probably from the same area of the same object), while the pixels at different ends of rows or columns are relatively far away and thus less correlated. So, at the edge in transform domain, the output signal in the symmetric extension case is probably smoother than that in the periodic extension case, while the later is likely to have jitters. For better coding performance, we use symmetric extension except otherwise stated.

Similar edge extensions were used for arbitrarily shaped visual object coding in [7], and more edge extensions were discussed there. But to our knowledge, if we use linear phase FIR filter banks, the symmetric edge extension presented here is the best for image compression, and can retain PR no matter the input is of odd or even length.

The periodic extension and the symmetric periodic extension were also discussed in [8]. But for the symmetric periodic extension presented in [8], the first and the last input samples appear in the extended signal, without overlapping of these samples for flipping and periodic repeat, as shown in Figure 2.9. If the input is of odd length, it is not possible to get PR without expanding the signal length in WT, as explained at the end of section 2.8.

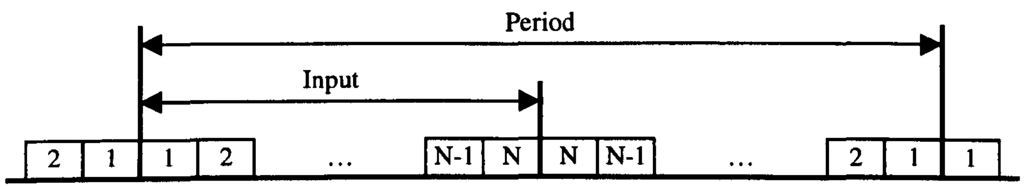


Figure 2.9 Symmetric extension without overlapping

If we use IIR filters, it is difficult to find an arrangement without expanding the signal length under the condition of PR.

2.8 Arrangements of wavelet coefficients

As discussed previously, the WT will keep the size of 2D signals. For a $M \times N$ image, the corresponding WCs are also of size $M \times N$. For a k-scale WT, it is easy to keep the signal size if M and N are integer multiples of 2^k. But for images of arbitrary size, the WT should be done carefully, especially the down-sampling and up-sampling. Here we present a scheme, as shown in Figure 2.10 and Figure 2.11. Input signal samples are numbered starting from 1. To use symmetric edge extension, the filters are symmetric or anti-symmetric, and of odd length $(2m+1)$. $2m+1$ neighbouring input samples are multiplied with the $2m+1$ filter coefficients in order, and sum up to an output sample. We align this output sample at the centre of the *2m+1* input samples. After low-pass analysis filtering using *La(z)*, we keep the odd numbered samples by down-sampling. After high-pass analysis filtering using *Ha(z)*, we keep the even numbered samples by down-sampling. The down-sampled outputs are also symmetric. The samples outside the boundary of the input are redundant. They are rejected, keeping the output the same size as the input. On the synthesis side, the H and L signals are up-sampled by inserting a zero between every sample, and then extending symmetrically at the edge to recover the symmetric periodic signal before synthesis filtering.

In Figure 2.10, the total number of input samples is odd *(2n+1)*. The high subband output signal (H) is one sample shorter than the low subband (L). The edge extension of L signal after up-sampling, for synthesis, is the same as that of the input for analysis. For H signal, an extra zero should be inserted at both ends before applying the same edge extension as that for the input.

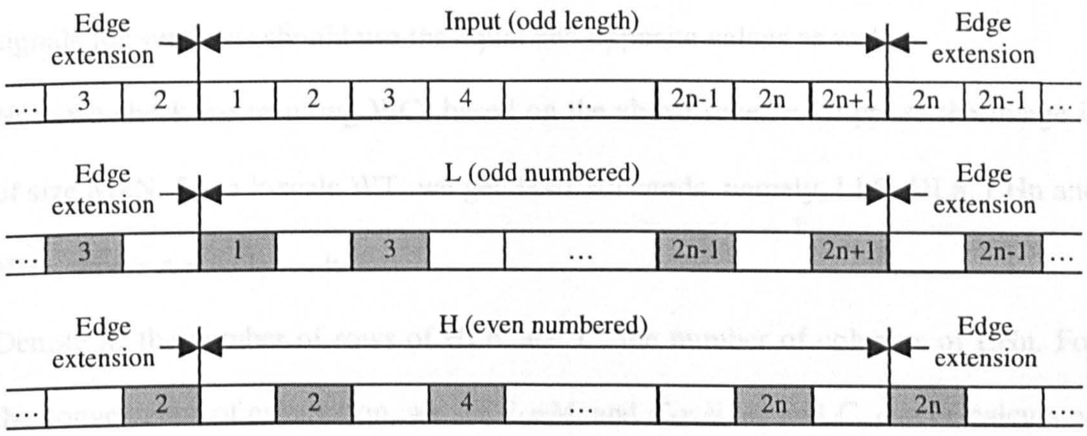


Figure 2.10 Down/up-sampling and edge extension in case of odd length input

In Figure 2.11, the input samples are of even length. The two output subband signals (H and L) are of the same length. For synthesis, after up-sampling, an extra zero is inserted at the right side of the L signal and at the left side of the H signal, then the H and L signals are extended symmetrically at the edge in the same way as that to the input for analysis.

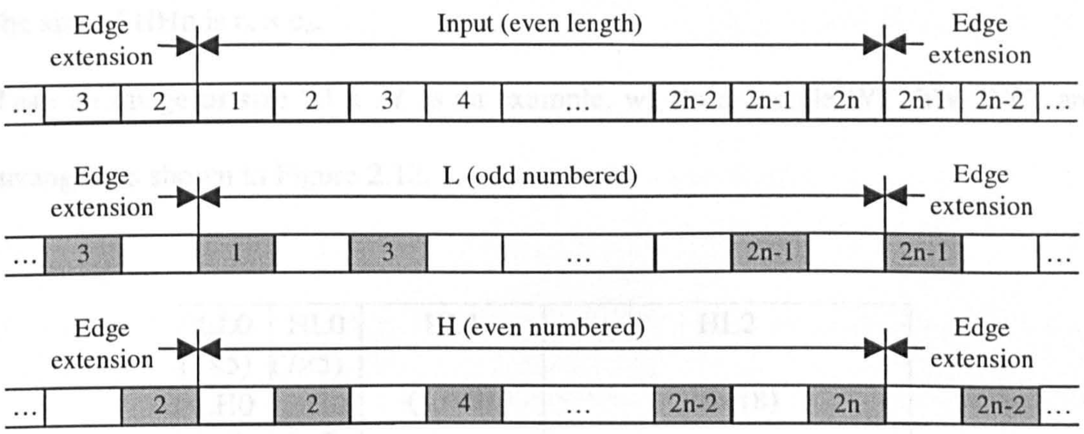


Figure 2.11 Down/up-sampling and edge extension in case of even length input

The edge extension in Figure 2.10 and Figure 2.11 is for symmetric filters. In the anti-symmetric case, the output samples outside the boundary are of the equal and opposite values as that shown in these figures. So, the edge extension of the H and L signals for synthesis should use the equal and opposite values as well.

Now we check the resulting WCs based on the above scheme. Suppose the image is of size M×N. For a k-scale WT, we get 3k+1 subbands, namely, LL0, HLn, LHn and HHn, where n = 0, 1, …, k-1.

Denote R_n the number of rows of HLn, and C_n the number of columns of LHn. For the convenience of calculation, we set $R_k=M$, and $C_k=N$. R_n and C_n can be calculated according to the following iteration, for n from k-1 to 0:

$$\begin{aligned} R_n &= \lfloor (R_{n+1}+1)/2 \rfloor \\ C_n &= \lfloor (C_{n+1}+1)/2 \rfloor \end{aligned} \qquad 2.65$$

Where $\lfloor x \rfloor$ denotes the integer part of x.

We also denote

$$\begin{aligned} r_n &= \lfloor R_{n+1}/2 \rfloor \\ c_n &= \lfloor C_{n+1}/2 \rfloor \end{aligned} \tag{2.66}$$

The size of LL0 is $R_0 \times C_0$, the size of HLn is $R_n \times c_n$, the size of LHn is $r_n \times C_n$, and the size of HHn is $r_n \times c_n$.

Take an image of size 50×37 as an example, we do a 3-scale WT. The WCs are arranged as shown in Figure 2.12.

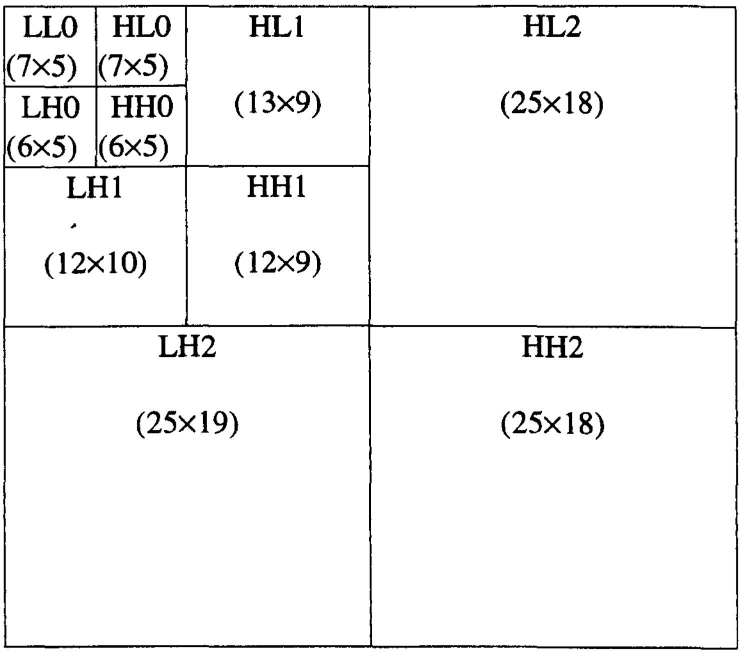


Figure 2.12 Results of 3-scale wavelet transform, 50 × 37 image

Now we explain why the symmetric edge extension without overlapping (shown in Figure 2.9) cannot get PR if we keep the output the same length as the input in WT, in case the input is of odd length. The length of the H and L signals has to be different, as shown in Figure 2.13. Here inside the boundary, down-sampling is the same as in Figure 2.10, keeping odd-numbered samples for L signal and even-numbered samples for H signal. But outside the boundary, the even-numbered samples are kept outside the L signal, and the odd-numbered samples are kept outside the H signal. The down-sampled output is no longer symmetric. We can still reject the output samples outside the boundary and get the H and L signals. But it is impossible to recover the rejected samples from the H and L signals. For synthesis, the edge extension after up-sampling has to be different with that in Figure 2.10. The edge extension of the up-sampled L signal can be the same as that in Figure 2.9, but an extra zero need to be inserted on each side of the up-sampled L signal before the same edge extension, as shown in Figure 2.13. After edge extension, the first and last samples of L signal are repeated continuously, and two continuous zeros appear at both ends of H signal. For signal reconstruction, the continuously repeated samples in the extended L signal mean unnecessary extra information or redundancy, while the continuos zeros in the extended H signal mean insufficient information. As the result, PR cannot be reached.

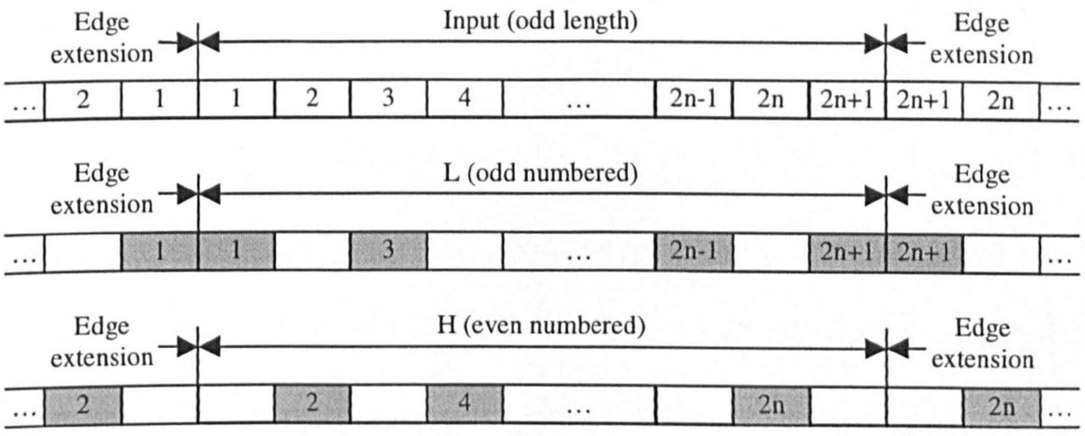


Figure 2.13 Edge extension without overlapping in case of odd length input

2.9 Summary

In this chapter, we have reviewed filter banks and the wavelet transform (WT). The necessary and sufficient conditions for perfect reconstruction (PR) were addressed.

We showed that the PR filter banks are biorthogonal, and showed the relations of the filters for PR FIR filter banks. We also showed the relationship between the filter banks and the WT. We built the biorthogonal WT from PR filter banks, and got PR filter banks from biorthogonal wavelet transform. Proofs for the above conditions and relations were presented, and many were proved using our own technique.

We went through separable two-dimensional multi-resolution WT using two-channel filter banks. It is the key for the transform coding of images. We also discussed the image edge extension for the WT. Various edge extension methods were given, together with our choice. The final arrangements of the WCs were presented.

References

[1] S.Mallat. A wavelet tour of signal processing. Academic Press, 1998.

[2] A.Cohen, I.Daubechies and J.-C.Feauveau, 'Biorthogonal bases of compactly supported wavelets', Communications on Pure and Applied Mathematics, Vol.XLV, pp.485-560, 1992.

[3] M.Vetterli and C.Herley, 'Wavelets and Filter banks: theory and design', IEEE Transactions on signal processing, Vol.40, No.9, pp.2207-32, September 1992.

[4] C.S.Burrus, R.A.Gopinath and H.Guo. Introduction to wavelets and wavelet transforms: A Primer. Prentice Hall, NJ, 1998.

[5] G.Strang and T.Nguyen. Wavelets and Filter banks. Wellesley-Cambridge Press, 1997.

[6] M.Vetterli and J.Kovacevic. Wavelets and subband coding. Prentice hall, NJ, 1995.

[7] S.Li and W.Li, 'Shape-adaptive discrete wavelet transform for arbitrarily shaped visual object coding', IEEE Transactions on circuits and systems for video technology, Vol.10, No.5, pp.725-43, August 2000.

[8] B.E.Usevitch, 'A tutorial on modern lossy wavelet image compression: foundation of JPEG 2000', IEEE Signal Processing Magazine, Vol.18, No.5, pp.22-35, September 2001.

Chapter 3

Statistical Properties of Wavelet Coefficients

To code a data set efficiently, we must know its statistical properties so as to choose the appropriate coding method. In this chapter, we check the distribution of the wavelet coefficients (WC) of typical natural images, and then explore their intra-subband, inter-subband (at the same level), and inter-level correlation. It will help us in understanding the techniques used in the transform coding of images, such as EZW (embedded zerotree wavelet coding) [1], SPIHT (set partitioning in hierarchical trees) [2], and EBCOT (embedded block coding with optimised truncation of the embedded bit-streams) [3], and in our own research.

We use Lena as an example image, as shown in Figure 3.1. It is often used as a test image in image coding. Its size is 512 × 512, and it is 8 bits per pixel (bpp), greyscale. We apply the 5-scale biorthogonal 9/7 wavelet transform (WT) to the image. The WT results in 16 subbands as shown in Figure 3.2: LL0, HLn, LHn, HHn, where n is the decomposition level (or resolution) numbered from 0 to 4. We also denote LL0 as LL, and refer HL, LH and HH to any or all (according to the relevant context) of HLn, LHn and HHn respectively.

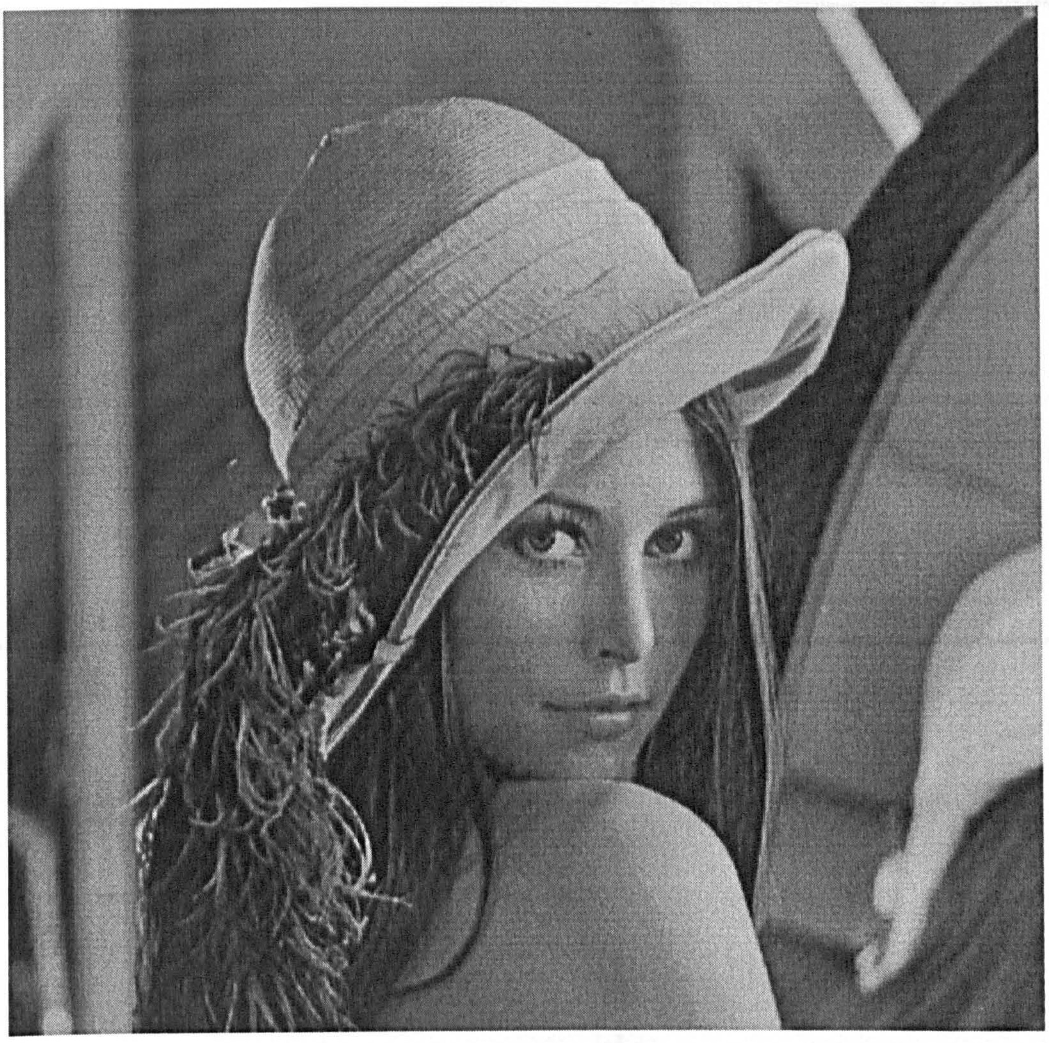


Figure 3.1 Lena (512×512, 8bpp)

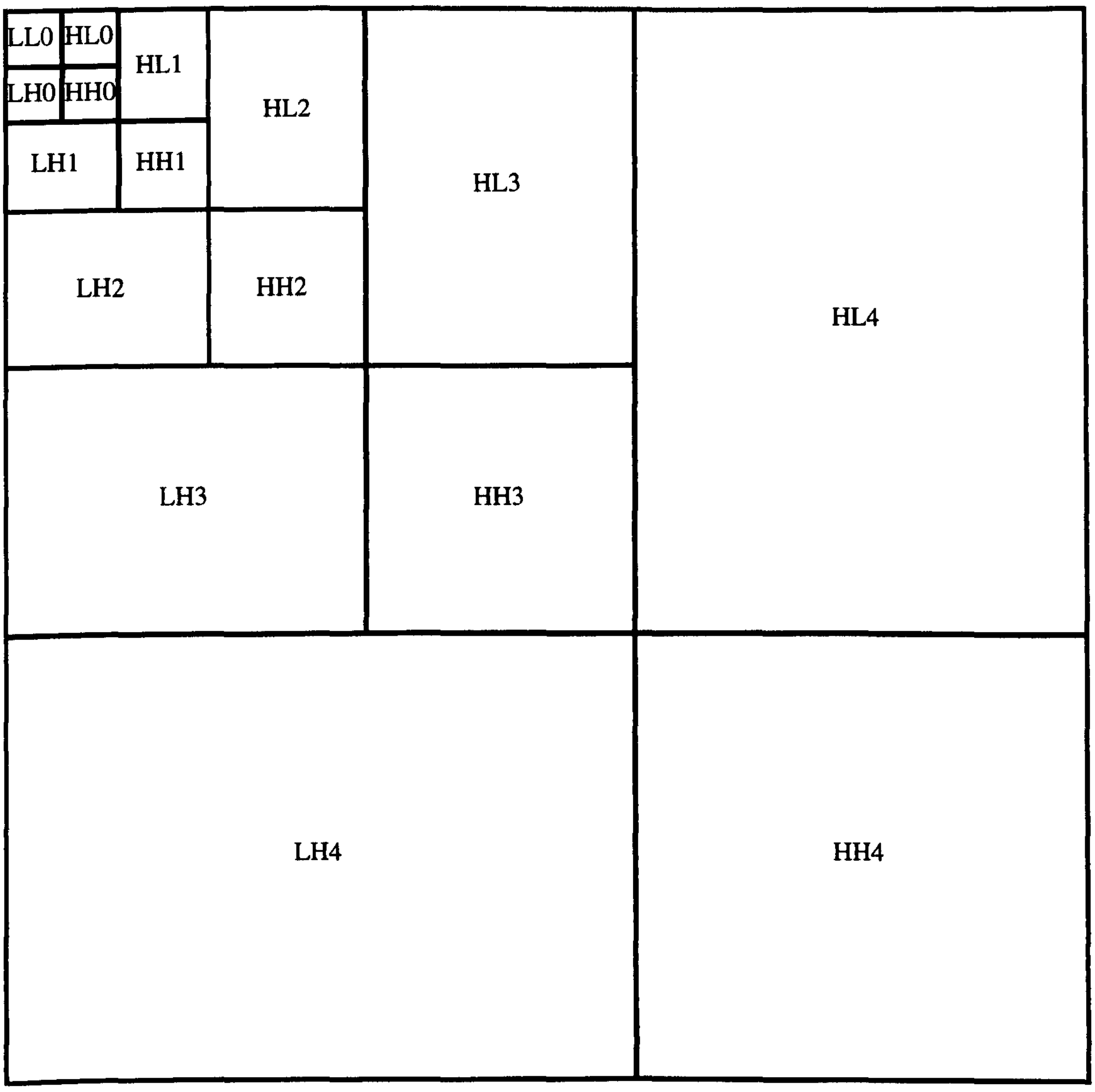


Figure 3.2 Subbands of 5-scale wavelet transform

3.1 Distribution of all wavelet coefficients

The maximum value of all WCs of Lena is 6760.5, and the minimum is -1337.3. Figure 3.3 shows the histogram of the WC values lying between neighbouring integers. In detail, for a positive integer k, the height of the bar at k represents the total number of WCs whose values are in the range [k, k+1), while the height of the bar at -k represents the total number of WCs whose value is in the range (-k-1, -k]. At 0, the height of the bar represents the total number of WCs whose value is in the range (-1, 1).

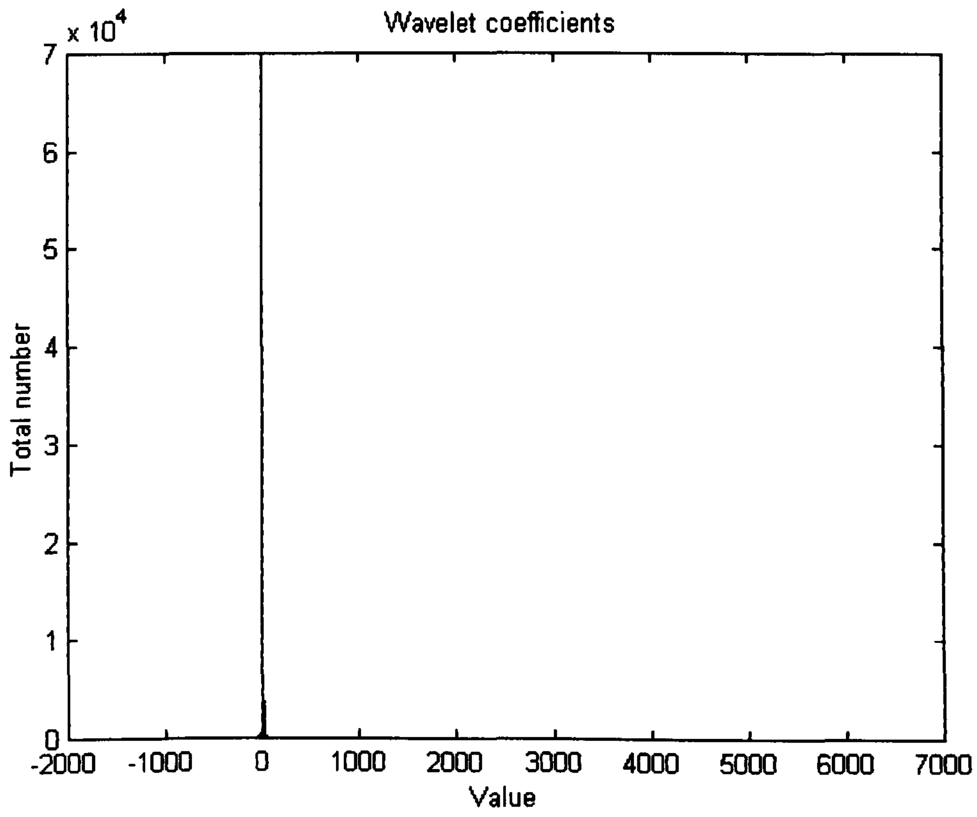


Figure 3.3 Distribution of all wavelet coefficients of Lena (linear axis)

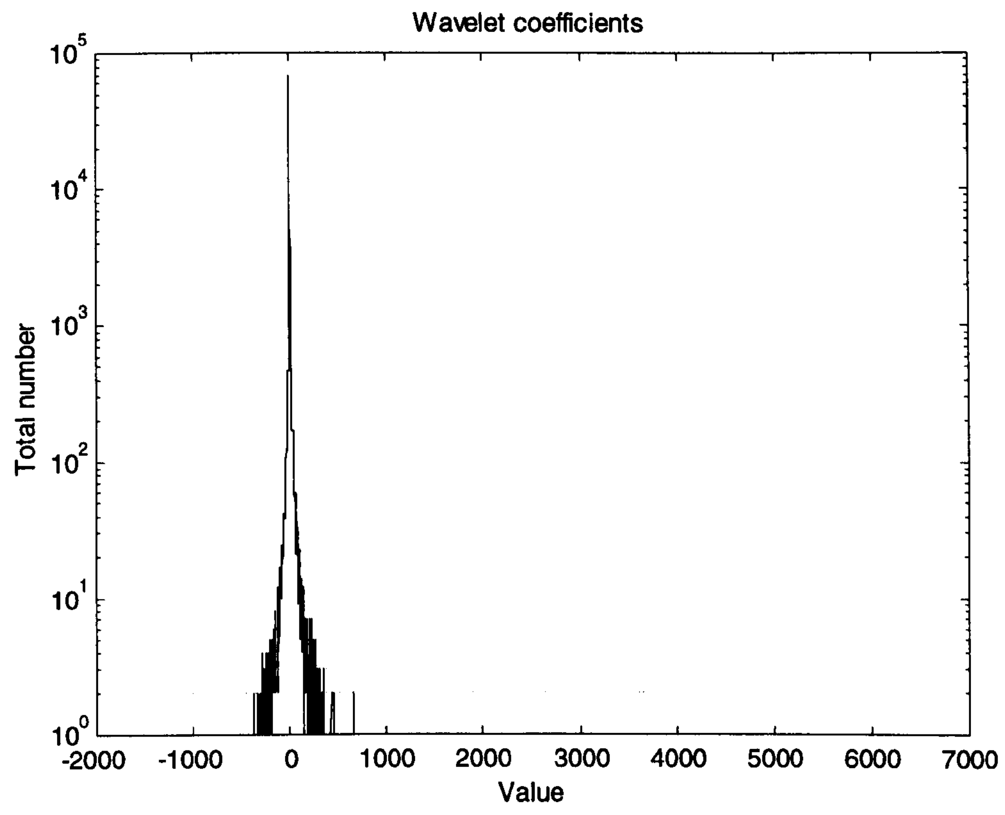


Figure 3.4 Distribution of all wavelet coefficients of Lena (log y-axis)

Figure 3.3 shows that most WCs gather round 0. To see the distribution clearly, we use a logarithmic scale for the vertical axis (y). For convenience, we plot a curve instead of the histogram. The result is shown in Figure 3.4. The curve is not continuous. This is because in some ranges, the total number of WCs is 0.

Figure 3.4 is better than Figure 3.3, but still not clear enough to see the distribution of WCs. Another way is to use a non-linear horizontal axis (x). We slice the WC value (x-axis) into non-linear divisions: $[2^k, 2^{k+1})$ (indexed as k+1), $(2^{k+1}, -2^k]$ (indexed as –k-1), and (-1, 1) (indexed as 0), where k is a non-negative integer. All the WCs of Lena lie in division –11 to 13. The resulting distribution is shown in Figure 3.5.

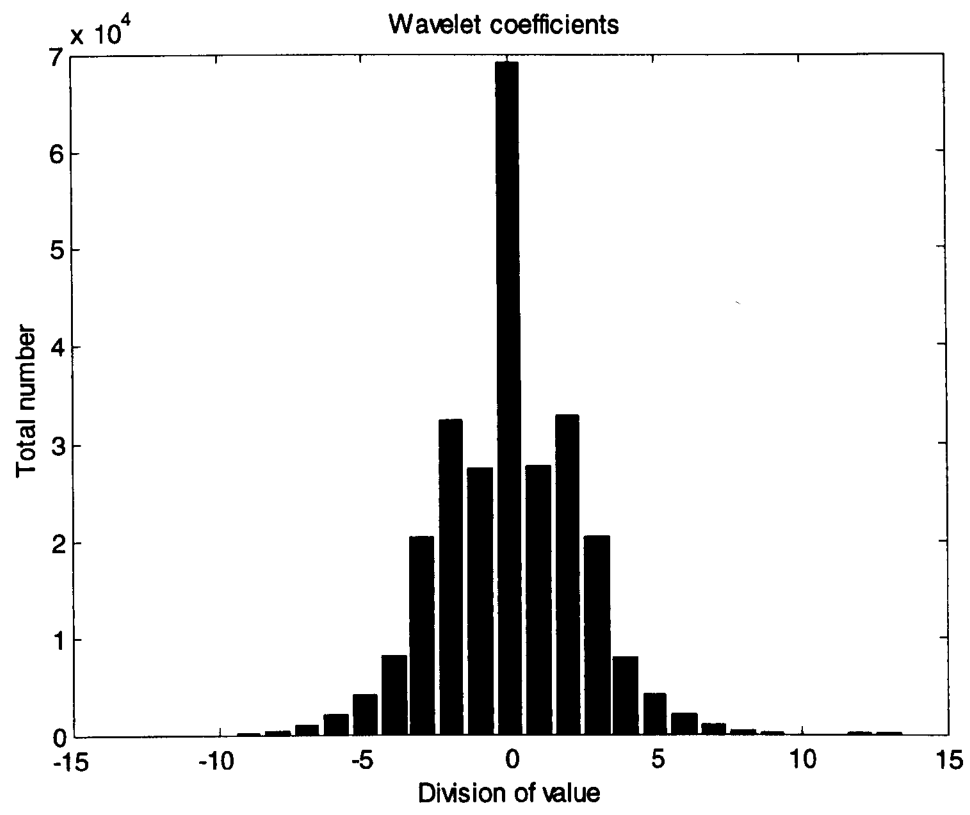


Figure 3.5 Distribution of all wavelet coefficients of Lena (non-linear x-axis)

Now we can see the distribution clearly from Figure 3.5. We ignore the three special divisions -1, 0, and 1, which stands for range (-2, -1], (-1, 1), and [1, 2) respectively. On the left negative side, as the division goes closer towards 0, although the interval of the division becomes shorter (the interval of division -k is half that of division -k-1, where k is an integer and k > 1), the total number of WCs in the division becomes larger. It is similar on the right positive side in Figure 3.5. This means that, the magnitude of most WCs is very small, while most of the signal energy is carried by a very few number of WCs. This is why we have changed the graph to view the distribution of WCs from Figure 3.3 to Figure 3.4 and then to Figure 3.5. This statistical property is good for coding.

The same co-ordinates (index of the specified non-linear divisions for x-axis) are used in the graphs of the distribution of the WCs later in this chapter, except as otherwise stated.

3.2 Distribution of wavelet coefficients on LL

Now we check the distribution of the WCs on a subband. We check LL first, shown in Figure 3.6. There are 256 WCs on LL altogether, the maximum value is 6760.5, the minimum is 1486.1, and the mean value is 3954.0. They lie in division 11 to 13 in Figure 3.6. To compare, we show the distribution of all other WCs (on HL, LH and HH, excluding LL) in Figure 3.7. Those WCs lie in division –11 to 11, with maximum 1200.5, minimum -1337.3, and mean -0.0119. Apparently, the WCs on LL are DC(direct current)-biased, while other WCs are DC-balanced. The minimum value of the WCs on LL is greater than the maximum value of other WCs. That is why there are some humps on the right end of Figure 3.5.

We can make the WCs on LL DC-balanced by deducting their mean value. This suggests a DC-level shifting. The distribution of the WCs on LL after DC-level shifting is shown in Figure 3.8. Now the maximum value becomes 2806.5, and the minimum becomes –2467.9. The WCs lie in the division –12 to 12.

As the WCs on LL are only 1/1024 of all WCs in number, and they are often not overlapped with other WCs in value, the DC-level shifting will not change the overall statistical property of the WCs very much as described previously in this section. However, it does reduce the total number of bit planes of the magnitude of all WCs by 1, and will favour the coding.

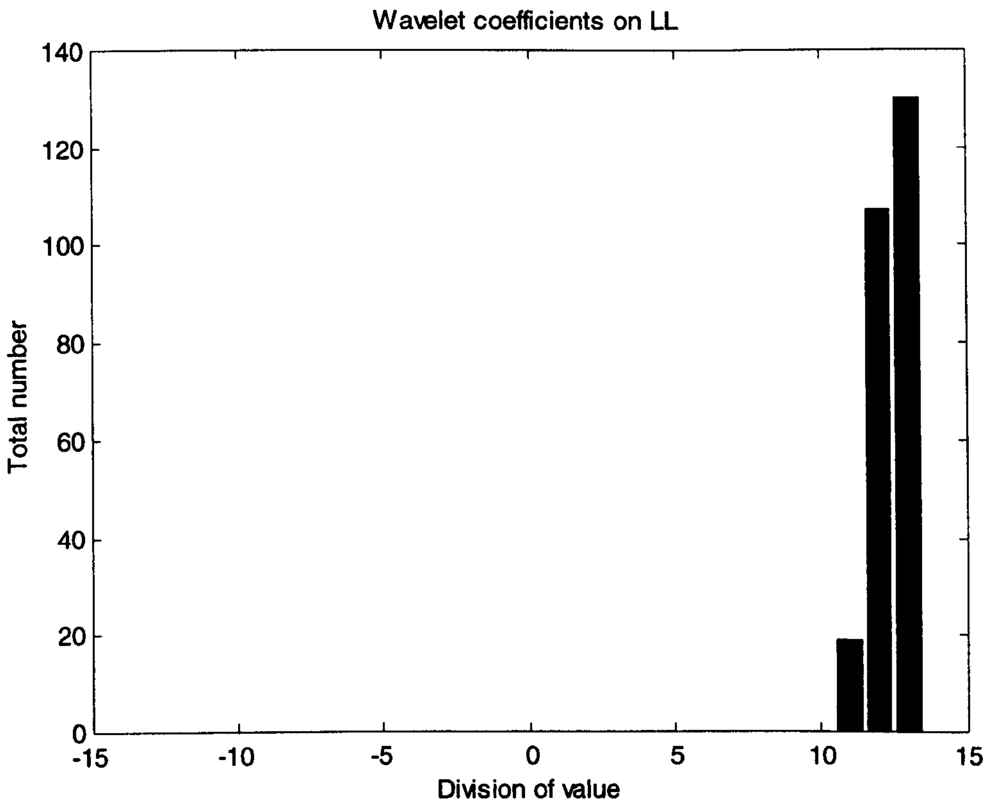


Figure 3.6 Distribution of wavelet coefficients on LL of Lena

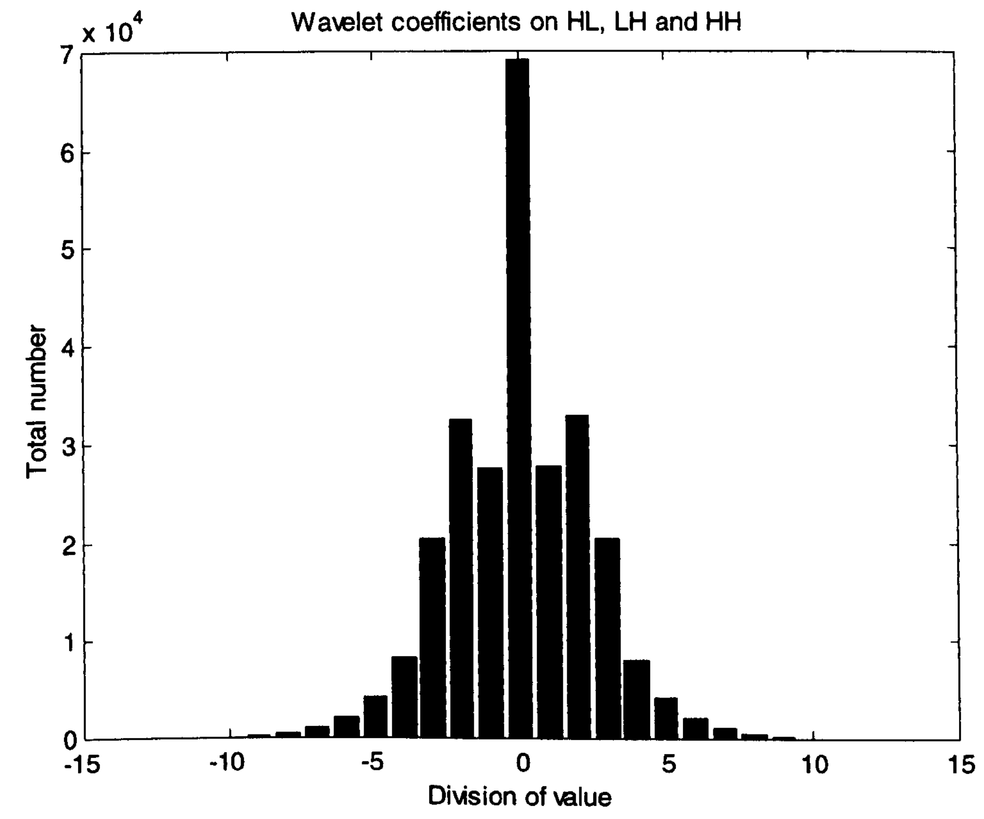


Figure 3.7 Distribution of all wavelet coefficients on HL, LH and HH of Lena

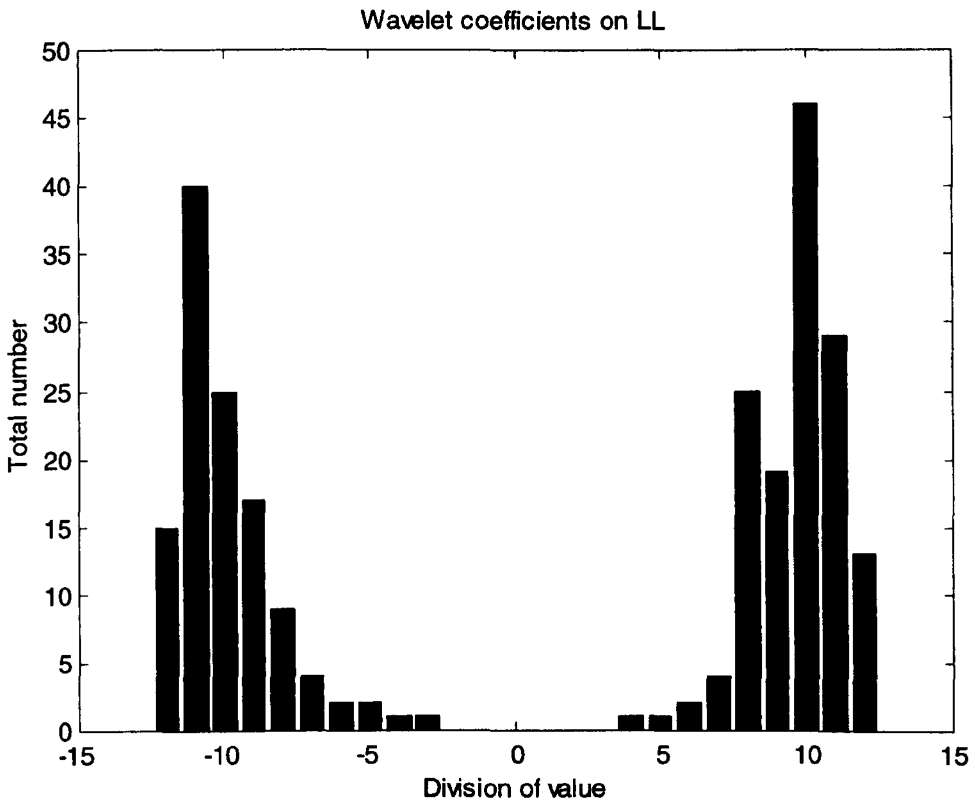


Figure 3.8 Distribution of wavelet coefficients on LL of Lena after DC-level shifting

3.3 Distribution of wavelet coefficients on HL, LH and HH

The WCs on LL spread around over their range as shown in Figure 3.8, while the WCs on HL, LH and HH gather round 0 as shown in Figure 3.7. We need to take this into consideration for coding, and use different strategies to encode them.

Now we take a close look at the distribution of the WCs on HL, LH and HH, from various points of view. We check individual subbands first. Figure 3.9 depicts the distributions of the WCs on HL4, LH4 and HH4. The three subbands are of the finest resolution – level 4, and the WCs on them are dominant in number. The three curves for subband HL4 (solid line), LH4 (dotted line) and HH4 (dashed line) are of similar shape, and are similar as that for all WCs on HL, LH and HH in Figure 3.7. Further studies show that the WCs of any HL, LH or HH subband are of similar distribution. Their distribution is similar in the sense that the majority of WCs gather round 0, and that the number of WCs in a division increases monotonically as the division getting closer towards 0 (except division –1, 0 and 1). We treat them as a whole later in this section.

Now we check the detailed distribution of these WCs inside each division, shown in Figure 3.10 – Figure 3.19. We use a linear horizontal axis (x) inside each division. We see again that in every division, the number of WCs in each of the equally spaced intervals increases monotonically as their value getting closer to 0, except in Figure 3.13 and Figure 3.19. For Figure 3.13, if we merge every two intervals, it becomes Figure 3.14, in which the number of WCs increases monotonically as the value goes towards 0. In Figure 3.15 – Figure 3.19, we merge the intervals in the same way. In Figure 3.19, the left negative side does not increase monotonically.

This is not surprising if we notice that the total number of WCs is only 44, which is not enough to observe the typical distribution in a wide range – (-1024, -512]. For the outermost divisions -11 and 11, which represent range (-2048, -1024] and [1024, 2048), the total number of WCs are 7 and 2 respectively. It is meaningless to discuss their distribution.

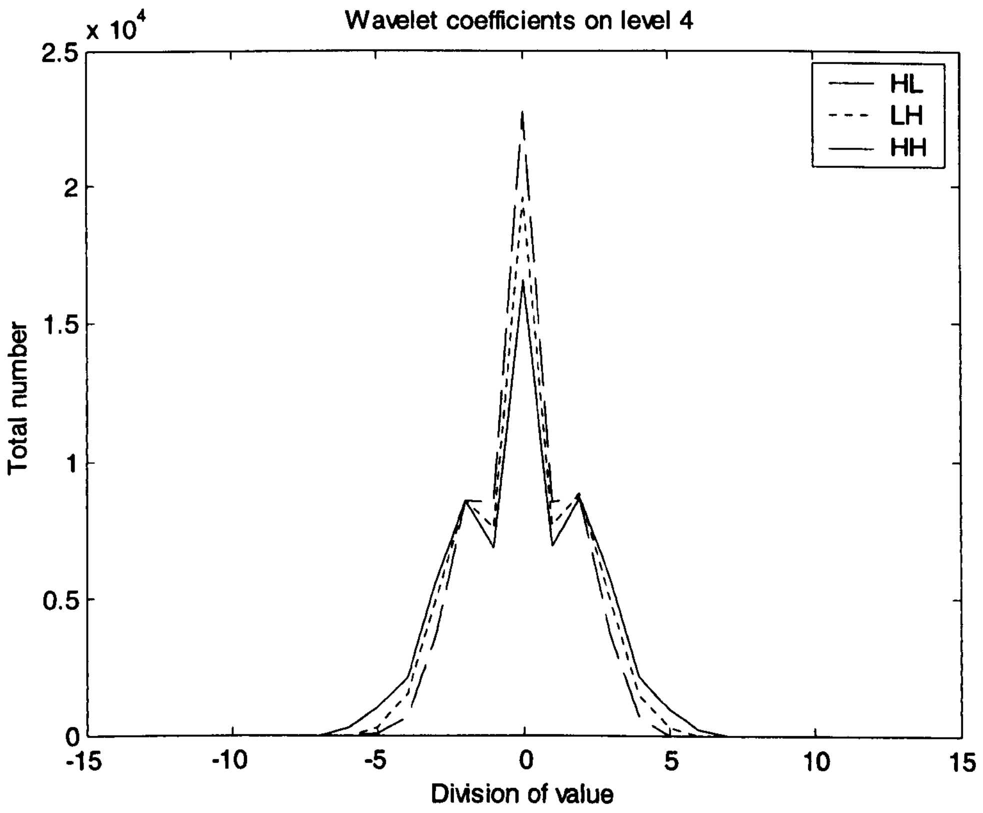


Figure 3.9 Distribution of wavelet coefficients on level 4 of Lena

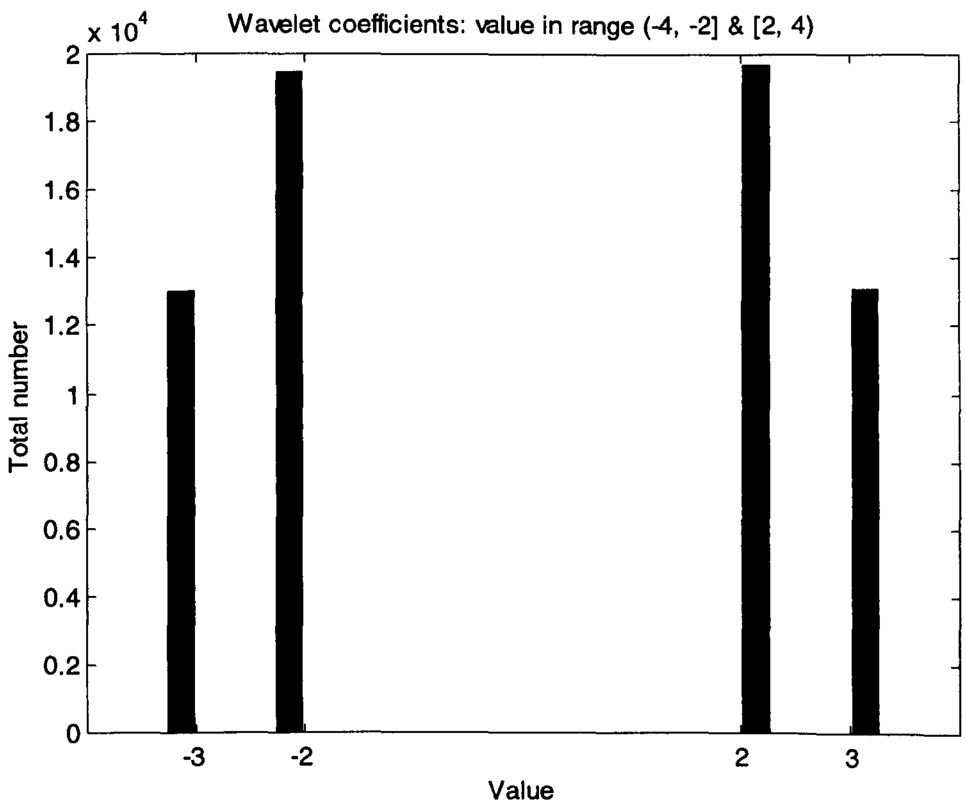


Figure 3.10 Distribution of wavelet coefficients of Lena in division ±2

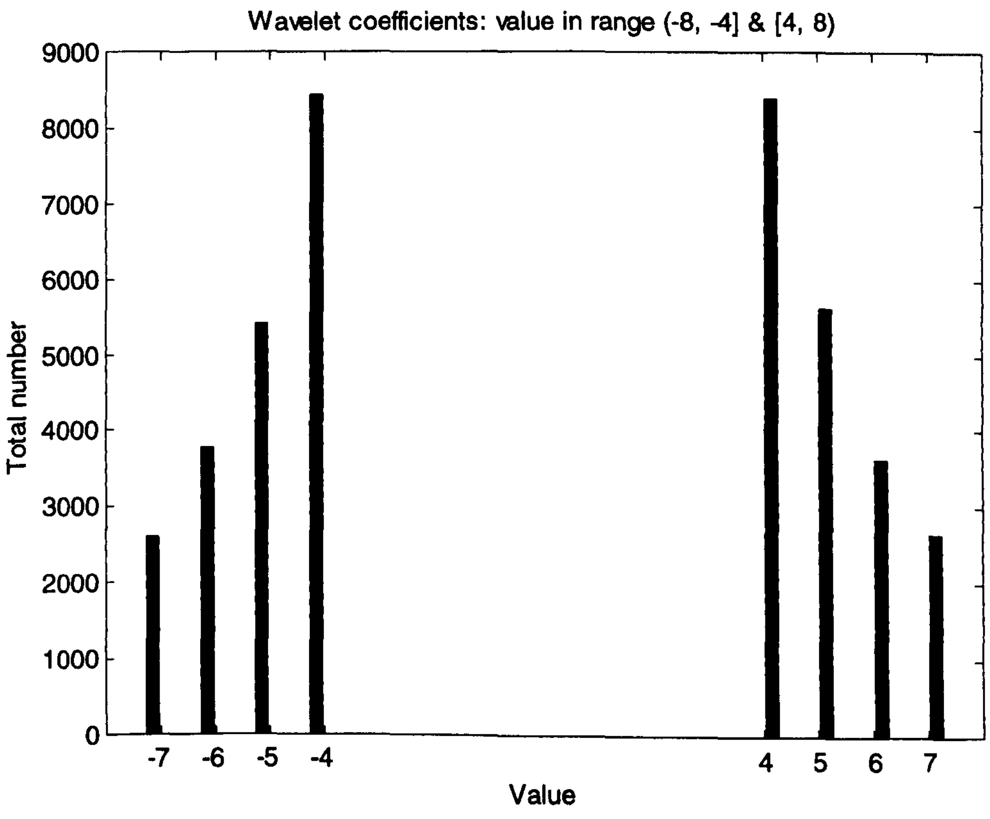


Figure 3.11 Distribution of wavelet coefficients of Lena in division ±3

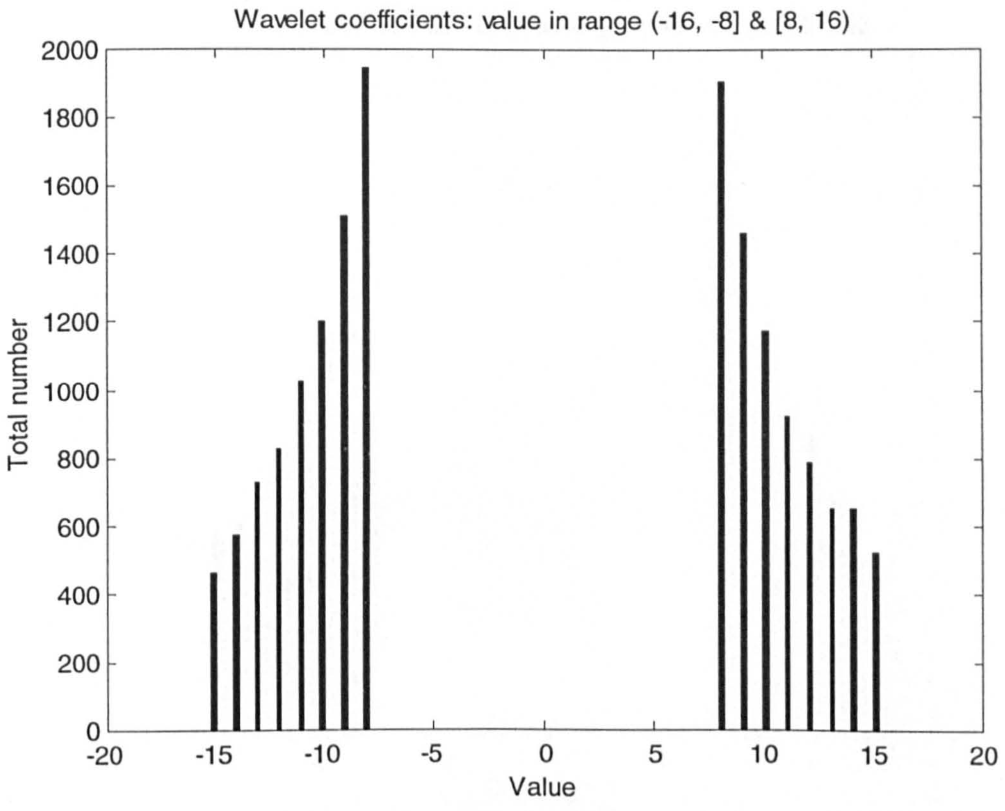


Figure 3.12 Distribution of wavelet coefficients of Lena in division ±4

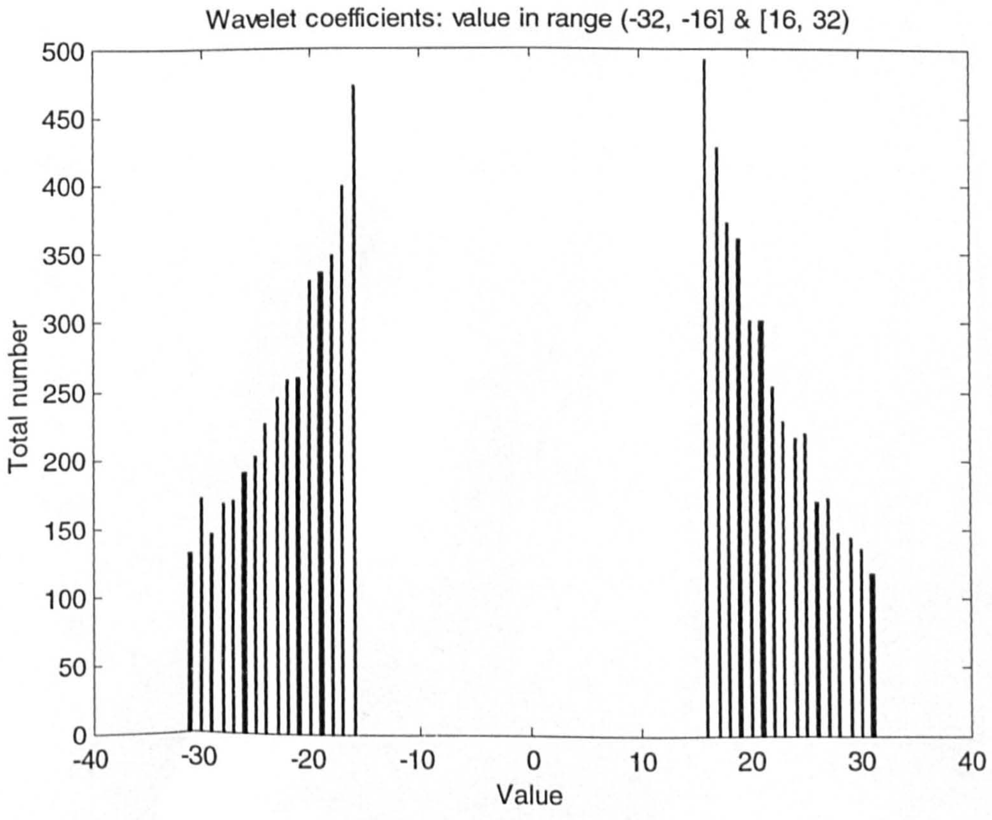


Figure 3.13 Distribution of wavelet coefficients of Lena in division ±5

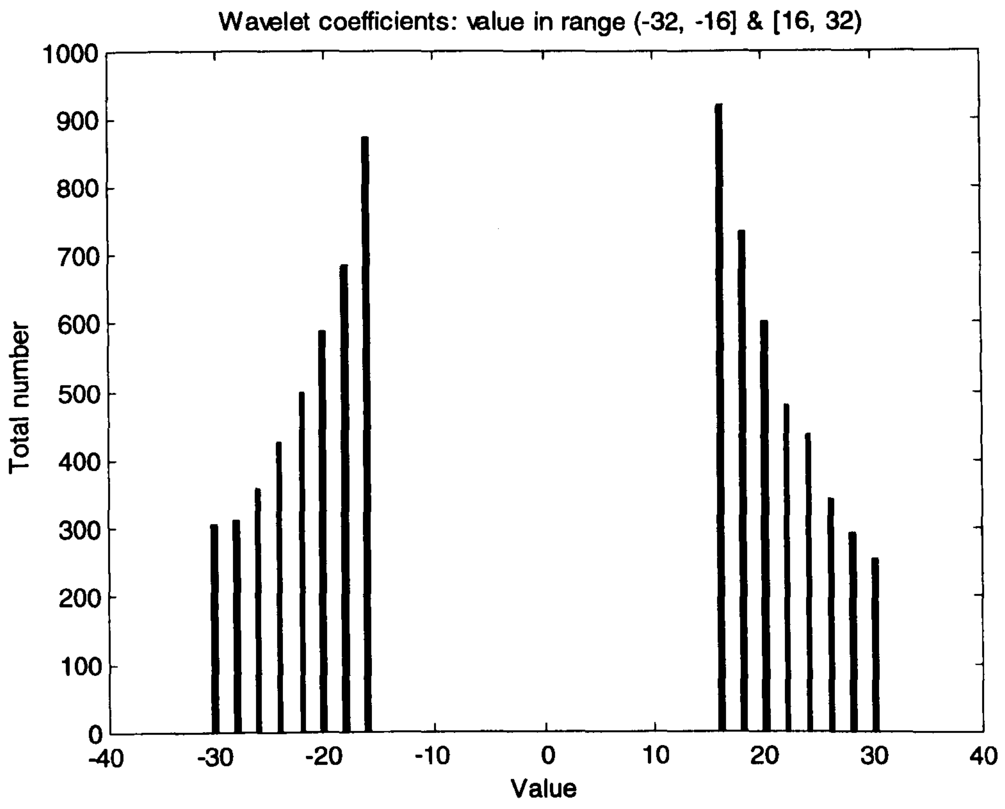


Figure 3.14 Distribution of wavelet coefficients of Lena in division ±5

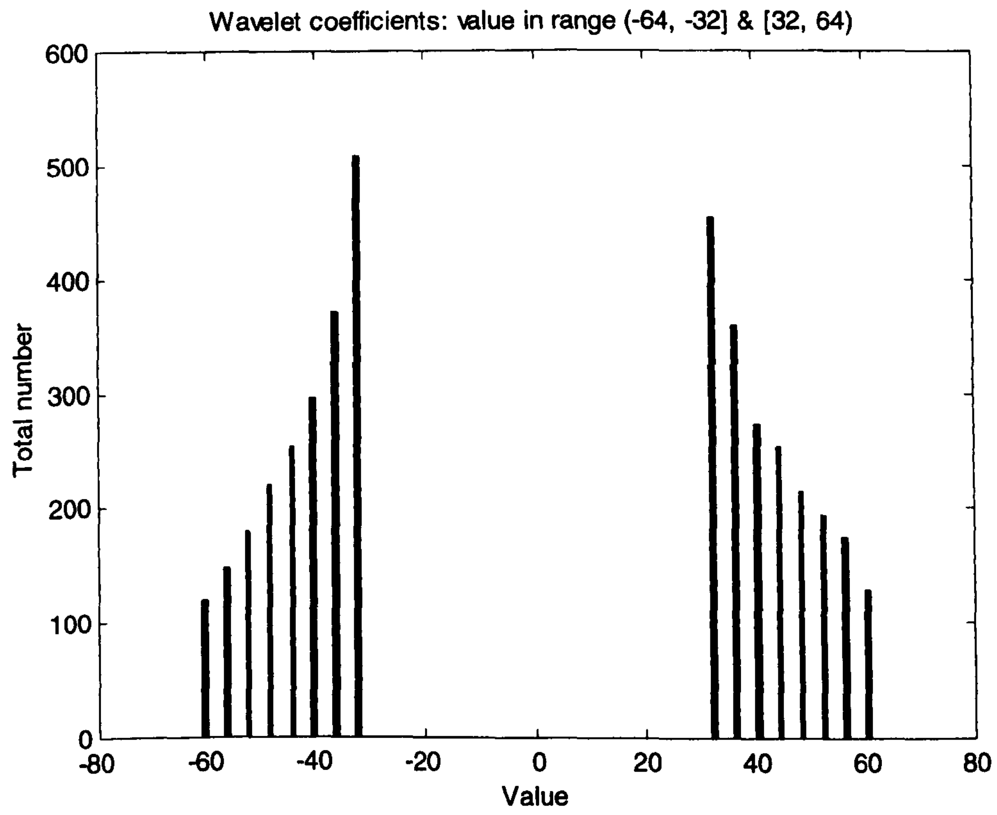


Figure 3.15 Distribution of wavelet coefficients of Lena in division ±6

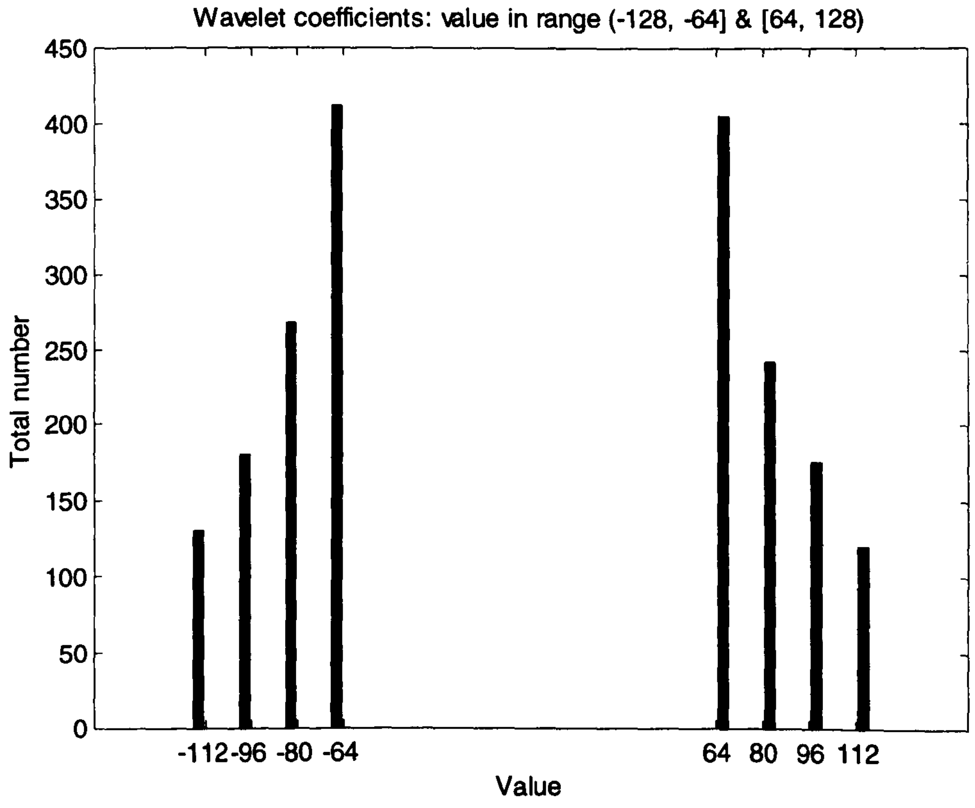


Figure 3.16 Distribution of wavelet coefficients of Lena in division ±7

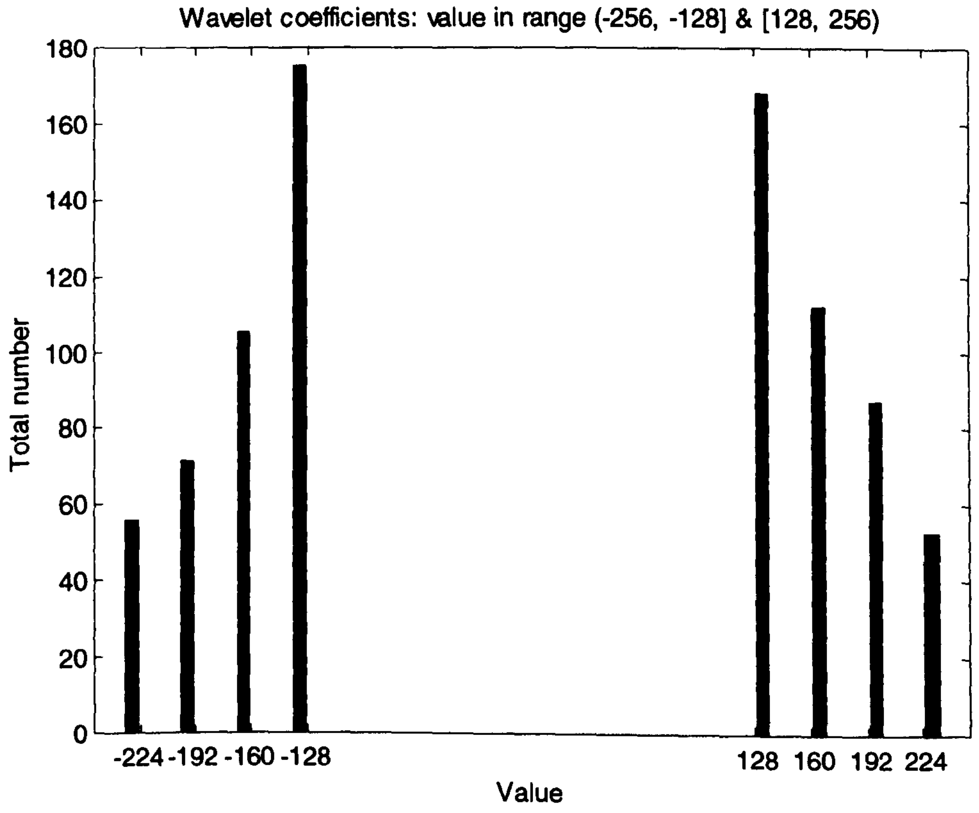


Figure 3.17 Distribution of wavelet coefficients of Lena in division ±8

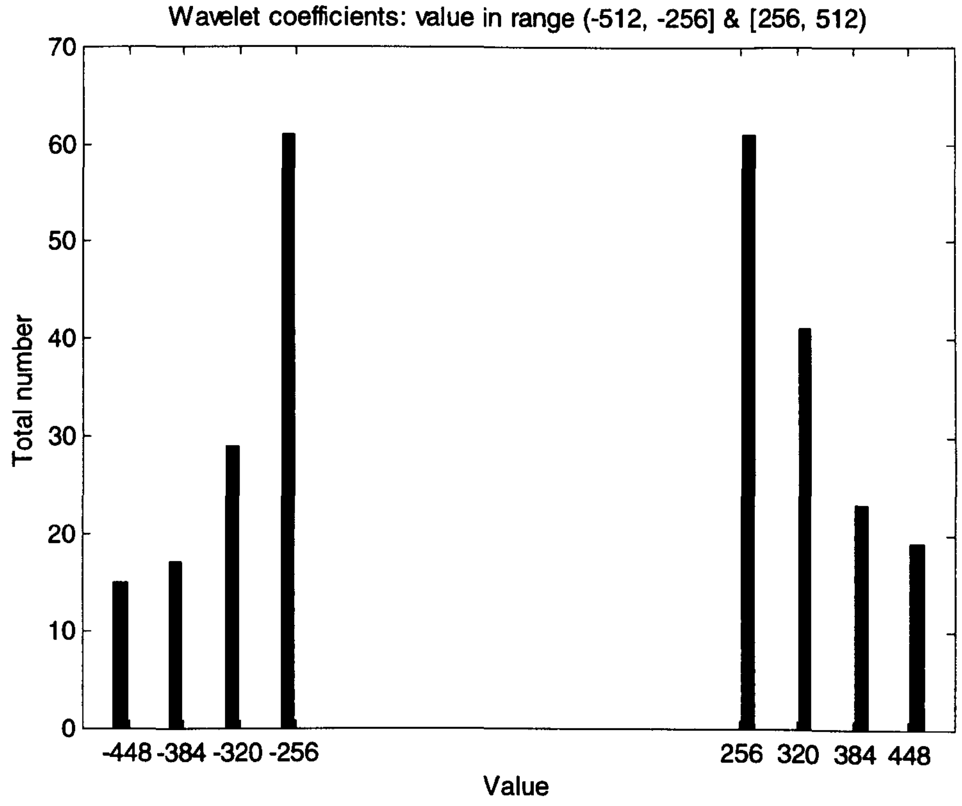


Figure 3.18 Distribution of wavelet coefficients of Lena in division ±9

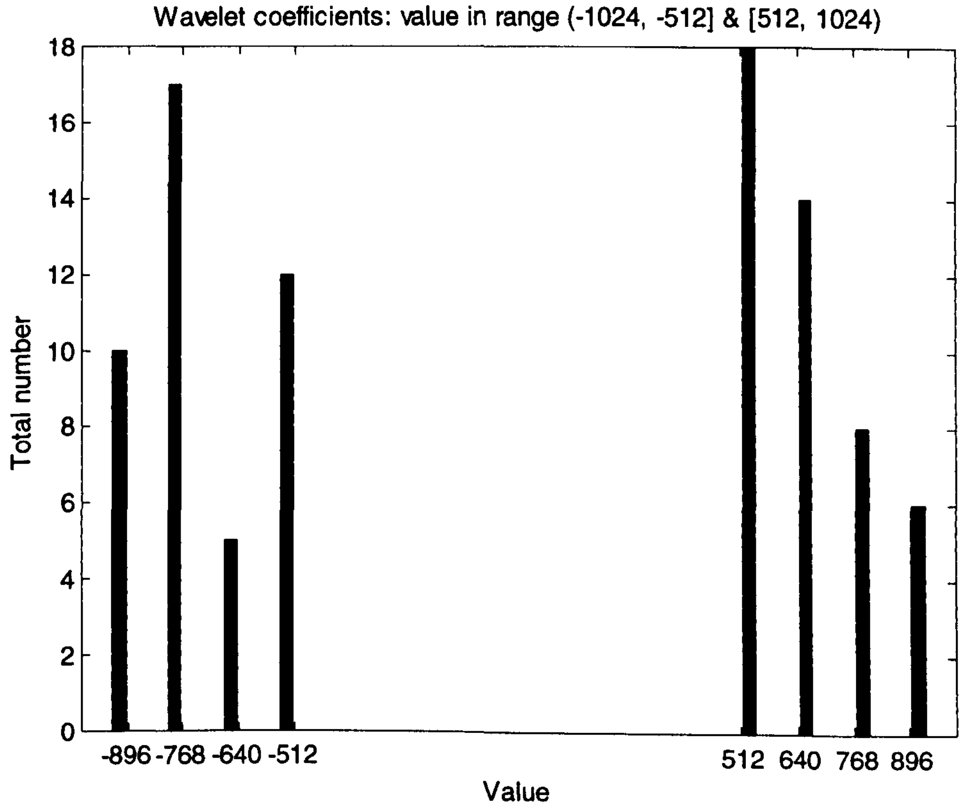


Figure 3.19 Distribution of wavelet coefficients of Lena in division ±10

Table 3.1 Mean value of the wavelet coefficients of Lena in each division

Division	Range	Centre	Mean
-11	(-2048, -1024]	-1536	-1124.6
-10	(-1024, -512]	-768	-776.37
-9	(-512, -256]	-384	-343.14
-8	(-256, -128]	-192	-175.33
-7	(-128, -64]	-96	-88.179
-6	(-64, -32]	-48	-43.915
-5	(-32, -16]	-24	-22.316
-4	(-16, -8]	-12	-10.967
-3	(-8, -4]	-6	-5.4905
-2	(-4, -2]	-3	-2.8692
2	[2, 4)	3	2.8671
3	[4, 8)	6	5.4948
4	[8, 16)	12	11.024
5	[16, 32)	24	22.07
6	[32, 64)	48	44.554
7	[64, 128)	96	87.607
8	[128, 256)	192	176.45
9	[256, 512)	384	349.43
10	[512, 1024)	768	699.42
11	[1024, 2048)	1536	1192.9

We list the mean value of the WCs in each division in Table 3.1, together with the central value of the division. As can be seen, the magnitude of the mean value is normally less than that of the central value (except in division -10).

3.4 Intra-subband correlation

The correlation between the WCs is very important for the strategy used to encode them. Higher correlation means more redundancy, and more compression could be achieved in coding if proper strategy is used.

To study the correlation of WCs for image coding, we are more interested in relations between signals than their absolute value. So, we use the correlation coefficient rather than the covariance. For the two-dimensional (2D) matrices **A** and **B**, of the same size, we define their correlation coefficient as

$$r = \frac{\sum_i \sum_j a(i,j) b(i,j)}{\sqrt{[\sum_i \sum_j a^2(i,j)][\sum_i \sum_j b^2(i,j)]}} \tag{3.1}$$

Where a(i, j) and b(i, j) are elements of **A** and **B** respectively at co-ordinates (i, j).

If **B** is equal to **A** shifted by *m* horizontally and shifted by *n* vertically, and the elements shifted in/out of the matrix boundaries are deleted, then *r* defined in (3.1) becomes the auto-correlation coefficient of **A** and is denoted as *r(m, n)*.

The value of the correlation coefficients is in the range [-1, 1].

We use these definitions throughout the rest of this chapter.

Figure 3.20 to Figure 3.22 show the auto-correlation coefficients of the WCs on HL, LH and HH of level 4 (the finest resolution). At the centre of the figures, *r(0, 0) = 1*, which has the maximum magnitude. In general, *r* located close to the centre has greater magnitude than *r* outside.

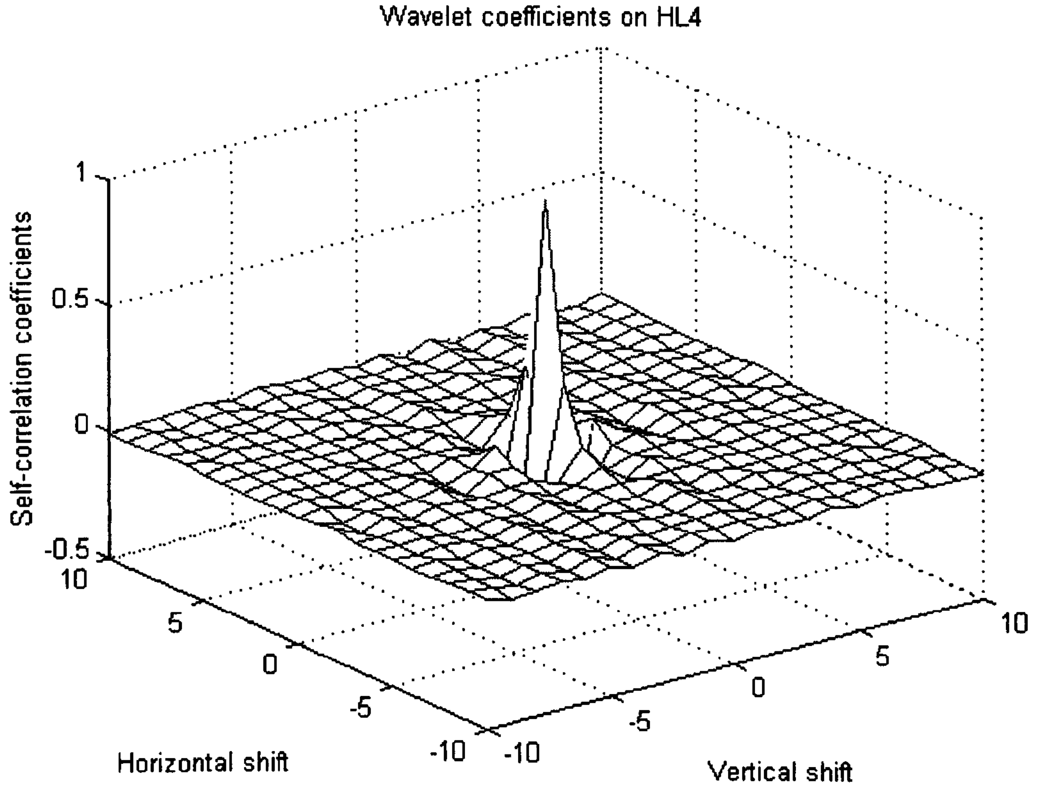


Figure 3.20 Intra-subband correlation of wavelet coefficients of Lena on HL4

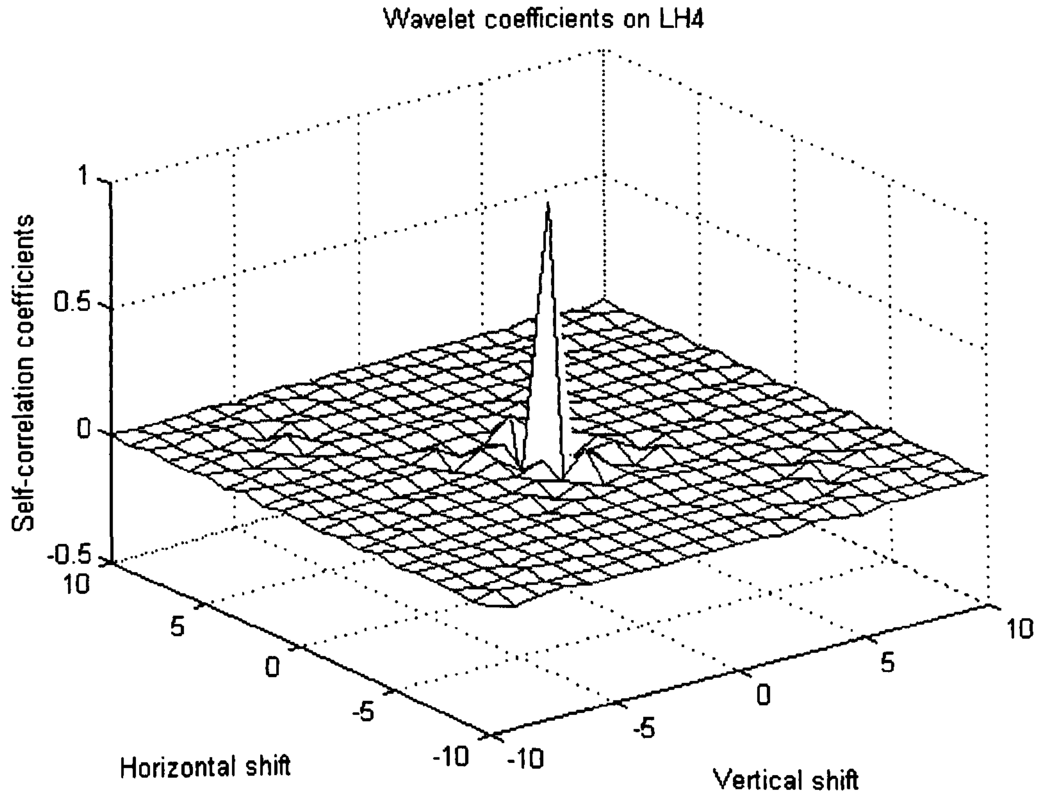


Figure 3.21 Intra-subband correlation of wavelet coefficients of Lena on LH4

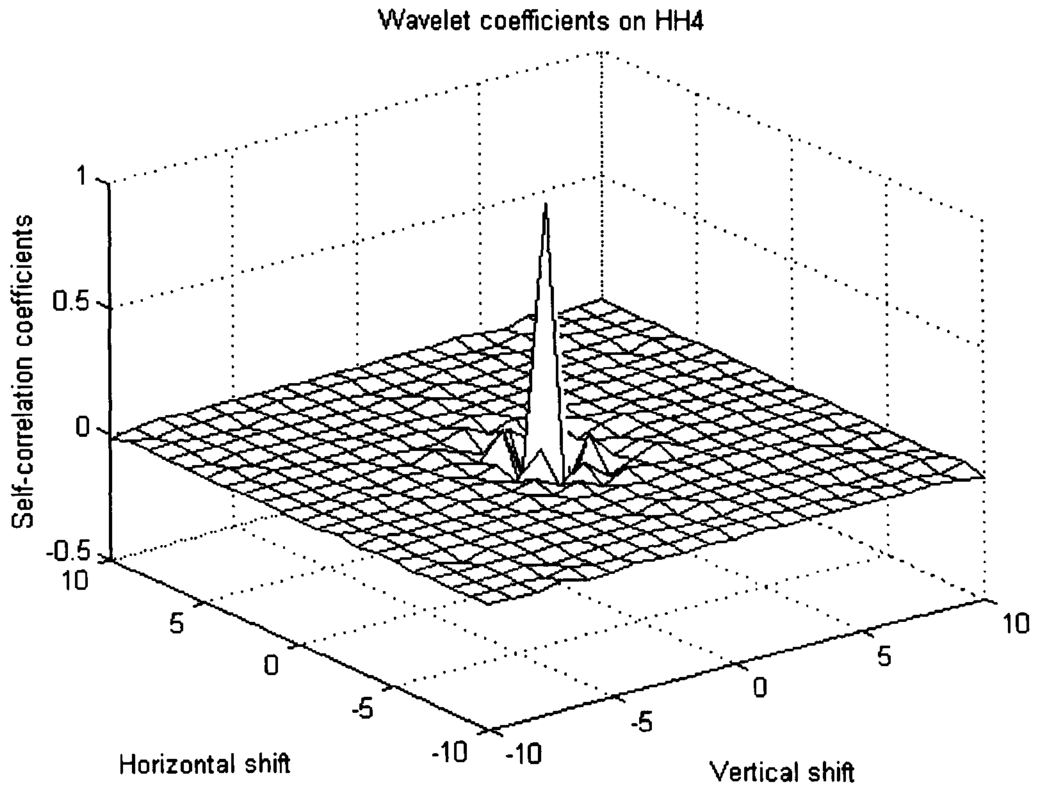


Figure 3.22 Intra-subband correlation of wavelet coefficients of Lena on HH4

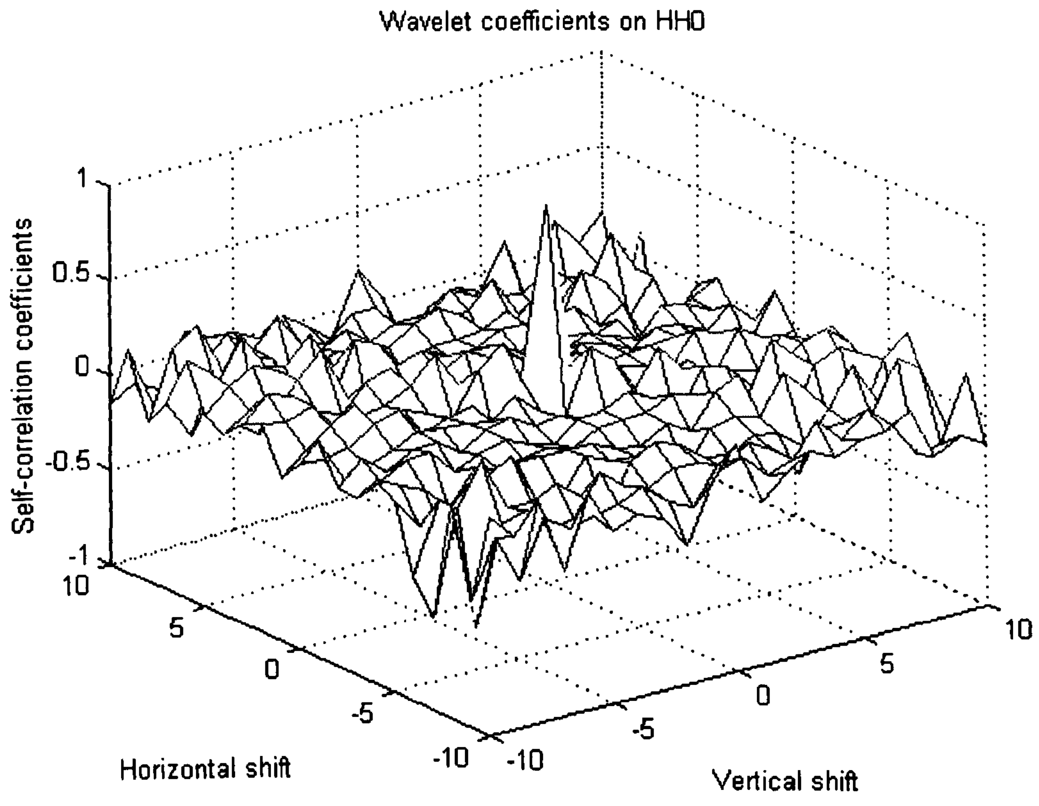


Figure 3.23 Intra-subband correlation of wavelet coefficients of Lena on HH0

Further studies show that as the resolution (decomposition level) decreases, the surface of *r(m, n)* becomes irregular. An extreme example is shown in Figure 3.23. We know that each WC corresponds to a square area of the original image. As the resolution decreases, the corresponding area increases, and *r(m ,n)* is becoming correlation between regions more than a correlation between pixels. The results here imply that neighbouring pixels are more correlated than regions. This property is typical for natural images.

Table 3.2 Auto-correlation coefficients of maximum magnitude (excluding 1) of wavelet coefficients of Lena on level 1 ~ 4

subband	***(m, n)***	***r(m, n)***
HL1	(-1, 0)	0.331395
HL2	(0, -1)	-0. 29049
HL3	(0, -1)	-0. 291538
HL4	(0, -1)	-0. 39712
LH1	(-1, 0)	-0. 382147
LH2	(-1, 0)	-0. 333728
LH3	(-1, 0)	-0. 424268
LH4	(-1, 0)	-0. 349077
HH1	(1, -1)	0. 213838
HH2	(-1, 0)	-0. 202929
HH3	(-1, 0)	-0. 241199
HH4	(-1, 0)	-0. 172336

However, the surface of *r(m, n)* is quite similar for resolution 1 ~ 4. We list *r(m, n)* of the maximum magnitude around the centre (excluding *r(0, 0)*) in Table 3.2. Here we are concerned about the magnitude other than the sign.

The average magnitude of *r(m, n)* in Table 3.2 is 0.30. We shall use this as a typical value for the intra-subband correlation coefficient.

3.5 Inter-subband correlation at same level

Now we use definition (3.1) to calculate the inter-subband correlation coefficients at the same level. Table 3.3 lists the results. The average magnitude is 0.029. We shall refer to it as typical value.

Table 3.3 Inter-subband correlation of wavelet coefficients of Lena on HL, LH and HH of same level

Level	LH & HH	HL & HH	HL & LH
1	-0.076130	-0.008845	0.066117
2	0.057193	-0.034465	0.041589
3	-0.011938	-0.005396	0.063084
4	-0.004136	-0.021111	0.049080

3.6 Inter-level correlation on HL, LH and HH

From chapter 2, we know that a WC of a coarse level corresponds to four WCs of the next fine level. To see the inter-level correlation, we correlate the WCs of the coarse level with the WCs at one of the four corresponding spatial orientations on the fine level. We get four correlation coefficients, and choose the value of maximum magnitude as the final result, as listed in Table 3.4. Their average magnitude is 0.13, which will be used as the typical value.

Table 3.4 Inter-level correlation coefficients of wavelet coefficients of Lena

Level	HL	LH	HH
0 & 1	0.214268	0.198743	-0.24192
1 & 2	0.103587	0.206566	-0.21096
2 & 3	0.167398	0.117892	-0.116673
3 & 4	0.114635	0.141768	0.0645661

3.7 Summary

Further experiments show that the WCs of other natural images have similar statistical properties. We present a group of example images here, of various types: Barbara (portrait, Figure 3.24), Boats (Figure 3.25), Goldhill (landscape, Figure 3.26), Mandrill (animal, Figure 3.27), Peppers (vegetable, Figure 3.28), and Zelda (portrait, Figure 3.29), etc. They are all of size 512×512, and are 8 bpp, greyscale. They will be used later in the thesis. 5-scale biorthogonal WT is used.

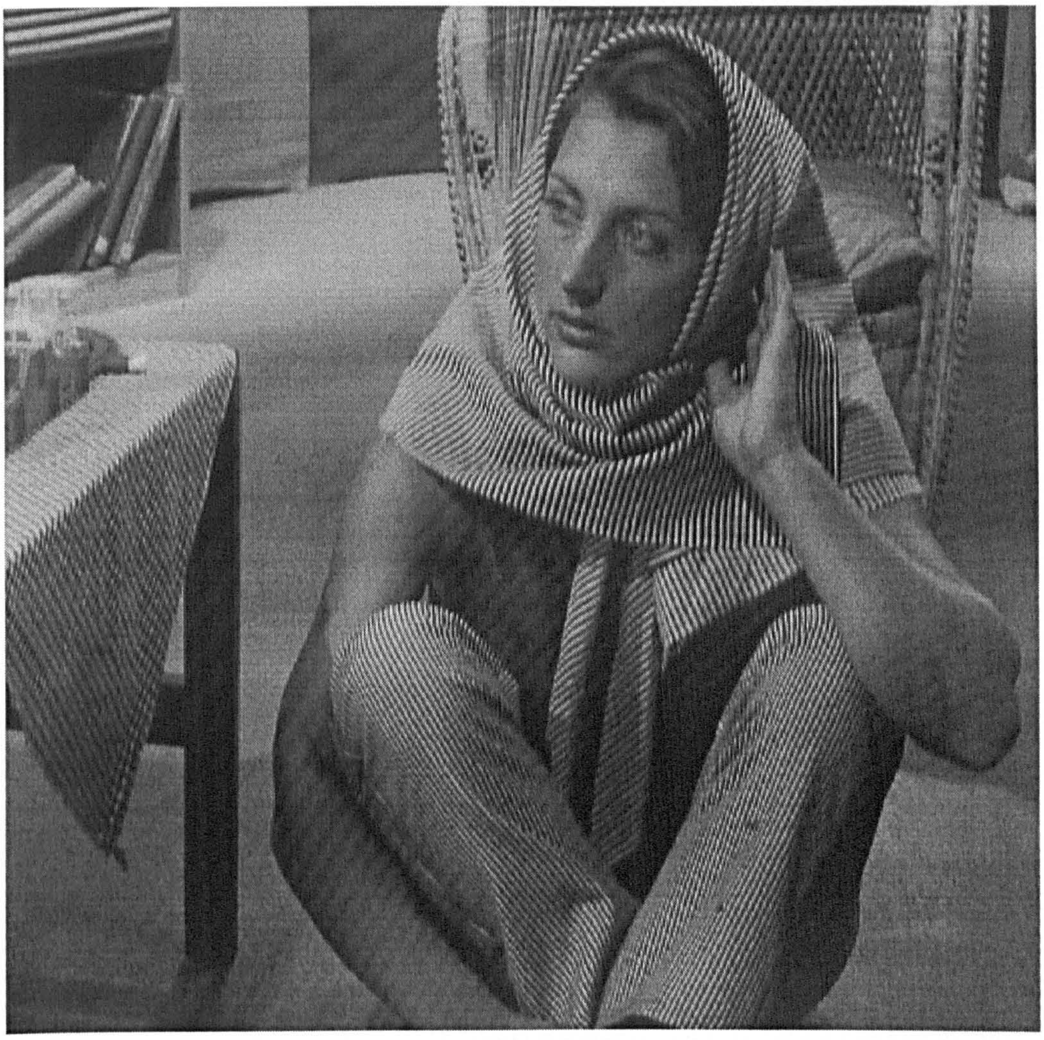


Figure 3.24 Barbara

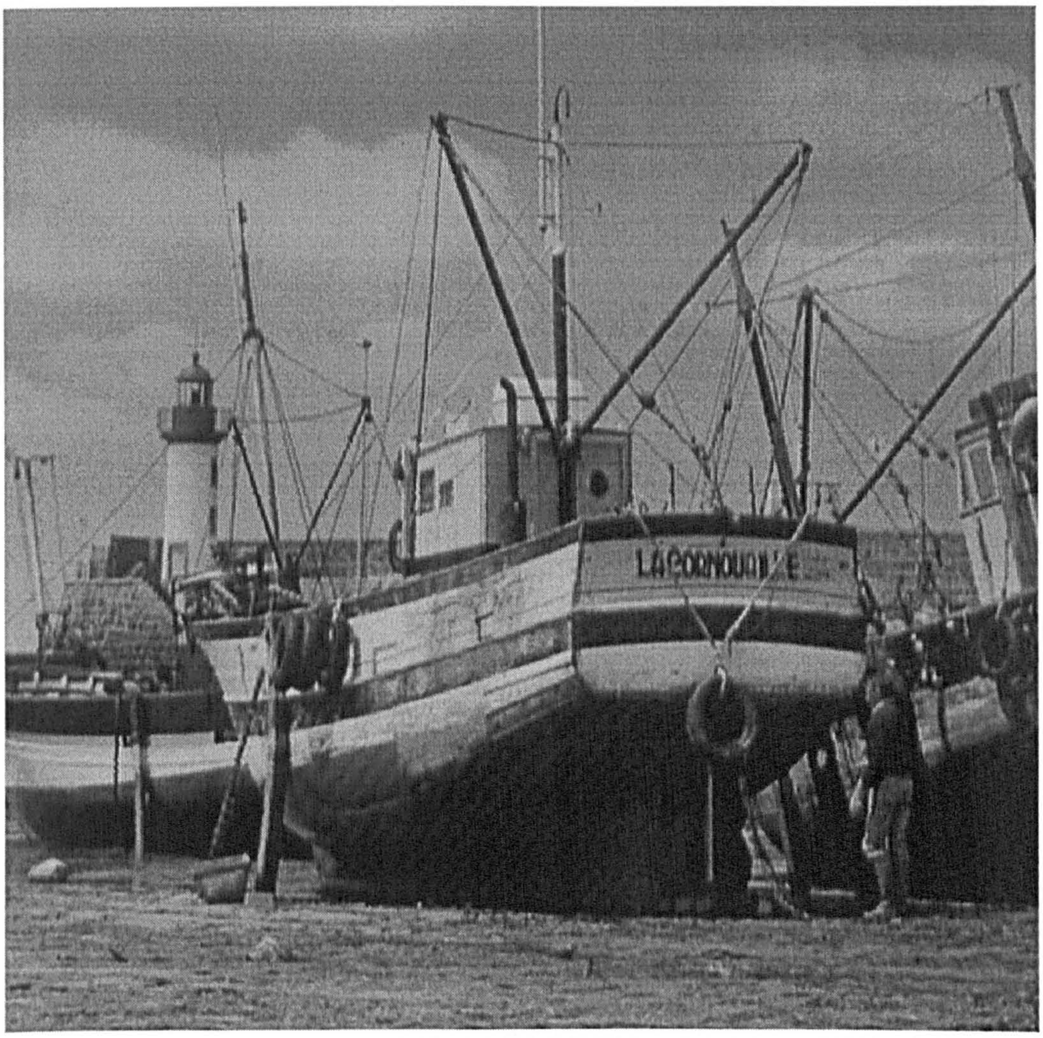


Figure 3.25 Boats

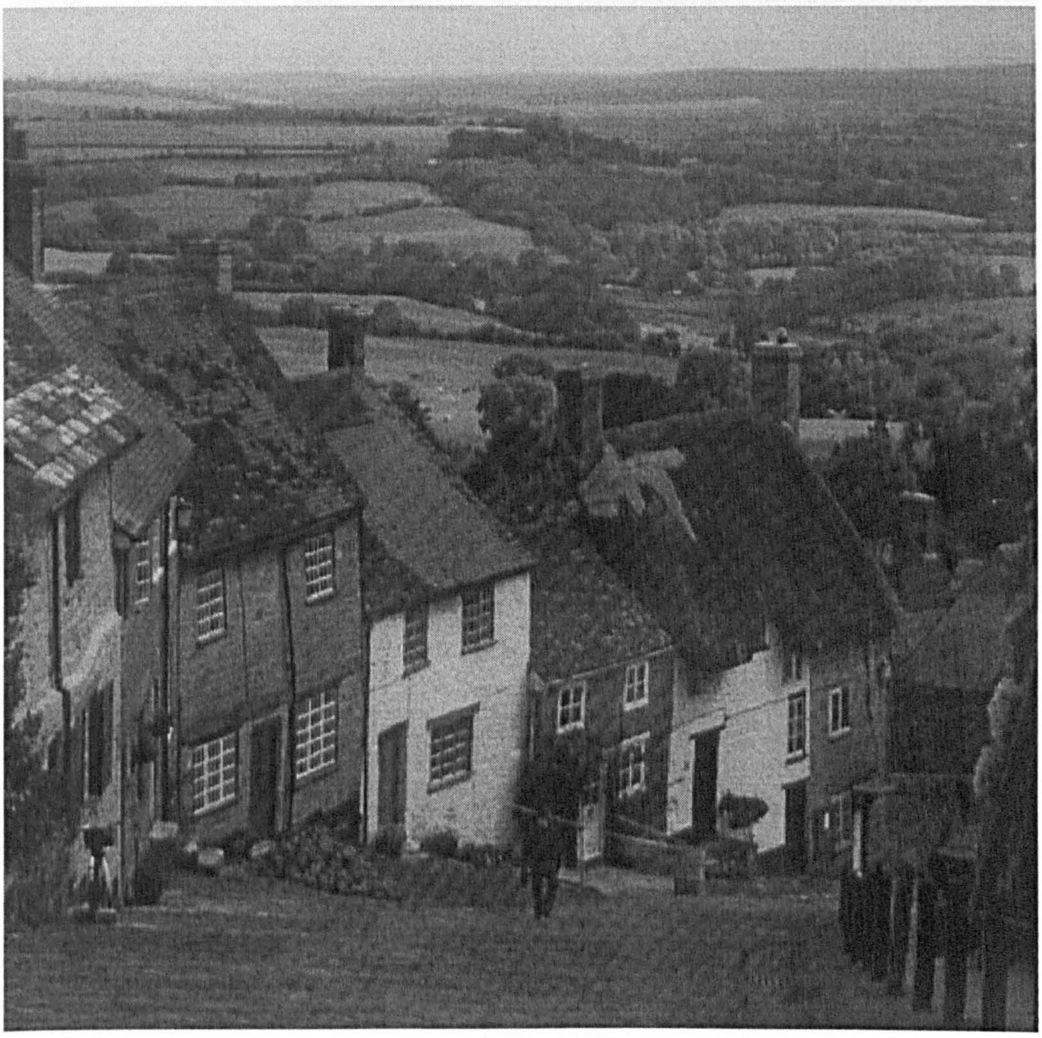


Figure 3.26 Goldhill

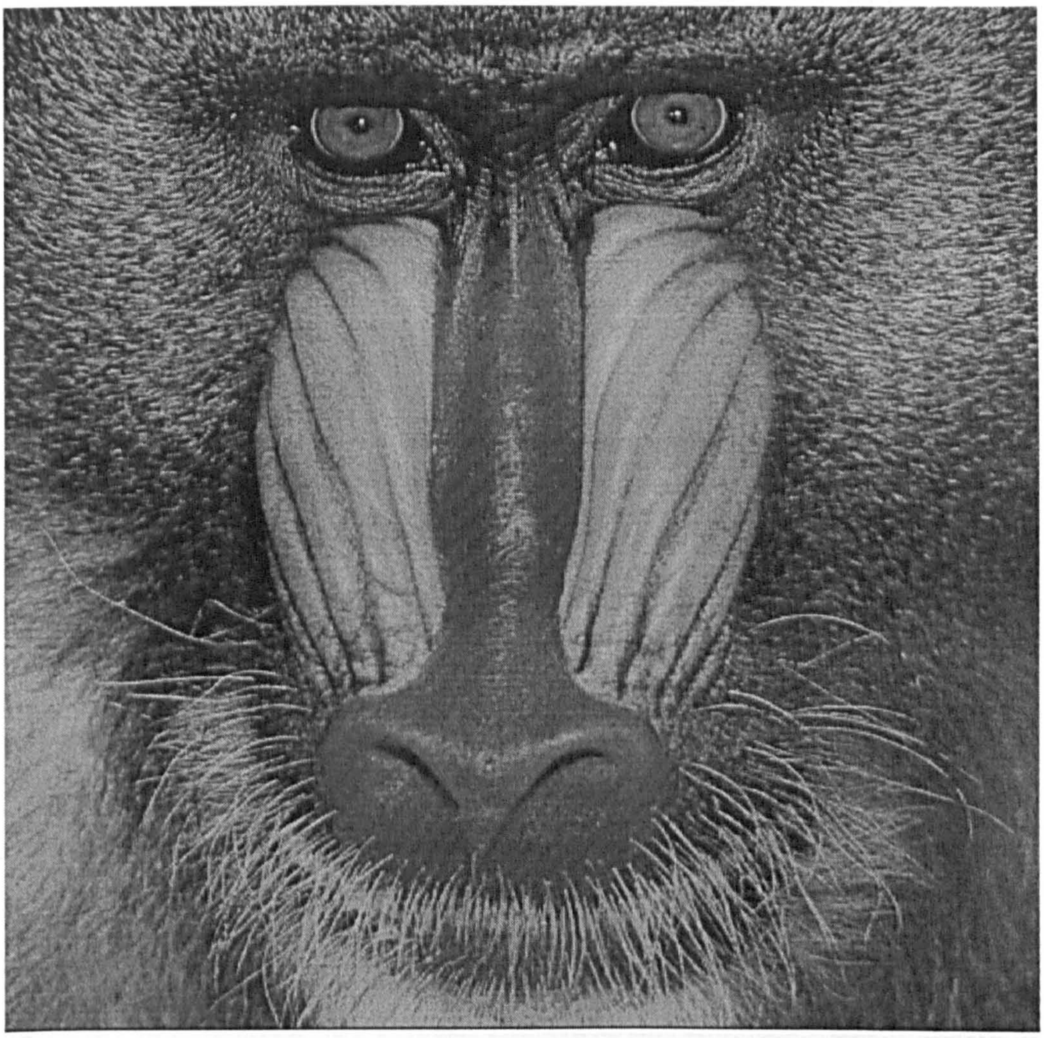


Figure 3.27 Mandrill

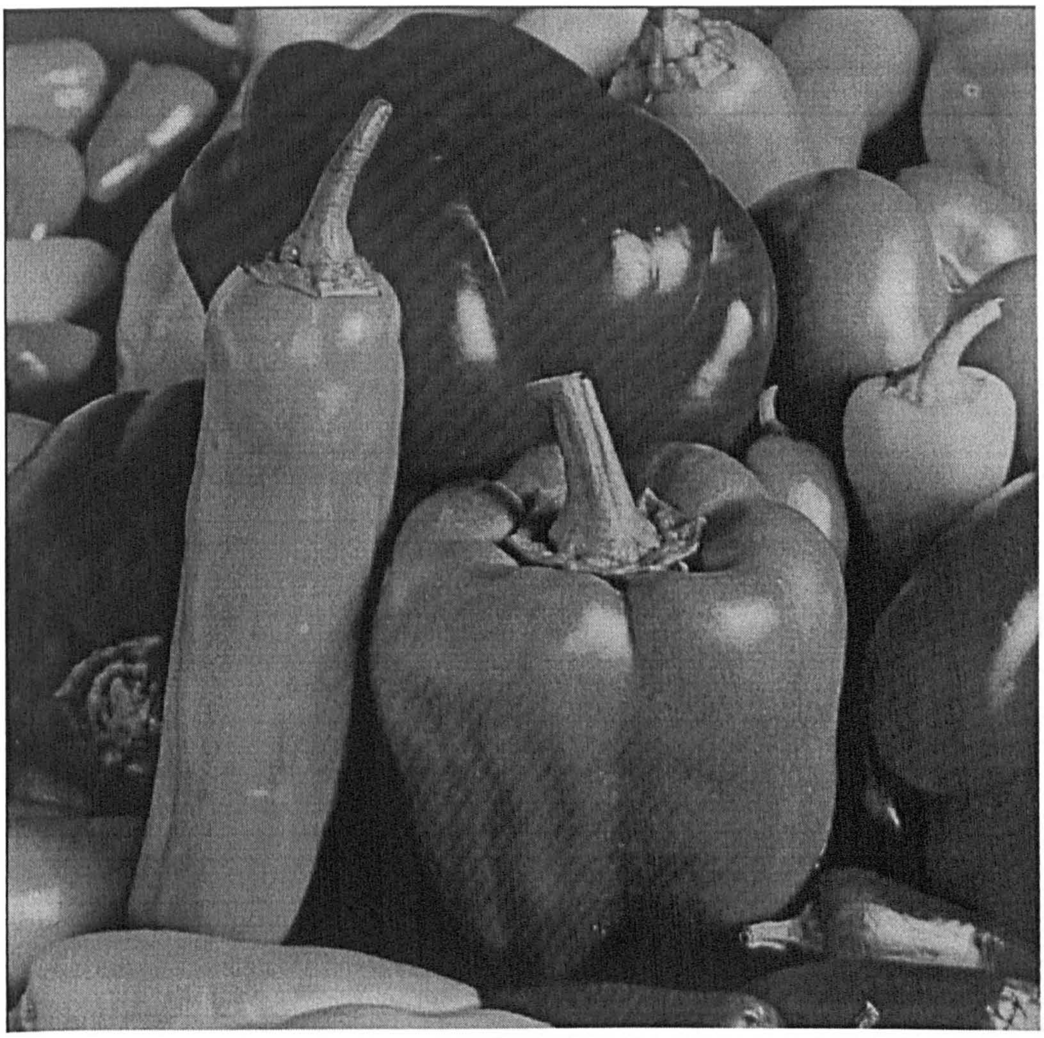


Figure 3.28 Peppers

Figure 3.29 Zelda

The distribution of the WCs on LL (after DC-level shifting) is shown in Figure 3.30, and Figure 3.31 shows the distribution of other WCs. As can be seen, the WCs on LL spread around, while other WCs gather round 0. For the WCs on HL, LH and HH, as the division of their values (except 0 and ±1) goes close towards 0, the width (range of values) of the division is halved, but the total number of WCs in the division increases. This statistical property explains the advantage of bit-plane coding of the WCs. In the successful transform coding algorithms, e.g., EZW, SPIHT, and EBCOT, the WCs are all encoded bit-plane by bit-plane, from the most significant bits (MSB) down to the least significant bits (LSB).

The mean value of the WCs in each division is listed in Table 3.5. It can be seen that the magnitude of the mean value is always lower than the magnitude of the central value of the division (with only one exception: division –10 of Lena). The difference between the two magnitudes increases as the range (width) of the division increases. Our work will exploit this property to increase the compression of natural images.

Table 3.6 lists the typical value of intra-subband, inter-subband (at the same level) and inter-level correlation coefficients. The average value for these images is 0.32, 0.022 and 0.10 respectively. Compared with the maximum value of 1, the intra-subband correlation is very high, while the inter-subband correlation is relatively low. This suggests that it could be most fruitful to take full advantage of the intra-subband for compression in image coding. Some more compression could still be obtained to exploit the inter-subband correlation at the same level, although it is the worst among the three. The inter-level correlation is always in the middle, between the intra-subband and the inter-subband correlation, but relatively close to the intra-subband and far away from inter-subband, which implies that it is also very important for image compression.

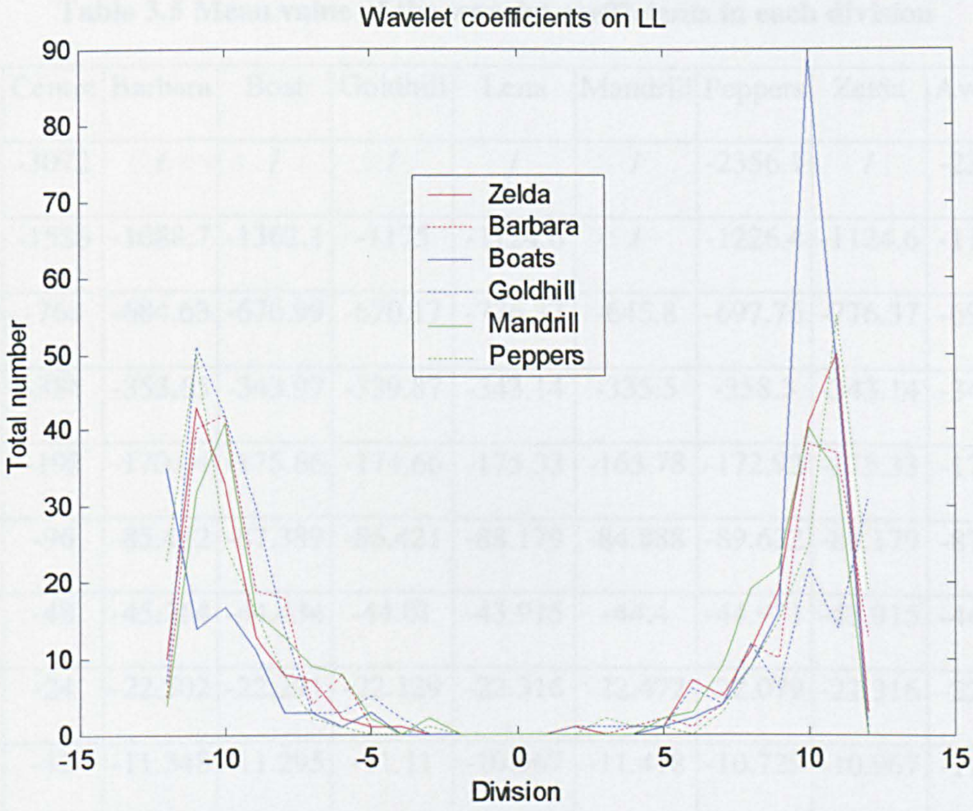


Figure 3.30 Distribution of wavelet coefficients on LL

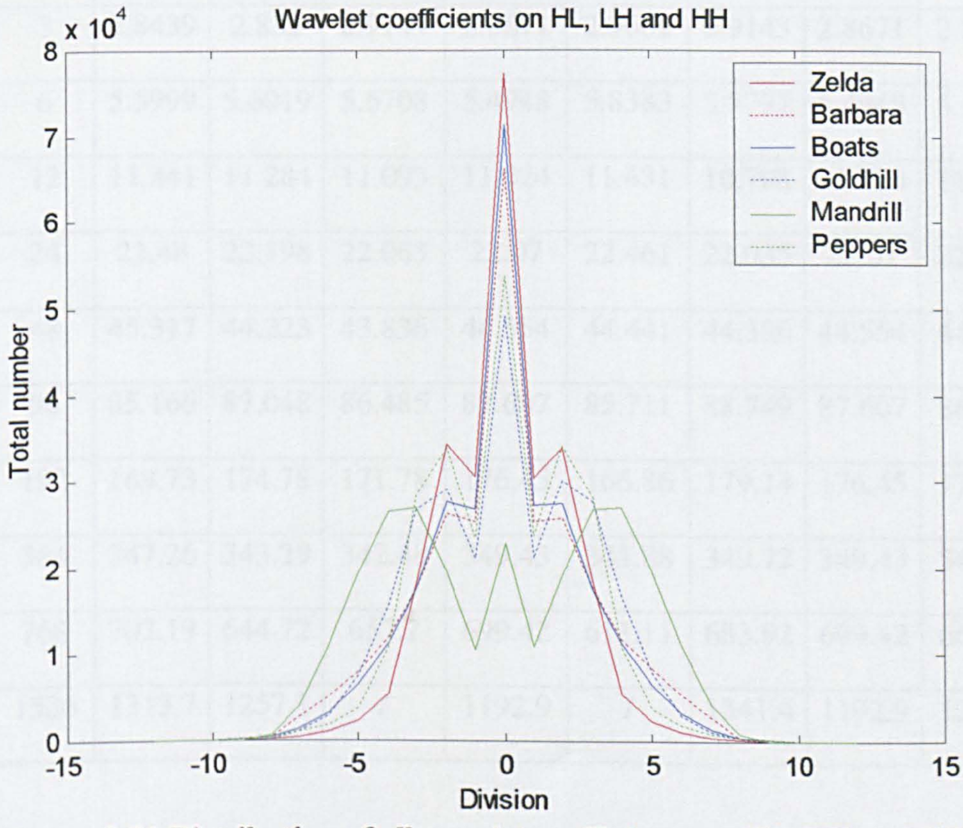


Figure 3.31 Distribution of all wavelet coefficients on HL, LH and HH

Table 3.5 Mean value of the wavelet coefficients in each division

Div	Centre	Barbara	Boat	Goldhill	Lena	Mandrill	Peppers	Zelda	Average
-12	-3072	/	/	/	/	/	-2356.1	/	-2356.1
-11	-1536	-1088.7	-1362.1	-1175	-1124.6	/	-1226.4	-1124.6	-1173.7
-10	-768	-684.63	-676.99	-670.17	-776.37	-645.8	-697.76	-776.37	-692.69
-9	-384	-353.65	-343.97	-339.87	-343.14	-335.5	-358.3	-343.14	-347.34
-8	-192	-170.04	-175.66	-174.66	-175.33	-163.78	-172.92	-175.33	-172.32
-7	-96	-85.412	-87.389	-86.421	-88.179	-84.888	-89.622	-88.179	-87.225
-6	-48	-45.264	-44.434	-44.01	-43.915	-44.4	-44.971	-43.915	-44.433
-5	-24	-22.502	-22.291	-22.129	-22.316	-22.477	-22.079	-22.316	-22.252
-4	-12	-11.348	-11.295	-11.11	-10.967	-11.418	-10.729	-10.967	-11.108
-3	-6	-5.6158	-5.6257	-5.6721	-5.4905	-5.8274	-5.5733	-5.4905	-5.5999
-2	-3	-2.8503	-2.8354	-2.907	-2.8692	-2.9716	-2.9074	-2.8692	-2.8836
2	3	2.8439	2.832	2.9144	2.8671	2.9662	2.9143	2.8671	2.8838
3	6	5.5999	5.6019	5.6708	5.4948	5.8383	5.5797	5.4948	5.5974
4	12	11.341	11.284	11.093	11.024	11.431	10.708	11.024	11.117
5	24	22.48	22.198	22.065	22.07	22.461	22.035	22.07	22.207
6	48	45.317	44.223	43.836	44.554	44.441	44.326	44.554	44.429
7	96	85.166	87.048	86.485	87.607	85.711	88.749	87.607	86.983
8	192	169.73	174.78	171.78	176.45	166.86	179.14	176.45	172.57
9	384	347.26	343.29	342.48	349.43	341.88	349.72	349.43	345.77
10	768	702.19	644.72	657.2	699.42	611.11	683.92	699.42	667.57
11	1536	1313.7	1257.1	/	1192.9	/	1341.4	1192.9	1244.6

Table 3.6 Typical correlation coefficients

	Intra-subband	Inter-subband	Inter-level
Barbara	0.36	0.035	0.11
Boat	0.42	0.026	0.097
Goldhill	0.35	0.012	0.086
Lena	0.30	0.029	0.13
Mandrill	0.19	0.015	0.063
Peppers	0.31	0.013	0.15
Zelda	0.34	0.027	0.098
Average	0.32	0.022	0.10

In fact, the zerotree technique used in EZW and SPIHT takes advantage of the inter-level correlation, and exploits intra-subband correlation to some limited extent. EBCOT exploits mainly the intra-subband correlation, more thoroughly and flexibly than the zerotree technique. The performance of EBCOT is similar to or a little better than that of EZW and SPIHT, and was adopted as the basic encoding engine of JPEG2000 [4]. This result reflects the relation of intra-subband and inter-level correlation. If we find a way to exploit intra-subband correlation more flexibly in SPIHT, the performance could be improved.

References

[1] J.M.Shapiro, 'Embedded Image Coding Using Zerotrees of Wavelet Coefficients', IEEE Transactions on Signal Processing, Vol.41, No.12, pp.3445-62, December 1993.

[2] A.Said and W.A.Pearlman, 'A new, fast, and efficient image codec based on set partitioning in hierarchical trees', IEEE Transactions on Circuits and Systems for Video Technology, Vol.6, No.3, pp.243-50, June 1996.

[3] D.Taubman, 'High performance scalable image compression with EBCOT', IEEE Transactions on Image Processing, Vol.9, No.7, pp.1158-70, July 2000.

[4] A.Skodras, C.Christopoulos and T.Ebrahimi, 'The JPEG2000 Still Image Compression Standard', IEEE Signal Processing Magazine, Vol.18, No.5, pp.36-58, September 2001.

Chapter 4

SPIHT Image Coding

The SPIHT (set partitioning in hierarchical trees) algorithm for image coding was introduced by A.Said and W.A.Pearlman [1]. It is a fine-tuned version of EZW (embedded zerotree wavelet) coding presented by J.M.Shapiro [2]. Both SPIHT and EZW are based on the zerotree technique, but the performance of SPIHT is much better than EZW. We review the SPIHT algorithm here, along with the zerotree technique.

The SPIHT image coding system is made up of three main parts: wavelet transform (WT), SPIHT coding and entropy coding, as indicated in Figure 4.1. The WT maps image pixels to wavelet coefficients (WC), as addressed in chapter 2. The SPIHT coding is the core of SPIHT image coding system, including the organisation of WCs, quantisation and ordered bit-plane coding. The resulting bit-stream can be compressed further by entropy coding, but this is optional. Arithmetic coding is used for the entropy coding in the SPIHT image coding system. We discuss the SPIHT coding and the arithmetic coding in detail in this chapter.

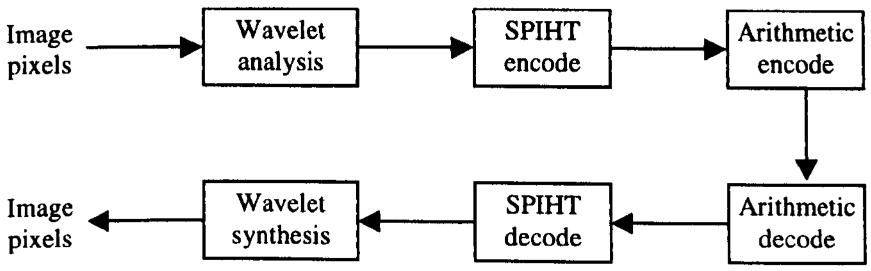


Figure 4.1 SPIHT image coding system

This chapter is organised as follows. Section 4.1 and 4.2 introduce organisation of WCs and quantisation in SPIHT coding. Section 4.3 explains the ordered bit-plane coding. The procedure of SPIHT coding is presented in section 4.4, and arithmetic coding in section 4.5. Section 4.7 summarises the chapter.

4.1 Organisation of wavelet coefficients

The WT maps image pixels to WCs. In chapter 2, we discussed the arrangement of WCs in matrix form. For a K-scale WT and a $R_K \times C_K$ image, subband LL*0* is of size $R_0 \times C_0$, HL*n* is of size $R_n \times c_n$, LH*n* is of size $r_n \times C_n$, and HH*n* is of size $r_n \times c_n$, where $n = 0, \ldots, K\text{-}1$, and

$$\begin{aligned} R_n &= \lfloor (R_{n+1}+1)/2 \rfloor \\ C_n &= \lfloor (C_{n+1}+1)/2 \rfloor \end{aligned}$$

$$\begin{aligned} r_n &= \lfloor R_{n+1}/2 \rfloor \\ c_n &= \lfloor C_{n+1}/2 \rfloor \end{aligned}$$

If the horizontal and vertical length of the image are integer multiples of 2^K, we have $R_n = r_n$ and $C_n = c_n$. For HL, LH or HH, corresponding to every WC on resolution level n, there are four WCs on level $n+1$ which have the same spatial orientation, as shown in Figure 4.2.

If the horizontal or vertical length of the image is not an integer multiple of 2^K, some R_n and C_n might be different from r_n and c_n. In this case, zeros are padded after the last row/column of each subband if necessary, for the wavelet coefficients to fit into a structure for a $(2^K \times R_0) \times (2^K \times C_0)$ image, so that the number of rows / columns is the same for all subbands on each level. After padding, for HL, LH or HH, corresponding to every WC on resolution level n ($n = 0, 1, \ldots, K\text{-}2$), there are four WCs on level $n+1$ which have the same spatial orientation, as in Figure 4.2.

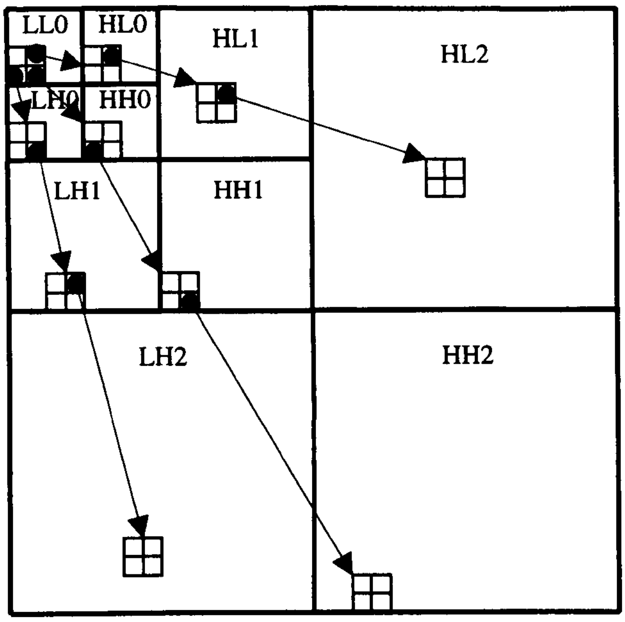


Figure 4.2 Correspondence of wavelet coefficients on different levels

Through padding (if needed), the organisation of WCs in the above two cases becomes identical. Suppose the size of subband LL*0* is $M_0 \times N_0$, then $R_n = r_n = M_0 \cdot 2^n$, $C_n = c_n = N_0 \cdot 2^n$, and the matrix of all WCs is of size $M \times N$, where $M = M_0 \cdot 2^K$ and $N = N_0 \cdot 2^K$. Denote ***W*** the matrix of WCs, and $w(i, j)$ its element, $0 \le i < M$ and $0 \le j < N$. The correspondence of WCs on neighbouring resolution levels which have same spatial orientation, shown in Figure 4.2, can be described as follows.

For $M_0 \le i < M/2$ or $N_0 \le j < N/2$, $w(i, j)$ is on subband HL*n*, LH*n* or HH*n* $(0 \le n < K-1)$. $w(2i, 2j)$, $w(2i, 2j+1)$, $w(2i+1, 2j)$ and $w(2i+1, 2j+1)$ are the four corresponding WCs in resolution level $n+1$. The four WCs, as a set, are denoted as $O(i, j)$.

Every element in $O(i, j)$, if it is not on resolution level $K-1$, can have four corresponding WCs of its own on the next finer resolution. Starting from $w(i, j)$, the correspondence can be iterated till resolution level $K-1$. All descendant WCs of

$w(i, j)$, as a set, are denoted as $D(i, j)$. Here is a tree structure, a hierarchical tree. Every node corresponds to a WC. The root of tree is at (i, j). The leaf nodes are on the finest resolution – level $K-1$. Every node, except the leaf nodes, has four branches, which lead to four direct descendants or offspring — $O(i, j)$. $D(i, j)$ is full as a tree, every node is occupied by a WC, except the root. The total number of resolution levels covered by a tree is called the depth of the tree. Normally, $O(i, j)$ is a subset of $D(i, j)$, its depth is 1. If (i, j) is on resolution level $K-2$, $O(i, j) = D(i, j)$.

Till now, the root of tree can only be on HL, LH and HH. It is extended to LL0 in the following. The WCs on LL0 can be denoted by $w(i, j)$, where $0 \leq i < M_0$ and $0 \leq j < N_0$. These WCs are grouped in four, as shown in Figure 4.2. A group is made up of $w(2i, 2j)$, $w(2i, 2j+1)$, $w(2i+1, 2j)$ and $w(2i+1, 2j+1)$, where $0 \leq i < M_0/2$ and $0 \leq j < N_0/2$.

HL0, LH0 and HH0 are of the same size as LL0. Their WCs are also grouped in four, the same as the WCs on LL0, shown in Figure 4.2. The four WCs in a group are $w(m_0+2i, n_0+2j)$, $w(m_0+2i, n_0+2j+1)$, $w(m_0+2i+1, n_0+2j)$ and $w(m_0+2i+1, n_0+2j+1)$, where $0 \leq i < M_0/2$, $0 \leq j < N_0/2$, and (m_0, n_0) is the index of the first WC at the top-left corner in relevant subband. For HL0, $m_0 = 0$, $n_0 = N_0$. For LH0, $m_0 = M_0$, $n_0 = 0$. For HH0, $m_0 = M_0$, $n_0 = N_0$.

Corresponding to each group on LL0, there is a group on HL0, LH0 and HH0, which has the same spatial orientation. The group of WCs on HL0, as a set, is defined as $O(2i, 2j+1)$, the group on LH0 is defined as $O(2i+1, 2j)$, and the group on HH0 as $O(2i+1, 2j+1)$. The roots of the three sets lie in the corresponding group on LL0, as shown in Figure 4.2. $D(2i, 2j+1)$, $D(2i+1, 2j)$ and $D(2i+1, 2j+1)$ are also defined accordingly. Note that the top-left WC at $(2i, 2j)$ in a group on LL0 does not have any descendant.

If M_0 or N_0 is odd, the last row or column of WCs on LL*0* cannot be grouped. In other words, these WCs do not have any descendant.

Another type of tree, $L(i, j)$, is also defined, as follows:

$$L(i, j) = D(i, j) - O(i, j)$$

$O(i, j)$, $D(i, j)$ or $L(i, j)$ refers to a specific tree with root at (i, j). Without index, O, D or L refers to the type of trees. D and L are two key type of trees to organise WCs on HL, LH and HH in the SPIHT coding. They are hierarchical trees in structure.

4.2 Successive approximation quantisation

Successive approximation quantisation is used in SPIHT image coding.

Denote a wavelet coefficient as x. For a given threshold T, x is significant if $/x/ \geq T$; otherwise, x is insignificant.

For $-2T < x < 2T$, x is quantised to y, as follows:

If $-2T < x \leq -T$ (significant), $y = -3T/2$;

If $-T < x < T$ (insignificant), $y = 0$;

If $T \leq x < 2T$ (significant), $y = 3T/2$;

For significant x, $/x/ - T$ is defined as the residue. It is always positive, no sign is needed.

The quantisation is applied to all WCs. Then the threshold is reduced by half and becomes $T/2$. The previous insignificant WCs ($-T < x < T$) will be quantised using the new threshold. The residue of the previous significant WCs will also be quantised using the new threshold, giving new quantisation results together with the previous quantisation.

The procedure repeats until the coding ends. As the threshold T becomes smaller, the quantisation becomes finer: the maximum quantisation error ($= T/2$) becomes smaller, and y approximates x more and more accurately. From this point of view, it is a successive approximation quantisation.

4.3 Ordered bit-plane coding

A binary symbol is coded every time the magnitude (or residue) of a WC is compared with the threshold during quantisation. If the WC (or residue) is significant, a binary symbol '1' is produced; if the WC (or residue) is insignificant, a binary symbol '0' is produced. The binary symbol indicates the significance of the WC (or residue). For a WC, another binary symbol is produced following the significance symbol, indicating the sign. A residue has no sign.

Normally, the thresholds used for quantisation are integer powers of 2: $T = 2^n$, n is an integer. In successive approximation quantisation, n decreases by 1 after each round.

We express the value of WCs with a sign bit and the magnitude in binary mode, and number the bits for magnitude from the least significant bit (LSB) up to the most significant bit (MSB) with integer starting from 0, as shown in Figure 4.3. Ignoring the sign bit, the coding with quantisation using threshold $T = 2^n$ is in fact getting the n-th bit of the magnitude. The coding with successive approximation quantisation is to get the magnitude bit by bit, from MSB to LSB. For the whole WCs, the coding is bit-plane by bit-plane, from MSB to LSB (shown in Figure 4.4). We call it ordered bit-plane coding.

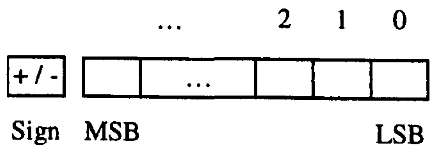


Figure 4.3 Value of wavelet coefficient in binary mode

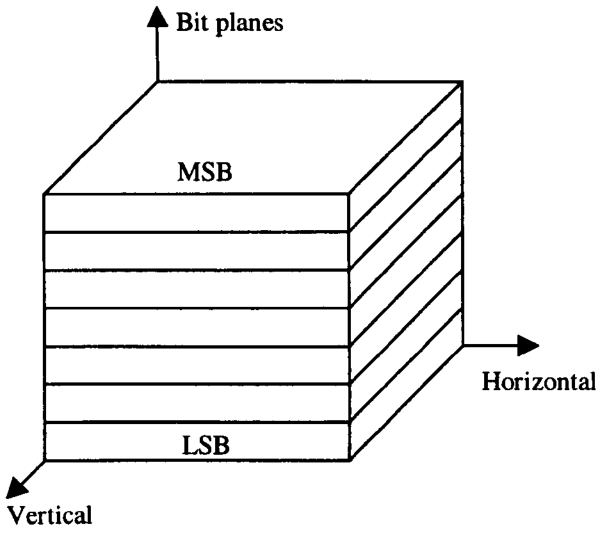


Figure 4.4 Bit-planes of wavelet coefficients

4.4 Procedure of SPIHT coding

As described in section 4.1, the WCs on HL, LH and HH are organised in hierarchical spatial-orientation trees. To apply the ordered bit-plane coding described in section 4.3, a tree is defined to be insignificant if all its WCs are insignificant; otherwise the tree is significant. In the SPIHT coding, an insignificant tree produces a binary symbol '0', and a significant tree produces a binary symbol '1'. If a tree is significant, it will be partitioned successively, resolution by resolution along the hierarchy towards the finest resolution, to reach the significant WCs. Individual WCs and/or subsets are produced during partitioning.

Three ordered lists are used in SPIHT coding, called list of insignificant sets (LIS), list of insignificant WCs (LIP, P for pixel – we keep the original name used in [1]), and list of significant WCs (LSP). The LIS lists all the trees which have not been found to be significant. There are two types of tree, *D* and *L*. Given the matrix of WCs, a tree in the LIS can be identified by its type and index of root. A WC in the

LIP or the LSP can be identified by its index provided that the matrix of WCs is known. For encoding, we can store the residues of significant WCs in the LSP instead of their index, as the residues are all what we need, and it is one value while the index contains two integers.

The three lists are scanned for coding, and each entry is tested to get its significance as in section 4.2. The order of WCs or trees in a list is important, it must be the same for encoding and decoding.

Now we introduce the procedure of SPIHT coding. It consists of four steps in order: initialisation, sorting pass, refinement pass, and quantisation-threshold update. They are discussed in the following paragraphs.

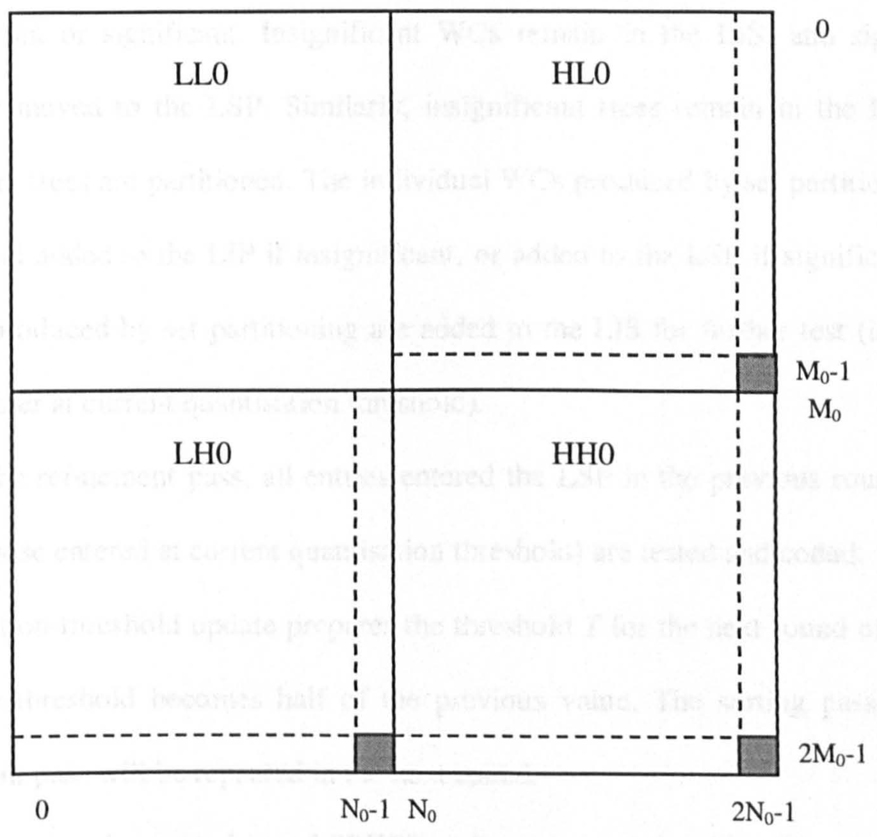


Figure 4.5 Last rows and columns of HL0, LH0 and HH0

During initialisation, the LIS, the LIP, the LSP and the quantisation threshold are initialised. All coordinates (index of the WCs) on LL*0* are added to the LIP, and those with descendants also to the LIS as trees of *D* type. If M_0 or N_0 is odd, neither the WCs in the last rows or columns on HL*0*, LH*0* and HH*0* (shown in Figure 4.5) nor their descendants are included in any tree with root on LL*0*, so their coordinates must be added to the LIP, and also to the LIS as trees of *D* type (for their desendants). The LSP is emptied. Suppose the maximum magnitude of all WCs (denoted as A) lie in the range $[2^n, 2^{n+1})$. That is, the total number of bit planes is $n = \lfloor log_2 A \rfloor$ Then the initial quantisation threshold is set to be $T = 2^n$.

The LIS and the LIP are processed in the sorting pass. The WCs or their sets are sorted into two categories according to the current quantisation threshold: insignificant or significant. Insignificant WCs remain in the LIS, and significant WCs are moved to the LSP. Similarly, insignificant trees remain in the LIS, and significant trees are partitioned. The individual WCs produced by set partitioning are tested, and added to the LIP if insignificant, or added to the LSP if significant. The subsets produced by set partitioning are added to the LIS for further test (including the test later at current quantisation threshold).

During the refinement pass, all entries entered the LSP in the previous rounds (i.e., except those entered at current quantisation threshold) are tested and coded.

Quantisation-threshold update prepares the threshold *T* for the next round of coding. The new threshold becomes half of the previous value. The sorting pass and the refinement pass will be repeated in the next round.

We summarise the procedure of SPIHT coding in the following. The significance value of WC *w(i, j)* is denoted as *S(i, j)*, and the significance value of tree *D(i, j)* and

$L(i, j)$ are denoted as $S(D(i, j))$ and $S(L(i, j))$ respectively. S is 1 (for the significant WC or tree) or 0 (for the insignificant WC or tree).

Procedure of SPIHT coding

(1) Initialisation:

LIP: (i, j), where i = $0, 1, \ldots, M_0\text{-}1$ and j = $0, 1, \ldots, N_0\text{-}1$ (for WCs on LL0);

$(M_0\text{-}1, j+N_0)$, $(2M_0\text{-}1, j)$ and $(2M_0\text{-}1, j+N_0)$, where j = $0, 1, \ldots, N_0\text{-}2$, if M_0 is odd (for last rows of HL0, LH0 and HH0, refer to Figure 4.5);

$(i, 2N_0\text{-}1)$, $(i+M_0, N_0\text{-}1)$ and $(i+M_0, 2N_0\text{-}1)$, where i = $0, 1, \ldots, M_0\text{-}2$, if N_0 is odd (for last columns of HL0, LH0 and HH0, refer to Figure 4.5);

$(M_0\text{-}1, 2N_0\text{-}1)$, $(2M_0\text{-}1, N_0\text{-}1)$ and $(2M_0\text{-}1, 2N_0\text{-}1)$, if M_0 or N_0 is odd (for the bottom-right corner of HL0, LH0 and HH0, refer to Figure 4.5).

LIS: $D(2i, 2j+1)$, $D(2i+1, 2j)$ and $D(2i+1, 2j+1)$, where i = $0, 1, \ldots, M_0/2\text{-}1$ and j =$0, 1, \ldots, N_0/2\text{-}1$ (for trees with roots on LL0);

$D(M_0\text{-}1, j+N_0)$, $D(2M_0\text{-}1, j)$ and $D(2M_0\text{-}1, j+N_0)$, where j = $0, 1, \ldots, N_0\text{-}2$, if M_0 is odd (for trees with roots in last rows of HL0, LH0 and HH0, refer to Figure 4.5);

$D(i, 2N_0\text{-}1)$, $D(i+M_0, N_0\text{-}1)$ and $D(i+M_0, 2N_0\text{-}1)$, where i = $0, 1, \ldots, M_0\text{-}2$, if N_0 is odd (for trees with roots in last columns of HL0, LH0 and HH0, refer to Figure 4.5);

$D(M_0\text{-}1, 2N_0\text{-}1)$, $D(2M_0\text{-}1, N_0\text{-}1)$ and $D(2M_0\text{-}1, 2N_0\text{-}1)$, if M_0 or N_0 is odd (for trees with roots at the bottom-right corner of HL0, LH0 and HH0, refer to Figure 4.5).

LSP: $\varnothing$.

$T = 2^n$, where $n = \lfloor \log_2 (\max_{(i,j)} (|w(i, j)|)) \rfloor$, $0 \leq i < M$ and $0 \leq j < N$.

```
(2) Sorting pass:
    (2.1) For each entry $(i, j)$ in the LIP do:
        (2.1.1) Output $S(i, j)$;
        (2.1.2) If $S(i, j) = 1$ then output the sign of $w(i, j)$, add $(i, j)$ to the LSP, and
                remove $w(i, j)$ from the LIP.
    (2.2) For each entry $(i, j)$ in the LIS do:
        (2.2.1) If the entry is of type $D$ then
            • Output $S(D(i, j))$;
            • If $S(D(i, j)) = 1$ then
                ♦ For each $w(k, l) \in O(i, j)$ do:
                    ▪ Output $S(k, l)$;
                    ▪ If $S(k, l) = 1$ then output the sign of $w(i, j)$, and add $(i, j)$ to the
                      LSP.
                    ▪ If $S(k, l) = 0$ then add $(i, j)$ to the LIP.
                ♦ If $(i, j)$ is not on the second finest resolution then add $L(i, j)$ to the
                  end of LIS.
                ♦ Remove $D(i, j)$ from the LIS.
        (2.2.2) If the entry is of type $L$ then
            • Output $S(L(i, j))$;
            • If $S(L(i, j)) = 1$ then
                ♦ For each $w(k, l) \in O(i, j)$, add $D(k, l)$ to the end of LIS;
                ♦ Remove $L(i, j)$ from the LIS.
(3) Refinement pass:
    For each entry $(i, j)$ in the LSP which entered before current round ($T > 2^n$),
    output the $n$-th bit of $w(i, j)$.
```

(4) Quantisation-threshold update:

Decrease n by 1 (halve T) and go to step (2).

In (2.2) of the procedure, "each entry in the LIS" includes those added to the end of the LIS at current quantisation threshold.

In the procedure, all branching conditions are based on the significance data S. S is calculated and output by the encoder, and is available to the decoder. If we replace all output by input, it becomes the procedure for decoding. Of course, some information such as image size, the number of WT scales, and the total number of bit planes of WCs, must be coded in the header and be available to the decoder in advance.

An additional task of the decoder is dequantisation – to map the binary code of WCs to an approximated value. All elements of W are initialised to 0. When (i, j) is moved to the LSP at quantisation threshold $T = 2^n$, it is known that $2^n \leq |w(i, j)| < 2^{n+1}$. So, the decoder set the magnitude of $w(i, j)$ to the middle value of the range, 1.5×2^n. The sign of $w(i, j)$ is also known from the input sign bit in the meanwhile. Similarly, during the refinement pass, the decoder adjusts the magnitude according to the input bit – n-th bit of $w(i, j)$: If the input bit is 0, subtract 2^{n-1} from the magnitude; otherwise, if the input bit is 1, add 2^{n-1} to the magnitude. In this manner, the distortion gradually decreases during both the sorting and refinement passes.

Due to the ordered bit-plane coding, the encoding and decoding process at a quantisation threshold reduces the largest distortion of the image pixels, measured by the mean square error (assuming the WT is unitary). The encoding and decoding can stop at any point and almost all the coding bits in the truncated bit-streams will contribute to the reconstructed image. This means the coding bit-streams are fully embedded.

4.5 A simple example

A simple example is used to demonstrate the procedure of SPIHT coding in this section. Consider the simple 2-scale wavelet transform of a 20×16 image. The array of wavelet coefficients is shown in Figure 4.6. The elements of the array are $a(i, j)$, where $0 \leq i \leq 20$, and $0 \leq j \leq 16$.

Since the largest coefficient magnitude is $/ a(3,2) / = 98$ (*1100010* in binary), there are seven bit-planes (*D0* to *D6*, where *D0* is LSB at the right end and *D6* is MSB at the left end), and the initial quantisation threshold is $T = 64$ (or 2^n, where $n = 6$).

The initial LIP is shown in Table 4.1. The order of the wavelet coefficients on LL0 in the LIP could be different with that in the table, but must be the same for encoding and decoding. In the table, they are grouped in four (2×2).

The initial LIS is shown in Table 4.2. An alternative order of trees is that the trees on HL with root at last row of HL0 succeed the trees on HL with root at LL0, and the same order applies to trees on LH and HH. The resulting order of trees is: No. 1 to 4, followed by No. 13 to 16, then No. 5 to 8 and No. 17 to 20, and then No. 9 to 12 and No. 21 to 24. Again the order must be the same for encoding and decoding.

In the sorting pass of SPIHT coding, the LIP is processed first. The entries in the LIP are scanned one by one in order. The first wavelet coefficient is $a(0, 0) = -14$. Its magnitude ($= 14$) is less than T ($= 64$ currently), so the encoder outputs a binary symbol 0 (for insignificant), and $a(0, 0)$ stays in the LIP. It is the same for the wavelet coefficients No. 2 to 14, and thirteen 0s are produced for them by the encoder.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-14	-52	-34	31	3	-7	0	1	1	2	0	-1	2	0	2	-2
1	-16	-54	-5	19	10	-4	-1	-11	-2	0	1	-2	0	-1	0	-2
2	-31	-45	9	31	13	3	-24	-9	1	0	1	2	-3	1	0	0
3	-32	-3	98	43	29	-15	-1	-14	3	0	2	-9	0	5	0	-1
4	5	-34	13	40	-9	-5	0	2	3	0	-2	2	-8	4	0	-3
5	0	-3	-8	0	-2	5	-8	-1	0	5	-6	0	0	0	0	-2
6	-7	1	9	4	-3	5	3	3	5	3	-6	0	-1	0	-2	-1
7	-16	-13	-13	3	-9	-12	-2	5	-3	0	3	-2	3	-4	2	0
8	1	21	0	-3	0	0	-5	-5	1	1	0	3	-4	-5	3	0
9	1	-34	-6	12	-2	-6	2	4	0	-2	1	0	0	-1	0	1
10	0	-1	0	1	3	2	1	0	1	0	0	1	-1	-1	0	0
11	0	-1	0	-4	-4	2	-2	0	1	0	0	4	0	2	0	0
12	0	0	-1	-8	2	-1	-4	-1	0	0	-1	-1	4	-1	0	1
13	-7	0	-1	3	-1	1	0	0	0	0	1	4	-3	0	0	0
14	8	0	5	20	0	0	-1	-1	0	0	1	-3	0	0	0	0
15	-1	0	-5	-5	2	0	-1	0	0	0	-1	0	0	0	1	0
16	-5	0	0	-5	-1	0	-3	0	-1	1	3	-2	2	0	0	0
17	-8	0	-1	11	0	0	0	1	5	0	0	1	0	-1	0	1
18	0	-6	3	-3	-4	3	0	-1	0	1	-4	3	-4	1	0	-1
19	0	3	-1	-7	1	1	0	0	0	-2	0	0	0	0	0	0

Figure 4.6 Example of 2-scale wavelet transform of a 20×16 image

Table 4.1 Initial LIP for the example image

No	Subband	Co-ordinates	Value	No	Subband	Co-ordinates	Value
1	LL0	*(0, 0)*	*-14*	**17**	LL0 (last row)	*(4, 0)*	*5*
2		*(0, 1)*	*-52*	**18**		*(4, 1)*	*-34*
3		*(1, 0)*	*-16*	**19**		*(4, 2)*	*13*
4		*(1, 1)*	*-54*	**20**		*(4, 3)*	*40*
5		*(0, 2)*	*-34*	**21**	Last row of HL0	*(4, 4)*	*-9*
6		*(0, 3)*	*31*	**22**		*(4, 5)*	*-5*
7		*(1, 2)*	*-5*	**23**		*(4, 6)*	*0*
8		*(1, 3)*	*19*	**24**		*(4, 7)*	*2*
9		*(2, 0)*	*-31*	**25**	Last row of LH0	*(9, 0)*	*1*
10		*(2, 1)*	*-45*	**26**		*(9, 1)*	*-34*
11		*(3, 0)*	*-32*	**27**		*(9, 2)*	*-6*
12		*(3, 1)*	*-3*	**28**		*(9, 3)*	*12*
13		*(2, 2)*	*9*	**29**	Last row of HH0	*(9, 4)*	*-2*
14		*(2, 3)*	*31*	**30**		*(9, 5)*	*-6*
15		*(3, 2)*	*98*	**31**		*(9, 6)*	*2*
16		*(3, 3)*	*43*	**32**		*(9, 7)*	*4*

Table 4.2 Initial LIS for the example image

No	Sub-band	Tree	Wavelet coefficients	No	Sub-band	Tree	Wavelet coefficients
1	HL (root at LL0)	*D(0, 1)*	*3,-7, 10,-4; 1, 2, 0,-1,-2, 0, 1,-2, 1, 0, 1, 2, 3, 0, 2,-9*	**13**	HL (root at last row of HL0)	*D(4, 4)*	*1, 1, 0,-2*
2		*D(0, 3)*	*0, 1,-1,-11; 2, 0, 2,-2, 0,-1, 0,-2,-3, 1, 0, 0, 0, 5, 0,-1*	**14**		*D(4, 5)*	*0, 3, 1, 0*
3		*D(2, 1)*	*13, 3, 29,-15; 3, 0,-2, 2, 0, 5,-6, 0, 5, 3,-6, 0,-3, 0, 3,-2*	**15**		*D(4, 6)*	*-4,-5, 0,-1*
4		*D(2, 3)*	*-24,-9,-1,-14; -8, 4, 0,-3, 0, 0, 0,-2,-1, 0,-2,-1, 3,-4, 2, 0*	**16**		*D(4, 7)*	*3, 0, 0, 1*
5	LH (root at LL0)	*D(1, 0)*	*0,-3,-7, 1; 0,-1, 0, 1, 0,-1, 0,-4, 0, 0,-1,-8,-7, 0,-1, 3*	**17**	LH (root at last row of LH0)	*D(9,0)*	*0.-6, 0, 3*
6		*D(1, 2)*	*-8, 0, 9, 4; 3, 2, 1, 0,-4, 2,-2, 0, 2,-1,-4,-1,-1, 1, 0, 0*	**18**		*D(9, 1)*	*3,-3,-1,-7*
7		*D(3, 0)*	*-16,-13, 1,21; 8, 0, 5,20,-1, 0,-5,-5,-5, 0, 0,-5,-8,0,-1, 11*	**19**		*D(9, 2)*	*-4, 3, 1, 1*
8		*D(3, 2)*	*-13, 3, 0,-3; 0, 0,-1,-1, 2, 0, -1, 0,-1, 0,-3, 0, 0, 0, 0, 1*	**20**		*D(9, 3)*	*0,-1, 0, 0*
9	HH (root at LL0)	*D(1, 1)*	*-2, 5,-3, 5; 1, 0, 0, 1, 1, 0, 0, 4, 0, 0,-1,-1, 0, 0, 1, 4*	**21**	HH (root at last row of HH0)	*D(9, 4)*	*0, 1, 0,-2*
10		*D(1, 3)*	*-8,-1, 3, 3; -1,-1, 0, 0, 0, 2, 0, 0, 4,-1, 0, 1,-3, 0, 0, 0*	**22**		*D(9, 5)*	*-4, 3, 0, 0*
11		*D(3, 1)*	*-9,-12, 0, 0; 0, 0, 1,-3, 0, 0, -1, 0,-1, 1, 3,-2, 5, 0, 0, 1*	**23**		*D(9, 6)*	*-4, 1, 0, 0*
12		*D(3, 3)*	*-2, 5,-5,-5; 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0,-1, 0, 1*	**24**		*D(9, 7)*	*0,-1, 0, 0*

For the fifth wavelet coefficient, $a(3, 2) = 98$, the magnitude ($= 98$) is greater than T, so the encoder outputs a 1 (for significant), followed by the sign bit 0 (for +), and $a(3, 2)$ is moved to the LSP, whose residue is 34 now.

The wavelet coefficients No. 16 to 32 are all insignificant. They stay in the LIP, and seventeen 0s are added to the encoded bit-stream for them.

In summary, the encoded bit-stream for the wavelet coefficients in the LIP at $T = 64$ consists of fourteen 0s followed by a 1 and then eighteen 0s, which is 0000, 0000, 0000, 0010, 0000, 0000, 0000, 0000, 0.

Then the trees in the LIS are processed. The entries in the LIS are scanned one by one in order. For any of the initial trees in the LIS, the maximum magnitude is less than T, so the trees are all insignificant. The trees stay in the LIS, and the encoder outputs a 0 for every tree. Twenty-four 0s are added to the encoded bit-stream.

In the refinement pass at $T = 64$, since the initial LSP is empty, the encoder does nothing.

That is the end of the first round of SPIHT coding ($T = 64$). The length of the encoded bit-stream is fifty-seven bits. All the binary symbols in the encoded bit-stream are 0 except the fifteenth.

Now the LIP is the same as that in Table 4.1 except that the entry No. 15 is removed. The LIS is exactly the same as that in Table 4.2. The LSP has one entry, $a(3, 2)$, whose residue is 34.

Then the quantisation threshold is reduced by half, $T = 32$, and $n = 5$. The sorting pass and the refinement pass are repeated at the new quantisation threshold in the second round.

The sorting pass in the second round is similar to that in the first round. The encoded bit-stream for the LIP is 0110, 1111, 0000, 1111, 0001, 0011, 0100, 0000, 1100, 0000, which consists of forty binary symbols. Nine entries are moved from the LIP to the LSP. Then twenty-four 0s are added to the encoded bit-stream for the twenty-four trees in the LIS which are all insignificant at $T = 32$.

To demonstrate the processing of a wavelet coefficient whose value is negative, the second entry in the LIP, $a(0,1) = -52$, is picked up as an example to be described here. It is significant at current quantisation threshold $T = 32$. The encoder outputs a 1 (for significant), followed by a 1 for the sign (-), and moves $a(0,1)$ to the LSP (its residue is 20).

The only entry in the LSP before the second round is $a(3, 2) = 98$, whose residue is 34 (or 10,0010 in binary). During the refinement pass of the second round, the encoder outputs a 1, which is $D5$ of the magnitude/residue of $a(3, 2)$. The new residue for $a(3,2)$ is 2 (or 0,0010 in binary).

That is the end of the second round. Sixty-five symbols are added to the output bit-stream. Now there are twenty-two wavelet coefficients in the LIP, twenty-four trees in the LIS, and ten entries in the LSP.

In the third round, $T = 16$, and $n = 4$. The encoded bit-stream for the LIP is 0111, 0010, 1100, 1000, 0000, 0000, 000, which consists of twenty-seven binary symbols. Five entries are moved from the LIP to the LSP. The processing of the LIS is illustrated in Table 4.3. Steps 3 and 27 are described in the following paragraphs to demonstrate the partitioning of D and L trees respectively.

Table 4.3 Processing of the LIS in the third round

Step	Trees to be processed	Output symbols	Trees staying in the LIS	Wavelet coefficients added to the LIP	Residues added to the LSP
1	*D(0, 1)*	0	*D(0, 1)*		
2	*D(0, 3)*	0	*D(0, 3)*		
3	*D(2, 1)*	100100		*a(2,4),a(2,5),a(3,5)*	*a(3,4)*
4	*D(2, 3)*	111000		*a(2,7),a(3,6),a(3,7)*	*a(2,6)*
5	*D(1, 0)*	0	*D(1, 0)*		
6	*D(1, 2)*	0	*D(1, 2)*		
7	*D(3, 0)*	1110010		*a(7,1),a(8,0)*	*a(7,0), a(8,1)*
8	*D(3, 2)*	0	*D(3, 2)*		
9	*D(1, 1)*	0	*D(1, 1)*		
10	*D(1, 3)*	0	*D(1, 3)*		
11	*D(3, 1)*	0	*D(3, 1)*		
12	*D(3, 3)*	0	*D(3, 3)*		
13	*D(4, 4)*	0	*D(4, 4)*		
14	*D(4, 5)*	0	*D(4, 5)*		
15	*D(4, 6)*	0	*D(4, 6)*		
16	*D(4, 7)*	0	*D(4, 7)*		
17	*D(9,0)*	0	*D(9,0)*		
18	*D(9, 1)*	0	*D(9, 1)*		
19	*D(9, 2)*	0	*D(9, 2)*		
20	*D(9, 3)*	0	*D(9, 3)*		
21	*D(9, 4)*	0	*D(9, 4)*		
22	*D(9, 5)*	0	*D(9, 5)*		
23	*D(9, 6)*	0	*D(9, 6)*		
24	*D(9, 7)*	0	*D(9, 7)*		
25	*L(2,1)*	0	*L(2,1)*		
26	*L(2,3)*	0	*L(2,3)*		
27	*L(3,0)*	1			
28	*D(7,0)*	0	*D(7,0)*		
29	*D(7,1)*	101000		*a(14,2),a(15,2),a(15,3)*	*a(14,3)*
30	*D(8,0)*	0	*D(8,0)*		
31	*D(8,1)*	0	*D(8,1)*		

In Step 3, *D(2, 1)* are scanned and processed. Since the maximum magnitude of wavelet coefficients on *D(2, 1)* is 29, which is greater than *T* (= *16*), *D(2, 1)* is significant. The encoder outputs a 1 (for significant). Then *D(2, 1)* is partitioned to *O(2, 1)* and *L(2, 1)*. The four wavelet coefficients on *O(2, 1)*, including *a(2, 4)*, *a(2, 5)*, *a(3, 4)* and *a(3, 5)*, are scanned and processed in the order. The value of *a(2, 4)*, *a(2, 5)* and *a(3, 5)* are *13*, *3* and *-15* respectively. They are insignificant at *T* = *16*, so the encoder outputs three 0s (one for each). They are added to the end of the LIP. *a(3, 5)* (= *29*) is significant at *T* = *16*, so a 1 is added to the output bit-stream, followed by a 0 for the sign (+). *a(3, 5)* is added to the end of the LSP. *L(2, 1)* is added to the end of the LIS, which will be processed latter in the same (third) round at Step 25. *D(2, 1)* is removed from the LIS in the end. In summary, the output symbols here for *D(2, 1)* are 100100.

In Step 27, *L(3, 0)*, which was partitioned from *D(3, 0)* earlier in the same (third) round at Step 7, are processed. The maximum magnitude of wavelet coefficients on *L(3, 0)* is 20, which is greater than *T*, so *L(3, 0)* is significant. The encoder outputs a 1 (for significant), and partitions *L(3, 0)* to four subtrees – *D(7, 0)*, *D(7, 1)*, *D(8, 0)* and *D(8, 1)*. The four subtrees are then added to the end of the LIS, which will be processed latter in the same (third) round at steps 28 to 31. *L(3, 0)* is removed from the LIS.

In the refinement pass of the third round, the ten entries in the LSP at the beginning of the round are scanned one by one in order, and ten binary symbols (0110,0000,00) are added to the output bit-stream as the result. The other ten entries added to the LSP in the sorting pass of the same (third) round (five from the LIP and five from the LIS) are not processed in this (third) round.

By the end of the third round, there are 211 encoded binary symbols in the output bit-stream, which is summarised in Table 4.4.

Table 4.4 Output bit-stream in the first three rounds of SPIHT encoding

T	List	Total	No.	Symbols
64	LIP	33	1~33	0000, 0000, 0000, 0010, 0000, 0000, 0000, 0000, 0
	LIS	24	34~57	0000, 0000, 0000, 0000, 0000, 0000
	LSP	0		
32	LIP	40	58~97	0110, 1111, 0000, 1111, 0001, 0011, 0100, 0000, 1100, 0000
	LIS	24	98~121	0000, 0000, 0000, 0000, 0000, 0000
	LSP	1	122	1
16	LIP	27	123~149	0111, 0010, 1100, 1000, 0000, 0000, 000
	LIS	52	150~201	0010, 0100, 1110, 0000, 1110, 0100, 0000, 0000, 0000, 0000, 0010, 1010, 0000
	LSP	10	202~211	0110, 0000, 00

The procedure can be repeated again and again, till the desired coding rate or PSNR is reached. Only integer values are listed in Figure 4.6. In fact, any real value can be encoded by SPIHT coding. For the fractional part, $T = 2^n$, where $n < 0$.

4.6 Arithmetic coding

The output of the SPIHT encoding process is a binary bit-stream. It can be compressed further using entropy coding (or Shannon coding) based on the probabilities of symbols. The arithmetic coding algorithm by Witten et al. [3] is used for entropy coding here.

The simplest implementation is to compress the output bit-stream of SPIHT encoding using arithmetic coding with an adaptive model of two symbols for binary 1 and 0. It is more efficient if various models are used to exploit various distribution patterns of trees and individual WCs in the SPIHT coding.

We review briefly arithmetic coding using the adaptive model. Full details of the implementation can be found in [3]. The set of coding symbols, x_n ($n = 1, 2, ..., M$), are predefined and ordered. The frequency of each symbol is counted as they appear, and every oncoming symbol is encoded and decoded based on the established statistical distribution. The probability of each symbol, denoted as p_n, is calculated according to the frequency of symbols, and mapped onto a range (denoted as P_n) between 0 and 1 in order, as indicated in Figure 4.7. The boundaries of P_n are U_{n-1} and U_n, where $U_0 = 0$ and $U_n = \sum_{m=1}^{n} p_m$ for $n = 1, 2, ..., M$. The width of P_n is $U_n - U_{n-1} = p_n$.

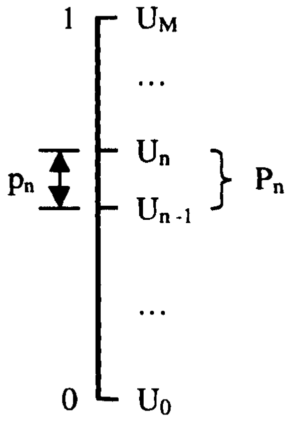


Figure 4.7 Models for the probability of symbols in arithmetic coding

Consider all the input symbols of the encoder as a whole, the probability q_n of the particular symbol series y_n (n is total number of symbols) is also mapped onto a range (denoted as Q_n) between 0 and 1 in order, as shown in Figure 4.8. The boundaries of Q_n are v_n and V_n. $Q_1 = P_m$ for $y_1 = \{x_m\}$. That is, $v_1 = U_{m-1}$ and $V_1 = U_m$. As a symbol x_m appears and is added to the series, $y_{n+1} = \{y_n, x_m\}$, the probability of the symbol series becomes $q_{n+1} = q_n \cdot p_m$, and its mapped range becomes Q_{n+1}, whose boundaries are $v_{n+1} = v_n + (V_n - v_n) \cdot U_{m-1}$ *and* $V_{n+1} = v_n + (V_n - v_n) \cdot U_m$. The width of Q_{n+1} is $V_{n+1} - v_{n+1} = q_{n+1} = q_n . p_m < q_n$. This means as n increases, the width of Q_n decreases thus V_n and v_n become closer. In binary, some more bits of v_n and V_n might become equal every time since a symbol is added, or some more bits of Q_n become fixed in other words. The encoder outputs these fixed bits and records the remaining vague range of Q_n for encoding of on-coming symbols.

The reverse is done in the decoder. The probability q_n and its mapped range Q_n are for the decoded symbol series. As more bits are input from the encoded bit-stream, Q_n is narrowed to Q_{n+1}. Q_{n+1} in Q_n is the range for a specific symbol, thus the symbol is decoded.

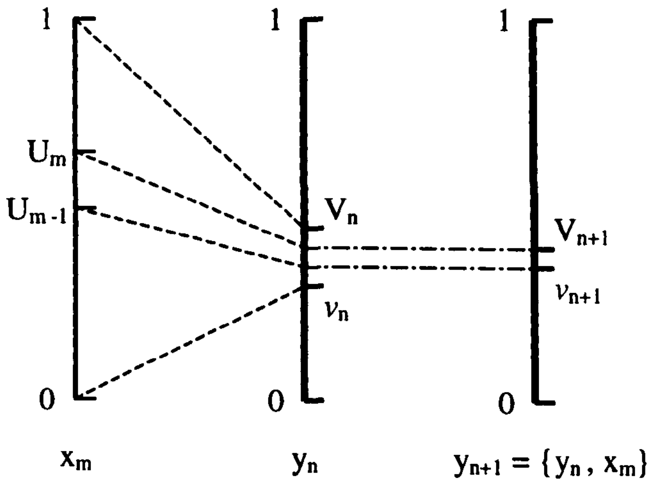


Figure 4.8 Probability of symbol series in arithmetic coding

For an adaptive model, the initial frequencies of all symbols should be same in the encoder and the decoder. Any predefined values can be used for the initial frequencies, but it is often set to 1 for each symbol, for quick adaptation in the beginning.

To avoid overflow, the frequency of each symbol is halved when the sum of frequencies for all symbols reaches a maximum value (called maximum frequency). The maximum frequency should also be same in the encoder and the decoder. This scheme has an extra advantage: more weight is put on the latest symbols for the adaptation of the probability distribution pattern.

The initial frequencies and the maximum frequency are two key parameters for an adaptive model.

Arithmetic coding can also use a fixed model. The procedure is the same as using an adaptive model, except that the probability of each symbol is fixed, thus it is not needed to count the frequency of symbols. Arithmetic coding using a fixed model is faster than that using an adaptive model.

4.6.1 Arithmetic coding models

In SPIHT coding, there are seven classes of binary symbols in the encoded bit-stream. They are the significance values and signs of WCs in the LIP, the significance values of D and L trees in the LIS, the significance values and signs of individual WCs partitioned from the D tree in LIS, and the significance values of WC residues in the LSP. Both the encoder and the decoder know the class of every symbol, so various models could be used in arithmetic coding, to track the distribution pattern of each class of symbols.

In [1], neighbouring WCs and *D* trees are grouped for arithmetic coding, to exploit their correlation. Initially, neighbouring WCs from LL in the LIP are grouped in four (2×2), and the three *D* trees with roots in a LIP group and descendants in HL, LH and HH respectively form a group in the LIS (refer to section 4.1 for grouping).

As significant WCs are moved out from the LIP during the sorting pass of SPIHT coding, the sizes of the groups they belong to are reduced to 3, 2 and 1, and then a group is removed from the LIP when all its four WCs become significant. The significance values of WCs in a group are coded as a single symbol in arithmetic coding. An adaptive model is used for each group of different sizes. It is similar for the groups of trees in the LIS.

On the other hand, more groups are produced during set partitioning. The four WCs of $O(m, n)$ partitioned from $D(m, n)$ are grouped together for arithmetic coding, and the four *D* subtrees partitioned from a *L* tree are also grouped together. This group of four *D* trees is unique due to its size. The group of four newly partitioned WCs can use either the same model as the group of four in LIP, or a different model of its own.

The *L* trees use an adaptive model of its own, with only two symbols for the significance values of trees.

Experiences show that little can be gained in using adaptive models for the significance values of WC residues in the LSP and the sign of WCs. So a simple fixed model with equal distribution between binary symbols 1 and 0 can be used for these two kinds of symbols in arithmetic coding.

4.7 Summary

We reviewed the procedure of SPIHT coding and arithmetic coding in this chapter. The organisation of wavelet coefficients in hierarchical trees as sets and the concept of successive approximation quantisation and ordered bit-plane coding were highlighted. Adaptive models used in arithmetic coding to compress the output of SPIHT coding were also discussed. The contents of this chapter are the basis of our research.

An example of a 20×16 image using 2-scale wavelet transform was presented to demonstrate the procedure of SPIHT coding. A simple solution to organise the wavelet coefficients of an arbitrarily sized image for SPIHT coding was discussed. These are not available in literature.

References

[1] A.Said and W.A.Pearlman, 'A new, fast, and efficient image codec based on set partitioning in hierarchical trees', IEEE Transactions on Circuits and Systems for Video Technology, Vol.6, No.3, pp.243-50, June 1996.

[2] J.M.Shapiro, 'Embedded Image Coding Using Zerotrees of Wavelet Coefficients', IEEE Transactions on Signal Processing, Vol.41, No.12, pp.3445-62, December 1993.

[3] I.H.Witten, R.M.Neal and J.G.Cleary, 'Arithmetic Coding for Data Compression', Communications of the ACM, Vol.30, No.6, pp.520-40, June 1987.

Chapter 5

Improvements to the SPIHT Algorithm

The SPIHT image coding [1] is very efficient, it outperforms the EZW (embedded zerotree wavelet) coding [2] and the image coding standard JPEG [3]. Though it can be improved and become more efficient. We present our improvements in this chapter. The DC-level shifting of image pixels or their wavelet coefficients (WC) is discussed in section 5.1. We introduce the new type of trees in section 5.2, propose a scheme to reduce the redundancy in the encoded symbols in section 5.3, and offset the quantisation in section 5.4. Section 5.5 addresses the optimisation of arithmetic coding, and section 5.6 summarises the chapter.

5.1 DC-level shifting

The original image data before coding is normally an array of unsigned values. The mean value of image pixels is positive. For example, the mean value of 8-bit grey image is around *127*. It is often inefficient to encode such non-negative data directly in zerotree-based transform coding [1][2]. So we shift the DC level before coding.

There are many schemes to shift the DC level. The first and simplest scheme is to deduct from the image pixels the middle value of the possible dynamic range. For example, use the value *127* for 8-bit grey image. Statistically, the mean value of the image pixels after the DC-level shift is close to zero, and thus won't have any major impact on the efficiency of coding.

The second scheme is to calculate the actual mean value and then deduct from the image pixels. The mean value needs to be coded together with the residue of image pixels. For simplicity, the mean value is rounded to its nearest integer.

The third scheme is to shift the DC level in the transform domain. As we know, the DC level will only affect the final LL subband in the transform domain. We can calculate the mean value of all WCs on LL, round to its nearest integer, and then deduct from the WCs on LL. This mean-value also need to be coded together with all WCs.

The difference between the three schemes is not very much. We use the first scheme in the thesis, except otherwise stated.

5.2 Introduce the virtual trees

Experiments show that at the beginning of SPIHT-encoded binary symbols, 0 is dominant, especially if the image is large in size and the number of wavelet transform (WT) scales is small. This can be explained by the statistical properties of wavelet coefficients (WC) described in chapter 3, as in the following.

The output symbols of the SPIHT encoder at the beginning are the coding results of the initial WCs in the LIP (list of insignificant WCs) and the initial sets in the LIS (list of insignificant sets). These initial WCs are from the LL subband, and the roots of those initial trees are on LL too.

Recall that the WCs on LL spread around over their range. Their typical distributions were shown in figure 3.29. We list the number of WCs on LL in the outermost divisions for each image in Table 5.1, together with their corresponding percentages in two hundred and fifty-six total WCs on LL. The WCs in the outermost divisions are a small ratio of total WCs on LL. In the first round of SPIHT coding, the quantisation threshold is $T = 2048$. These WCs are found to be significant and produce symbol 1 for significance, and the rest produce symbol 0. As the result, 0 is dominant in the encoded symbols for initial WCs in the LIP.

Table 5.1 Wavelet coefficients on LL in the outermost divisions

Image	Outermost divisions	Range of magnitude	Wavelet coefficients	
			Number	Percentage
Lena	±12	[2048, 4096)	28	11%
Barbara	±12	[2048, 4096)	18	7%
Boat	±12	[2048, 4096)	35	14%
Goldhill	±12	[2048, 4096)	38	15%
Mandrill	±12	[2048, 4096)	4	2%
Peppers	±12	[2048, 4096)	42	16%

Similarly, we list the WCs on HL, LH and HH in their outermost divisions in Table 5.2. There are sixty-four initial trees for HL, LH, or HH, 64 × 3 altogether in the LIS. The same initial quantisation threshold, $T = 2048$, is used for the initial WCs in the LIP and the initial trees in the LIS. In the first round of SPIHT coding, the encoded symbols for all initial trees are 0, except one or two trees on HL of image Peppers which include the two WCs in divisions ±12.

In the second round, the WCs in divisions ±11 become significant. Half of the outermost divisions listed in Table 5.2 are ±10. In these cases, there is no WC in divisions ±11, so the encoded symbols for the relevant trees are 0s. In other cases, the total number of WCs in divisions ±11 (9 for LH of Peppers) is small compared with the total number of trees on relevant subband in the LIS (64, maybe more for LH of Peppers). Even if each of these WCs belongs to a different tree, the significant trees are still of a small ratio, thus 0 is dominant in the encoded symbols.

Table 5.2 Number of wavelet coefficients (NWC) on HL, LH and HH in the outermost divisions (OMD)

Image	HL		LH		HH	
	OMD	NWC	OMD	NWC	OMD	NWC
Lena	±11	8	±10	7	±11	1
Barbara	±11	5	±10	25	±10	5
Boat	±11	2	±11	10	±10	2
Goldhill	±11	2	±10	14	±10	2
Mandrill	±10	17	±10	11	±10	1
Peppers	±11	16	±12	2	±11	2

We know the binary coding is not efficient if one symbol (0 or 1) is much more in number than the other. We can take advantage of the unbalanced distribution of symbols for compression in several ways, such as run length coding. Another way is to set the initial frequencies of adaptive models to match the distribution in arithmetic coding. But the extremely unbalanced initial frequencies can not easily adapt to the distribution of oncoming symbols later without causing any coding inefficiency, and this method does not benefit the SPIHT coding without arithmetic coding.

We work out a solution by introducing the virtual trees. To define the virtual trees, we extend *D(i, j)* to $D_n(i, j)$. We observe the following facts of WCs and *D(i, j)*. For

K-scales WT, the resolutions of WCs are from level 0 to level $K-1$. The root of $D(i, j)$ is at (i, j), and its leaves are on the finest resolution – level $K-1$. Suppose (i, j) is on level M $(M < K-1)$, then the depth of $D(i, j)$ is $K-1-M$. We rewrite $D(i, j)$ as $D_{K-1-M}(i, j)$, including the depth information in its denotation. Similarly, for a generic depth, $D_n(i, j)$ is the same as $D(i, j)$, except that its leaves are on level $M+n$ $(M+n < K)$. In other words, $D_n(i, j)$ is a subset of $D(i, j)$, with root at (i, j) and depth of n. In particular, $D_0(i, j) = \varnothing$.

Figure 5.1 shows an example of the WCs of a 64×64 image after a 4-scale WT. There are four resolutions: level 0 to level 3. $D(2, 5)$ includes four WCs on HL1, sixteen WCs on HL2, and thirty-two leaves on HL3. Its depth is 3, so we also write it as $D_3(2, 5)$. $D_2(2, 5)$ has the four WCs on HL1, and the sixteen WCs on HL2 which are leaves. $D_1(2, 5)$ has only the four WCs on HL1, which are leaves. $D_2(2, 5)$ and $D_1(2, 5)$ are subsets of $D(2, 5)$.

We also define $G_{m,n}(i, j) = D_m(i, j) - D_n(i, j)$, where $n \leq m$. $G_{m,n}(i, j)$ is a general form for trees, with root at (i, j) on level M and depth of $m-n$. The WCs of $G_{m,n}(i, j)$ are on level $M+n+1$ to $M+m$. As examples, in Figure 5.1, $G_{3,2}(2, 5)$ includes the sixty-four WCs on HL3, $G_{3,1}(2, 5)$ includes the sixteen WCs on HL2 and the sixty-four WCs on HL3, and $G_{2,1}(2, 5)$ includes the sixteen WCs on HL2.

All type of trees can be written in $G_{m,n}(i, j)$:

$$D(i, j) = G_{K-M-1,0}(i, j) = D_{K-M-1}(i, j)$$

$$O(i, j) = G_{1,0}(i, j) = D_1(i, j)$$

$$L(i, j) = G_{K-M-1,1}(i, j) = D_{K-M-1}(i, j) - D_1(i, j)$$

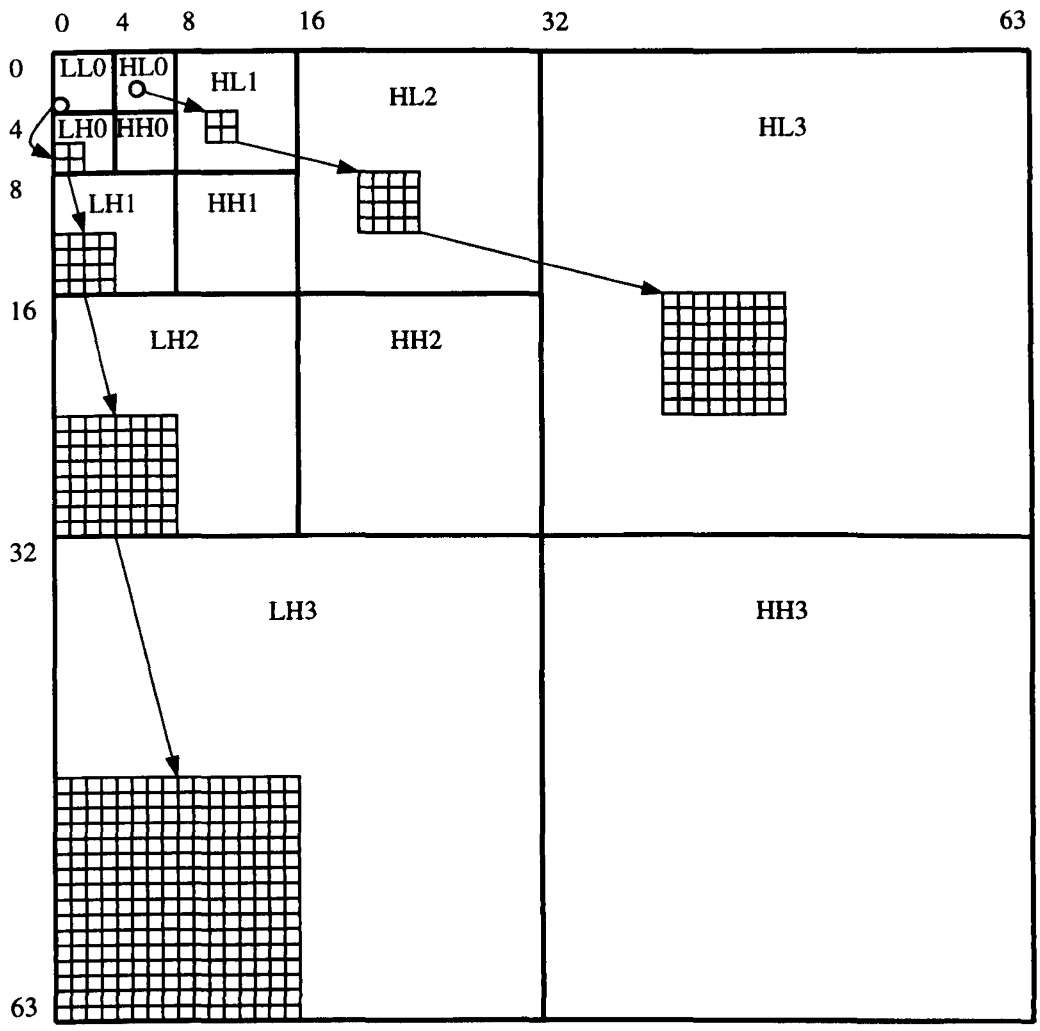


Figure 5.1 Wavelet coefficients of 64×64 image after 4-scale wavelet transform

For example, in Figure 5.1, $D(2, 5) = G_{3,0}(2, 5)$, $O(2, 5) = G_{1,0}(2, 5)$, and $L(2, 5) = G_{3,1}(2, 5)$.

The initial trees in the LIS, with roots on LL, are of D type: $D(i, j) = G_{K,0}(i, j)$. Their roots are defined specially. In Figure 5.1, $D(3, 0)$ is such an example, which includes the four WCs on LH0, the sixteen WCs on LH1, the sixty-four WCs on LH2, and the two hundred and fifty-six WCs on LH3. According to the general rules of other trees, these roots are on resolution level -1.

Now we define a virtual tree using $G_{m,n}(i, j)$:

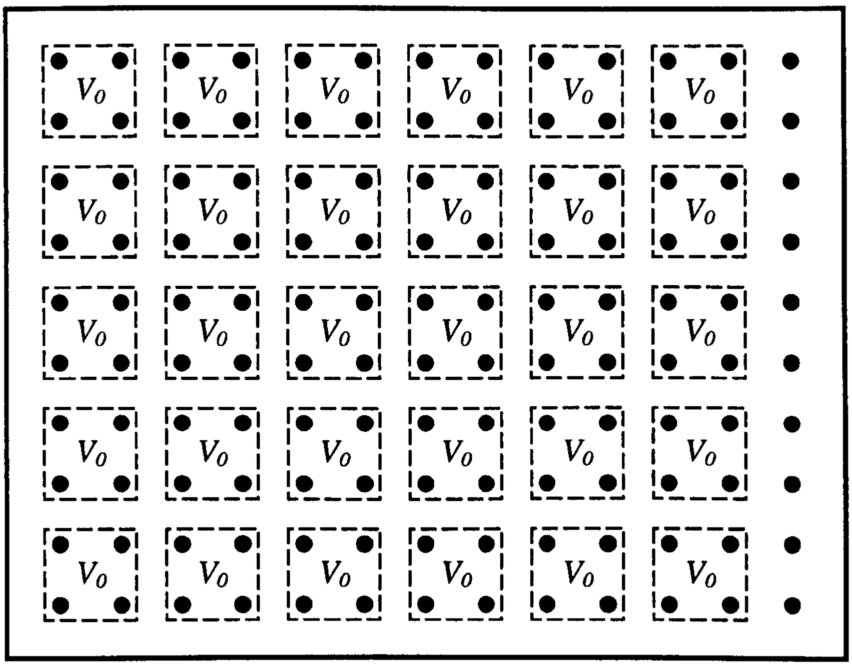
$$V_n(i, j) = G_{K+n,n}(i, j) = D_{K+n}(i, j) - D_n(i, j)$$

Where $n > 0$. The depth of $V_n(i, j)$ are K, which means that the WCs of $V_n(i, j)$ covers all resolutions from level 0 to level $K\text{-}1$. This implies that the leaves of $D_n(i, j)$ are on level -1, thus its root (i, j) is on level $-n-1$. $D_n(i, j)$ does not actually exist. But $V_n(i, j)$ is of tree structure indeed, like $L(i, j)$. So we call $V_n(i, j)$ the virtual tree, and identify it using V or V_n as a tree type.

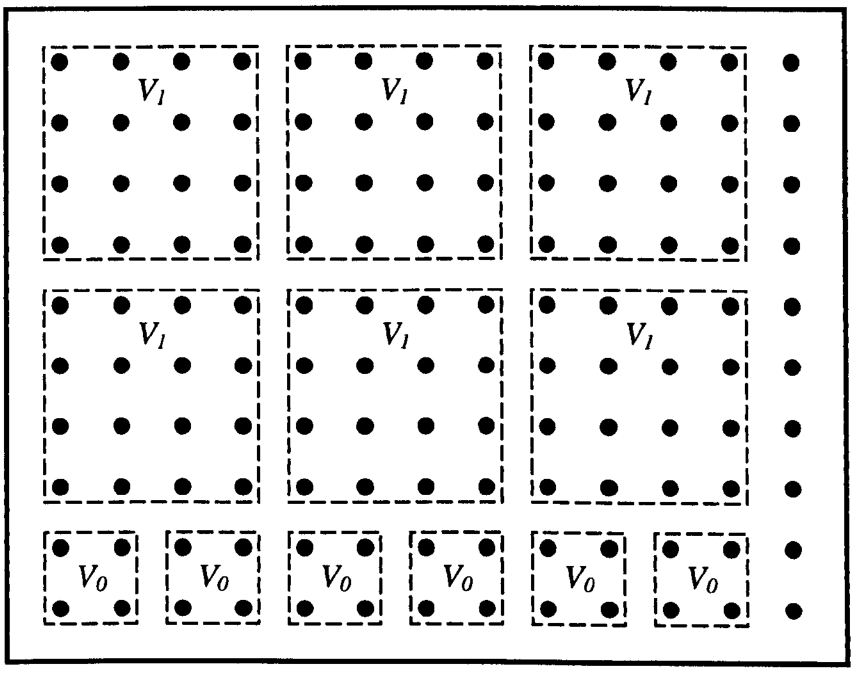
$V_0(i, j) = G_{K,0}(i, j)$. It is the same as the initial D trees in the LIS with root on LL in the original SPIHT. Like those initial D trees, we define the root of $V_n(i, j)$ specially. But unlike those initial D trees with root defined on LL, here the root of $V_n(i, j)$ is defined as the co-ordinates (index) of the top-left WC on resolution level 0 in the set. For consistent, we include $V_0(i, j)$ in $V_n(i, j)$ $(n \geq 0)$. $V_n(i, j)$ can be uniquely identified by its type V_n and root (i, j).

We look at the examples in Figure 5.1 again. $V_0(4, 0) = D(3, 0)$, and $V_1(4, 0)$ includes all WCs on LH (LH0, LH1, LH2 and LH3).

$V_n(i, j)$ is made up of 4^n neighbouring $V_0(i, j)$ in a square spatial orientation region. During the initialisation of SPIHT coding, $V_n(i, j)$ is used instead of $V_0(i, j)$. The initial V trees are set to be as large as possible. This can be done as in the following. Starting from $V_0(i, j)$, we keep on merging every neighbouring four $V_n(i, j)$ in a square spatial orientation region to a $V_{n+1}(i, j)$, until there are no more 2x2 neighbouring V trees of the same size to merge. Figure 5.2 illustrates the merging procedure. Only HL0 is shown, whose size is 10×13. We start from V_0 in figure (a), some are merged to V_1 in figure (b), and then V_2 in figure (c). Figure (c) is the final result.

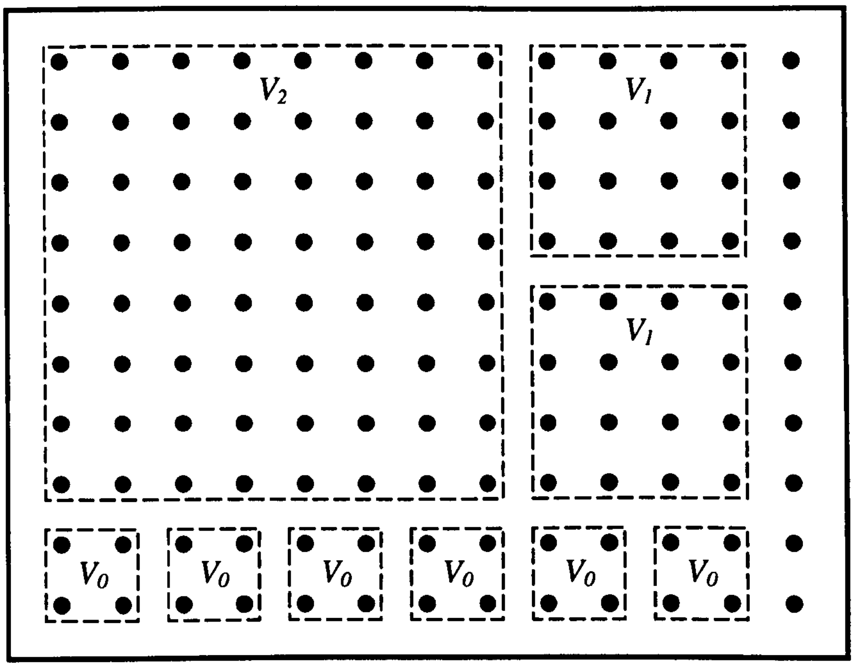


(a)



(b)

Figure 5.2 Merging of *Vn* to *Vn+1* for initial trees



(c)

Figure 5.2 Merging of V_n to V_{n+1} for initial trees

As the result, the total number of initial trees is very much reduced compared with Said and Pearlman's initialisation in [1] (described in chapter 4 of this thesis, we refer to their algorithm the original SPIHT). In the example of Figure 5.2, the number of trees for HL is reduced from thirty to nine. In case of a 512×512 image and 5-scales WT, there are only three initial trees in the LIS in our scheme – $V_3(0, 16)$, $V_3(16, 0)$ and $V_3(16, 16)$, while there are 64×3 (=192) initial trees in the original SPIHT algorithm.

$V_n(i, j)$ $(n > 0)$ is processed in the similar way as $L(i, j)$ in the sorting pass. If $V_n(i, j)$ is significant, it is partitioned to four subtrees – $V_{n-1}(k, l)$, where $k = i$ or $i+2^n$ and $l = j$ or $j+2^n$.

$V_0(i, j)$ is in fact a D tree, except its root is defined specially. It is processed in the same way as $D(i, j)$. Significant $V_0(i, j)$ is partitioned to a L tree – $G_{K,1}(i, j)$ with root at resolution level -1, and four individual WCs at (k, l), where $k = i$ or $i+2^0$ and

$l = j$ or $j+2^0$. Here (i, j) is in fact on level 0, so $L(i, j) = G_{K-1,1}(i, j)$. To distinguish the partitioned $G_{K,1}(i, j)$ with $L(i, j)$, we denote it $U(i, j)$.

$U(i, j)$ is also a virtual tree, with virtual root defined specially. Similarly, significant $U(i, j)$ is partitioned to four $D(k, l)$, where $k = i$ or $i+1$ and $l = j$ or $j+1$.

$D(i, j)$ and $L(i, j)$ are processed as usual. Significant $D(i, j)$ is partitioned to $L(i, j)$ and four individual WCs at (k, l), where $k = 2i$ or $2i+1$ and $l = 2j$ or $2j+1$. Significant $L(i, j)$ is partitioned to four $D(k, l)$, where $k = 2i$ or $2i+1$ and $l = 2j$ or $2j+1$.

In summary, significant trees are partitioned, producing new trees of different type, as shown in Figure 5.3. Individual WCs are also produced when partitioning $V_0(i, j)$ or $D(i, j)$.

$$\begin{tikzcd} V_n \arrow[r] & V_{n-1} \arrow[r] & \cdots \arrow[r] & V_0 \arrow[r] & U \arrow[r] & D \arrow[r] & L \arrow[l, bend right] \end{tikzcd}$$

Figure 5.3 Partitioning of trees (including virtual trees)

When any of the above trees is scanned during the sorting pass of SPIHT coding, a binary symbol is coded for the significance of the tree.

Due to the small amount of initial trees in the LIS, the proposed scheme is sure to produce less encoded symbols in the beginning than the original SPIHT algorithm.

For the initial WCs in the LIP, similar virtual trees can be used. As their distribution probability, which can be exploited in compression, is mainly for the outermost divisions, for simplicity, we choose a sort of run length coding, and limit its usage to the first round of scan in the sorting pass. To be compatible with the arithmetic coding, we group the initial WCs in four (2×2). We substitute the four encoded 0s for a group of initial WCs with one binary symbol 0. If the encoded symbols for the group are not four 0s, a binary symbol 1 is inserted before these four symbols. The coding result of this run length coding is in fact the same as using a virtual O tree for the initial WCs in the LIP.

As these proposed schemes reduce the encoded symbols mainly in the beginning of SPIHT coding, they are particularly effective for low bit rate image coding.

5.3 Omit the predictable symbols

In the sorting pass of SPIHT coding, if a tree is significant, it is partitioned into subtrees and/or individual WCs. At least one of the partitioned subtrees or WCs is significant. This means there is redundancy in the encoded symbols. Entropy coding can take advantage of some of the redundancy, but not all. We propose an approach to reduce the redundancy without entropy coding.

First, we re-arrange the processing of $V_n(i, j)$, $U(i, j)$ and $L(i, j)$, which are denoted in this paragraph as $G_{m,n}(i, j)$ in general form with $n \geq 1$. We reserve 4^n-1 spaces following the position of $G_{m,n}(i, j)$ in the LIS, so that $G_{m,n}(i, j)$ occupies 4^n entries altogether in the LIS. During the LIS scan in the sorting pass, the reserved spaces are skipped if $G_{m,n}(i, j)$ is insignificant. If $G_{m,n}(i, j)$ is significant, it is partitioned into four subtrees. The subtrees fill in the spaces occupied by $G_{m,n}(i, j)$, and are scanned immediately after $G_{m,n}(i, j)$.

Now we check and change the encoding when partitioning a significant tree. If $V_n(i, j)$, $U(i, j)$ or $L(i, j)$ is significant, its significance value is coded, and it is partitioned to four subtrees. Then the four subtrees are scanned and coded. If the former three subtrees are all insignificant, the fourth must be significant. In this case, the output bit pattern (binary symbols) is 10001, with the first symbol for the significance of the partitioned tree, followed by the significance values of four subtrees. As the last symbol is predictable from previous four symbols, we omit it.

The trees are grouped for arithmetic coding (refer to section 4.5 for details). A group of four trees can only be four *D*-trees partitioned from a significant *L* or *U* – tree, or four *V* or *U* – trees partitioned from a significant *V* - tree. At lease one of the coding symbols for the group of four trees is 1. If the coding symbols are 0001, the last symbol 1 is predictable and thus can be omitted. This suggests another implementation of the above scheme. Here *V(i, j)*, *U(i, j)* and *L(i, j)* do not need extra spaces (empty entries) in the LIS, and their subtrees are added to the end of the LIS as in the original SPIHT algorithm.

The second case is *D(i, j)* with root *(i, j)* on resolution level *K-2*. It is in fact *O(i, j)*. If it is significant, it is partitioned to four individual WCs. If the former three WCs are all insignificant, the fourth must be significant. Thus the relevant output bit pattern is 10001. The last symbol is predictable, so we omit it too.

The third case is *D(i, j)* in general (excluding the *D(i, j)* in the second case). If it is significant, it is partitioned to four individual WCs and one *L(i, j)*. If the four individual WCs are all insignificant, *L(i, j)* must be significant. In this case, we do not code the significance of the *L(i, j)*, and do not add the *L(i, j)* to the end of the LIS (refer to the original SPIHT for details). Instead, we partition *L(i, j)* immediately to four *D(k, l)*, and add them to the end of the LIS.

The schemes in the former two cases are only for the SPIHT image coding system without arithmetic coding. In the third case, it does not matter whether the arithmetic coding is used or not.

Despite the above, all other processes in the sorting pass remain the same as in the original SPIHT. As the omitted symbols are predictable, we can easily decode the symbols without extra loss, following the same procedure as in the encoding.

5.4 Offset the quantisation

Recall the quantisation in the original SPIHT. For threshold T $(T > 0)$, insignificant WCs in the range $(-T, T)$ are quantised to 0; significant WCs in the range $(-2T, -T]$ are quantised to $-3T/2$; and significant WCs in the range $[T, 2T)$ are quantised to $3T/2$. Is the quantisation optimal?

PSNR as defined in equation 1.2 of chapter 1 is used to measure the performance of image coding. The higher the PSNR, the better the reconstructed image. As equation (1.2) indicated, to get maximum PSNR, minimum MSE is wanted.

The optimisation here is: finding a quantisation value to approximate all the WCs in a division, to get the minimum MSE.

It is a least mean square (LMS) problem. As we know, the solution is the mean value of these WCs.

For convenience, we discuss only the positive WCs in the following. It is similar for the negative WCs.

As the distribution of WCs in chapter 3 shows, for natural image, there are more WCs in the range $[0, T)$ than WCs in the range $[T, 2T)$. It is also true that there are more WCs in the range $[T, 3T/2)$ than WCs in the range $[3T/2, 2T)$. The mean value of the WCs in the range $[T, 2T)$ is less than the geometric centre of the division (division centre in short), $3T/2$. So, the quantisation in the original SPIHT, where the quantised value of WCs in each division is the division centre, is not optimal.

Table 5.3 Average mean value of WCs in each division

Division	Centre	Average	Difference	*T/8*
-12	-3072	-2356.1	715.9	256
-11	-1536	-1173.7	362.34	128
-10	-768	-692.69	75.313	64
-9	-384	-347.34	36.657	32
-8	-192	-172.32	19.682	16
-7	-96	-87.225	8.7754	8
-6	-48	-44.433	3.5674	4
-5	-24	-22.252	1.7479	2
-4	-12	-11.108	0.89237	1
-3	-6	-5.5999	0.40013	0.5
-2	-3	-2.8836	0.11642	0.25
2	3	2.8838	0.11619	0.25
3	6	5.5974	0.40257	0.5
4	12	11.117	0.88318	1
5	24	22.207	1.7931	2
6	48	44.429	3.5709	4
7	96	86.983	9.0174	8
8	192	172.57	19.425	16
9	384	345.77	38.234	32
10	768	667.57	100.43	64
11	1536	1244.6	291.36	128

The distribution of WCs is different for different images. For various test images, the mean value of WCs in each division is calculated, to see if there is a common solution for the quantisation optimisation. The results are listed in table 3.5 of chapter 3. We extract the average mean value of WCs in each division for the test images and list in Table 5.3 (as "Average" in short). As can be seen in the table, the magnitude of the average mean value is always less than the magnitude of the division centre ("Centre" in the table). Their difference is calculated and listed in the table (as "Difference").

Studying the data in Table 5.3, we found that the difference between the average mean value and the division centre can be approximated by one eighth of the relevant quantisation threshold. Denote the difference as Δ, we have $\Delta \approx T/8$. The approximation is very good if $8 \leq T \leq 128$ (e.g., for division ±4, ±5, ±6, ±7 and ±8). In fact, that covers the most useful range of thresholds in applications, as is to be explained later.

We offset the quantisation of SPIHT coding. The magnitude of WCs in the range $[T, 2T)$ and $(-2T, T]$ is quantised to $3T/2 - \Delta$, instead of the magnitude of division centre $3T/2$.

The quantisation offset affects only the decoder in implementation. We adjust the decoding procedure at threshold T. During the sorting pass, the magnitude of significant WCs, which is in the range $[T, 2T)$, is set as $3T/2 - \Delta$ instead of $3T/2$. The WCs with magnitude no less than $2T$ are in the LSP. For those with magnitude in the range $[2T, 4T)$, they are scanned for the first time in the refinement pass. Their magnitudes are refined to the geometric centre, $5T/2$ or $7T/2$ according to their range $[2T, 3T)$ or $[3T, 4T)$ indicated by the relevant coding symbol. Notice that they were moved to the LSP in the previous round, their magnitudes were offset from the division centre, and the quantisation offset is removed here. Other processes remain the same as in the original SPIHT. We see that only the WCs with magnitude in the range *[T, 4T)* can be affected by the quantisation offset at threshold *T*.

Table 5.4 Encoded length of Lena after the scan of a list at each threshold in the original SPIHT

Threshold	Encoded image length (bits)		
	LIP	LIS	LSP
2048	284	476	476
1024	773	1,009	1,037
512	1,294	1,920	2,026
256	2,359	3,969	4,236
128	5,116	9,741	10,310
64	12,776	22,046	23,476
32	28,884	46,867	50,238
16	60,920	94,229	101,749
8	122,490	191,479	207,097
4	252,318	415,082	447,040
2	559,611	702,978	775,514
1	915,160	957,669	1,095,491

Take image Lena as an example. Table 5.4 lists its encoded length after a list (LIP/LIS/LSP) has been scanned at each threshold in the original SPIHT. Before $T = 64$, the maximum encoded length is 10,310 bits, the coding rate is 0.04 bit per pixel (bpp). Figure 5.4 shows the reconstructed image at this rate. The distortion is large. The image quality is not acceptable in most applications.

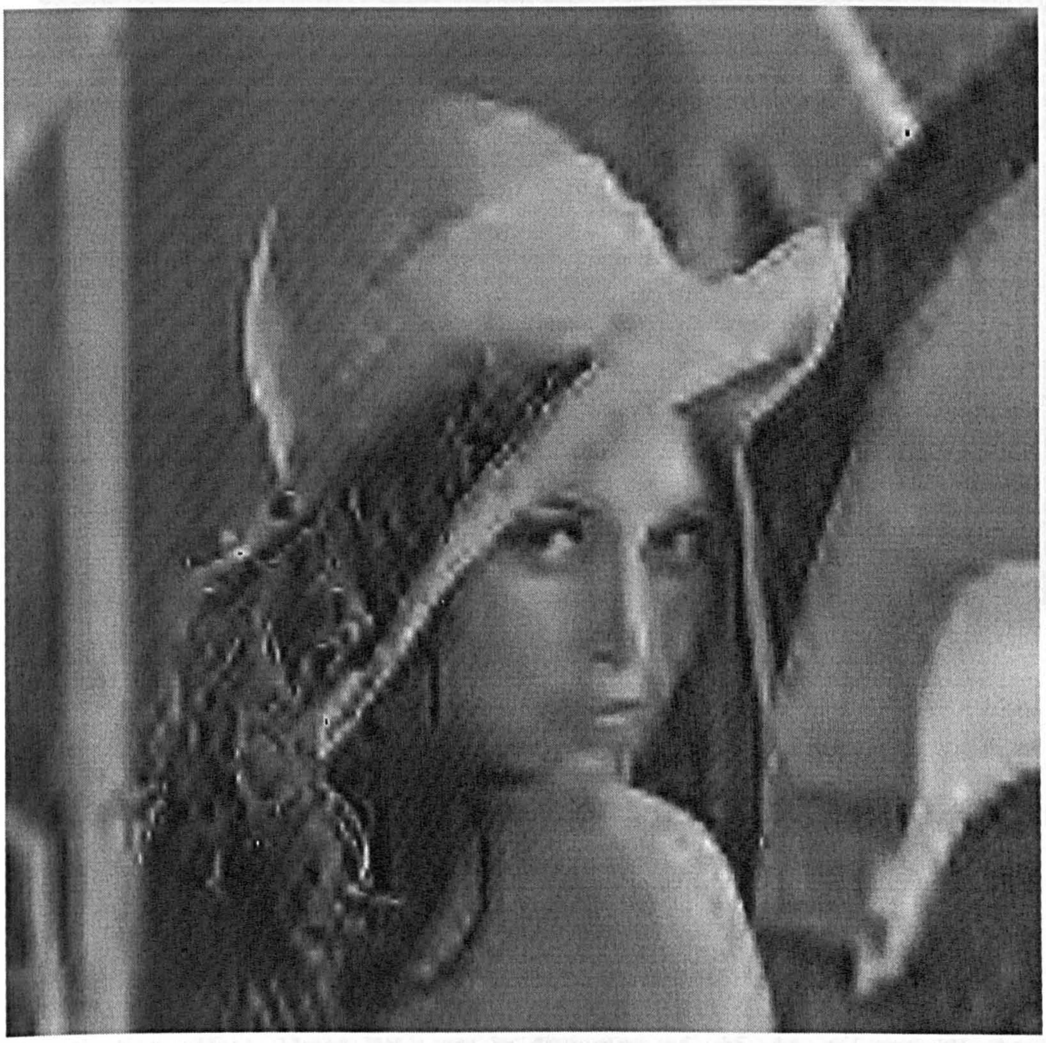


Figure 5.4 Reconstructed Lena at 0.04 bpp

After $T = 8$, the encoded length of Lena is 207,097 bits. The coding rate is 0.8 bpp. Figure 5.5 shows the reconstructed image at this rate. Compared with the original Lena (refer to figure 3.1 in chapter 3), one can hardly tell any difference. The image quality is good enough for most applications where lossy image coding is eligible.

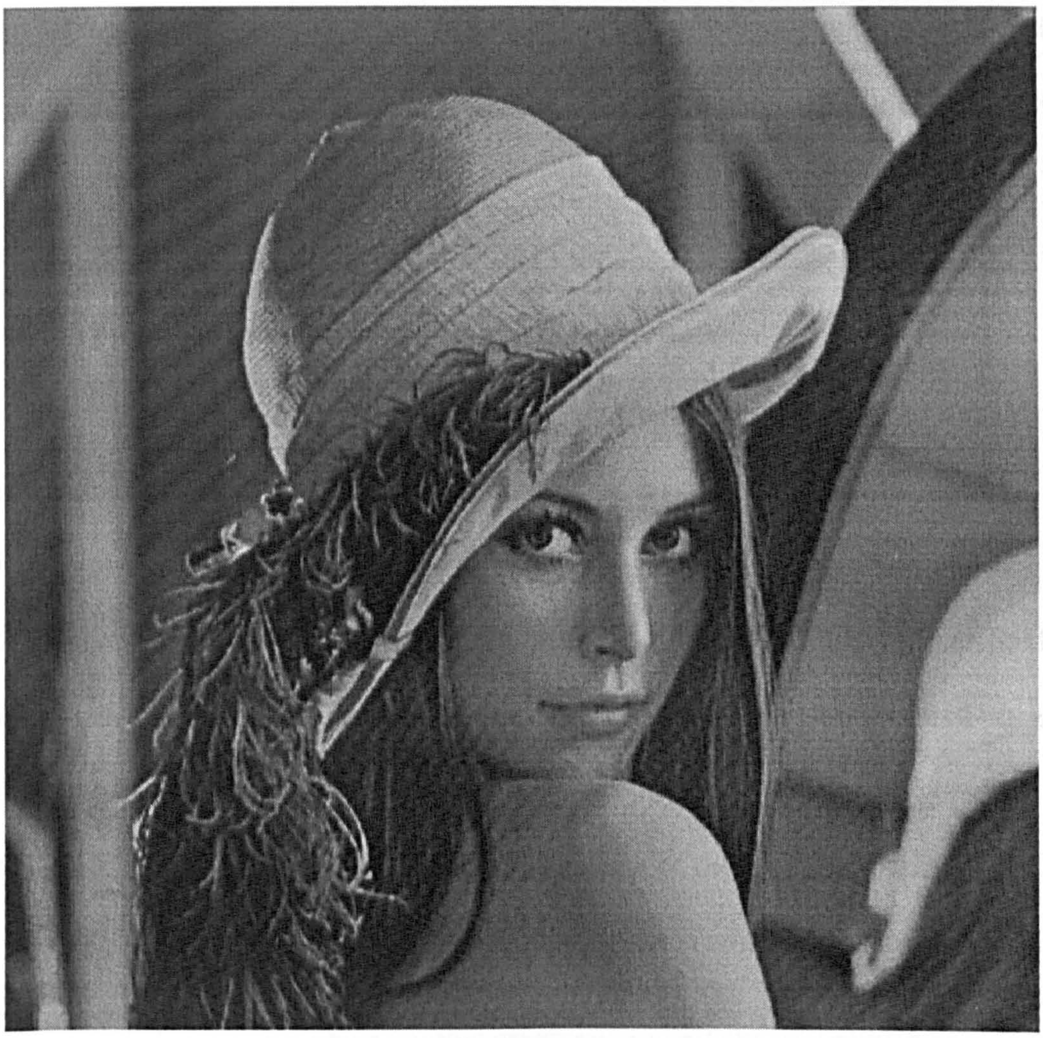


Figure 5.5 Reconstructed Lena at 0.8 bpp

In summary, for image Lena, the important bit rates of concern lie mostly in the range *(0.04, 0.8)*. This is relevant to the SPIHT coding stopping at $T \in [8, 64]$. The WCs in the range $[T, 4T)$, which is $[8, 256)$ for $T \in [8, 64]$, can be affected by the quantisation offset. These WCs are in divisions ±4, ±5, ±6, ±7 and ±8. As stated previously, in these divisions, $\Delta = T/8$ approximates the difference in Table 5.3 very well. It is concluded that the quantisation offset of $\Delta = T/8$ is good enough for most applications.

$\Delta = T/8$ can be calculated by 3 bits shift to the right (least significant bit) of T, which is very simple and fast. That is the reason $\Delta = T/8$ *is* chosen, other than any other more accurate values around $T/8$.

For $T < 8$, since integer operation is often used, $\Delta = T/8$ is rounded to 0, or $\Delta = \lfloor T/8 \rfloor$. That means there is no offset for the quantisation if $T < 8$.

For $T > 128$, $\Delta = T/8$ is less than the difference in Table 5.3. That means the performance of SPIHT with the quantisation offset is not optimal, but better than that without quantisation offset.

So, the quantisation offset $\Delta = T/8$ is appropriate to be applied for any threshold T during the whole SPIHT coding procedure, although it is not optimal at the beginning and the end.

For a selected offset model $\Delta = f(T)$, the parameters of the model (function f) can be optimised for typical test images using LMS criteria. Though our approximation is simple and effective in practice.

5.5 Optimise the arithmetic coding

For the adaptive models of arithmetic coding used in the SPIHT image coding system, the initial frequencies and the maximum frequency can be optimised for each model.

An extra symbol is added in the arithmetic coding model to mark the end of bitstream (EOB) in implementation. The frequency of the EOB symbol is set to 1 initially and never changes before the arithmetic coding ends. Normally, this value is trivial among the frequencies of all symbols. But if we set all the initial frequencies to be 1, the EOB symbol does change the distribution of symbols, and the effect can not be ignored in the beginning of arithmetic coding using adaptive models, or in case a fixed model is used. In both cases, the initial frequencies of other symbols (excluding EOB) can be increased to reduce the effect of the EOB symbol.

Now we take a close look at some of the adaptive models used. For the initial WCs in the LIP grouped in four, an adaptive model with sixteen symbols (0000 – 1111) is used to code their significance values. As the cause of introducing the virtual trees in section 5.1, symbol 0000 is dominant in the beginning of SPIHT coding. We set the initial frequency of symbol 0000 higher than others to track this unbalanced initial distribution.

D trees grouped in four come only from partitioned significant *L* or *U* trees. As discussed in section 5.3, these four trees in the group can not be all insignificant, so there are fifteen possible symbols (0001 – 1111) for their significance values. We can also use a model of sixteen symbols (0000 – 1111), but set the initial frequency of 0000 as 0. It is similar for a group of four *V* trees, and for a group of four WCs partitioned from a significant *D* tree which is in fact an *O* tree.

Experiments show that the distribution pattern of significance values of *L* trees changes rapidly during the procedure of SPIHT coding. The distribution changes dramatically when the quantisation threshold changes, but there is still some similarity between successive round of sorting pass. We can use a small maximum frequency for *L* trees, and/or halve all the frequencies at the end of each round.

For a group of *n* WCs (or trees), the co-ordinates of WCs (or tree roots) may occupy any *n* of four possible orientation positions. This orientation occupation may contain texture information. We can use an adaptive model for each group with different orientation occupation.

Experiments show that these approaches above do get some performance gain, but the gain is very limited.

More models can be used in arithmetic coding to track the distribution patterns of the symbols of SPIHT coding. But we must be aware that the total number of symbols produced by the SPIHT encoding for an image is limited. As the number of models increases, the number of symbols for each model decreases. On the other hand, an adaptive model needs enough symbols to track their distribution pattern, the more the better. The arithmetic coding is often inefficient at the beginning before the models have adapted well to the distribution pattern. We should take both aspects into consideration and compromise in setting arithmetic coding models.

The recent embedded block coding with optimised truncation (EBCOT) [4] uses context-based models for arithmetic coding, to exploit the intra-subband correlation of WCs. Only immediate neighbours are used as context. The context models are classified into categories, and an arithmetic coding model is used for each of the context categories. Hence the total number of arithmetic coding models are reduced dramatically. Similar context-based models can be used for the WCs and trees in the arithmetic coding of SPIHT. EBCOT outperforms most state-of-the-art image coding algorithms, and was adopted as the coding engine of JPEG-2000. We know that the SPIHT algorithm exploits the inter-level correlation of WCs successfully. The combination of context-based arithmetic coding and the SPIHT algorithm can exploit both the intra-subband correlation and inter-level correlation efficiently, and is promising.

5.6 A simple example

An example is presented in this section to demonstrate the improved SPIHT coding, using the same 20×16 image used in section 4.5 of chapter 4. For convenience, the wavelet coefficients in Figure 4.6 are repeated in Figure 5.6.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-14	-52	-34	31	3	-7	0	1	1	2	0	-1	2	0	2	-2
1	-16	-54	-5	19	10	-4	-1	-11	-2	0	1	-2	0	-1	0	-2
2	-31	-45	9	31	13	3	-24	-9	1	0	1	2	-3	1	0	0
3	-32	-3	98	43	29	-15	-1	-14	3	0	2	-9	0	5	0	-1
4	5	-34	13	40	-9	-5	0	2	3	0	-2	2	-8	4	0	-3
5	0	-3	-8	0	-2	5	-8	-1	0	5	-6	0	0	0	0	-2
6	-7	1	9	4	-3	5	3	3	5	3	-6	0	-1	0	-2	-1
7	-16	-13	-13	3	-9	-12	-2	5	-3	0	3	-2	3	-4	2	0
8	1	21	0	-3	0	0	-5	-5	1	1	0	3	-4	-5	3	0
9	1	-34	-6	12	-2	-6	2	4	0	-2	1	0	0	-1	0	1
10	0	-1	0	1	3	2	1	0	1	0	0	1	-1	-1	0	0
11	0	-1	0	-4	-4	2	-2	0	1	0	0	4	0	2	0	0
12	0	0	-1	-8	2	-1	-4	-1	0	0	-1	-1	4	-1	0	1
13	-7	0	-1	3	-1	1	0	0	0	0	1	4	-3	0	0	0
14	8	0	5	20	0	0	-1	-1	0	0	1	-3	0	0	0	0
15	-1	0	-5	-5	2	0	-1	0	0	0	-1	0	0	0	1	0
16	-5	0	0	-5	-1	0	-3	0	-1	1	3	-2	2	0	0	0
17	-8	0	-1	11	0	0	0	1	5	0	0	1	0	-1	0	1
18	0	-6	3	-3	-4	3	0	-1	0	1	-4	3	-4	1	0	-1
19	0	3	-1	-7	1	1	0	0	0	-2	0	0	0	0	0	0

Figure 5.6 Example of 2-scale wavelet transform of a 20 × 16 image

Only the SPIHT encoding without arithmetic coding is discussed here in the example, so the relevant improvements include the virtual trees and omitting the predictable symbols.

The initial LIP is the same for the improved and original SPIHT coding. In the improved and original SPIHT encoding, the processing of the LSP is the same, and the processing of the LIP is also the same except in the first round of sorting pass. Only the differences are discussed here, that is the processing of the LIP in the first round and the LIS.

For convenience, the initial LIP in Table 4.1 is repeated in Table 5.5. There is a little difference between the two tables. The wavelet coefficients in Table 5.5 are grouped in four.

In the first round of the sorting pass, the wavelet coefficients in the initial LIP (Table 5.5) are scanned and processed group by group in order. For the first group, the four wavelet coefficients are all insignificant at the initial quantisation threshold $T = 64$. The encoder outputs a 0 (for insignificant), and the four wavelet coefficients stay in the LIP. The results are the same for groups 2 and 3. For group 4, the third wavelet coefficient is significant. The encoder outputs a 1 (for significant), followed by the coding symbols for the four individual wavelet coefficients in the group, which are 00100. Groups 5 to 8 are all insignificant, and four 0s are encoded for them.

In summary, the encoded bit-stream for the LIP in the first round of SPIHT encoding is 0001,0010,0000,0, which contains thirteen binary symbols. Recall that the original SPIHT encoding produces thirty-three binary symbols for the LIP in the first round. The improvement reduces the length of the output bit-stream by twenty bits, which is more than 60%.

Table 5.5 Initial LIP for the example image

Group	Subband	Co-ordinates	Value	Group	Subband	Co-ordinates	Value
1	LL0	*(0, 0)*	*-14*	5	LL0 (last row)	*(4, 0)*	*5*
		(0, 1)	*-52*			*(4, 1)*	*-34*
		(1, 0)	*-16*			*(4, 2)*	*13*
		(1, 1)	*-54*			*(4, 3)*	*40*
2		*(0, 2)*	*-34*	6	Last row of HL0	*(4, 4)*	*-9*
		(0, 3)	*31*			*(4, 5)*	*-5*
		(1, 2)	*-5*			*(4, 6)*	*0*
		(1, 3)	*19*			*(4, 7)*	*2*
3		*(2, 0)*	*-31*	7	Last row of LH0	*(9, 0)*	*1*
		(2, 1)	*-45*			*(9, 1)*	*-34*
		(3, 0)	*-32*			*(9, 2)*	*-6*
		(3, 1)	*-3*			*(9, 3)*	*12*
4		*(2, 2)*	*9*	8	Last row of HH0	*(9, 4)*	*-2*
		(2, 3)	*31*			*(9, 5)*	*-6*
		(3, 2)	*98*			*(9, 6)*	*2*
		(3, 3)	*43*			*(9, 7)*	*4*

The initial LIS for the improved SPIHT coding is listed in Table 5.6. For convenience, the co-ordinates are listed for the wavelet coefficients on the virtual trees, while the values are listed for that on *D*-trees.

Table 5.6 Initial LIS for the example image

No	**Sub-band**	**Tree**	**Wavelet coefficients**	*No*	**Sub-band**	**Tree**	**Wavelet coefficients**
1	HL	$V_l(0,4)$	$a(i, j): i = 0 \sim 3, j = 4 \sim 7;$ $a(i, j): i = 0 \sim 7, j = 8 \sim 15$	*4*	HL (root at last row of HL0)	*D(4, 4)*	*1, 1, 0,-2*
				5		*D(4, 5)*	*0, 3, 1, 0*
				6		*D(4, 6)*	*-4,-5, 0,-1*
				7		*D(4, 7)*	*3, 0, 0, 1*
2	LH	$V_l(5,0)$	$a(i, j): i = 5 \sim 8, j = 0 \sim 3;$ $a(i, j): i = 10 \sim 17, j = 0 \sim 7$	*8*	LH (root at last row of LH0)	*D(9,0)*	*0.-6, 0, 3*
				9		*D(9, 1)*	*3,-3,-1,-7*
				10		*D(9, 2)*	*-4, 3, 1, 1*
				11		*D(9, 3)*	*0,-1, 0, 0*
3	HH	$V_l(5,4)$	$a(i, j): i = 5 \sim 8, j = 4 \sim 7;$ $a(i, j): i = 10 \sim 17, j = 8 \sim 15$	*12*	HH (root at last row of HH0)	*D(9, 4)*	*0, 1, 0,-2*
				13		*D(9, 5)*	*-4, 3, 0, 0*
				14		*D(9, 6)*	*-4, 1, 0, 0*
				15		*D(9, 7)*	*0,-1, 0, 0*

In the first two rounds of SPIHT encoding, all trees are insignificant, and fifteen 0s are added to the output bit-stream in each round, instead of twenty-four in the original SPIHT encoding.

The processing of the LIS in the third round is illustrated in Table 5.7. Steps 1, 18 and 26 are described in the following paragraphs to demonstrate the partitioning of *V* and *U* trees.

There are fifteen entries in the LIS at the beginning of the third round. In step 1, the first entry – $V_1(0, 4)$ is scanned and processed. Since the maximum magnitude of wavelet coefficients on $V_1(0, 4)$ is 29, which is greater than T (= 16), $V_1(0, 4)$ is significant. The encoder outputs a 1 (for significant), and partitions $V_1(0, 4)$ to four subtrees – $V_0(0, 4)$, $V_0(0, 6)$, $V_0(2, 4)$ and $V_0(2, 6)$, which are added to the end of the LIS and processed latter in the same (third) round at steps 16 to 19.

In step 18, $V_0(2, 4)$ is processed. The maximum magnitude of wavelet coefficients on $V_0(2, 4)$ is 29, which is greater than T, so $V_0(2, 4)$ is significant. The encoder outputs a 1 (for significant), and partitions $V_0(2, 4)$ to $U(2, 4)$ and four individual wavelet coefficients – $a(2, 4)$, $a(2, 5)$, $a(3, 4)$ and $a(3, 5)$, which are then scanned and processed in the order that they are listed. $a(2, 4)$, $a(2, 5)$ and $a(3, 5)$ are insignificant, so they are added to the end of the LIP. $a(3, 5)$ is significant, so it is added to the end of the LSP. The encoded binary symbols for the four wavelet coefficients are 00100. $U(2, 4)$ is added to the end of the LIS and processed latter in the same (third) round at step 24. In fact, the processing of $V_0(2, 4)$ is the same as that of $D(2, 1)$ in the original SPIHT encoding (step 2 in Table 4.3), except that the names of trees and step numbers are different.

In step 26, $U(7, 0)$ is processed, in the same way as L(3, 0) in the original SPIHT encoding (step 27 in Table 4.3).

Table 5.7 Processing of the LIS in the third round

Step	Trees to be processed	Output symbols	Trees staying in the LIS	Wavelet coefficients added to the LIP	Residues added to the LSP
1	$V_I(0,4)$	1			
2	$V_I(5,0)$	1			
3	$V_I(5,4)$	0	$V_I(5,4)$		
4	$D(4, 4)$	0	$D(4, 4)$		
5	$D(4, 5)$	0	$D(4, 5)$		
6	$D(4, 6)$	0	$D(4, 6)$		
7	$D(4, 7)$	0	$D(4, 7)$		
8	$D(9,0)$	0	$D(9,0)$		
9	$D(9, 1)$	0	$D(9, 1)$		
10	$D(9, 2)$	0	$D(9, 2)$		
11	$D(9, 3)$	0	$D(9, 3)$		
12	$D(9, 4)$	0	$D(9, 4)$		
13	$D(9, 5)$	0	$D(9, 5)$		
14	$D(9, 6)$	0	$D(9, 6)$		
15	$D(9, 7)$	0	$D(9, 7)$		
16	$V_O(0,4)$	0	$V_O(0,4)$		
17	$V_O(0,6)$	0	$V_O(0,6)$		
18	$V_O(2,4)$	100100		$a(2,4),a(2,5),a(3,5)$	$a(3,4)$
19	$V_O(2,6)$	111000		$a(2,7),a(3,6),a(3,7)$	$a(2,6)$
20	$V_O(5,0)$	0	$V_O(5,0)$		
21	$V_O(5,2)$	0	$V_O(5,2)$		
22	$V_O(7,0)$	1110010		$a(7,1),a(8,0)$	$a(7,0)$, $a(8,1)$
23	$V_O(7,2)$	0	$V_O(7,2)$		
24	$U(2,4)$	0	$U(2,4)$		
25	$U(2,6)$	0	$U(2,6)$		
26	$U(7,0)$	1			
27	$D(7,0)$	0	$D(7,0)$		
28	$D(7,1)$	101000		$a(14,2),a(15,2),a(15,3)$	$a(14,3)$
29	$D(8,0)$	0	$D(8,0)$		
30	$D(8,1)$	0	$D(8,1)$		

There is an alternative processing order for the trees listed in Table 5.7. Each V_l – tree in the LIS is followed by three extra empty entries. In the first step, the partitioned $V_0(0, 4)$, $V_0(0, 6)$, $V_0(2, 4)$ and $V_0(2, 6)$ replace the entry for $V_0(0, 4)$ and the three reserved empty entries, and the processing of steps 16 to 19 in Table 5.7 follows immediately. It is similar for $V_0(5, 0)$ of entry 2 in Table 5.7. After the processing of $V_0(5, 4)$ (entry 3 in Table 5.7), the three reserved empty entries are skipped because $V_1(5, 4)$ is insignificant. Processing of other entries is the same as in Table 5.7. The resulting output bit-stream for the LIS in the third round is 1001, 0010, 0111, 0001, 0011, 1001, 0000, 0000, 0000, 0000, 0010, 1010, 0000, which consists of forty-two binary symbols. For the two processing orders, the overall output symbols are the same, but in a different orders.

Table 5.8 Output bit-stream in the first three rounds of the improved SPIHT encoding

T	List	Total	No.	Symbols
64	LIP	13	1~13	0001,0010,0000,0
	LIS	15	14~28	0000, 0000, 0000, 000
	LSP	0		
32	LIP	40	29~68	0110, 1111, 0000, 1111, 0001, 0011, 0100, 0000, 1100, 0000
	LIS	15	69~83	0000, 0000, 0000, 000
	LSP	1	84	1
16	LIP	27	85~111	0111, 0010, 1100, 1000, 0000, 0000, 000
	LIS	42	112~153	1001, 0010, 0111, 0001, 0011, 1001, 0000, 0000, 0000, 0000, 0010, 1010, 0000
	LSP	10	154~163	0110, 0000, 00

In summary, by the end of the third round, there are one hundred and sixty-three encoded binary symbols in the output bit-stream, which is summarised in Table 5.8. For comparison, we list the coding results of both the original and the improved SPIHT (OSPIHT and ISPIHT) in Table 5.9. The table shows that the length of the output bit-stream is reduced significantly by the improvements.

Table 5.9 Length of output bit-stream in the first three round of the original and the improved SPIHT encoding

T	List	Each part				Overall			
		OSPIHT	ISPIHT	Save		OSPIHT	ISPIHT	Save	
64	LIP	33	13	20	61%	33	13	20	61%
	LIS	24	15	9	38%	57	28	29	51%
	LSP	0	0			57	28	29	51%
32	LIP	40	40			97	68	29	30%
	LIS	24	15	9	38%	121	83	38	31%
	LSP	1	1			122	84	38	31%
16	LIP	27	27			149	111	38	26%
	LIS	52	42	10	19%	201	153	48	24%
	LSP	10	10			211	163	48	23%

To demonstrate the improvement of omitting the predictable coding symbols, the encoding procedure is carried on to the fourth round at $T = 8$. We check the trees $V_0(5, 0)$ and $D(8, 1)$ remaining in the LIS after the third round (entries 20 and 30 in Table 5.7).

$V_0(5, 0)$ is significant at $T = 8$. It is partitioned into $U(5, 0)$ and four individual wavelet coefficients – $a(5, 0)$, $a(5, 1)$, $a(6, 0)$ and $a(6, 1)$. The values of the four wavelet coefficients are *0, -3, -7* and *1*, which are all insignificant. So, the output coding symbols are 10000. The four wavelet coefficients are added to the end of the LIP. *U(5, 0)* is known to be significant at this point, and it is partitioned immediately to four subtrees – *D(5, 0)*, *D(5, 1)*, *D(6, 0)* and *D(6, 1)*. The four subtrees are added to the end of the LIS to be processed latter in the same (forth) round. On the contrary, in the original SPIHT coding, *U(5, 0)* is not partitioned and is added to the end of the LIS. *U(5, 0)* will be scanned and processed latter in the same round, which will produce a predictable 1 and be partitioned into the four subtrees. The output symbol 1 for *U(5, 0)* is omitted in the improved SPIHT coding.

D(8, 1) consists of four wavelet coefficients – *a(16, 2)*, *a(16, 3)*, *a(17, 2)* and *a(17, 3)*, whose values are *0, -5, -1* and *11*. *D(8, 1)* is significant at $T = 8$, and is partitioned to the four individual wavelet coefficients. The output coding symbols are 1000. For comparison, in the original SPIHT coding, the output symbols are 10001. The last symbol is omitted in the improved SPIHT coding.

More coding symbols are omitted in the fourth round and beyond in the improved SPIHT coding. As the result, the length of the output bit-stream is reduced without sacrificing the PSNR. Or, for the same coding length (thus the same coding rate), the improved SPIHT algorithm gets higher PSNR than the original SPIHT algorithm.

5.7 Summary

DC-level shifting for the zerotree-based transform coding was discussed in this chapter. We selected the simplest method, deducting the image pixels by a fixed value – the centre of the dynamic range. As we shall see in chapter 7, this is also the best among the three discussed methods in most cases. The other two methods are: deducting the image pixels by their mean value and deducting the WCs on LL by their mean value. For these two methods, the mean value must be coded together with the WCs.

Three measures were presented in this chapter to improve the SPIHT coding. First, the virtual trees, V_n and U, were introduced, together with the run length coding for initial WCs in the LIP. Second, the procedure of set partitioning was re-arranged to omit the predictable symbols. Third, the quantised value of a division was moved from the geometric centre to the statistical centre. A simple example is given to demonstrate the procedure of the improved SPIHT coding. The performance gain of these improvements will be demonstrated in chapter 7.

Ways to optimise the arithmetic coding are also discussed. A promising approach to combine the context-based arithmetic coding (as used in EBCOT) and the SPIHT coding is pointed out.

References

[1] A.Said and W.A.Pearlman, 'A new, fast, and efficient image codec based on set partitioning in hierarchical trees', IEEE Transactions on Circuits and Systems for Video Technology, Vol.6, No.3, pp.243-50, June 1996.

[2] J.M.Shapiro, 'Embedded Image Coding Using Zerotrees of Wavelet Coefficients', IEEE Transactions on Signal Processing, Vol.41, No.12, pp.3445-62, December 1993.

[3] W.B.Pennebaker and J.L.Mitchell. JPEG still image data compression standard. Van Nostrand Reinhold, New York, 1993.

[4] D.Taubman, 'High performance scalable image compression with EBCOT', IEEE Transactions on Image Processing, Vol.9, No.7, pp.1158-70, July 2000.

Chapter 6

Pre-processing for SPIHT Coding

6.1 Speed up the judgement of significance of trees

During SPIHT coding, the significance value of every tree in the list of insignificant sets (LIS) is required. The direct solution is to check the nodes on the tree and compare the wavelet coefficients (WC) with the quantisation threshold.

If a tree is insignificant with respect to the current threshold, all the nodes on the tree will be checked. Later for the next smaller threshold, the nodes on the tree will be checked again. If the tree is insignificant again, all the nodes will be checked for the second time, and so on. As the result, the same set of WCs may be compared individually with different thresholds again and again.

If a tree is found to be significant after checking many of the nodes on the tree, it will be partitioned into four subtrees as well. Then for the subtrees, their nodes will be checked, while many of the WCs have been compared with the same threshold previously. As the result, some WCs may be compared with the same threshold again and again.

So, the direct solution to judge the significance of trees is not efficient. Can the comparison be done only once for each WC? The answer is yes. A new scheme is developed here.

For an image of size $M*N$, we define a matrix $\boldsymbol{M_{max}}$ whose size is $M/2 \times N/2$. The element of $\boldsymbol{M_{max}}$, $m_{max}(i, j)$ ($0 \leq i < M/2$ and $0 \leq j < N/2$), is the maximum magnitude of $D(i, j)$ if (i, j) is not on LL0. For (k, l) on LL0, $m_{max}(k, l)$ is the maximum magnitude of $V_n(i, j)$ (if it exists), where $k = \lfloor i/2^{(n+1)} \rfloor$ and $l = \lfloor j/2^{(n+1)} \rfloor$. After the wavelet transform during the SPIHT image coding, we calculate $\boldsymbol{M_{max}}$ first, and then use $\boldsymbol{M_{max}}$ to judge the significance of all type of trees as in the following paragraphs. Denote the current quantisation threshold as T.

(1) For $D(i, j)$, compare $m_{max}(i, j)$ with T.

If $m_{max}(i, j) < T$, $D(i, j)$ is insignificant;

If $m_{max}(i, j) \geq T$, $D(i, j)$ is significant.

(2) For $L(i, j)$, it is made up of four subtrees: $D(k, l)$, where $k = 2i$ *or* $2i+1$, and $l = 2j$ or $2j+1$. Compare $m_{max}(k, l)$ with T.

If all the four $m_{max}(k, l) < T$, $L(i, j)$ is insignificant;

If any of the four $m_{max}(k, l) \geq T$, $L(i, j)$ is significant.

We denote L_{max} the maximum value of the four $m_{max}(k, l)$ here.

(3) For $U(i, j)$, it is made up of four subtrees: $D(k, l)$, where $k = i$ *or* $i+1$, and $l = j$ or $j+1$. Compare $m_{max}(k, l)$ with T.

If all the four $m_{max}(k, l) < T$, $U(i, j)$ is insignificant;

If any of the four $m_{max}(k, l) \geq T$, $U(i, j)$ is significant.

We denote U_{max} the maximum value of the four $m_{max}(k, l)$ here.

(4) For $V_n(i, j)$, compare $m_{max}(k, l)$ with T, where $k = \lfloor i/2^{(n+1)} \rfloor$ and $l = \lfloor j/2^{(n+1)} \rfloor$

If $m_{max}(k, l) < T$, $V_n(i, j)$ is insignificant;

If $m_{max}(k, l) \geq T$, $V_n(i, j)$ is significant.

To get $\boldsymbol{M_{max}}$, we calculate from the bottom up. That is, start from $m_{max}(i, j)$ where $M/4 \le i < M/2$ and $N/4 \le j < N/2$. Here $D(i, j)$ is $O(i, j)$, which has four WCs only. $m_{max}(i, j)$ is the maximum magnitude of the four WCs.

For $i < M/4$ and $j < N/4$, $D(i, j) = O(i, j) + L(i, j)$. It is easy to calculate the maximum magnitude of the four WCs belong to $O(i, j)$, denoted as O_{max}. $L(i, j)$ is made up of four subtrees: $D(k, l)$, where $k = 2i$ or $2i+1$, and $l = 2j$ or $2j+1$. m_{max}(k, l) is known already. Determine L_{max}, the maximum value among the four $m_{max}(k, l)$. $m_{max}(i, j)$ is the larger value of O_{max} and L_{max}.

For (k, l) on LL0, $m_{max}(k, l)$ related with $V_n(i, j)$ is calculated similarly. $m_{max}(k, l)$ related with $V_0(i, j)$ comes from relevant O_{max} and U_{max}. There is no $O(i, j)$ for $V_n(i, j)$ $(n > 0)$, so the relevant $m_{max}(k, l)$ equals to V_{max}.

In summary, $\boldsymbol{M_{max}}$ is calculated before the SPIHT coding procedure. With this overhead of processing, the judgement of the significance of trees is simplified significantly, thus reducing the computation load. But extra memory space for $\boldsymbol{M_{max}}$ is required. That is an exchange of space (memory) for time (speed).

An example is given here to show $\boldsymbol{M_{max}}$ and its usage to judge the significance of trees. For the same 20×16 image used in chapters 4 and 5 (the wavelet coefficients are repeated in Figure 6.1), $\boldsymbol{M_{max}}$ is shown in Figure 6.2. $m(0, 0)$ and $m(4, j)$, where $j = 0 \sim 3$, do not exist.

To judge the significance of $V_1(0, 4)$, we just check the significance of $m(0, 1)$ against T. Similarly, for $D(4, 4)$ and $V_0(0, 4)$, we check $m(4, 4)$ and $m(0, 2)$ for the relevant significance. For $U(2, 4)$, we check $m(2, 4)$, $m(2, 5)$, $m(3, 4)$ and $m(3, 5)$ – if any of the four is significant, $U(2, 4)$ is significant; otherwise $U(2, 4)$ is insignificant. We do not have an L – tree in the example.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-14	-52	-34	31	3	-7	0	1	1	2	0	-1	2	0	2	-2
1	-16	-54	-5	19	10	-4	-1	-11	-2	0	1	-2	0	-1	0	-2
2	-31	-45	9	31	13	3	-24	-9	1	0	1	2	-3	1	0	0
3	-32	-3	98	43	29	-15	-1	-14	3	0	2	-9	0	5	0	-1
4	5	-34	13	40	-9	-5	0	2	3	0	-2	2	-8	4	0	-3
5	0	-3	-8	0	-2	5	-8	-1	0	5	-6	0	0	0	0	-2
6	-7	1	9	4	-3	5	3	3	5	3	-6	0	-1	0	-2	-1
7	-16	-13	-13	3	-9	-12	-2	5	-3	0	3	-2	3	-4	2	0
8	1	21	0	-3	0	0	-5	-5	1	1	0	3	-4	-5	3	0
9	1	-34	-6	12	-2	-6	2	4	0	-2	1	0	0	-1	0	1
10	0	-1	0	1	3	2	1	0	1	0	0	1	-1	-1	0	0
11	0	-1	0	-4	-4	2	-2	0	1	0	0	4	0	2	0	0
12	0	0	-1	-8	2	-1	-4	-1	0	0	-1	-1	4	-1	0	1
13	-7	0	-1	3	-1	1	0	0	0	0	1	4	-3	0	0	0
14	8	0	5	20	0	0	-1	-1	0	0	1	-3	0	0	0	0
15	-1	0	-5	-5	2	0	-1	0	0	0	-1	0	0	0	1	0
16	-5	0	0	-5	-1	0	-3	0	-1	1	3	-2	2	0	0	0
17	-8	0	-1	11	0	0	0	1	5	0	0	1	0	-1	0	1
18	0	-6	3	-3	-4	3	0	-1	0	1	-4	3	-4	1	0	-1
19	0	3	-1	-7	1	1	0	0	0	-2	0	0	0	0	0	0

Figure 6.1 Example of 2-scale wavelet transform of a 20×16 image

	0	1	2	3	4	5	6	7
0		29	10	11	2	2	2	2
1	21	12	29	24	3	9	5	1
2	8	9	5	8	5	6	8	3
3	21	13	12	5	5	6	4	2
4					2	3	5	3
5	1	4	4	2	1	4	2	0
6	7	8	2	4	0	4	4	1
7	8	20	2	1	0	3	0	1
8	8	11	1	3	5	3	2	1
9	6	7	4	1	2	4	4	1

Figure 6.2 M_{max} for the 20 × 16 example image

6.2 Pre-processing for SPIHT

It is mentioned in section 5.3 of chapter 5 that the SPIHT image coding stopping at $T = 8$ was good enough for most applications. The relevant maximum coding error of the WCs (wavelet coefficients) may be 8 or even 16, as we shall see from Table 6.1.

Studying the procedure of SPIHT coding, some interesting cases are found. For example, on tree *D(i, j)*, suppose only one of the WCs on a leaf node is significant, but its magnitude x is just a little above the threshold T:

$$x = T + \delta (\delta \text{ is a small value, } 0 \leq \delta < T)$$

Then the whole tree is significant, and will be partitioned until the leaf is reached at this threshold.

In this special case, if the significance of the sole significant WC is ignored, the tree will remain as an insignificant tree and stay in the LIS. The symbols to encode the tree are saved significantly. Table 6.1 lists the coding errors of the specially treated WC and other WCs during the scan of a list (LIP – list of insignificant WCs, LIS – list of insignificant sets, or LSP – list of significant WCs) at *T*. The results are for the SPIHT without the quantisation offset (refer to section 5.3 of chapter 5 for the offset). The coding error of the specially treated WC is not significant compared with the maximum coding error of other WCs.

Table 6.1 Maximum coding errors of wavelet coefficients during the scan of a list at threshold *T*

	Magnitude of wavelet coefficient	**Coding errors**		
		LIP	**LIS**	**LSP**
The specially-treated wavelet coefficient	$T+\delta$	$T+\delta$	$T+\delta$	$T+\delta$
	$2T+\delta$	$2T+\delta$	$2T+\delta$ $(\Rightarrow T/2+\delta)$	$T/2+\delta$
	$2^nT+\delta$ $(n>1)$	$T+\delta$	$T+\delta$	$T+\delta$ $(\Rightarrow T/2+\delta)$
Other wavelet coefficients		$< 2T$	$< 2T$	$< T$

This method is applied to all the sole significant WCs on the trees. Recall that in section 6.1, $\boldsymbol{M_{max}}$ is used to judge the significance of a tree. $\boldsymbol{M_{max}}$ can be modified to ignore the significance of any sole significant WC on the trees. For bit-plane coding, the threshold $T = 2^n$ (n is a positive integer). In fact the calculation of $\boldsymbol{M_{max}}$ is modified, as follows.

For $D(i, j)$, the maximum magnitude of the four WCs from $O(i, j)$ is O_{max}. Let $m = \lfloor log_2(O_{max}) \rfloor$ and $s = 2^m$. Then $s \le O_{max} < 2s$. We also define a constant β to be the up-limit of δ, so that $\delta < \beta$. Normally, $0 < \beta < 8$.

For $M/4 \le i < M/2$ and $N/4 \le j < N/2$, $D(i, j)$ is $O(i, j)$, which has four WCs only. If $O_{max} = s + \delta$, but only the magnitude of one WC is of the value and the magnitude of the other three WCs is less than s, then force $m_{max}(i, j) = s - 1$ (the maximum value below s). Otherwise, $m_{max}(i, j) = O_{max}$, as in section 6.1.

For $i < M/4$ and $j < N/4$, $D(i, j) = O(i, j) + L(i, j)$. $L(i, j)$ is made up of four subtrees: $D(k, l)$, where $k = 2i$ or $2i+1$, and $l = 2j$ or $2j+1$. The maximum value among the four $m_{max}(k, l)$ is L_{max}. If $O_{max} = s + \delta$, but L_{max} and the magnitude of three WCs of $O(i, j)$ is less than s, then force $m_{max}(i, j) = s - 1$. Otherwise $m_{max}(i, j)$ is the larger value of O_{max} and L_{max}, as in section 6.1.

For $V_n(i, j)$, the relevant $m_{max}(i, j)$ is calculated in the same way as in section 6.1.

The processing is done before the bit-plane coding in SPIHT. It is called pre-processing for SPIHT.

If $\beta = 0$, δ does not exist (since $0 \le \delta < \beta$). Then there will not be any pre-processing in fact.

Take the example in section 6.1. Set $\beta = 1$. $\boldsymbol{M_{max}}$ after pre-processing is shown in Figure 6.3.

We examine the encoding of $V_0(5, 0)$ at $T = 8$ (the forth round). $m(2, 0) = 8$ before pre-processing, so $V_0(5, 0)$ was significant and partitioned, producing five binary coding symbols immediately and more latter in the same round for the partitioned subtrees (refer to chapter 5 for details). Now $m(2, 0) = 7$ after pre-processing, thus $V_0(5, 0)$ is insignificant and remains in the LIS, producing only one binary coding symbols in this round. The price for the shortened output bit-stream is a little bit degrading of the PSNR, which can not be recovered in decoding.

	0	1	2	3	4	5	6	7
0		29	10	11	2	1	1	2
1	21	12	29	24	3	9	5	0
2	7	9	5	7	5	6	7	3
3	21	13	12	5	5	6	3	2
4					1	3	5	3
5	1	3	3	1	1	3	1	0
6	7	7	1	3	0	3	3	0
7	7	20	1	1	0	3	0	0
8	7	11	0	3	5	3	1	0
9	6	7	3	0	1	3	3	0

Figure 6.3 M_{max} after pre-processing for the 20 × 16 example image

6.3 Summary

A pre-processing technique has been proposed for SPIHT in this chapter. A matrix, $\boldsymbol{M}_{max}$, is used to hold the maximum magnitude of wavelet coefficients on D and V_n trees. During the calculation of $\boldsymbol{M}_{max}$, some elements whose values are higher than but very close to one of the quantisation thresholds are cut down, to reduce the impact of sole large wavelet coefficient of a tree on coding efficiency.

As to be shown in chapter 7, the pre-processing can significantly reduce the computation needed to judge the significance of trees, and improve the efficiency of SPIHT coding.

Chapter 7

Performance of SPIHT Image Coding

In chapter 5 and 6, we proposed the improvements to the SPIHT (set partitioning in hierarchical trees) image coding, introducing virtual trees, omitting predictable coding symbols, quantisation offset, using a matrix to speed up the judgement of the significance of trees, and pre-processing. We also discussed the DC-level shifting, and explored ways to optimise the arithmetic coding. Now we show the performance of the SPIHT algorithm and the performance gain of these improvements.

7.1 Implementation of the SPIHT algorithm

The programs for the original SPIHT image coding are available on the Internet (Said&Pearlman's implementation, [1]). There are two programs executable on the PC (.exe) for the SPIHT algorithm without the arithmetic coding, and two for the SPIHT algorithm with the arithmetic coding. They are listed in Table 7.1.

Table 7.1 Programs for the original SPIHT algorithm available on Internet

Program Name		**Entropy Coding Method**
Encoder	**Decoder**	
fastcode	fastdecd	none
codetree	decdtree	arithmetic

The source code of these programs is not available, and the implementation details are not clear. To carry out the improvements, we write our own programs using C++, each with a MATLAB interface so that we can do various experiments easily in MATLAB. We separate the wavelet transform (WT) from the SPIHT coding, so that we can use various WTs and DC-level shifting. The programs for the WT are DWA (for analysis) and DWS (for synthesis), and the programs for the SPIHT coding are listed in Table 7.2. They are dynamic-linked libraries (.dll). The programs for the original SPIHT algorithm (SPIHTencode, SPIHTdecode, SPIHTacEncode, and SPIHTacDecode) are written strictly according to the procedure described in [2].

Table 7.2 Programs for the SPIHT coding

Program Name		SPIHT algorithm	Entropy Coding Method
Encoder	Decoder		
SPIHTencode	SPIHTdecode	original	none
SPIHTacEncode	SPIHTacDecode	original	arithmetic
ISPIHTencode	ISPIHTdecode	improved	none
ISPIHTacEncode	ISPIHTacDecode	improved	arithmetic

We use 5-scale bi-orthogonal 9/7 WT [3] for the SPIHT algorithm in the experiments, except as stated otherwise. The test images shown in chapter 3 (Barbara, Boat, Goldhill, Lena, Mandrill, Peppers, and Zelda) are used for the simulations.

Table 7.3 Performance of the SPIHT algorithm without the arithmetic coding for Goldhill

Coding Rate	PSNR (dB)		
(bpp)	Our implementation	Said & Pearlman's	difference
0.05	25.8346	26.0370	0.2024
0.10	27.5103	27.6737	0.1634
0.15	28.5424	28.7150	0.1726
0.20	29.3337	29.5289	0.1952
0.25	30.1003	30.2157	0.1154
0.30	30.7426	30.8348	0.0922
0.35	31.2318	31.3345	0.1027
0.40	31.6580	31.8073	0.1493
0.45	32.1063	32.2752	0.1689
0.50	32.5484	32.7064	0.1580
0.55	32.9554	33.0854	0.1300
0.60	33.3908	33.5135	0.1227
0.65	33.7877	33.8682	0.0805
0.70	34.1347	34.2178	0.0831
0.75	34.4676	34.5524	0.0848
0.80	34.7453	34.8429	0.0976
0.85	35.0219	35.1372	0.1153
0.90	35.2997	35.4293	0.1296
0.95	35.5614	35.7109	0.1495
1.00	35.8463	36.0027	0.1564
Average	32.2410	32.3744	0.1335

To see the performance of our implementation, we use SPIHTencode and SPIHTdecode (our implementation of the SPIHT algorithm without the arithmetic coding) to encode and decode the test images. The decoder stops after decoding a given length of the coded bit-streams, which is equivalent to a relevant coding rate. The results of PSNR at twenty equally-spaced coding rates of 0.05 ~ 1 bpp for Goldhill are listed in Table 7.3.

We also list the results using fastcode and fastdecd (Said and Pearlman's implementation of the SPIHT algorithm without the arithmetic coding) in Table 7.3 for comparison. As can be seen, Said and Pearlman's implementation outperforms our implementation at all coding rates. The difference is about 0.20 dB maximum, and 0.13 dB in average. The difference is also depicted in Figure 7.1.

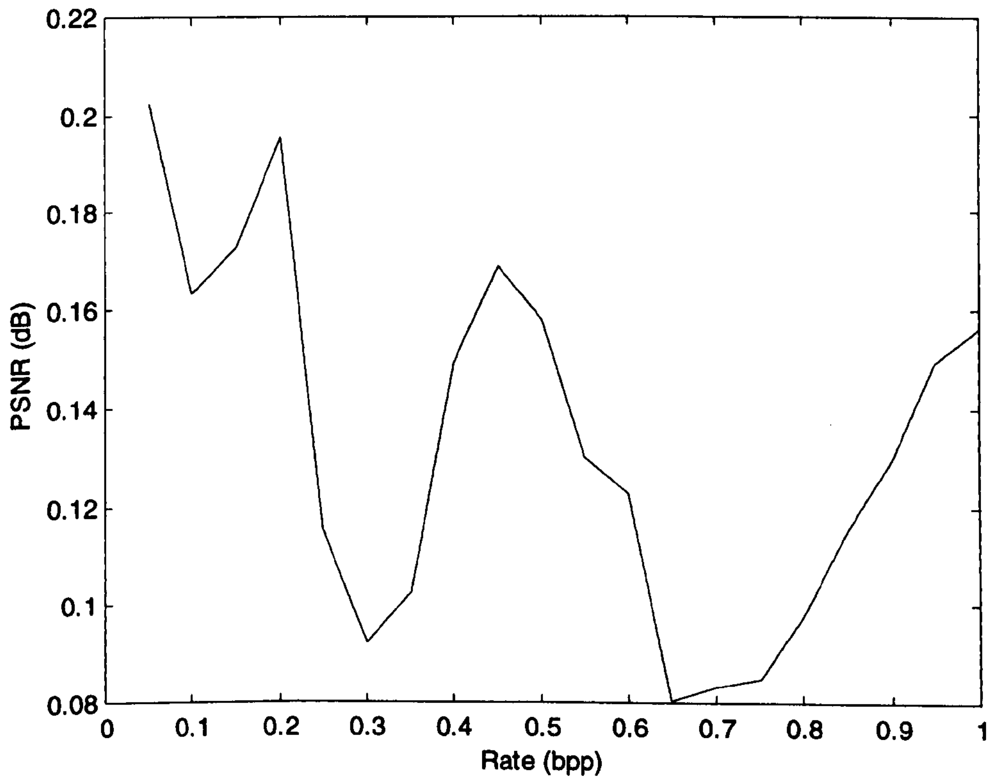


Figure 7.1 Performance difference of the SPIHT algorithm between Said&Pearlman's and our implementation for Goldhill

In the case of other test images, the results are similar. Said and Pearlman's implementation outperforms our implementation in all cases. We list the average differences of PSNR in Table 7.4. The differences are small. We use our implementation of the original SPIHT algorithm as the base to measure the performance gain of the improvements.

Table 7.4 Average difference of PSNR for various test images

Image	Barbara	Boat	Goldhill	Lena	Mandrill	Peppers	Zelda
Difference	0.1571	0.1606	0.1335	0.1227	0.1449	0.1148	0.1145

7.2 Performance of the SPIHT algorithm

In this section we study the performance of the SPIHT algorithm. First, we compare the SPIHT algorithm with the international image coding standards JPEG [4] and JPEG2000 [5]. Then we investigate the performance of the SPIHT algorithm using various levels of wavelet transform.

We use Said&Pearlman's programs "codetree" and "decdtree" in Table 7.1 for the SPIHT algorithm, the software from the independent JPEG group for JPEG [6], and the JasPer software for JPEG2000 [7]. The rate-distortion performance for Goldhill is depicted in Figure 7.2, and that for Zelda is in Figure 7.3. We can see that the SPIHT algorithm outperforms JPEG, and that the PSNR difference is about 2 dB. The performance of SPIHT and JPEG2000 are very close. The PSNR of SPIHT is a little bit higher than that of JPEG2000 at the ends of low and high coding rates.

The results for other test images are similar.

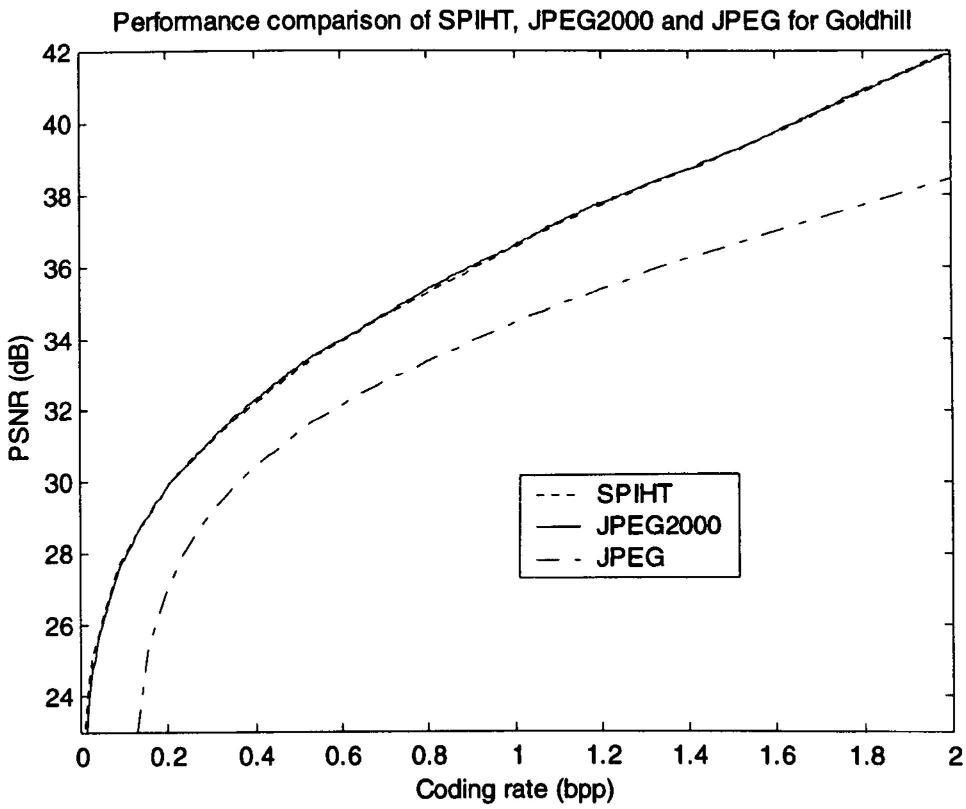


Figure 7.2 Performance comparison of SPIHT, JPEG and JPEG2000 image coding for Goldhill

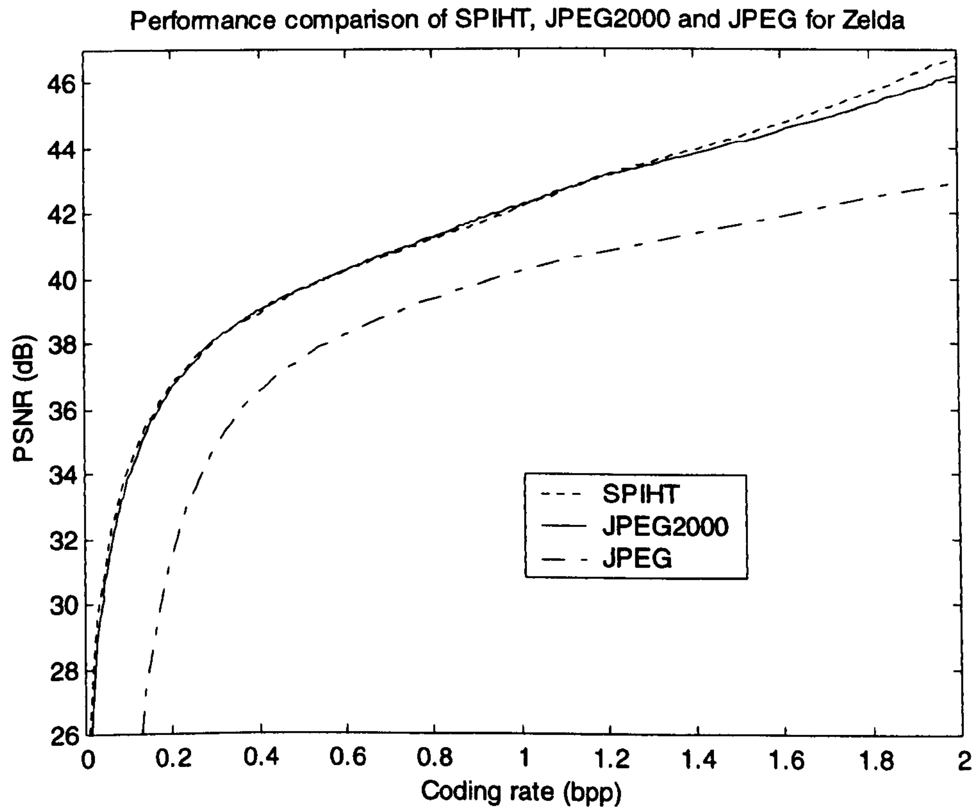


Figure 7.3 Performance comparison of SPIHT, JPEG and JPEG2000 image coding for Zelda

The basic encoding engine of JPEG2000 is based on the EBCOT algorithm [5]. JPEG2000 modified the EBCOT algorithm to reduce complexity [8]. The performance of the SPIHT algorithm and the EBCOT algorithm are also very close, as shown in Table 7.5. The results for EBCOT and SPIHT in the table are from [8]. The results for EBCOT are for bit-stream with only one layer, which are the best using octave decomposition but not SNR scalable. We calculate the results for SPIHT using "codetree" and "decdtree", and find that the results for Lena are the same as that from [8], but the results for Barbara are quite different with that from [8]. The results for Barbara calculated using "codetree" and "decdtree" are listed in Table 7.5 under SPIHT*. The difference may be due to different versions of the SPIHT algorithm or the Barbara image.

Table 7.5 Performance of SPIHT and EBCOT for Lena and Barbara

Coding Rate (bpp)	PSNR (dB)				
	Lena		Barbara		
	EBCOT	SPIHT	EBCOT	SPIHT	SPIHT*
0.0625	28.30	28.38	23.45	23.35	23.77
0.125	31.22	31.10	25.55	24.86	25.38
0.25	34.38	34.11	28.55	27.58	28.13
0.5	37.43	37.21	32.48	31.39	32.11
1.0	40.61	40.41	37.37	36.41	37.45

As mentioned in section 7.1, we usually use the 5-scale wavelet transform for the SPIHT algorithm in the experiments. Now we check the performance using various levels of wavelet transform. Since we cannot change the wavelet transform in Said and Pearlman's implementation (in fact we do not know the number of wavelet transform levels used in their programmes), we use our implementation here. Figure 7.4 and Figure 7.5 show the results for Goldhill and Zelda. The results for other test images are similar.

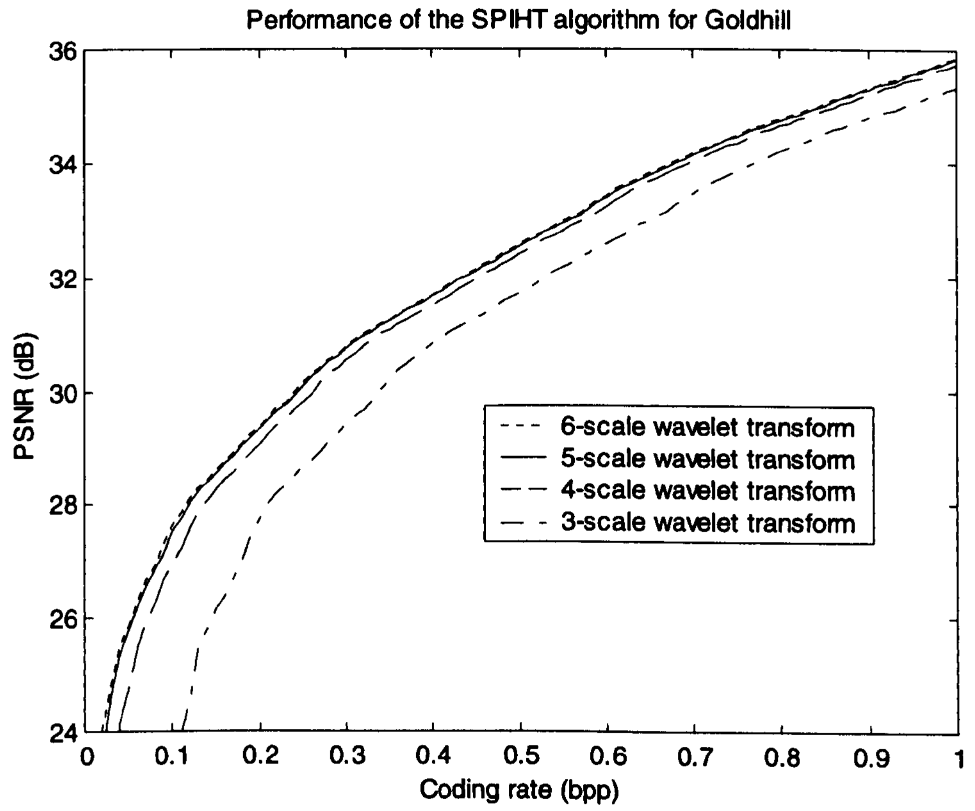


Figure 7.4 Performance of the SPIHT algorithm under various levels of wavelet transform for Goldhill

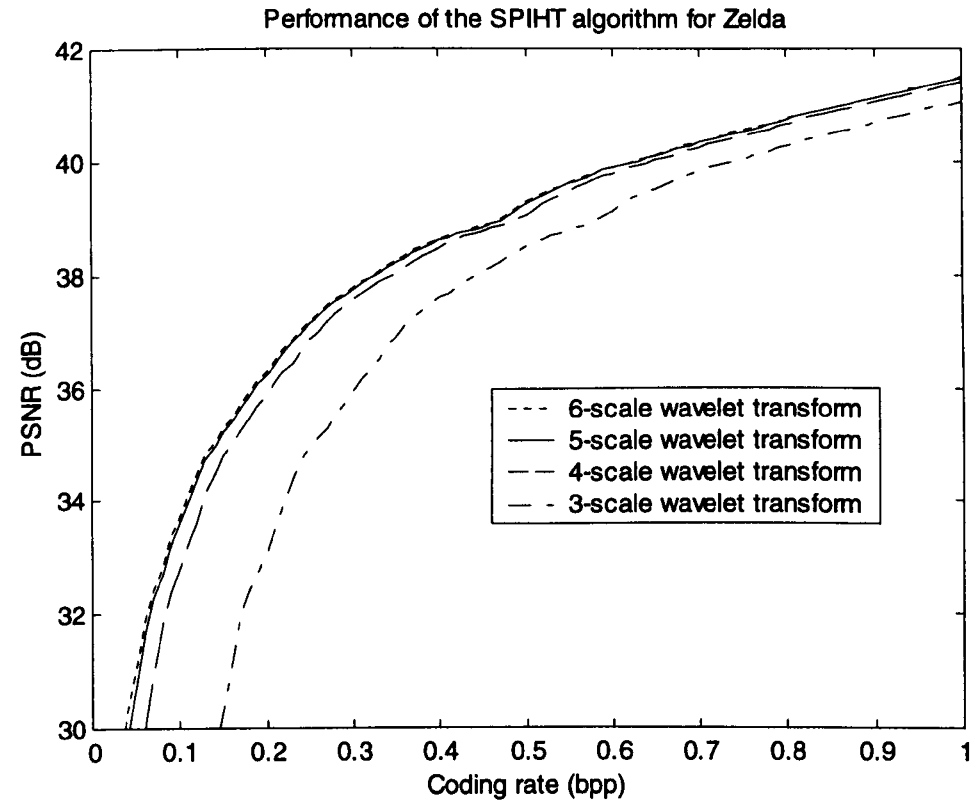


Figure 7.5 Performance of the SPIHT algorithm under various levels of wavelet transform for Zelda

Figure 7.4 and Figure 7.5 show that the performance of the SPIHT algorithm using high scale wavelet transform is better than that using low scale wavelet transform. But the performance difference between high levels of wavelet transform (such as 5 and 6-scale) is very small. We can also use 7 or even 8-scale wavelet transform for the test image whose size is 512 × 512, but their performance differences with 6-scale wavelet transform are so minute that we cannot differentiate them if they were to be depicted in Figure 7.4 and Figure 7.5. The 5-scale wavelet transform is good enough in the SPIHT algorithm for images of size 512 × 512.

7.3 Performance gain of the improvements to the SPIHT algorithm

The proposed improvements to the SPIHT algorithm include: (1) DC-level shifting; (2) Virtual trees; (3) Redundancy reduction by omitting predictable coding symbols; (4) Quantisation offset; (5) Pre-processing (for lossy image coding); (6) Speeding up the judgement of the significance of trees; (7) optimisation of the arithmetic coding. We examine the performance gain of individual and all improvements.

7.3.1 DC-level shifting

We compare the performance of the SPIHT algorithm with and without DC-level shifting. Various DC-level shifting schemes (described in chapter 5) are studied. SPIHTencode and SPIHTdecode are used for encoding and decoding of the wavelet coefficients (WC) after a 3-scale wavelet transform. The numerical results for Lena are shown in Table 7.6.

As Table 7.6 shows, with DC-level shifting, PSNR are higher than that without DC-level shifting (about 0.65 dB in average). The performance gains of various DC-level shifting schemes are depicted in Figure 7.6. The differences between the three DC-level shifting schemes are so small (no more than 0.007 dB in average) that the three curves in Figure 7.6 are almost identical. In general, the performance gain at low coding rate is higher than that at high coding rate.

The results for Lena are typical. The results for other test images are similar. Table 7.7 lists the average PSNR of the test images using various DC-level shifting. In most cases (5 out of 7), the SPIHT algorithm with fixed DC-level shifting in the image domain gives the best performance.

Table 7.6 Performance comparison for DC-level shifting (Lena)

Coding Rate (bpp)	PSNR (dB)			
	No DC-level shifting	Image Domain DC-level shifting		Transform Domain DC-level shifting
		Fixed (127)	Mean	
0.05	16.6260	21.0700	21.0540	21.0540
0.10	24.0120	26.0010	26.0200	26.0190
0.15	27.4100	28.6930	28.6940	28.6920
0.20	29.9240	30.6430	30.6410	30.6410
0.25	31.2730	31.9930	31.9910	31.9890
0.30	32.6850	33.2740	33.2730	33.2720
0.35	33.5900	33.9970	33.9990	33.9990
0.40	34.3290	34.7490	34.7530	34.7520
0.45	35.0360	35.4260	35.4250	35.4240
0.50	35.7740	36.0820	36.0820	36.0820
0.55	36.3200	36.5510	36.5470	36.5470
0.60	36.7080	36.9230	36.9220	36.9210
0.65	37.1130	37.3150	37.3140	37.3130
0.70	37.5090	37.7050	37.7060	37.7050
0.75	37.8600	37.9920	37.9920	37.9930
0.80	38.1350	38.2630	38.2640	38.2640
0.85	38.4400	38.6610	38.6590	38.6590
0.90	38.8550	39.0140	39.0130	39.0130
0.95	39.1420	39.2940	39.2930	39.2930
1.00	39.4230	39.5770	39.5770	39.5770
Average	34.0081	34.6611	34.6610	34.6605

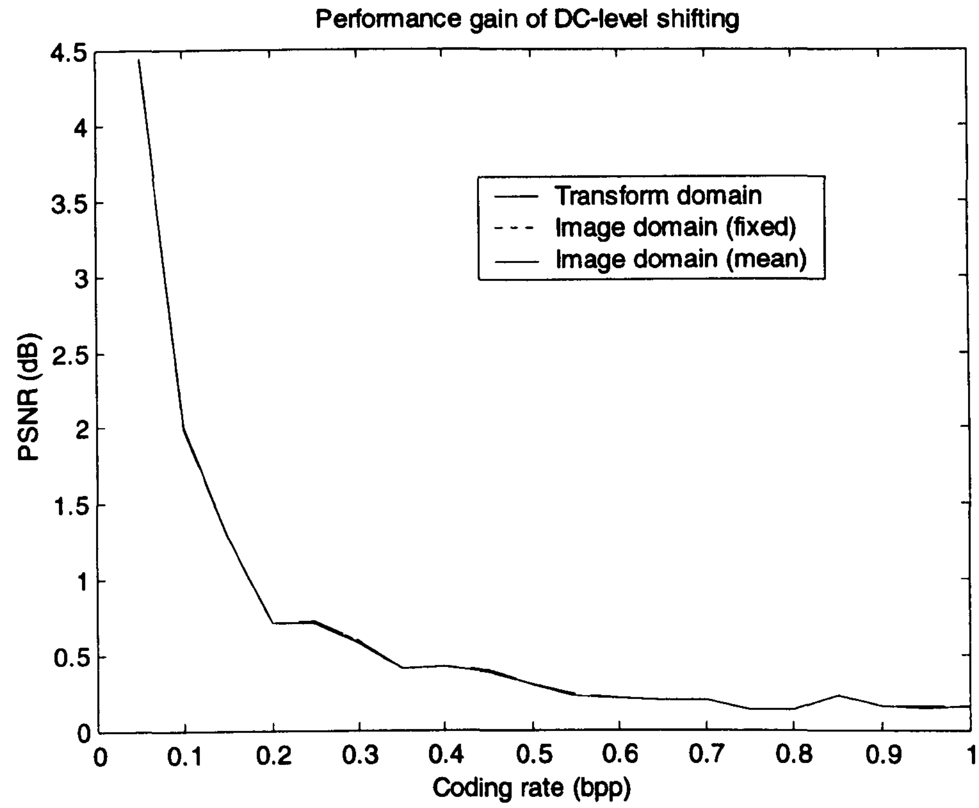


Figure 7.6 Performance gain of various DC-level shifting schemes for Lena

Table 7.7 Performance comparison for DC-level shifting (various test images)

Image	**Average PSNR (dB)**			
	No DC-level shifting	Image Domain DC-level shifting		Transform Domain DC-level shifting
		Fixed (127)	Mean	
Barbara	29.5321	30.1077	30.1077	**30.1119**
Boat	31.5664	**32.1732**	32.1537	32.1486
Goldhill	30.7096	**31.3514**	31.3492	31.3509
Lena	34.0081	**34.6611**	34.6610	34.6605
Mandrill	24.0390	**24.4393**	24.4382	24.4386
Peppers	32.9977	32.9989	**33.5976**	33.5958
Zelda	36.4599	**37.2203**	37.2114	37.2086

We conclude that DC-level shifting does improve the performance of the SPIHT algorithm, especially at low coding rates. The performance improvement at low coding rates is higher than that at high coding rates. We choose the fixed DC-level shifting in the image domain for the rest of the experiments, because it is the best in most cases and is the simplest. The maximum and the average performance gains of the fixed DC-level shifting in the image domain for various test images using 3-scale wavelet transform are listed in Table 7.8.

Table 7.8 Performance gain of DC-level shifting

Image	PSNR (dB)	
	Maximum	Average
Barbara	3.737897	0.575585
Boat	3.792362	0.606830
Goldhill	6.074123	0.641783
Lena	4.443772	0.653007
Mandrill	3.187370	0.400251
Peppers	0.298223	0.001281
Zelda	5.877013	0.760401

Now we vary the number of wavelet transform levels and check the performance gain. The results for Lena are depicted in Figure 7.7.

Figure 7.7 shows that the performance gains of DC-level shifting in the SPIHT algorithm changes a lot with the levels of wavelet transform. The performance gain using the low-scale wavelet transform is higher than that using the high-scale wavelet transform.

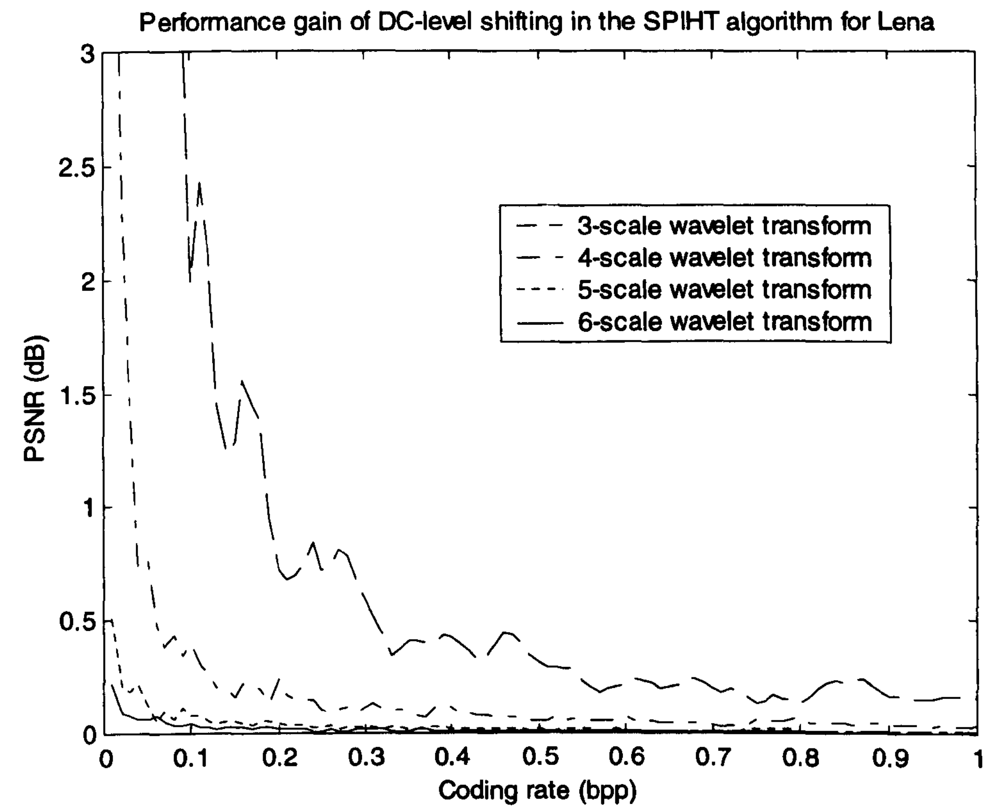


Figure 7.7 Performance gain of DC-level shifting in the SPIHT algorithm for Lena

The results for other test images are similar, except Peppers. There is little performance gain of DC-level shifting for Peppers in any case.

7.3.2 Virtual trees

In this section we use the virtual trees in SPIHT coding and check the performance of the improvement. The results for Boat using the 3-scale wavelet transform are listed in Table 7.9. The performance of the original SPIHT algorithm is also listed in the table.

Table 7.9 Performance of the SPIHT algorithm using the virtual trees for Boat

Coding Rate (bpp)	PSNR (dB)		
	Original SPIHT	Virtual tree	Difference
0.05	16.9517	20.7275	3.7759
0.10	22.5659	24.8655	2.2996
0.15	25.4006	26.7450	1.3444
0.20	26.9088	28.3708	1.4620
0.25	28.4762	29.3340	0.8579
0.30	29.3604	30.2867	0.9263
0.35	30.3386	31.2563	0.9178
0.40	31.3134	32.0209	0.7075
0.45	32.0381	32.5781	0.5400
0.50	32.5895	33.1471	0.5576
0.55	33.1588	33.7310	0.5722
0.60	33.7531	34.2944	0.5413
0.65	34.3242	34.9591	0.6349
0.70	34.9825	35.4376	0.4551
0.75	35.4579	35.9404	0.4825
0.80	35.9534	36.3217	0.3683
0.85	36.3192	36.7349	0.4157
0.90	36.7303	37.1308	0.4005
0.95	37.1264	37.5781	0.4517
1.00	37.5783	37.9765	0.3982
Average	31.5664	32.4718	0.9055

Table 7.9 shows that the improved SPIHT algorithm using the virtual trees outperforms the original SPIHT algorithm at all coding rates. The PSNR difference is about 3.8 dB maximum and 0.9 dB in average. The PSNR difference is also depicted in Figure 7.8.

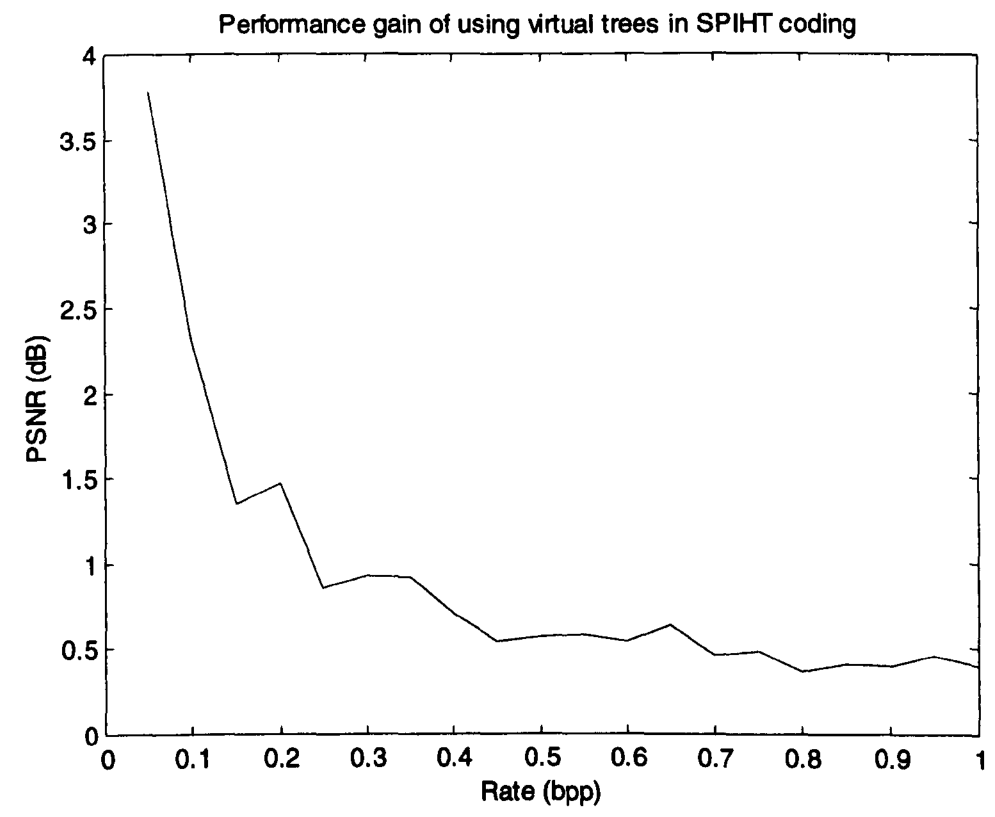


Figure 7.8 Performance gain of using the virtual trees in the SPIHT coding for Boat

As Figure 7.8 shows, the performance gain of using the virtual trees in the SPIHT coding at low coding rates is higher than that at high coding rates in general. This suggests that the virtual tree technique is more efficient for the low-rate image coding than for the high-rate. This can be explained by the fact that the virtual tree technique mainly reduces the coding symbols at the beginning of the SPIHT coding. The total number of symbols reduced is limited. As the coding rate goes higher, the ratio of the reduced symbols in total coding symbols becomes less, so that the reduction becomes less important.

Now we vary the number of wavelet transform levels and check the performance gain. The results are depicted in Figure 7.9.

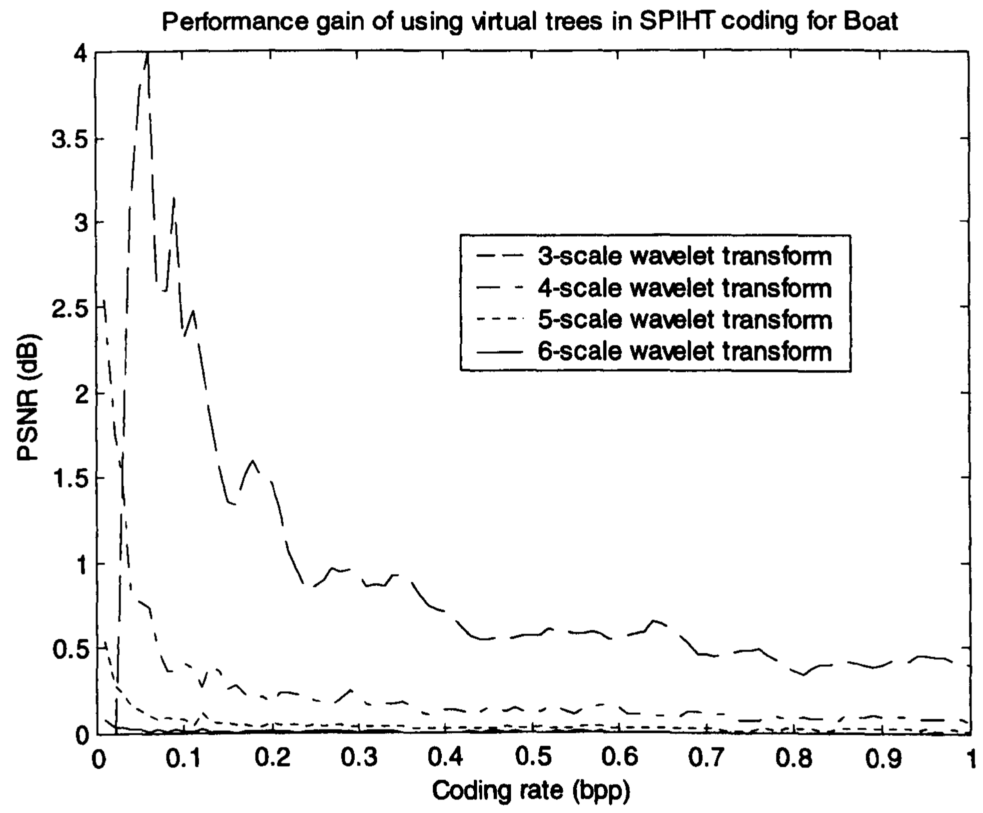


Figure 7.9 Performance gain of using virtual trees in SPIHT coding for Boat

Figure 7.9 shows that the performance gain of using the virtual trees in the SPIHT algorithm changes a lot with levels of wavelet transform. The performance gain using the low-scale wavelet transform is higher than that using the high-scale wavelet transform.

This is explained in the following discussion. Recall that the virtual tree technique reduces the total number of initial trees in the LIS (list of the insignificant sets). As the number of wavelet transform levels increases, the size of subbands LL, HL0, LH0 and HH0 decreases, and the number of initial trees in the original SPIHT algorithm also decreases, thus there is less to be gained by using the virtual trees. In the extreme case, for the 512 × 512 image, after an 8-scale wavelet transform, the size of LL, HL0, LH0 or HH0 is 2 × 2. There exists no other initial virtual tree in the LIS except V_0, which is actually the initial D tree in the original SPIHT algorithm. So, there is no performance gain by using the virtual trees in the extreme case.

In Figure 7.9, there is no performance gain at very low coding rates (e.g. 0.02 bpp) when using the 3-scale wavelet transform. This is because the SPIHT coding starts from the LIP, which includes all the wavelet coefficients on LL initially. The virtual trees in the LIS have not been coded yet during this period. The coding rate at the beginning of the LIS scan depends on the size of LL. For 3-scale wavelet transform, the size of LL is 64×64, the coding rate of the initial wavelet coefficients in the LIP is about 0.02 bpp for a 512×512 image, or between 1/64 bpp and 1/32 bpp accurately. For higher scale wavelet transforms, this coding rate is too low to be shown in Figure 7.9.

The results for Boat are typical. The results for other test images are similar. Table 7.10 lists the maximum and the average performance gain of using the virtual trees in the SPIHT coding for various test images. The 3-scale wavelet transform is used here.

Table 7.10 Performance gain of using the virtual trees in the SPIHT coding for various test images

Image	PSNR (dB)	
	Maximum	Average
Barbara	3.696666	0.902046
Boat	3.775896	0.905480
Goldhill	5.485875	0.844814
Lena	4.583156	1.013042
Mandrill	2.648991	0.489282
Peppers	4.494274	0.892464
Zelda	4.966644	1.109960

7.3.3 Omit predictable coding symbols

In this section we omit the predictable symbols in the SPIHT coding and check the performance of the improvement. The results for Barbara using the 5-scale wavelet transform are listed in Table 7.11. The performance of the original SPIHT algorithm is also listed in the table.

Table 7.11 shows that the improved SPIHT algorithm omitting the predictable symbols outperforms the original SPIHT algorithm at all coding rates. The PSNR difference is about 0.14 dB maximum and 0.105 dB in average. The PSNR difference is also depicted in Figure 7.10 (the dotted line).

Table 7.11 Performance of the SPIHT algorithm omitting the predictable symbols for Barbara

Coding Rate (bpp)	PSNR (dB)		
	Original SPIHT	Omit predictable	Difference
0.05	23.1742	23.1977	0.0234
0.10	24.3055	24.4138	0.1084
0.15	25.4207	25.5598	0.1391
0.20	26.5952	26.6583	0.0631
0.25	27.5392	27.6244	0.0853
0.30	28.4817	28.5721	0.0904
0.35	29.3690	29.5092	0.1402
0.40	30.2752	30.3638	0.0886
0.45	30.8993	30.9981	0.0987
0.50	31.5772	31.6474	0.0703
0.55	32.1018	32.2339	0.1321
0.60	32.7024	32.8026	0.1002
0.65	33.2964	33.4335	0.1371
0.70	33.8864	34.0216	0.1353
0.75	34.4859	34.6215	0.1356
0.80	34.9636	35.0753	0.1117
0.85	35.4169	35.5558	0.1389
0.90	35.9623	36.0483	0.0860
0.95	36.3544	36.4610	0.1067
1.00	36.7688	36.8799	0.1112
Average	31.1788	31.2839	0.1051

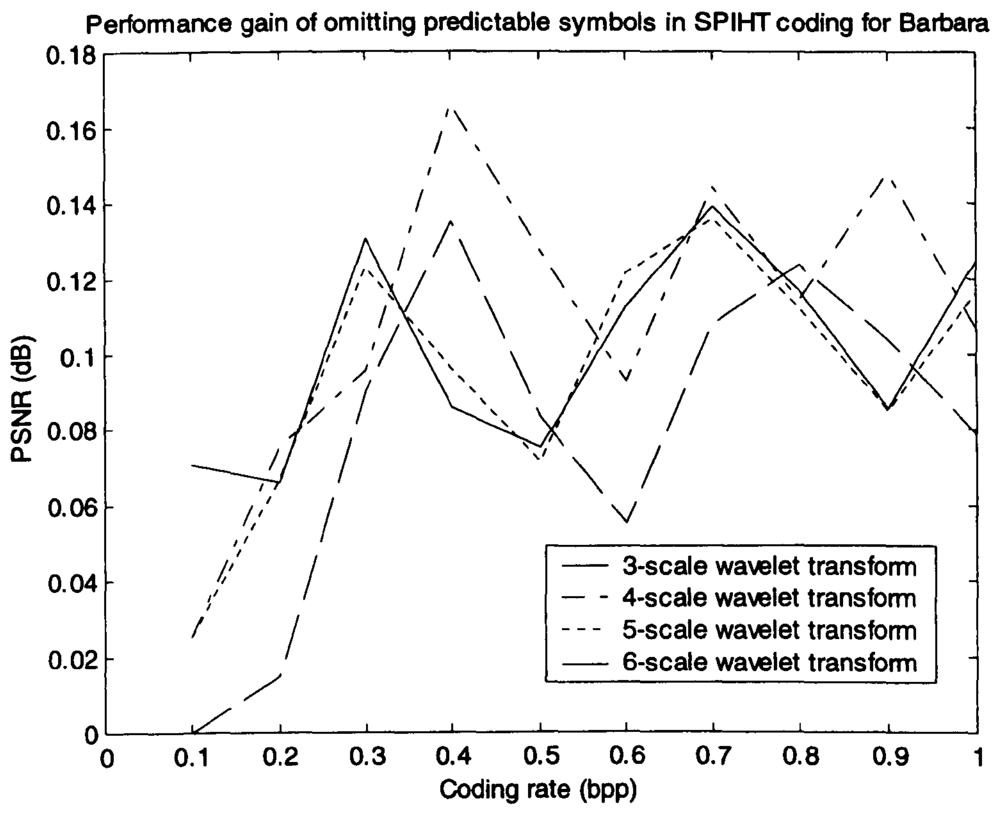


Figure 7.10 Performance gain of omitting the predictable symbols in SPIHT coding for Barbara

Now we vary the number of wavelet transform levels and check the performance gain. The results are depicted in Figure 7.10. As can be seen, the performance gain changes with the levels of wavelet transform, but the difference is not significant. In fact, there are only ten values for each curve in the figure. If more values were depicted, the curves would mix together that we could hardly distinguish any curve from the others.

The results for Barbara are typical. The results for other test images are similar. Table 7.12 lists the maximum and the average performance gain of omitting the predictable symbols in the SPIHT coding for various test images. The 5-scale wavelet transform is used here.

Table 7.12 Performance gain of omitting the predictable symbols in the SPIHT coding for various test images

Image	PSNR (dB)	
	Maximum	Average
Barbara	0.145604	0.084623
Boat	0.098055	0.046216
Goldhill	0.091744	0.040003
Lena	0.099213	0.037865
Mandrill	0.120768	0.056696
Peppers	0.130776	0.036815
Zelda	0.075692	0.024998

7.3.4 Quantisation offset

In this section we offset the quantisation of the wavelet coefficients in the SPIHT coding and check the performance of the improvement. The results for Mandrill using the 5-scale wavelet transform are listed in Table 7.13. The performance of the original SPIHT algorithm is also listed in the table.

Table 7.13 shows that the improved SPIHT algorithm with the quantisation offset outperforms the original SPIHT algorithm at all coding rates. The PSNR difference is about 0.12 dB maximum and 0.05 dB in average. The PSNR difference is also depicted in Figure 7.11 (the dotted curve).

Table 7.13 Performance of the SPIHT algorithm with the quantisation offset for Mandrill

Coding Rate (bpp)	PSNR (dB)		
	Original SPIHT	Quantisation offset	Difference
0.05	20.0730	20.0967	0.0237
0.10	21.0524	21.0795	0.0271
0.15	21.6538	21.6899	0.0362
0.20	22.2370	22.2875	0.0505
0.25	22.7641	22.8345	0.0704
0.30	23.3261	23.3630	0.0369
0.35	23.8040	23.8474	0.0433
0.40	24.2129	24.2617	0.0489
0.45	24.6180	24.6726	0.0546
0.50	25.0210	25.0835	0.0625
0.55	25.4234	25.4954	0.0721
0.60	25.8275	25.9135	0.0860
0.65	26.2347	26.3362	0.1015
0.70	26.5537	26.6730	0.1193
0.75	26.9899	27.0141	0.0242
0.80	27.3037	27.3286	0.0248
0.85	27.6189	27.6456	0.0266
0.90	27.9491	27.9785	0.0294
0.95	28.2585	28.2923	0.0338
1.00	28.5403	28.5784	0.0380
Average	24.9731	25.0236	0.0505

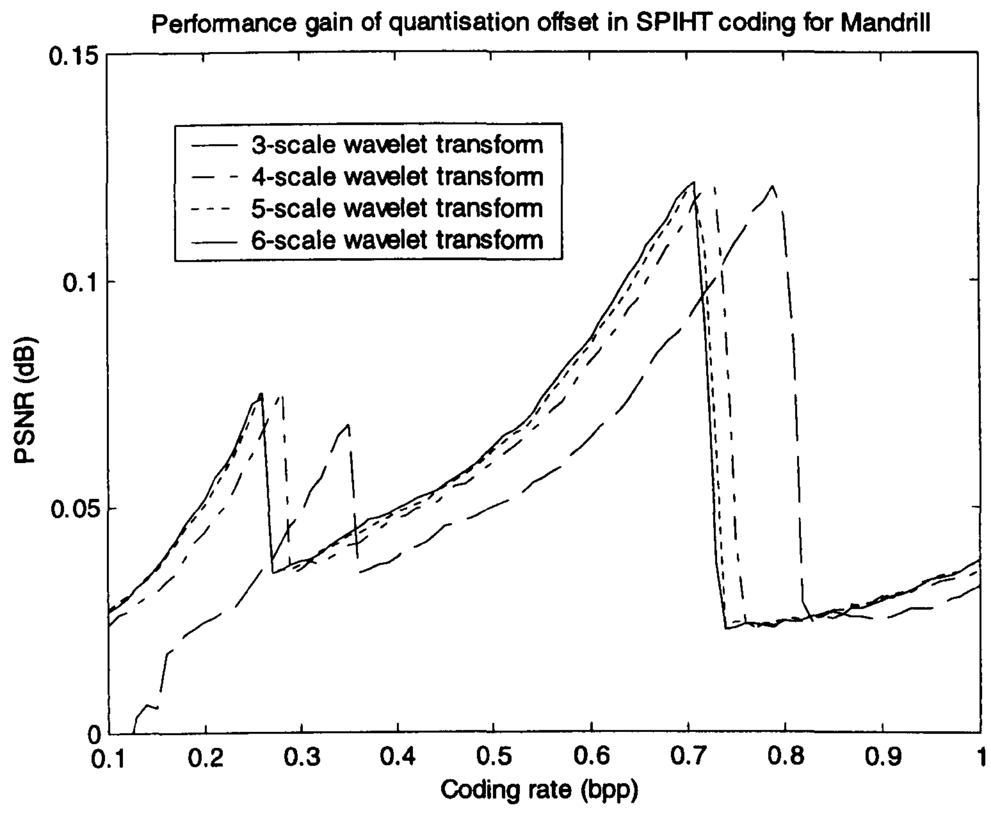


Figure 7.11 Performance gain of the quantisation offset in the SPIHT coding for Mandrill

Now we vary the number of wavelet transform levels and check the performance gain. The results are depicted in Figure 7.11. As can be seen, the performance gain changes with the levels of wavelet transform, but the difference is not significant. The results for Mandrill are typical. The results for other test images are similar. Table 7.14 lists the maximum and the average performance gain of the quantisation offset in the SPIHT coding for various test images, using the 5-scale wavelet transform.

Table 7.14 Performance gain of the quantisation offset in the SPIHT coding for various test images

Image	PSNR (dB)	
	Maximum	Average
Barbara	0.110525	0.024564
Boat	0.063191	0.029683
Goldhill	0.065569	0.033288
Lena	0.059797	0.024154
Mandrill	0.119306	0.050491
Peppers	0.088732	0.029296
Zelda	0.050236	0.029173

7.3.5 Pre-processing for lossy image coding

In this section we apply pre-processing to the SPIHT algorithm and check the performance of the improvement. The parameter β for pre-processing is set to be 2. The results for Peppers using the 5-scale wavelet transform are listed in Table 7.15. The performance of the original SPIHT algorithm is also listed in the table.

Table 7.15 shows that the improved SPIHT algorithm with pre-processing outperforms the original SPIHT algorithm at almost all coding rates (with only one exception at 0.25 bpp). The PSNR difference is about 0.41 dB maximum and 0.16 dB in average. The PSNR difference is also depicted in Figure 7.12 (the dotted curve).

Table 7.15 Performance of the SPIHT algorithm with pre-processing for Peppers

Coding Rate (bpp)	PSNR (dB)		
	Original SPIHT	Pre-processing	Difference
0.05	26.1578	26.1640	0.0062
0.10	29.0917	29.1475	0.0559
0.15	30.6949	30.7341	0.0392
0.20	32.0638	32.1671	0.1033
0.25	33.0086	32.9956	-0.0130
0.30	33.5823	33.6950	0.1126
0.35	34.2085	34.2889	0.0804
0.40	34.6631	34.8558	0.1927
0.45	35.1805	35.2286	0.0481
0.50	35.5239	35.5572	0.0332
0.55	35.7850	35.8588	0.0737
0.60	36.0230	36.1756	0.1526
0.65	36.2701	36.4346	0.1646
0.70	36.4936	36.6865	0.1929
0.75	36.6780	36.9741	0.2961
0.80	36.8881	37.2709	0.3828
0.85	37.0801	37.4929	0.4128
0.90	37.2780	37.6859	0.4079
0.95	37.5560	37.8602	0.3042
1.00	37.8815	38.0384	0.1569
Average	34.6054	34.7656	0.1602

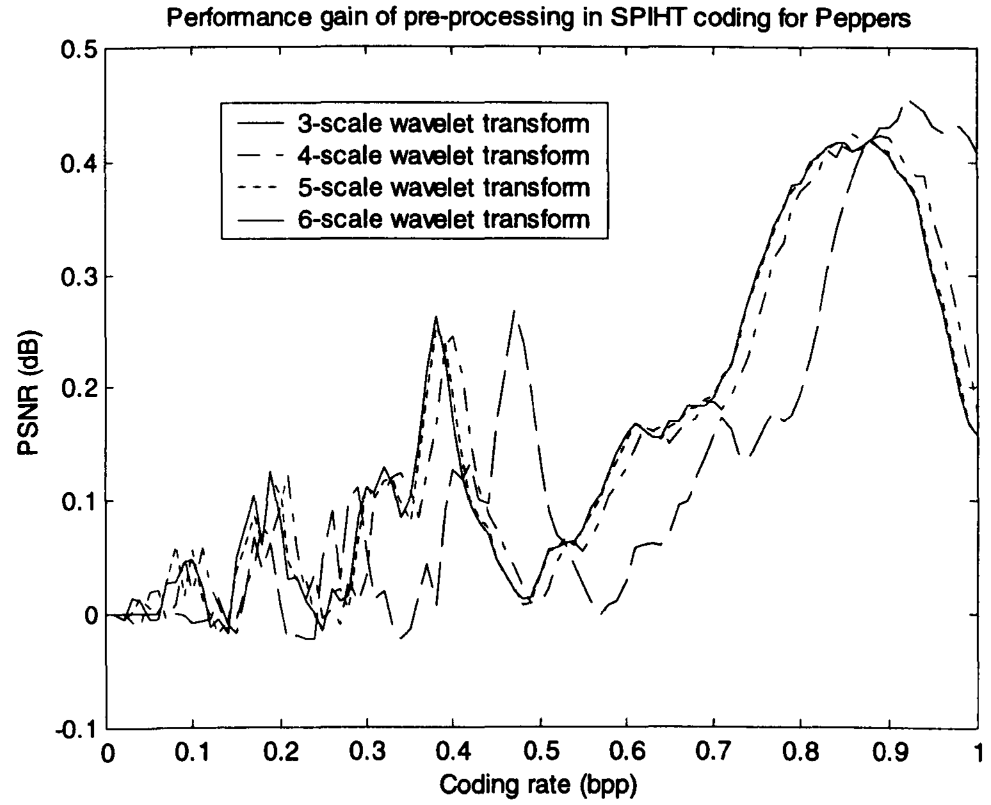


Figure 7.12 Performance gain of pre-processing in the SPIHT algorithm for Peppers

Now we vary the number of wavelet transform levels and check the performance gain. The results are depicted in Figure 7.12. As can be seen, the performance gain changes with the levels of wavelet transform, but the difference is not significant. The results for Peppers are typical. The results for other test images are similar. Table 7.16 lists the maximum and the average performance gain of the pre-processing in the SPIHT coding for various test images. The 5-scale wavelet transform is used here.

Table 7.16 Performance gain of the pre-processing in the SPIHT algorithm for various test images

Image	PSNR (dB)	
	Maximum	Average
Barbara	0.284992	0.102971
Boat	0.386628	0.150213
Goldhill	0.340984	0.120651
Lena	0.321143	0.128116
Mandrill	0.215864	0.071373
Peppers	0.412769	0.160151
Zelda	0.453334	0.192245

7.3.6 Arithmetic coding

In this section we optimise arithmetic coding for the SPIHT algorithm and check the performance of the optimisation. For the groups of wavelet coefficients (WC) in the LIP (list of insignificant WC), we use one model for each pattern of spatial orientation occupation, instead of one model for each group containing the same number of WCs. We also adjust the initial frequencies and the maximum frequency for each arithmetic coding model. The optimisation is made for the 5-scale wavelet transform. The results for Goldhill are listed in Table 7.17. The performance of the SPIHT algorithm without optimisation for arithmetic coding is also listed in the table.

Table 7.17 Optimisation of arithmetic coding (Goldhill)

Coding Rate (bpp)	PSNR (dB)		
	Original	Optimised	Difference
0.05	25.9534	25.9716	0.0182
0.10	27.7620	27.7864	0.0245
0.15	28.7258	28.7467	0.0209
0.20	29.6183	29.6464	0.0280
0.25	30.4265	30.4583	0.0318
0.30	31.0227	31.0459	0.0232
0.35	31.4819	31.5215	0.0396
0.40	31.9560	31.9959	0.0399
0.45	32.4406	32.4814	0.0407
0.50	32.9226	32.9610	0.0384
0.55	33.4013	33.4745	0.0732
0.60	33.8103	33.8463	0.0360
0.65	34.1662	34.2015	0.0354
0.70	34.5159	34.5475	0.0316
0.75	34.8009	34.8454	0.0445
0.80	35.1073	35.1590	0.0518
0.85	35.4043	35.4427	0.0384
0.90	35.6983	35.7487	0.0504
0.95	36.0057	36.0533	0.0476
1.00	36.3132	36.3622	0.0489
Average	32.5767	32.6148	0.0382

Table 7.17 shows that the optimisation gets some performance gain at all coding rates. The PSNR difference is about 0.07 dB maximum and 0.04 dB in average. The PSNR difference is also depicted in Figure 7.13.

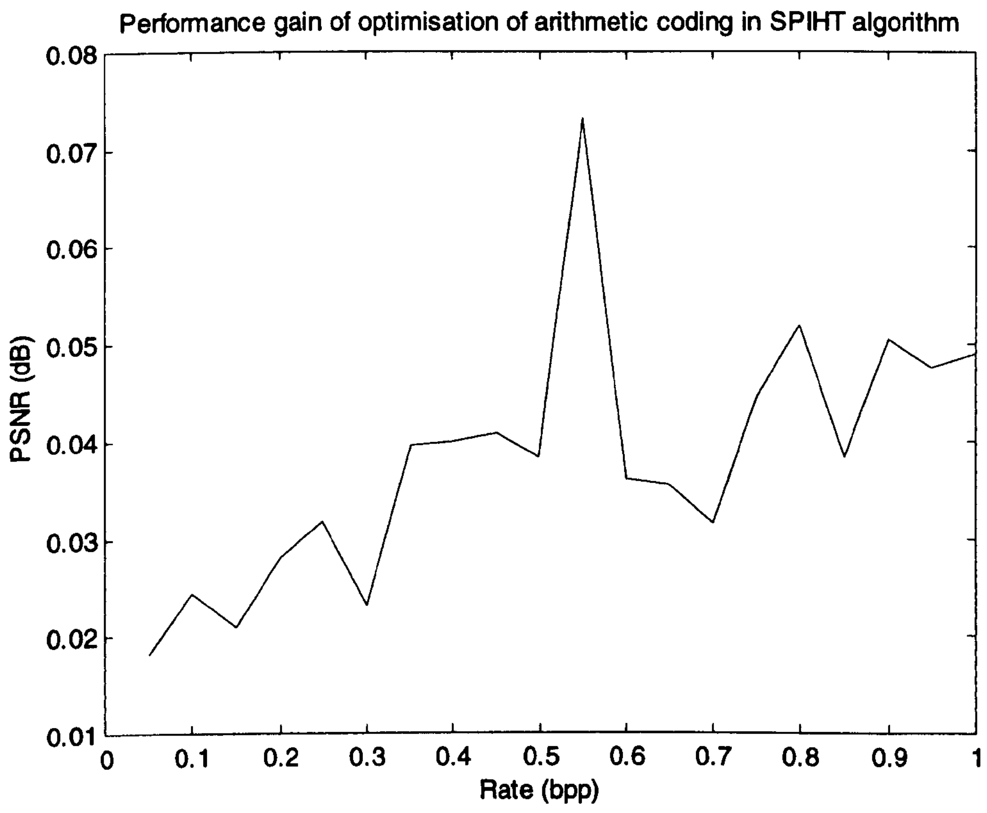


Figure 7.13 Performance gain of the optimisation of the arithmetic coding in the SPIHT algorithm for Goldhill

The results for Goldhill are typical. The results for other test images are similar. Table 7.18 lists the maximum and the average performance gain of the optimisation of arithmetic coding in the SPIHT algorithm for various test images.

Table 7.18 Performance gain of the optimisation of arithmetic coding in the SPIHT algorithm for various test images

Image	PSNR (dB)	
	Maximum	Average
Barbara	0.066176	0.043427
Boat	0.102048	0.051910
Goldhill	0.073152	0.038158
Lena	0.056916	0.039462
Mandrill	0.060098	0.035692
Peppers	0.050691	0.034181
Zelda	0.063511	0.034469

7.3.7 Judgement of the significance of trees

In this section we compare the speed of the SPIHT encoding using two different schemes to judge the significance of trees, namely, the proposed scheme and the direct judgement.

We measure the speed by the running time of the relevant program modules. The programs run on a PC with a GenuineIntel 350MHz CPU, 64M byte memory and Windows NT 4.0. To get accurate results, we close all other applications in the system. But we still get different running times for the same program in repeated experiments.

The results of the original SPIHT algorithm judging the significance of trees directly for Goldhill (using 5-scale wavelet transform) at coding rate of 1.0 bpp in twenty repeated experiments are listed in Table 7.19. The running time is for SPIHT encoding only, excluding the wavelet transform and the MATLAB interface.

Table 7.19 Speed of the original SPIHT algorithm judging the significance of trees directly (Goldhill @ 1.0 bpp)

Experiment	Running time of SPIHT encoding (s)
1	0.220
2	0.280
3	0.221
4	0.221
5	0.220
6	0.220
7	0.220
8	0.210
9	0.220
10	0.210
11	0.220
12	0.211
13	0.220
14	0.220
15	0.211
16	0.221
17	0.210
18	0.210
19	0.211
20	0.220

Without interference and measuring error, the actual running time of the program with the same input data should be the same in the repeated experiments. The measurement is done by reading the clock of the PC at the beginning and the end of the program and calculating the difference. A tick of the clock is 0.01 seconds. So the reading error of the beginning and the end time is less than 0.01 seconds (±0.01 s), the error of the measured running time is less than 0.02 seconds (±0.02 s), and the difference between the measured running time should be less than 0.04 seconds (±0.02 s). But this is not true in Table 7.19, as well as in other experiments. We conclude that there must be some interference in the system.

Since there is no other application program running on the PC while we are measuring running time in MATLAB, the only possible interference is from the operating system. It is a networked PC, so this could cause the interference. Anyway, the interference can only slow down the running of the program and increase the measured running time. So, the true running time is within the 0.04 seconds above the minimum measured results (valid range).

If we repeat the measuring experiments, the actual running time is likely to appear most frequently in the measured results. Studying Table 7.19 and the results of other experiments, we see that the measured running time which appears most frequently is the minimum or close to the minimum. We take the most frequent running time as the final result. An alternative is to calculate the average value of the measured running times in the valid range. The two methods get (almost) the same results. For example, it is 0.22 seconds in Table 7.19.

Table 7.20 and Table 7.21 list the running time of the SPIHT encoding for the test images at various coding rates, judging the significance of trees directly and by the proposed scheme respectively. We can see that in each table, for various images, the running time of the SPIHT encoding is almost the same at the same coding rate.

Table 7.20 Running time (seconds) of the original SPIHT encoding judging the significance of trees directly

Rate (bpp)	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Barbara	0.12	0.15	0.17	0.19	0.21	0.23	0.24	0.26	0.28	0.29
Boat	0.13	0.15	0.18	0.20	0.22	0.24	0.25	0.27	0.29	0.30
Goldhill	0.13	0.16	0.18	0.20	0.22	0.24	0.25	0.27	0.28	0.30
Lena	0.14	0.17	0.19	0.21	0.22	0.24	0.26	0.28	0.29	0.30
Mandrill	0.13	0.17	0.19	0.21	0.22	0.24	0.26	0.27	0.29	0.31
Peppers	0.14	0.17	0.19	0.21	0.23	0.24	0.25	0.28	0.29	0.31
Zelda	0.14	0.17	0.19	0.21	0.23	0.25	0.26	0.28	0.29	0.30

Table 7.21 Running time (seconds) of the SPIHT encoding using the proposed scheme to judge the significance of trees

Rate (bpp)	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Barbara	0.07	0.08	0.10	0.11	0.12	0.13	0.15	0.16	0.18	0.19
Boat	0.07	0.08	0.09	0.11	0.13	0.14	0.15	0.17	0.18	0.19
Goldhill	0.07	0.08	0.10	0.11	0.13	0.14	0.15	0.17	0.18	0.20
Lena	0.07	0.08	0.10	0.11	0.12	0.14	0.15	0.17	0.18	0.20
Mandrill	0.07	0.08	0.10	0.11	0.13	0.14	0.15	0.17	0.18	0.19
Peppers	0.07	0.08	0.10	0.12	0.13	0.14	0.15	0.17	0.18	0.20
Zelda	0.07	0.08	0.10	0.11	0.12	0.14	0.15	0.17	0.18	0.19

We depict the running time of SPIHT encoding at various coding rates for Goldhill in Figure 7.14. As can be seen, SPIHT encoding using the proposed scheme to judge the significance of trees runs much faster than that by direct judgement.

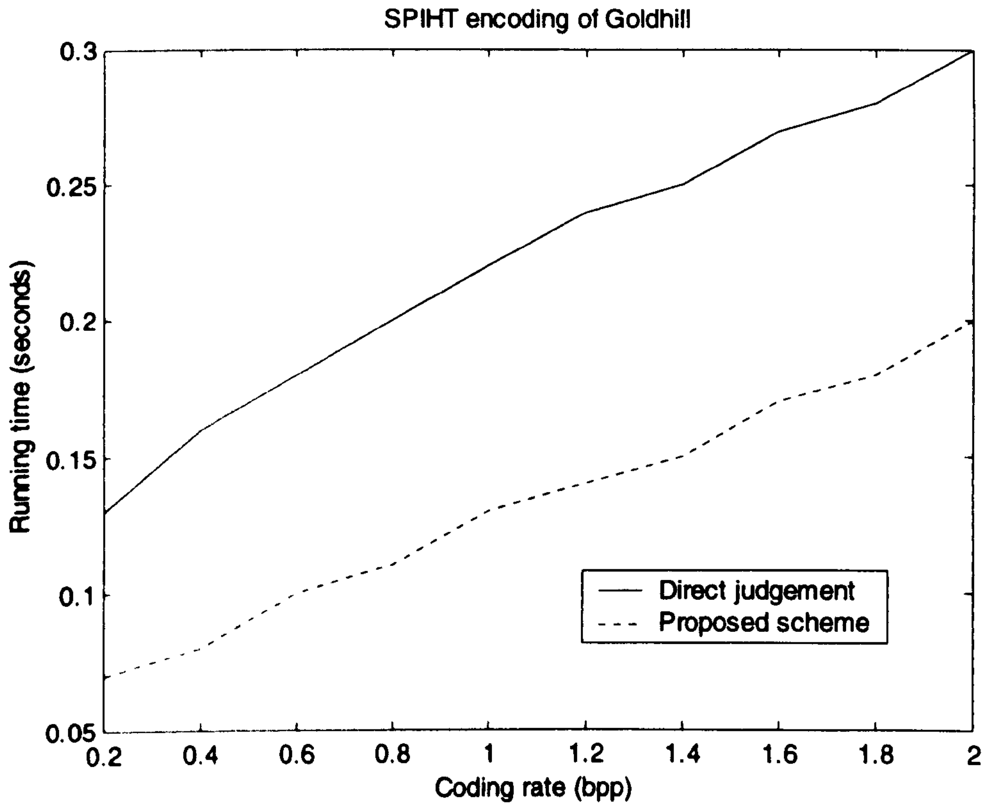


Figure 7.14 Running time of SPIHT encoding for Goldhill

7.3.8 Overall rate-distortion performance gain

In this section we apply all the improvements to the SPIHT algorithm and check the rate-distortion performance. The results for Zelda, using the 3-scale wavelet transform, and without arithmetic coding, are depicted in Figure 7.15. The performance of the original SPIHT algorithm is also depicted in the figure.

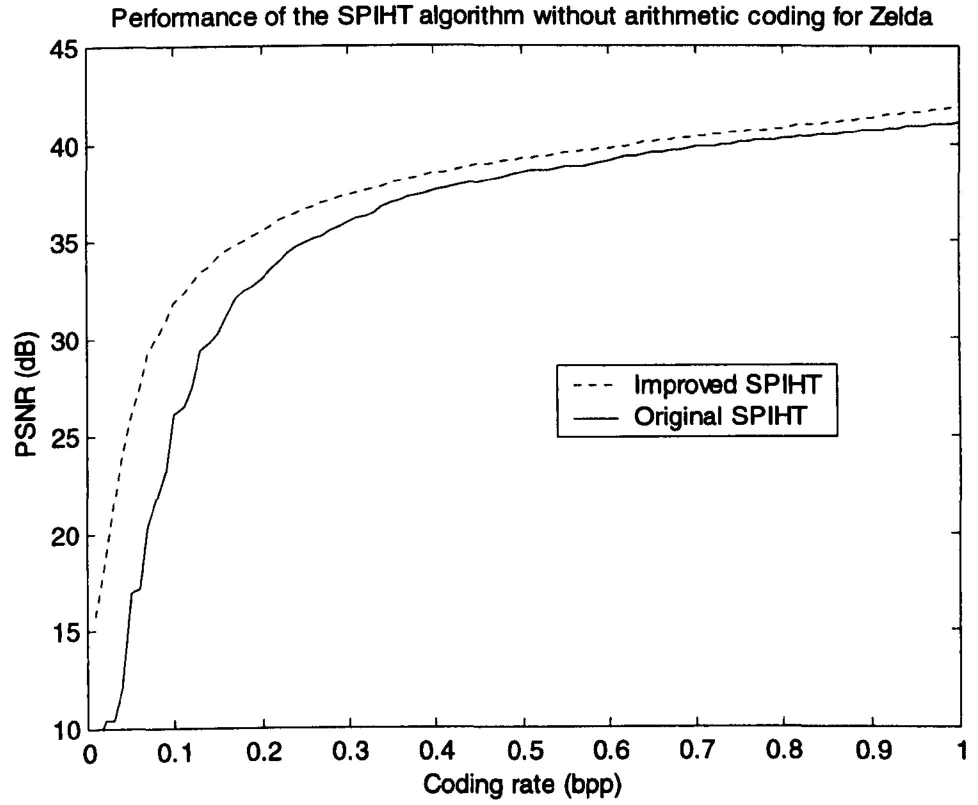


Figure 7.15 Performance of the SPIHT algorithm without arithmetic coding for Zelda

The results show that the improved SPIHT algorithm outperforms the original SPIHT algorithm at all coding rates. Figure 7.16 depicts the performance gain (the dashed curve). The curve is very much alike with that for DC-level shifting and the virtual trees. In fact, the performance gain for 3-scale wavelet transform is mainly the results of DC-level shifting and the virtual tree technique, especially at low coding rates. The performance gain is up to 11.8 dB at 0.04 bpp.

Figure 7.17 and Figure 7.18 show the reconstructed image of Zelda coded by the original and the improved SPIHT at 0.1 bpp respectively. Figure 7.18 is much clearer than Figure 7.17, due to the improvements to the SPIHT algorithm.

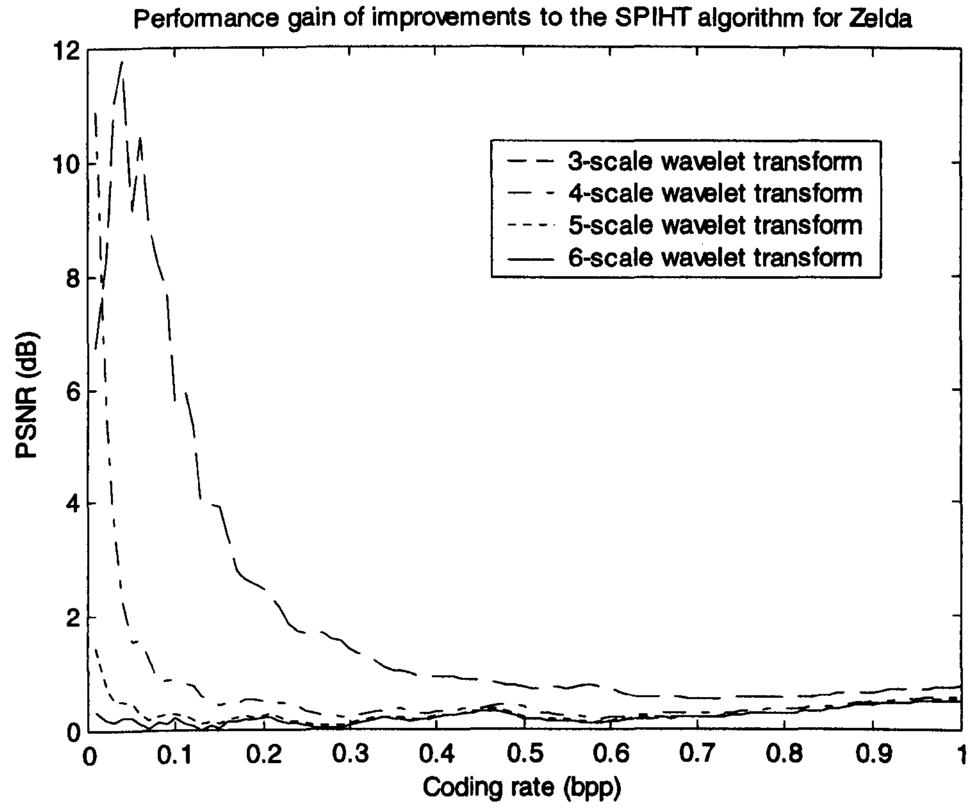


Figure 7.16 Performance gain of the improvements to the SPIHT algorithm without arithmetic coding for Zelda

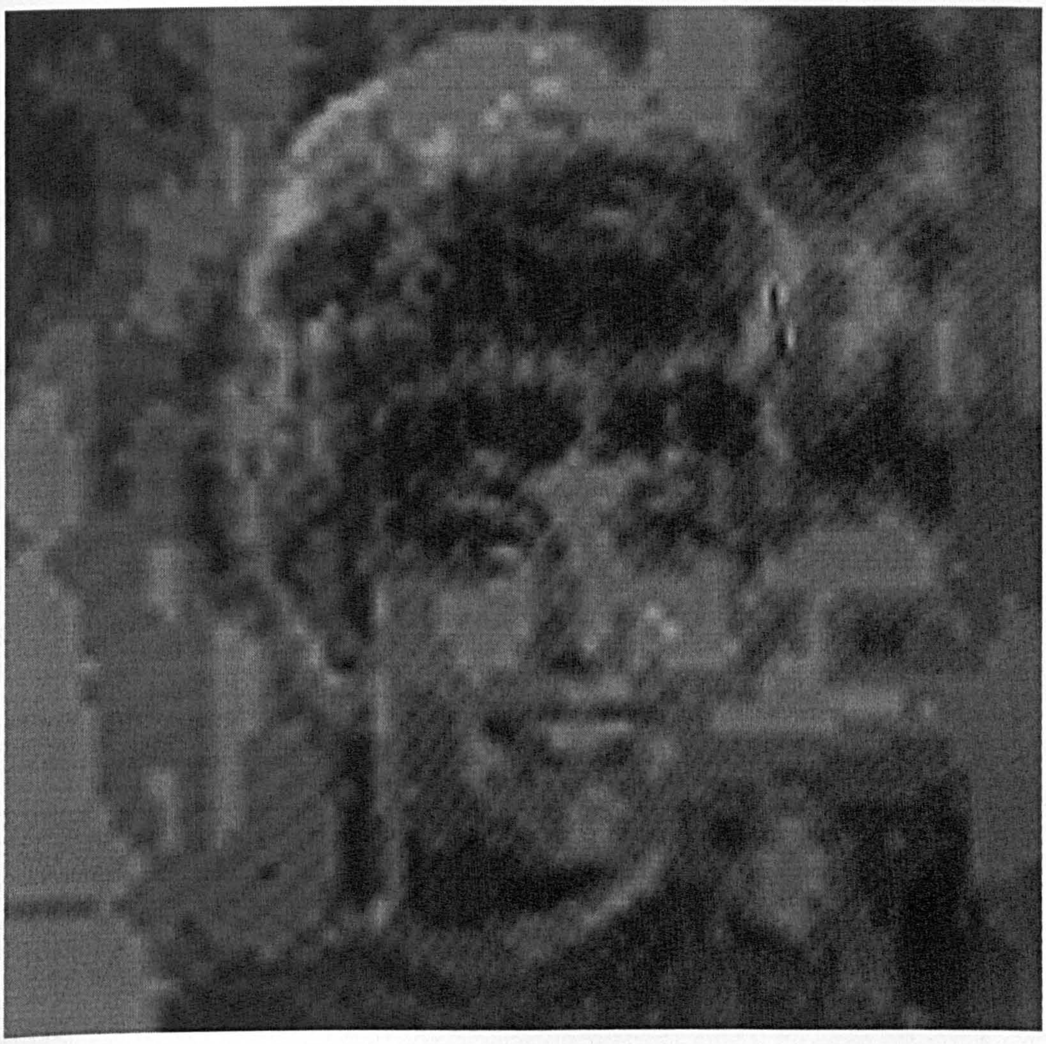


Figure 7.17 Reconstructed Zelda, coded by the original SPIHT at 0.1 bpp

(PSNR 26.1 dB)

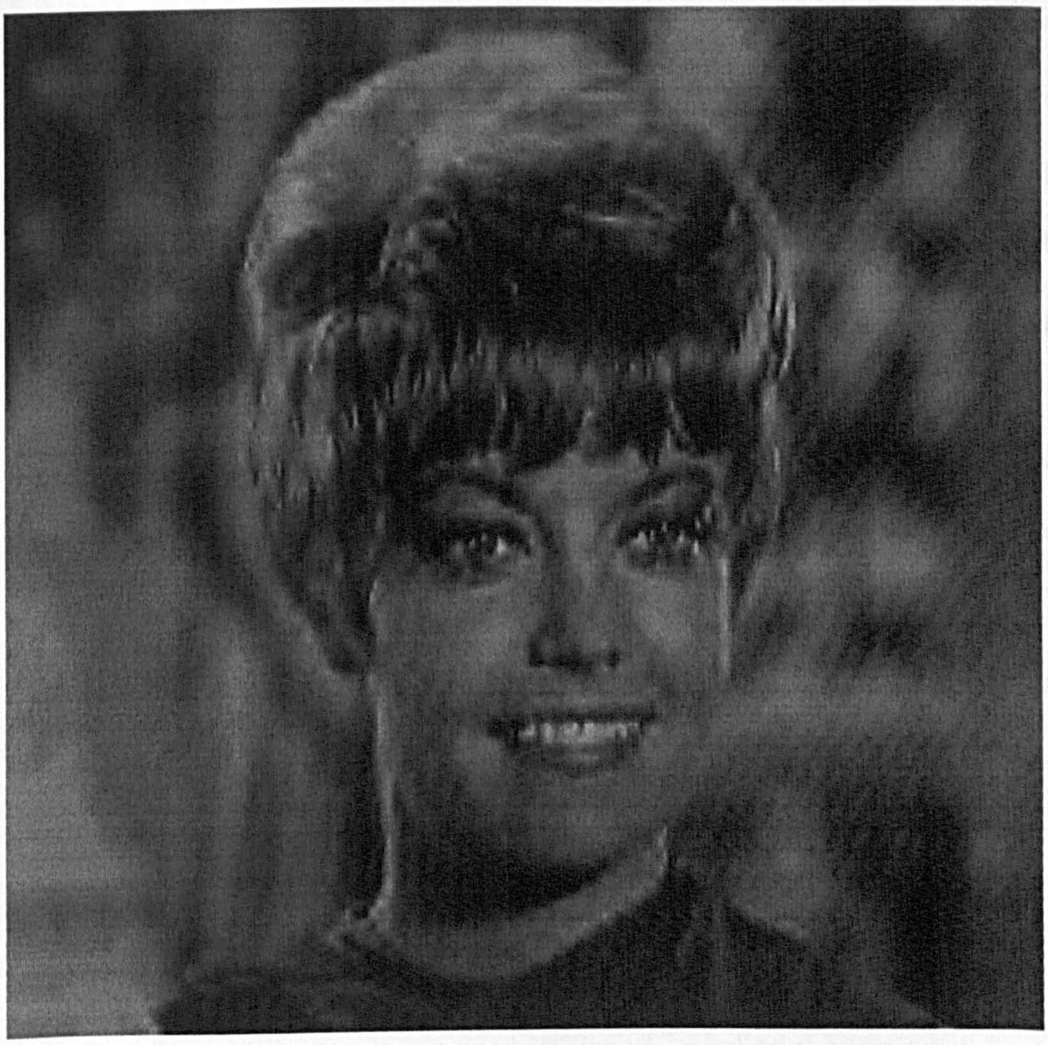


Figure 7.18 Reconstructed Zelda, coded by the improved SPIHT at 0.1 bpp (PSNR 31.9 dB)

Now we vary the number of wavelet transform levels and check the performance gain. The results are depicted in Figure 7.16. As can be seen, the performance gain changes a lot with the levels of wavelet transform. The performance gain using the low-scale wavelet transform is higher than that using the high-scale wavelet transform. This is due to the performance gains contributed by DC-level shifting and the virtual tree technique, which are the major parts of the overall performance gain for the low-scale wavelet transform. For the high-scale wavelet transform, the contributions of other improvements form the dominant parts of the overall performance gain.

The performances of the original and improved SPIHT algorithm with arithmetic coding are depicted in Figure 7.19. The 5-scale wavelet transform is used here.

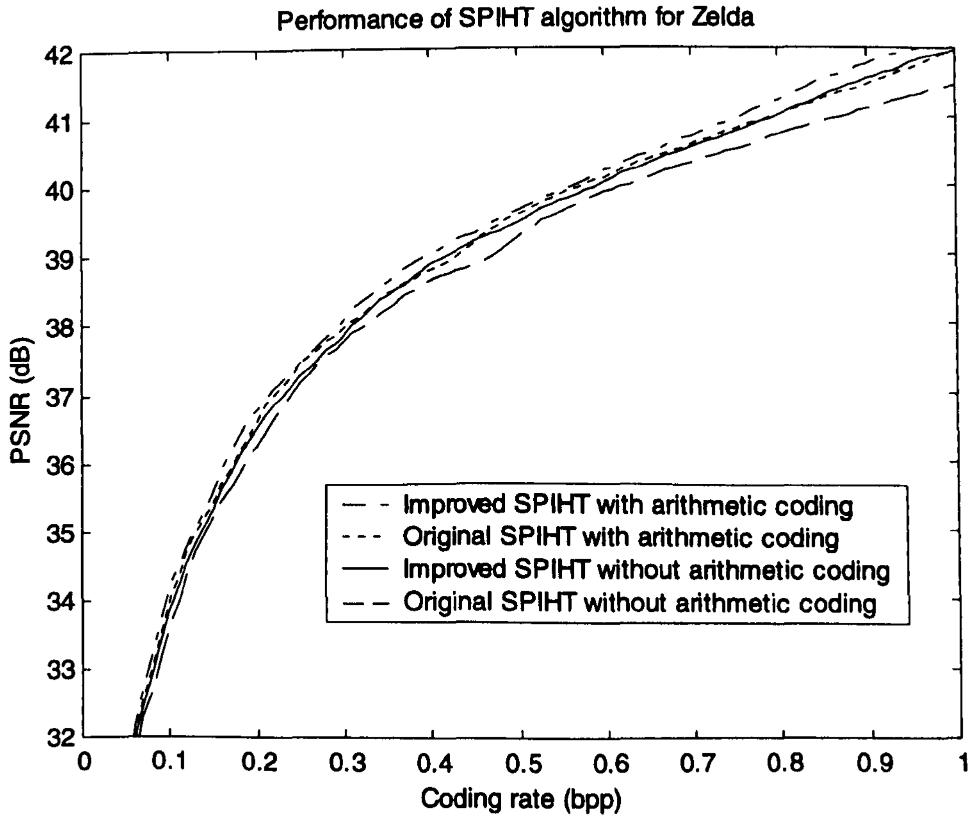


Figure 7.19 Performance of the SPIHT algorithm for Zelda

Figure 7.19 shows that with arithmetic coding, the improved SPIHT algorithm (the dash-dotted curve) outperforms the original SPIHT algorithm (the dotted curve) at any coding rate. The performance gain is depicted in Figure 7.20 (the dotted curve).

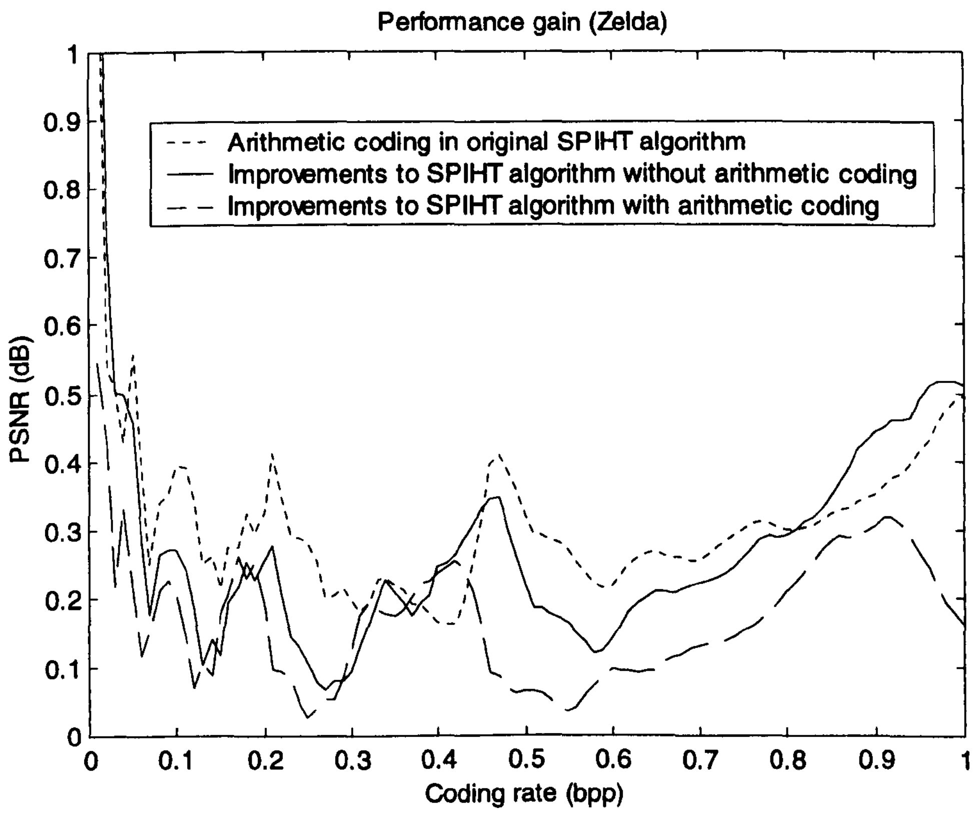


Figure 7.20 Performance gain of arithmetic coding and the improvements to the SPIHT algorithm using the 5-scale wavelet transform for Zelda

The performances of the SPIHT algorithm without arithmetic coding are also depicted in Figure 7.19. It is interesting to see that the rate-distortion curve of the improved SPIHT algorithm without the arithmetic coding (the solid line) is almost identical to that of the original SPIHT with arithmetic coding (the dotted line). The performance gain of the arithmetic coding in the SPIHT algorithm (the dashed line in Figure 7.20) is 0.31 dB in average (for the 100 coding rates equally spaced between 0.01 ~ 1.00 bpp), and that of the improvements (the solid line in Figure 7.20) is 0.28

dB. Their difference is only 0.03 dB. In other words, the improvements explore and exploit successfully the redundancy in the SPIHT encoded bit-streams (without arithmetic coding) for compression, and approach the performance of the arithmetic coding.

The performance gain of the improvements to the SPIHT algorithm with arithmetic coding is also depicted in Figure 7.20. We can see that the performance gain of the improvements to the SPIHT algorithm with arithmetic coding (the dotted line) is less than that without arithmetic coding (the solid line) at most coding rates. This means that there is less to be gained by arithmetic coding in the SPIHT algorithm after the improvements. This also shows the success of the improvements to the SPIHT algorithm.

We get similar results for other test images. The performance gains of the improvements to the SPIHT algorithm without the arithmetic coding (using 5-scale wavelet transform) for various test images are shown in Figure 7.21.

In section 7.1, we showed that Said and Pearlman's implementation of the SPIHT algorithm outperforms our implementation using the 5-scale wavelet transform. The improved SPIHT algorithm (using 5 or higher scale wavelet transform) outperforms the original SPIHT algorithm, no matter whose implementation it is, as shown by the examples in Table 7.22 – the performance of the improved SPIHT algorithm (ISPIHT: ISPIHTencode and ISPIHTdecode) and that of Said & Pearlman's implementation (OSPIHT: "codetree" and "decdtree") for Goldhill.

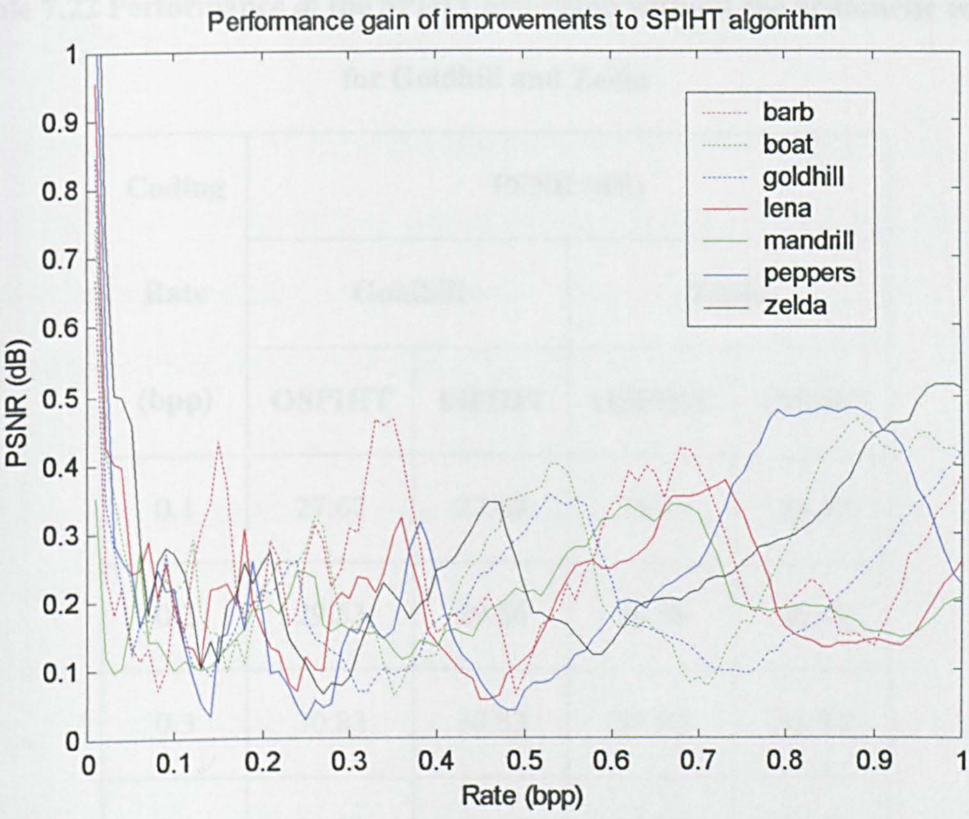


Figure 7.21 Performance gain of the improvements to the SPIHT algorithm without arithmetic coding for various test images

Table 7.22 Performance of the SPIHT algorithm without the arithmetic coding for Goldhill and Zelda

Coding Rate (bpp)	PSNR (dB)			
	Goldhill		Zelda	
	OSPIHT	ISPIHT	OSPIHT	ISPIHT
0.1	27.67	27.69	33.75	33.87
0.2	29.53	29.56	36.38	36.49
0.3	30.83	30.83	37.82	37.84
0.4	31.81	31.84	38.69	38.86
0.5	32.71	32.83	39.33	39.46
0.6	33.51	33.62	39.94	40.03
0.7	34.22	34.26	40.41	40.56
0.8	34.84	34.89	40.82	41.03
0.9	35.43	35.57	41.24	41.56
1.0	36.00	36.27	41.61	41.98

7.4 Speed of the SPIHT algorithm

In this section we apply all the improvements to the SPIHT algorithm and check the speed. Table 7.23 lists the running time of the improved SPIHT encoder (with all the improvements but without arithmetic coding). We can see that the encoding speed of the improved SPIHT algorithm is almost the same for various images.

Table 7.23 Running time of the improved SPIHT encoding

Rate (bpp)	**0.2**	**0.4**	**0.6**	**0.8**	**1.0**	**1.2**	**1.4**	**1.6**	**1.8**	**2.0**
Barbara	0.08	0.09	0.10	0.12	0.13	0.15	0.16	0.17	0.18	0.20
Boat	0.08	0.09	0.11	0.12	0.14	0.15	0.16	0.17	0.19	0.20
Goldhill	0.08	0.09	0.10	0.12	0.14	0.15	0.17	0.18	0.19	0.20
Lena	0.08	0.09	0.11	0.12	0.14	0.15	0.16	0.18	0.19	0.21
Mandrill	0.08	0.09	0.11	0.12	0.14	0.15	0.16	0.18	0.20	0.21
Peppers	0.08	0.09	0.11	0.12	0.14	0.15	0.16	0.18	0.19	0.21
Zelda	0.08	0.09	0.11	0.12	0.13	0.15	0.16	0.18	0.20	0.21

In Figure 7.22, we depict the running time of various versions of the SPIHT encoder. The dashed line is for the improved SPIHT algorithm, and the dotted line is for the SPIHT algorithm using the proposed scheme to judge the significance of trees. We can see that other improvements (except the proposed judgement) slow down the SPIHT encoding a little. But the improved SPIHT encoding is still much faster than the original SPIHT encoding judging the significance of trees directly (the running time for the latter is the solid line in Figure 7.22).

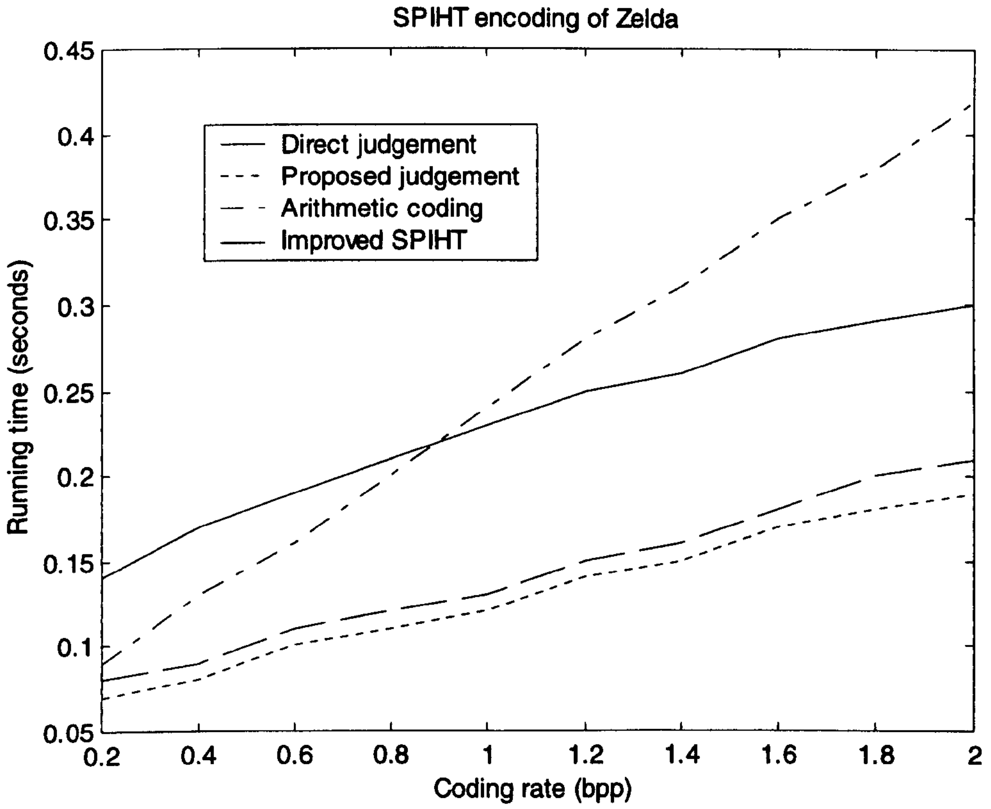


Figure 7.22 Speed of SPIHT encoding for Zelda

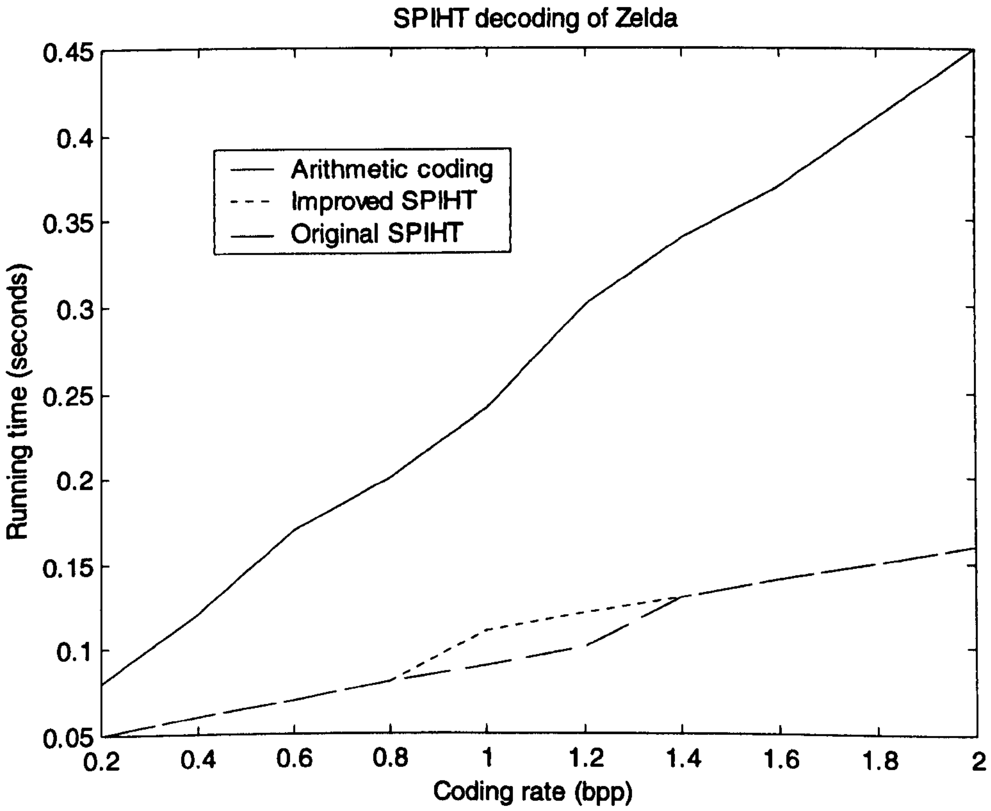


Figure 7.23 Speed of the SPIHT decoding for Zelda

The dash-dot line in Figure 7.22 shows the running time of the SPIHT encoder with arithmetic coding (using the proposed judgement). We see that the SPIHT algorithm with the arithmetic coding runs much slower that that without arithmetic coding, especially at the high coding rates. This is also true for the decoder, as shown in Figure 7.23.

Figure 7.23 also shows that the decoding speed is almost the same for the improved and the original SPIHT algorithm. The SPIHT decoder does not need to judge the significance of trees, which is time-consuming.

We can see from Figure 7.22 and Figure 7.23 that the improved SPIHT algorithm without arithmetic coding is much faster than the original SPIHT algorithm with arithmetic coding. On the other hand, the performance of the two algorithms are almost the same as shown in Figure 7.19 (their PSNR difference is only 0.03 dB in average).

One of the major advantages of the SPIHT algorithm is that it can get very good performance without using arithmetic coding. The improvements enhance the advantage. The improved SPIHT algorithm without arithmetic coding gets almost the same performance as the original SPIHT algorithm with arithmetic coding, but is much faster.

The running time of the wavelet transform is also checked, and the results for Zelda are listed in Table 7.24. We see that the running time increases as the number of wavelet transform levels increases from 1, but is fixed (in the measurement accuracy of 0.01 seconds) if the number of wavelet transform levels is 3 or larger. For SPIHT image coding, we usually use large scales of wavelet transform in practice, and the running time is fixed.

Table 7.24 Speed of wavelet transform (Zelda)

Scale		1	2	3	4	5	6
Running time (s)	Analysis	0.39	0.47	0.48	0.49	0.49	0.49
	Synthesis	0.48	0.58	0.60	0.60	0.60	0.60

The wavelet transform runs much slower than SPIHT coding. The fast wavelet transform is the most important to increase the speed of SPIHT image coding. Some work was done recently on the fast implementation of the biorthogonal 9/7 wavelet transform in literature [9].

For other test images, the running time for the SPIHT algorithm is similar.

7.5 Summary

We have presented the performance of the SPIHT algorithm and the performance gain of the improvements in this chapter. The results revealed that the improved SPIHT algorithm outperforms the original SPIHT algorithm at any coding rate, and is faster than the original SPIHT algorithm. We also showed that the improved SPIHT algorithm without arithmetic coding approximates the performance of the original SPIHT algorithm with arithmetic coding. This is essential, as we do not need to use complex arithmetic coding in order to get the same performance, and the running speed becomes fast.

References

[1] A.Said and W.A.Pearlman. http://www.cipr.rpi.edu/research/SPIHT/. The Center for Image Processing Research at the Rensselaer Polytechnic Institute.

[2] A.Said and W.A.Pearlman, 'A new, fast, and efficient image codec based on set partitioning in hierarchical trees', IEEE Transactions on Circuits and Systems for Video Technology, Vol.6, No.3, pp.243-50, June 1996.

[3] A.Cohen, I.Daubechies and J.-C.Feauveau, 'Biorthogonal bases of compactly supported wavelets', Communications on Pure and Applied Mathematics, Vol.XLV, pp.485-560, 1992.

[4] W.B.Pennebaker and J.L.Mitchell. JPEG still image data compression standard. Van Nostrand Reinhold, New York, 1993.

[5] A.Skodras, C.Christopoulos and T.Ebrahimi, 'The JPEG2000 Still Image Compression Standard', IEEE Signal Processing Magazine, Vol.18, No.5, pp.36-58, September 2001.

[6] T.G.Lane, et al. The Independent JPEG Group's JPEG software, release 6b of 27-Mar-1998. http://www.ijg.org/.

[7] M.D.Adams. Jasper software – JPEG2000 codec. http://www.ece.uvic.ca/~mdadams/jasper/.

[8] D.Taubman, 'High performance scalable image compression with EBCOT', IEEE Transactions on Image Processing, Vol.9, No.7, pp.1158-1170, July 2000.

[9] H.Meng and Z.Wang, 'Fast spatial combinative lifting algorithm of wavelet transform using the 9/7 filter for image block compression', IEE Electronics Letters, Vol.36, No.21, pp.1766-7, 12 October 2000.

Chapter 8

Lossless image coding using the SPIHT algorithm

Transform coding is good at lossy image coding, while predictive coding such as CALIC (context-based adaptive lossless image coding [1]) is good at lossless image coding. But sometimes it is desirable to support both lossless and lossy image coding in the same system. We consider transform coding for lossless image coding here. There are two typical approaches to use transform coding in lossless image coding. One is a lossy transform coding plus a lossless coding for the image residue [2]. The other needs a reversible wavelet transform that maps integers to integers. We discuss lossless image coding using the SPIHT algorithm (set partitioning in hierarchical trees [3]) and the reversible integer-to-integer wavelet transforms in this chapter. This chapter also presents some attempts on image coding based on physical models – content-based SPIHT coding for the CT image of head.

8.1 Reversible integer-to-integer wavelet transforms

The lifting factorisation can be used to build the reversible integer-to-integer wavelet transforms [4]. We use some of the examples in [4] whose impulse response is symmetric that are therefore suitable for symmetric extension on the image edge. For forward transforms, we denote:

Input signal: $s_0(n), n=1, 2, ..., N$

Low pass output: $\kappa \cdot s_1(n), n=1, 2, ..., N/2-1$

High pass output: $d_1(n)/\kappa, n=1, 2, ..., N/2-1$

Here N is an even positive integer, κ is the scaling factor. We can ignore κ in practice, just bearing in mind its existence for the inverse wavelet transform. This is similar to the non-unitary expansion.

We use $\lfloor x \rfloor$ to denote the largest integer not exceeding x.

The forward transforms of the integer-to-integer wavelet transforms we use are listed in the following.

8.1.1 Interpolating transform (2,2): IPT22

$$d_1(n) = s_0(2n+1) - \lfloor (s_0(2n) + s_0(2n+2))/2 + 1/2 \rfloor$$

$$s_1(n) = s_0(2n) + \lfloor (d_1(n-1) + d_1(n))/4 + 1/2 \rfloor$$

8.1.2 Interpolating transform (4,2): IPT42

$$d_1(n) = s_0(2n+1) - \lfloor 9/16 \cdot (s_0(2n) + s_0(2n+2)) - (s_0(2n-2) + s_0(2n+4)/16 + 1/2 \rfloor$$

$$s_1(n) = s_0(2n) + \lfloor (d_1(n-1) + d_1(n))/4 + 1/2 \rfloor$$

8.1.3 Interpolating transform (2,4): IPT24

$$d_1(n) = s_0(2n+1) - \lfloor (s_0(2n) + s_0(2n+2))/2 + 1/2 \rfloor$$

$$s_1(n) = s_0(2n) + \lfloor 19/64 \cdot (d_1(n-1) + d_1(n)) - 3/64 (d_1(n-2) + d_1(n+1)) + 1/2 \rfloor$$

8.1.4 Interpolating transform (6,2): IPT62

$$d_1(n) = s_0(2n+1) - \lfloor 75/128 \cdot (s_0(2n) + s_0(2n+2)) - 25/256 \cdot (s_0(2n-2) + s_0(2n+4)) + 3/256 \cdot (s_0(2n-4) + s_0(2n+6)) + 1/2 \rfloor$$

$$s_1(n) = s_0(2n) + \lfloor (d_1(n-1) + d_1(n))/4 + 1/2 \rfloor$$

8.1.5 Interpolating transform (4,4): IPT44

$$d_1(n) = s_0(2n+1) - \lfloor 9/16\cdot(s_0(2n) + s_0(2n+2)) - (s_0(2n-2) + s_0(2n+4))/16+1/2\rfloor$$

$$s_1(n) = s_0(2n) + \lfloor 9/32\cdot(d_1(n-1) + d_1(n)) - (d_1(n-2) + d_1(n+1))/32 + 1/2\rfloor$$

8.1.6 S+P transform (2+2,2): SP222

$$d_1^{(1)}(n) = s_0(2n+1) - \lfloor (s_0(2n) + s_0(2n+2))/2 + 1/2\rfloor$$

$$s_1(n) = s_0(2n) + \lfloor (d_1^{(1)}(n-1) + d_1^{(1)}(n))/4 + 1/2]$$

$$d_1(n) = d_1^{(1)}(n) - \lfloor (s_1(n) + s_1(n+1))/16 - (s_1(n-1) + s_1(n+2))/16 + 1/2\rfloor$$

8.1.7 Bi-orthogonal transform (9,7): Bio97

$$d_1^{(1)}(n) = s_0(2n+1) - \lfloor \alpha\cdot(s_0(2n) + s_0(2n+2)) + 1/2\rfloor$$

$$s_1^{(1)}(n) = s_0(2n) + \lfloor \beta\cdot(d_1^{(1)}(n-1) + d_1^{(1)}(n)) + 1/2\rfloor$$

$$d_1(n) = d_1^{(1)}(n) - \lfloor \gamma\cdot(s_1^{(1)}(n) + s_1^{(1)}(n+1)) + 1/2\rfloor$$

$$s_1(n) = s_1^{(1)}(n) + \lfloor \delta\cdot(d_1(n-1) + d_1(n)) + 1/2\rfloor$$

Where

$\alpha = -1.586134342$

$\beta = -0.05298011854$

$\gamma = 0.8829110762$

$\delta = 0.4435068522$

$\kappa = 1.149604398$

For multi-resolution transforms, the filtering is repeated on the low pass output $s_1(n)$. The inverse transform can be obtained directly by rewriting the equations of forward transforms in the reverse order. For example, the inverse Bio97 transform is

$$s_1^{(1)}(n) = s_1(n) - \lfloor \delta(d_1(n-1) + d_1(n)) + 1/2\rfloor$$

$$d_1^{(1)}(n) = d_1(n) + \lfloor \gamma(s_1^{(1)}(n) + s_1^{(1)}(n+1)) + 1/2 \rfloor$$

$$s_0(2n) = s_1^{(1)}(n) - \lfloor \beta(d_1^{(1)}(n-1) + d_1^{(1)}(n)) + 1/2 \rfloor$$

$$s_0(2n+1) = d_1^{(1)}(n) + \lfloor \alpha(s_0(2n) + s_0(2n+2)) + 1/2 \rfloor$$

8.2 Performance evaluation of the integer wavelet transforms for the SPIHT algorithm

The performance evaluation of the integer wavelet transforms can be found in the literature for JPEG-2000 [5] but none for the SPIHT algorithm. Due to the success of the SPIHT algorithm, the evaluation is worth doing.

We use the reversible integer-to-integer wavelet transforms listed in section 8.1 for the improved SPIHT image coding (refer to chapter 5 for details). The pre-processing in chapter 6 is not eligible for lossless image coding, so the parameter β is set to be 0. The compression ratios of various test images are listed in Table 8.1. In the experiments, the 5-scale wavelet transform is used, and entropy coding is not used. All improvements are applied to the SPIHT algorithm except DC-level shifting. DC-level shifting makes the compression ratio much lower for Bio97, and makes little improvements for other wavelet transforms. What makes Bio97 different from others is that Bio97 is not unitary because we ignore the scaling factor κ in the implementation.

For each image in Table 8.1, we mark the maximum compression ratio with **bold**, and the minimum with *italic*. Among the seven symmetric integer wavelet transforms, bi-orthogonal 9/7 is always the worst for all the test images, and the IPT44 and the SP222 are the best for natural images. This conclusion is the same as that in [5]. By choosing the proper wavelet transform, we can reduce the size of the seven encoded natural images by 1.2 to 6.7 Kbytes. That is 0.6% to 4.8%, or 3.1% in average.

Table 8.1 Compression ratio of lossless image coding using the improved SPIHT algorithm

	IPT22	IPT42	IPT24	IPT62	IPT44	SP222	Bio97
Barbara	1.634	1.664	1.638	**1.672**	**1.672**	1.665	*1.618*
Boat	1.731	1.736	1.730	1.733	1.741	**1.744**	*1.669*
Goldhill	1.598	1.598	1.593	1.593	1.598	**1.601**	*1.564*
Lena	1.780	1.788	1.775	1.784	**1.790**	**1.790**	*1.724*
Mandrill	1.292	1.294	1.290	1.292	**1.296**	1.295	*1.288*
Peppers	**1.653**	1.649	**1.653**	1.645	**1.653**	**1.653**	*1.615*
Zelda	1.891	1.898	1.894	1.894	**1.907**	1.900	*1.820*
CT-head	4.803	**4.983**	4.686	4.957	4.944	4.872	*4.223*
CT-liver	3.787	**3.808**	3.737	3.780	3.793	3.781	*3.620*

Other transforms may be the best for a particular class of images. For example, the IPT42 is the best for the CT images of head and liver (Figure 8.1 and Figure 8.2). The size difference of the head image encoded using various wavelet transforms is up to 18%, and that of the liver image is up to 5.2%. This suggests that it is more important to use a proper wavelet transform for CT images than that for natural images. In fact, the lossless image coding is required more for medical images than for natural images.

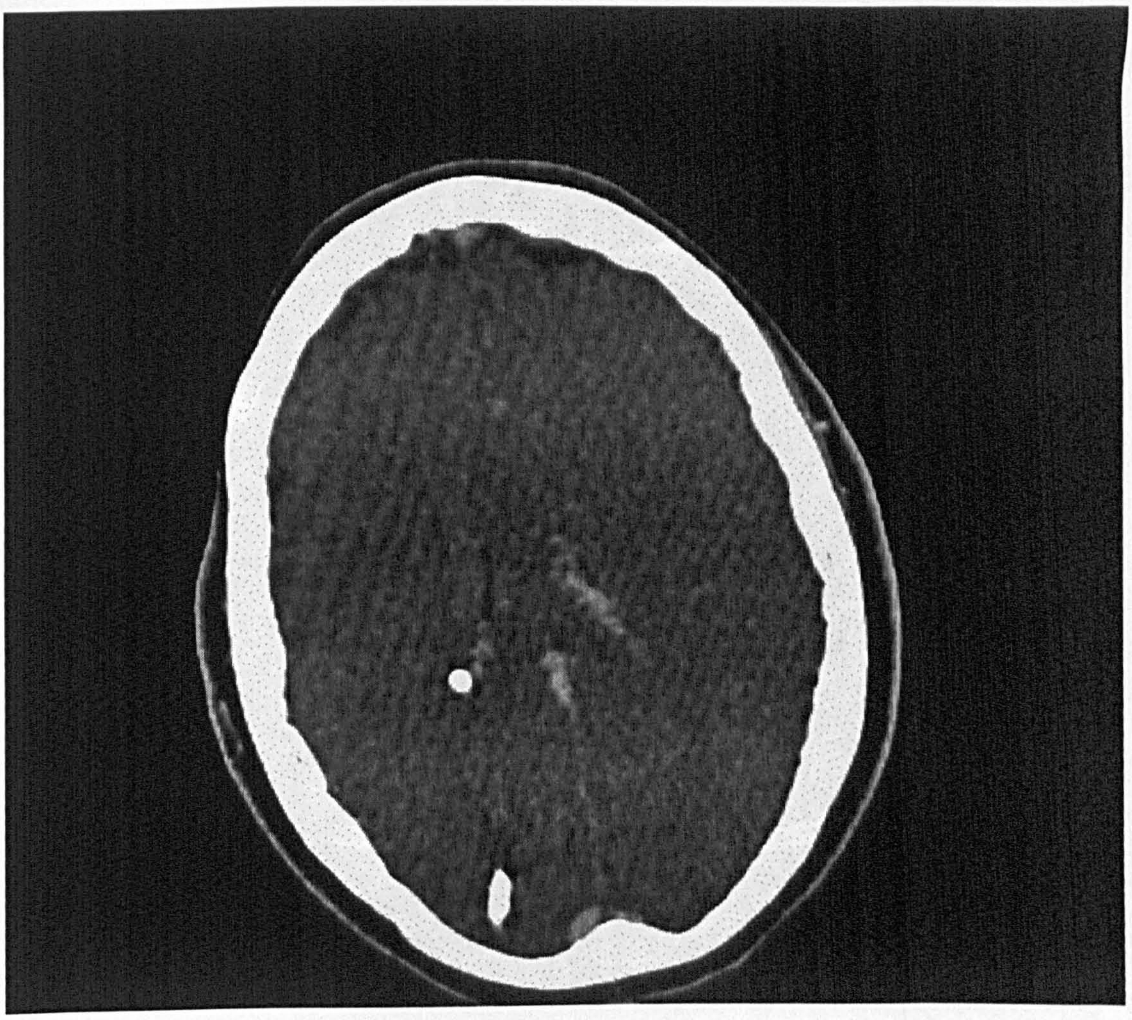


Figure 8.1 CT image of head

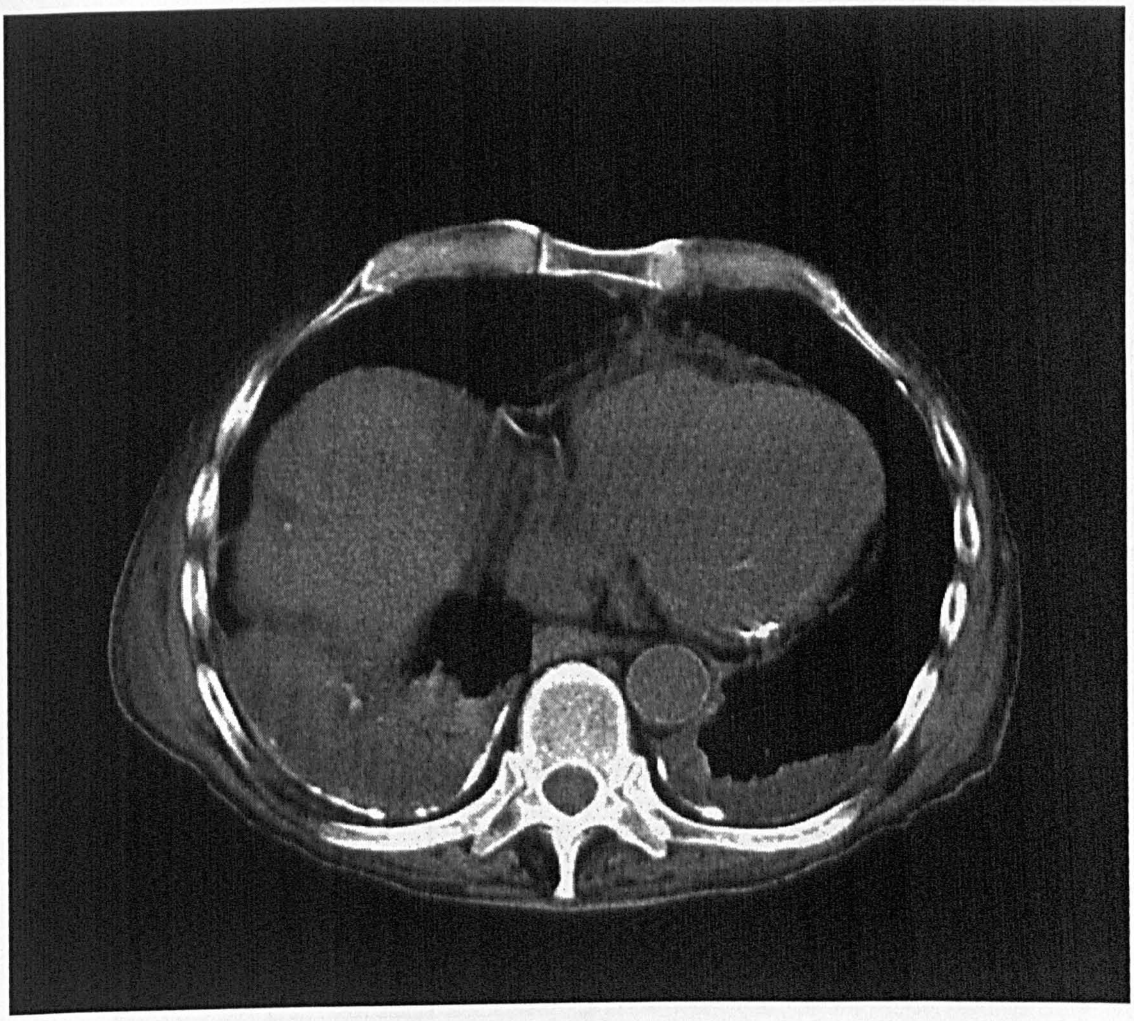


Figure 8.2 CT image of liver

To show the effect of the improvements to the SPIHT algorithm on the lossless image coding, we also apply the original SPIHT algorithm in the lossless image coding. The best results and the relevant wavelet transform are listed in Table 8.2. We can see that the improvements to the SPIHT algorithm reduce the length of the encoded image by about 1%.

Table 8.2 Comparison of the original (OSPIHT) and the improved SPIHT algorithm (ISPIHT) for lossless image coding

Image	Wavelet transform	Length of the encoded image (bits)			
		OSPIHT	ISPIHT	Difference	%
Barbara	IPT44	1269185	1254688	14497	1.2
Boat	SP222	1217817	1202837	14980	1.3
Goldhill	SP222	1323959	1309605	14354	1.1
Lena	IPT44	1187054	1171386	15668	1.3
Mandrill	IPT44	1633713	1618855	14858	0.9
Peppers	SP222	1285321	1268419	16902	1.3
Zelda	IPT44	1116116	1099801	16315	1.5
CT-head	IPT42	494006	488923	5083	1.0
CT-liver	IPT42	662880	655957	6923	1.1

8.3 Content-based lossless image coding using the SPIHT algorithm

Observing Figure 8.1, we see two distinguishing regions: the black outlying and the white ring. We draw the histogram of the pixels in Figure 8.3. As can be seen in

Figure 8.3, there are two outstanding bars at 0 and 245. 60% of the pixels are 0 (which are black in Figure 8.1), 8.4% are 245 (which are white in Figure 8.1), and the rest lie between 1 and 244 (which are grey in Figure 8.1).

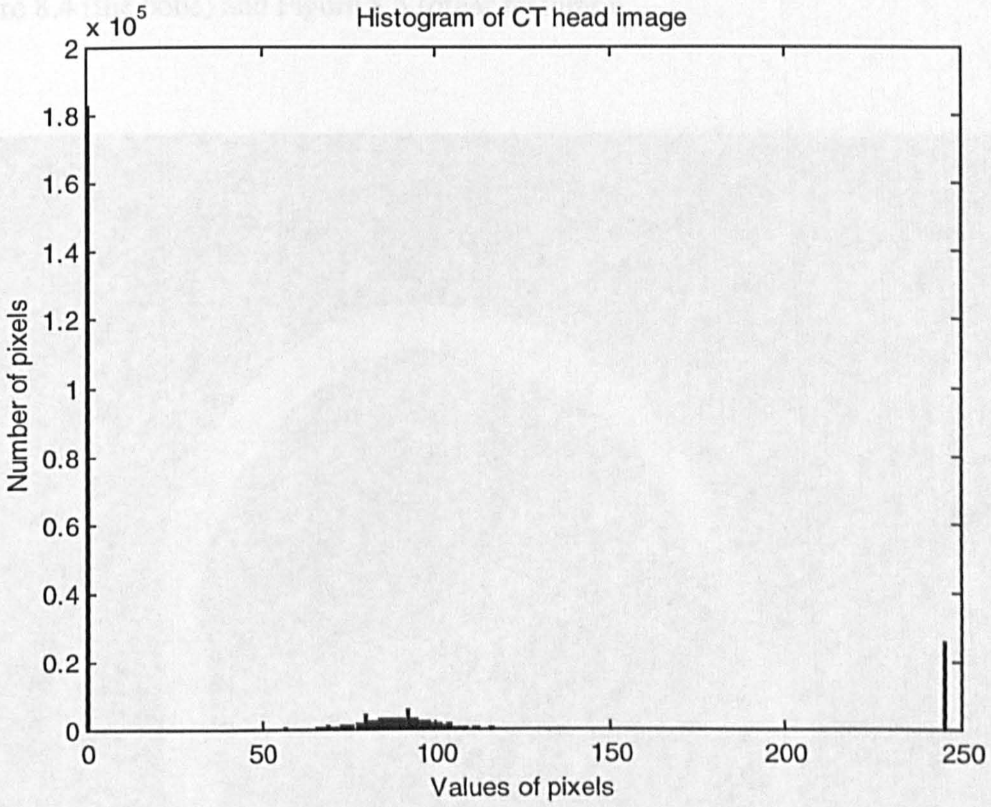


Figure 8.3 Histogram of CT head image

The bright white ring represents the image of bone (skull). It has no detailed information except its shape and position. This is true also for the black outlying region. The bone is much harder than the surrounding tissues thus the relevant pixel values are much larger on the CT image. To get a clear image of the tissues, the pixels representing the bone have to be saturated to the maximum value of 245 (the ten largest values of the 8-bit code are reserved).

The pixel values at the edge of the bone on the CT image are likely to jump. This jump produces large wavelet coefficients in the subbands produced by highpass filtering (i.e., HL, LH and HH). As we know, this is not good for the coding using the SPIHT algorithm.

We segment the bone from the other features for coding, by separating the pixels whose values are 245 from the others. The results of segmentation are shown in Figure 8.4 (the bone) and Figure 8.5 (other features).

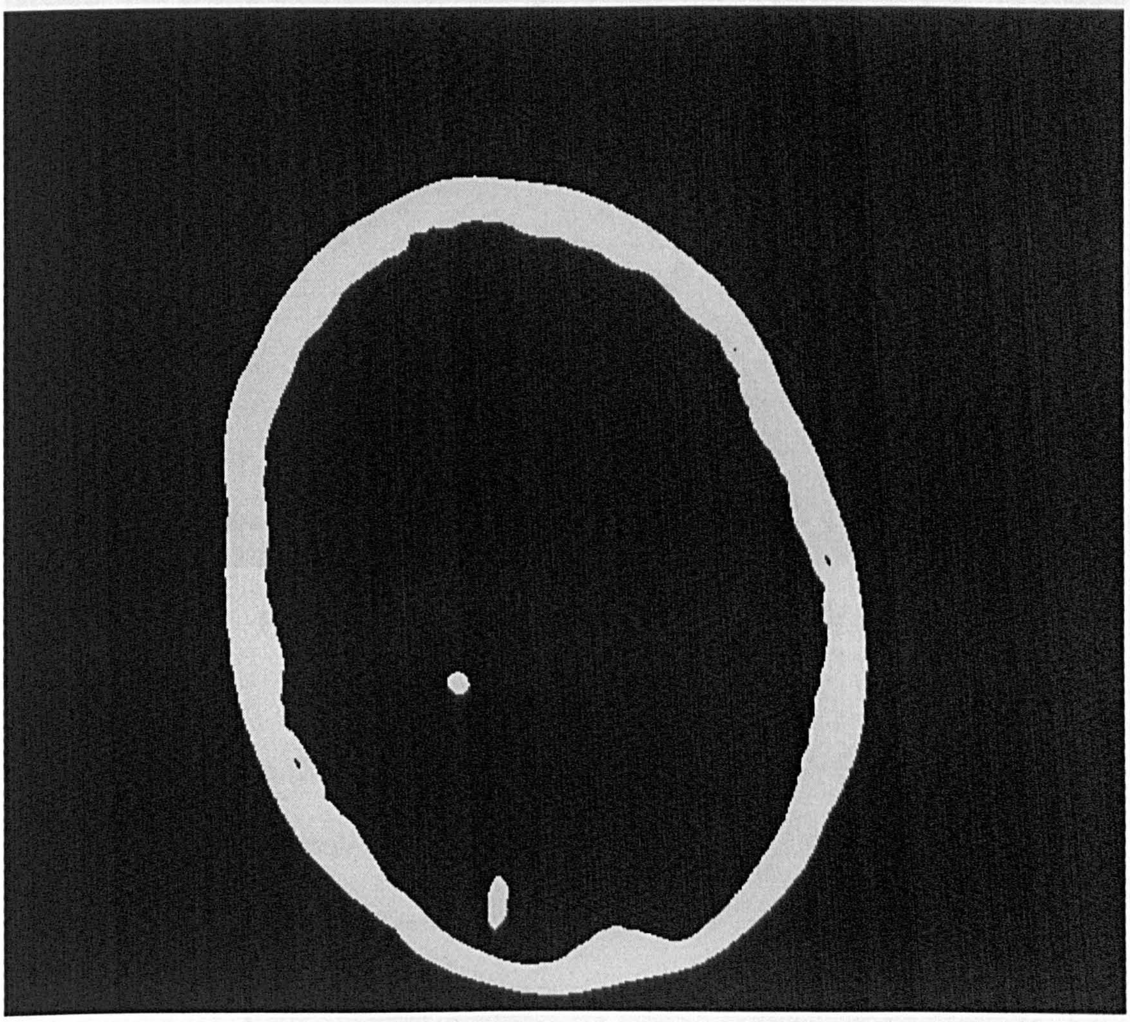


Figure 8.4 CT image of head: bone

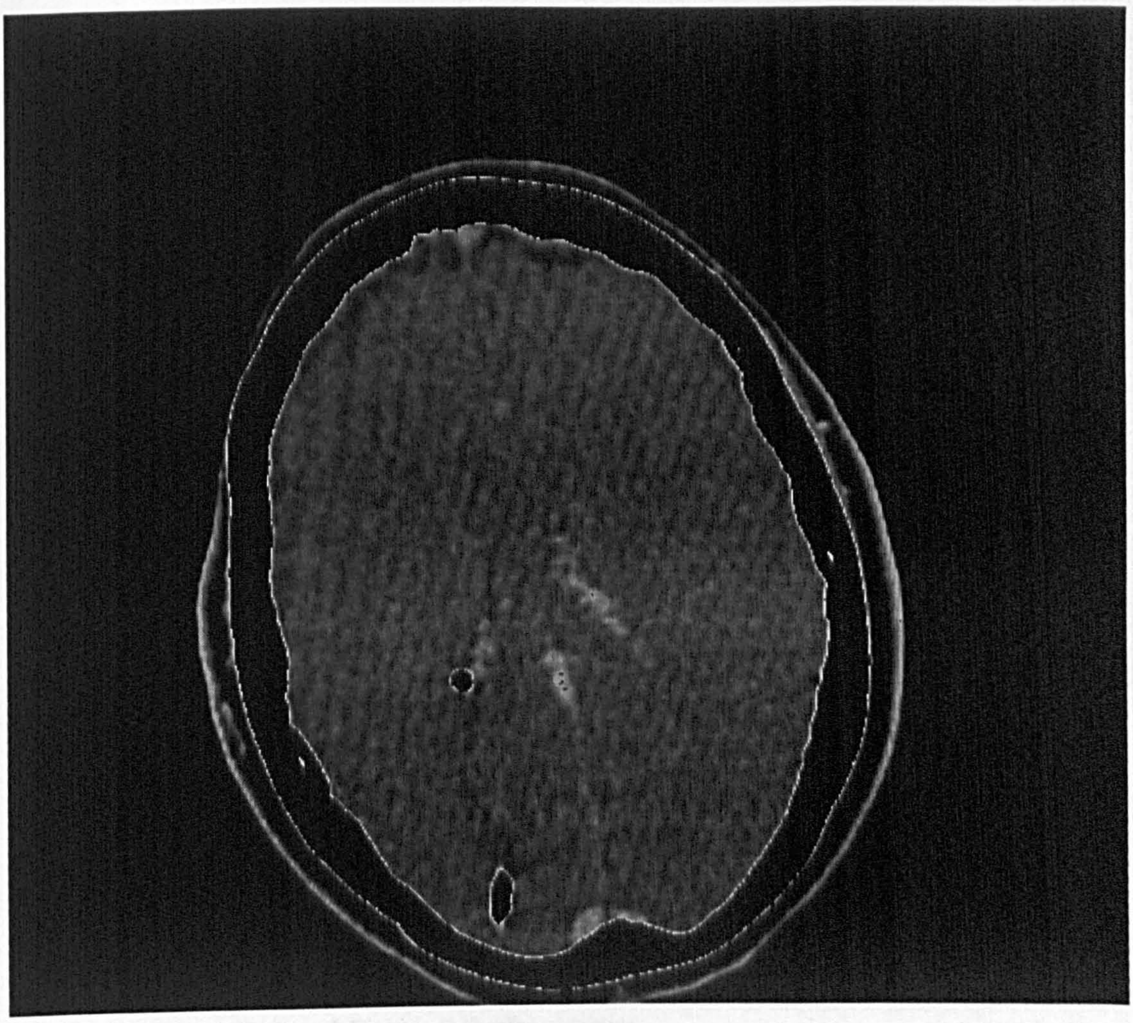


Figure 8.5 CT image of head excluding bone

We encode the bone (Figure 8.4) and other features (Figure 8.5) separately. For the bone, only the shape needs to be coded. We use the context-based arithmetic coding [6]. Each pixel can only be one of the two values: 0 or 245. The context information of four previously coded neighbours (indicated in Figure 8.6) is used to classify the probability of current pixels. There are sixteen models for arithmetic coding. Every pixel is coded according to its conditional distribution under its context.

×	×	×
×	O	

Figure 8.6 Context to model the arithmetic coding of shape

O – pixel to be coded, × – context.

As for the other part of the head image excluding the bone (Figure 8.5), we see that there are still white pixels at the edge of the bone. A further check reveals that although their values are less than 245, those white pixels are still not continuous in value from the neighbouring area. This is due to the sub-pixel alignment of the bone edge. The bone occupies part of a pixel, and the value of the pixel is the mean value of the bone and the neighbouring tissue at the position. So again, we separate the direct neighbours of the bone from the other part, including the horizontal, vertical and diagonal neighbouring pixels (indicated in Figure 8.7). The results are shown in Figure 8.8 (the edge) and Figure 8.9 (the tissues).

D	V	D
H	O	H
D	V	D

Figure 8.7 Direct neighbouring pixels

O – original pixel, H – horizontal neighbour, V – vertical neighbour,

D – diagonal neighbour.

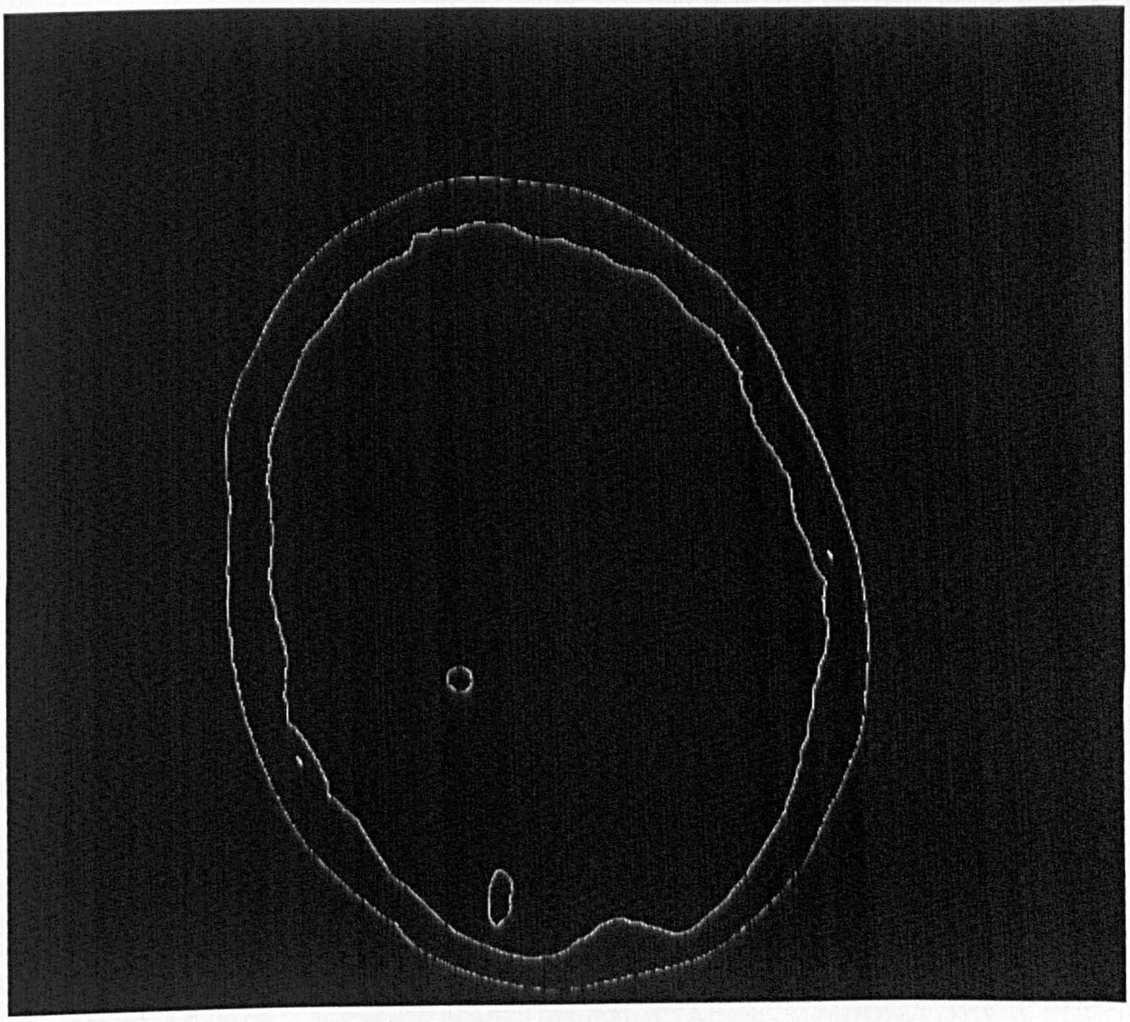


Figure 8.8 CT image of head: edge of bone

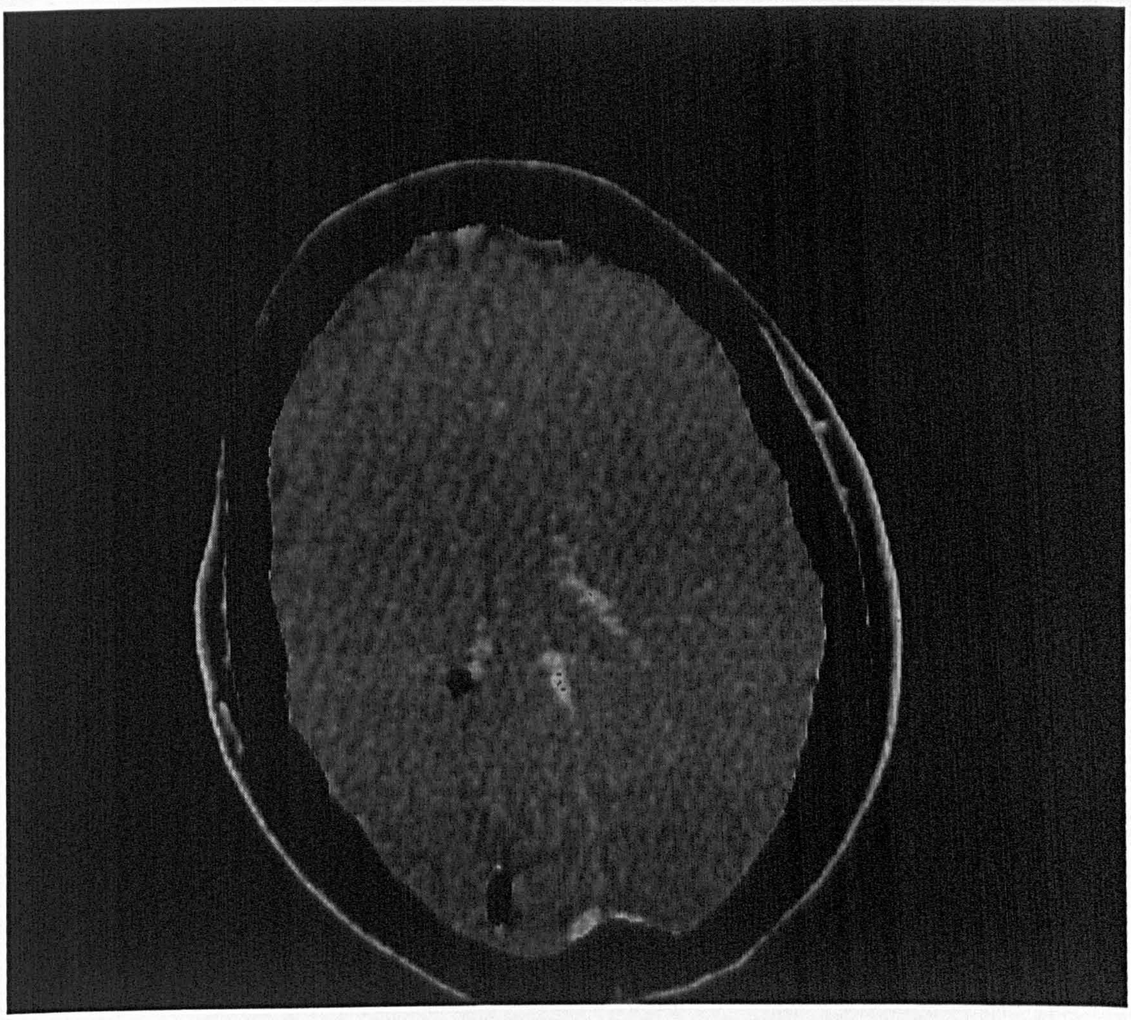


Figure 8.9 CT image of head: tissues

Now what remains in Figure 8.9 are the tissues of the head excluding the bone. There is no bright white edge. As for the edge of bone in Figure 8.8, both the encoder and the decoder know its position from the shape of the bone. We define a scan order (e.g. raster), and code the pixel value of the edge using arithmetic coding, or just 8 bits each. As the total number of pixels of the edge is very small compared with the whole image, the two coding methods make little difference.

To code the tissues in Figure 8.9, we use the SPIHT algorithm and the wavelet transform IPT42. At the edge of the bone, we can extend the tissues symmetrically, the same as at the edge of the whole image. Alternatively, we fill the edge pixels (at the position indicated in Figure 8.8) with half the mean value of its direct horizontal and vertical neighbouring pixels of the tissues. For simplicity, we use the second method.

We can remove the blank black outlying region in Figure 8.9 before applying the SPIHT algorithm. We need to code its shape, and then we can extend the edge of the remaining image symmetrically along the shape for the wavelet transform. For simplicity, we reduce the blank outlying region by cutting the margin squarely and leaving a reduced square image (Figure 8.10) to be coded using the SPIHT algorithm. In this case, we need to code only the four widths of the margins cutting away at the top, bottom, left and right. In practice, we cut the margins before the segmentation, so that the shape coding of bone can benefit from it as well.

The results of lossless coding based on the segmentation and the margin cutting are listed in Table 8.3. The results show that the coding benefits from both the segmentation and the margin cutting. The segmentation and the margin cutting reduce the coding rate by 2.4%, or equivalently, increase the compression ratio by

2.4%. Table 8.3 also shows the performance gain of the improvements to the SPIHT algorithm again.

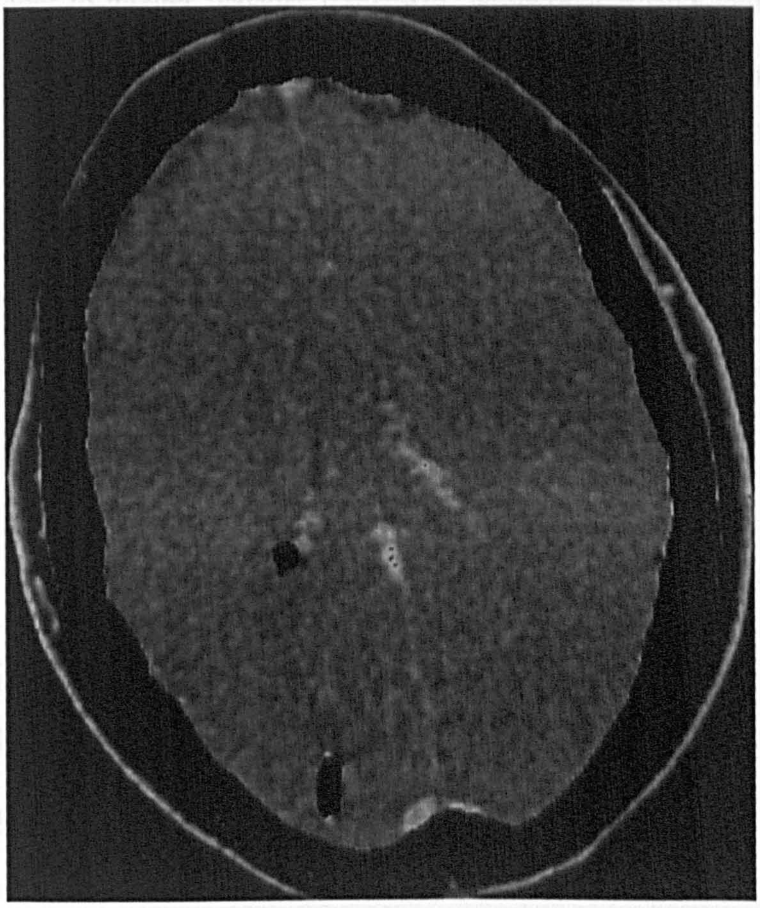


Figure 8.10 CT image of head: tissues (without blank margin)

Table 8.3 Lossless coding rate (bpp) of CT head image using the SPIHT algorithm based on segmentation and margin cutting

SPIHT algorithm	None	Segmentation	Margin cutting	Both
Original	1.622	1.611	1.599	1.583
Improved	1.606	1.594	1.583	1.567

8.4 Summary

We have applied the improved SPIHT algorithm to lossless image coding using the reversible integer-to-integer wavelet transform. Numerical results showed that the IPT44 and the SP222 are the best for natural images among the symmetric transforms used in the experiments, and the IPT42 is the best for CT images. The difference of the compression ratio using various transforms in the SPIHT algorithm is about 3% for natural images, and the difference is much larger for CT images. It has also been shown that the improvements to the SPIHT algorithm increase the compression ratio by about 1%.

Some of our attempts on image coding based on physical models have also been presented in this chapter – the content-based SPIHT coding for the CT image of head. We separated the image into four parts: the blank outlying margin, the bone, the edge of the bone, and the tissues. Each part is coded using a particular scheme. The blank outlying margins are cut away, leaving a reduced square image containing the desired information. Four parameters are coded, indicating the widths of the margins on each side. The bone is coded by its shape, using the context-based shape coding in MPEG-4. The edge of the bone is coded using arithmetic coding. The tissues are coded using the SPIHT algorithm. Compared with the SPIHT algorithm for the whole image, the content-based image coding increases the compression ratio by 2.4% for lossless coding. Although the performance gain is not very high, the experiments demonstrate the potential advantage of the content-based image coding. More complicated image processing techniques can be used for compression.

References

[1] X.Wu and N.Memon, 'Context-based, Adaptive, Lossless Image Codec', IEEE Transactions on Communications, Vol.45, No.4, pp.437-44, April 1997.

[2] A.Said and W.A.Pearlman, 'An image multiresolution representation for lossless and lossy compression', IEEE Transactions on Image Processing, Vol.5, No.9, pp.1303-10, September 1996.

[3] A.Said and W.A.Pearlman, 'A new, fast, and efficient image codec based on set partitioning in hierarchical trees', IEEE Transactions on Circuits and Systems for Video Technology, Vol.6, No.3, pp.243-50, June 1996.

[4] A.R.Calderbank, I.Daubechies, W.Sweldens and B.L.Yeo, 'Wavelet Transforms that Map Integers to Integers', Applied and Computational Harmonic Analysis, Vol.5, pp.332-369, July 1998.

[5] M.D.Adams and F.Kossentini, 'Reversible Integer-to-Integer Wavelet Transforms for Image Compression: Performance Evaluation and Analysis', IEEE Transactions On Image Processing, Vol.9, No.6, pp.1010-24, June 2000.

[6] T.Ebrahimi and C.Horne, 'MPEG-4 Natural Video Coding - An Overview', Signal Processing: Image Communication, Vol.15, No.4-5, pp.365-85, Elsevier Science, January 2000.

Chapter 9

Discussion and Conclusions

This thesis has studied the novel scalable image coding using the SPIHT (set partitioning in hierarchical trees [1]) algorithm based on wavelet transform. This is accomplished by the analysis of the wavelet transform, statistical properties of the wavelet coefficients (being the output of wavelet transform) and the procedure of SPIHT coding, followed by the improvements to the SPIHT algorithm and the applications of the SPIHT algorithm in lossless image coding.

This chapter presents a discussion of the work that was undertaken in this research project. The results of this work are summarised and suggestions for further work proposed.

9.1 Summary and conclusions

The main original contribution of this thesis is the improvements to the SPIHT algorithm, including DC-level shifting, virtual trees, omitting the predictable coding symbols, quantisation offset, pre-processing, and optimisation of arithmetic coding. Further contributions include a solution of wavelet transform and SPIHT coding for arbitrarily-sized image, study of statistical properties of wavelet coefficients for natural images, evaluation of various integer wavelet transforms for lossless image coding using the SPIHT algorithm, and tries on content-based image coding.

Chapter 1 introduced the requirements of image coding, including the latest new features such as scalability, error-resilience and region-of-interest. The methodologies of image coding were also discussed, which are categorised into statistical and physical models.

The wavelet transform is the basis of the newly emerged image coding algorithms such as EZW (embedded zerotree wavelet coding [2]), SPIHT and EBCOT (embedded block coding with optimised truncation [3]). The wavelet transform can be constructed from two-channel filter banks. To get perfect reconstruction, two-channel filter banks have to be biorthogonal. Biorthogonal FIR (finite impulse response) filter banks are often used for image coding because they have the advantage of linear phase. The necessary and sufficient conditions for perfect reconstruction, and the relations between filter banks and wavelet transform were proofed in z-domain in Chapter 2, using our own techniques.

Multi-resolution decomposition can be made using two-channel filter banks as a building block. Octave decomposition is usually used in image coding. It is implemented by repeating the decomposition on the low subband.

Image is two-dimensional (2D) signal, which needs 2D wavelet transform. A simple solution is 2D separable wavelet transform, which applies a one-dimensional (1D) wavelet transform to each dimension of the image.

The length of an image signal is limited in any of the two dimensions. To keep the signal length unchanged after wavelet transform, and without losing any information, the image signal is extended at the edge for calculation. After wavelet transform, extra samples at the edge (due to extension) are discarded. Several edge extension methods were checked, and a periodic extension and a symmetric extension were found to be suitable for arbitrary image size. Because of the nature of natural images, the symmetric edge extension normally gets better performance in image coding than the periodic extension. Symmetric edge extension requires the wavelet transform to be linear phase. That is why linear phase is desirable in wavelet transform for image coding.

The energy of an image is localised after wavelet transform. That is the key for the success of transform coding. Statistical properties of wavelet coefficients were explored in chapter 3, including the distribution of values of each subband, the intra-subband correlation, inter-subband correlation, and inter-level correlation. The results for natural images show that the energy is highly concentrated. A small amount of values carry most of the energies. This suggests that successive approximation quantisation (or bit-plane coding) is efficient in coding. This is because the distribution of 1s and 0s is highly biased (most 0s, and only a few 1s) for significant bits of the values (in binary). The entropy coding can take advantage of this distribution and compress the data efficiently. In fact, bit-plane coding is used in EZW, SPIHT and EBCOT. The energy of wavelet coefficients is also concentrated on low level (resolution) subbands.

The typical value of intra-subband correlation coefficient is 0.32, and those for inter-level and inter-subband are 0.10 and 0.022 respectively. The inter-subband correlation is relatively trivial, and the intra-subband correlation is the largest. The SPIHT algorithm takes advantage of the inter-level correction by organising the wavelet coefficients at the same spatial orientation on various decomposition levels as a set in a hierarchical tree. This tree structure also contains the neighbouring wavelet coefficients on each of the relevant subbands in a set, thus the intra-subband correlation is exploited to some limited extend for coding. The coding and partitioning of trees also benefits from the energy concentration on low level subbands.

The EBCOT algorithm takes advantage of the intra-subband correlation by context-based arithmetic coding. The modelling of arithmetic coding is so successful that the rate-distortion performance of EBCOT is better than that of SPIHT at some coding rates in the extreme case, although EBCOT exploits intra-subband correlation only. In this extreme case, the EBCOT coded bit-stream contains only one layer, which is not optimal for truncation. The results of EBCOT show the potential compression in image coding exploiting the intra-subband correlation.

Statistical studies also reveal that not only the overall energy of wavelet coefficients is concentrated, but also the distribution of values in a sub-range (relevant to a bit plane) is biased – more values on the small-magnitude half. The gravity centre of a typical sub-range is offset from the geometrical centre by about one eighth of the total length of the range. This property is exploited in the improvements to the SPIHT algorithm for decoding.

The key techniques of SPIHT coding were highlighted in chapter 4, including organising the wavelet coefficients as a set in hierarchical trees, successive approximation quantisation, and ordered bit-plane coding. The detailed implementation procedure was also presented in chapter 4.

Observing the encoded bit-stream, and studying the procedure of SPIHT coding, it was found that the SPIHT algorithm could be improved for better rate-distortion performance and faster running speed. The improvements were described in chapter 5 and 6, including DC-level shifting, virtual trees, omitting predictable coding symbols, quantisation offset, and pre-processing. Simple and direct efforts were also made to optimise the arithmetic coding for the SPIHT algorithm.

Numerical results in chapter 7 showed the success of the improvements to the SPIHT algorithm. With the improvements, the running time of the SPIHT algorithm is reduced almost by half, and the rate-distortion performance is also improved significantly, especially at low coding rates. The performance gain is very important for applications such as image transmission in mobile communications.

The performance of the SPIHT algorithm was also compared with that of JPEG [4] and JPEG2000 [5]. SPIHT is the best, outperforms JPEG and JPEG2000.

It is sometimes desired to use the same system for lossy and lossless image coding. Chapter 8 addressed the lossless image coding using reversible integer wavelet transform and SPIHT coding. Performance was evaluated for lossless SPIHT image coding using different integer wavelet transforms. The results revealed that the coding rate could be saved by several percent by choosing proper integer wavelet transform, and the best wavelet transform varied depending on the type of image. In our experiments, the interpolating transform (4, 4) and the S+P transform (2+2, 2) [6] are often the best for the natural images, and the popular bi-orthogonal 9/7 transform is the worst. For the CT images, the interpolating transform (4, 2) [6] is the best.

Content-based video coding has been adopted in MPEG-4 [7], and has proved to be successful. Content-based image coding was tried in this research project, for both 2D and 3D (volumetric) medical images. The idea here is to separate the special object and region from the background of an image, and applying various coding schemes respectively.

Content-based coding for the 2D CT image of head was presented in chapter 8. The image was segmented into four parts, which are the outside blank margins, the bones (skull), the outside edge of bones, and the remaining tissues. The margins are square regions with pixel value of 0, which are encoded by the width on each side of the image. The bones consist of pixels whose values are 245 – the maximum pixel value.

The outside edge of bones includes the neighbouring pixels of the bones, which are formed partly by the bones. The shape of bones is encoded using context-based arithmetic coding, and the values of its outside edge are encoded using arithmetic coding. The remaining tissues are encoded using the SPIHT algorithm. Numerical results show that the content-based coding scheme reduces the lossless coding length of the CT head image by several percent. Further improvements could be made by using arbitrary-shaped margins, and by using the wavelet transform and the SPIHT algorithm for arbitrary-shaped object (or region).

For 3D volumetric CT images, neighbouring images were encoded using prediction. Various prediction techniques were tried, both in image domain and wavelet transform domain. An iterative robust estimation algorithm using an optical flow model [8] was also tried in the prediction. But these efforts failed to get any performance gain in coding. The reason is that the prediction (or shape mapping) can not be absolutely accurate. Significant prediction errors are produced at the edge of the object, resulting in high energy in the high subband of wavelet coefficients, which is not good for SPIHT. These failed attempts are not presented in the thesis.

9.2 Future work

As discussed in section 9.1, the primary researches on content-based image coding have shown its potential advantage. Further compression could be reached from content-based image coding, or image coding based on physical models in general. Even in the primary researches, some further work could be done, as pointed out in section 9.1. Content-based image coding is especially good for special applications such as lossless coding of some medical images.

SPIHT exploits the inter-level correlation efficiently, and exploits the intra-subband correlation partially. The statistical properties of the wavelet coefficients show that the intra-subband correlation is the most important among the intra-subband, the inter-level and the inter-subband correlation. The key technique of EBCOT is the modelling of arithmetic coding, which exploits the intra-subband correlation thoroughly by the successful context-based adaptive arithmetic coding. Similar techniques (i.e., efficient modelling of context-based adaptive arithmetic coding) could be used to code the individual wavelet coefficients and the significance of trees in SPIHT.

References

[1] A.Said and W.A.Pearlman, 'A new, fast, and efficient image codec based on set partitioning in hierarchical trees', IEEE Transactions on Circuits and Systems for Video Technology, Vol.6, No.3, pp.243-50, June 1996.

[2] J.M.Shapiro, 'Embedded Image Coding Using Zerotrees of Wavelet Coefficients', IEEE Transactions on Signal Processing, Vol.41, No.12, pp.3445-62, December 1993.

[3] D.Taubman, 'High performance scalable image compression with EBCOT', IEEE Transactions on Image Processing, Vol.9, No.7, pp.1158-70, July 2000.

[4] W.B.Pennebaker and J.L.Mitchell. JPEG still image data compression standard. Van Nostrand Reinhold, New York, 1993.

[5] A.Skodras, C.Christopoulos and T.Ebrahimi, 'The JPEG2000 Still Image Compression Standard', IEEE Signal Processing Magazine, Vol.18, No.5, pp.36-58, September 2001.

[6] A.R.Calderbank, I.Daubechies, W.Sweldens and B.L.Yeo, 'Wavelet Transforms that Map Integers to Integers', Applied and Computational Harmonic Analysis, Vol.5, pp.332-369, July 1998.

[7] T.Ebrahimi and C.Horne, 'MPEG-4 Natural Video Coding - An Overview', Signal Processing: Image Communication, Vol.15, No.4-5, pp.365-85, Elsevier Science, January 2000.

[8] J.M.Odobez and P.Bouthemy, 'Robust multiresolution estimation of parametric motion models', Journal of Visual Communication and Image Representation, Vol.6, No.4, pp.348-365, December 1995.

Bibliography

[1] Michael D. Adams and Andreas Antoniou, 'Reversible EZW-based image compression using best transform selection and selective partial embedding', IEEE Transactions on Circuits and Systems – II. Analog and Digital Processing, Vol.47, No.10, pp.1119-22, October 2000.

[2] Philippe Aigrain, Hongjiang Zhang and Dragutin Petkovic, 'Content-based representation and retrieval of visual media: A state-of-the-art review', Multimedia Tools and Applications, Vol.3, pp.179-202, 1996.

[3] Maaruf Ali, 'Medical image compression using set partitioning in hierarchical trees for (military) telemedicine applications', IEE Colloquium on Time-Scale and Time-Frequency Analysis and Applications, pp.22/1-5, London, 29 February 2000.

[4] Marc Antonini, Michel Barlaud, Pierre Mathieu and Ingrid Daubechies, 'Image coding using wavelet transform', IEEE Transactions on Image Processing, Vol.1, No.2, pp.205-20, April 1992.

[5] Joel Askelof, Mathias Larsson Carlander and Charilaos Christopoulos, 'Region of interest coding in JPEG 2000', Signal Processing: Image Communication, Vol.17, No.1, pp.105-111, Elsevier Science, January 2002.

[6] Olivier Avaro, Alexandros Eleftheriadis, Carsten Herpel, Ganesh Rajan and Liam Ward, 'MPEG-4 Systems: Overview', Signal Processing: Image Communication, Vol.15, No.4-5, pp.281-98, Elsevier Science, January 2000.

[7] Noel Brady, 'MPEG-4 standardized methods for the compression of arbitrarily shaped video objects', IEEE Transactions on Circuits and Systems for Video Technology, Vol.9, No.8, pp.1170-89, December 1999.

[8] Patrick Bouthemy, Marc Gelgon and Fabrice Ganansia, 'A unified Approach to shot change detection and camera motion characterization', IEEE Transactions on Circuits and Systems for Video Technology, Vol.9, No.7, pp.1030-44, October 1999.

[9] Zhaohui Cai, K.R.Subramanian and Tee Hiang Cheng, 'Modifications on morphology-based image coding', IEE Electronics Letters, Vol.37, No.7, pp.421-2, 29 March 2001.

[10] Zuo-Dian Chen, Ruey-Feng Chang and Wen-Jia Kuo, 'Adaptive predictive multiplicative autoregressive model for medical image compression', IEEE Transactions on Medical Imaging, Vol.18, No.2, pp.181-4, February 1999.

[11] Sungdae Cho and William A. Pearlman, 'A full-Featured, Error-Resilient, Scalable Wavelet Video Codec Based on the Set Partitioning in Hierarchical Trees (SPIHT) algorithm', IEEE Transactions on Circuit and Systems for Video Technology, Vol.12, No.3, pp.157-71, March 2002.

[12] Seung-Jong Choi and John W. Woods, 'Motion-compensated 3-D subband coding of video', IEEE Transaction on Image Processing, Vol.8, No.3, pp.155-67, February 1999.

[13] Charilaos Christopoulos, Athanassios Skodras and Touradj Ebrahimi, 'The JPEG2000 still image coding system: An overview', IEEE Transactions on Consumer Electronics, Vol.46, No.4, pp.1103-27, November 2000.

[14] Christos Chrysafis and Antonio Ortega, 'Efficient context-based entropy coding for lossy wavelet image compression', IEEE Proceedings of Data Compression Conference (DCC'97), pp.241-50, 1997.

[15] Charles D. Creusere, 'Fast embedded compression for video', IEEE Transaction on Image Processing, Vol.8, No.12, pp.1811-16, December 1999.

[16] Edouard Francois and Patrick Bouthemy, 'Derivation of qualitative information in motion analysis', Image and Vision Computing, Vol.8, No.4, pp.279-88, November 1990.

[17] Vivek K. Goyal, 'Theoretical foundations of transform coding', IEEE Signal Processing Magazine, Vol.18, No.5, pp.9-21, September 2001.

[18] Bilge Gunsel, A. Murat Tekalp and Peter J.L. van Beek, 'Content-based access to video objects: Temporal segmentation, visual summarization, and feature extraction', Signal Processing, Vol.66, pp.261-80, Elsevier Science, 1998.

[19] C. Herpel and A. Eleftheriadis, 'MPEG-4 Systems: Elementary stream management', Signal Processing: Image Communication, Vol.15, No.4-5, pp.299-320, Elsevier Science, January 2000.

[20] S.W.Hong and P.Bao, 'An edge-preserving subband coding model based on non-adaptive and adaptive regularization', Image and Vision Computing, Vol.18, pp.573-82, Elsevier Science, 2000.

[21] Ashraf A. Kassim and Lifeng Zhao, 'Rate-scalable object-based wavelet codec with implicit shape coding', IEEE Transactions on Circuits and Systems for Video Technology, Vol.10, No.7, pp.1068-79, October 2000.

[22] Andre Kaup, 'Object-based texture coding of moving video in MPEG-4', IEEE Transactions on Circuits and Systems for Video Technology, Vol.9, No.1, pp.5-15, February 1999.

[23] E. Khan and M. Ghanbari, 'Very low bit rate video coding using virtual SPIHT', IEE Electronics Letters, Vol.37, No.1, pp.40-42, 4 January 2001.

[24] Omid E. Kia and David S. Doermann, 'Residual coding in document image compression', IEEE Transactions on Image Processing, Vol.9, No.6, pp.961-9, June 2000.

[25] Beong-Jo Kim, Zixiang Xiong and William A. Pearlman, 'Low bit-rate scalable video coding with 3-D set partitioning in hierarchical trees (3-D SPIHT)', IEEE Transactions on Circuits and Systems for Video Technology, Vol.10, No.8, pp.1374-87, December 2000.

[26] Hyo Joon Kim and Choong Woong Lee, 'Efficient significance map coding using block-based zerotree and quadtree', IEE Electronics Letters, Vol.36, No.13, pp.1110-1, 22 June 2000.

[27] Youngseop Kim and William A. Pearlman, 'Lossless volumetric medical image compression', SPIE Conference on Applications of Digital Image Processing, Vol.3803, pp.305-12, Denver, Colorado, July 1999.

[28] Eleftherios Kofidis, Nicholas Kolokotronis, Aliki Vassilarakou, Sergios Theodoridis and Dionisis Cavouras, 'Wavelet-based medical image compression', Future Generation Computer Systems, Vol.15, pp.223-43, Elsevier Science, 1999.

[29] M.H.Lee and K.N.Ngan, 'Video coding with a variable block-sizing technique in the wavelet transform domain', Journal of Electronic Imaging, Vol.7, No.3, pp.539-47, 1998.

[30] Jin Li and Shawmin Lei, 'An embedded still image coder with rate-distortion optimization', IEEE Transactions on Image Processing, Vol.8, No.7, pp.913-23, July 1999.

[31] Jiebo Luo, Xiaohui Wang, Changwen Chen and Kevin J. Parker, 'Volumetric medical image compression with three-dimensional wavelet transform and octave zerotree coding', SPIE Proceedings of Visual Communications and Image Processing, Vol.2727, pp.579-90, 1996.

[32] Stephane G. Mallat, 'A theory for multiresolution signal decomposition: The wavelet representation', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.11, No.7, pp.674-93, July 1989.

[33] Hong Man, Faouzi Kossentini and Mark J.T. Smith, 'A family of efficient and channel error resilient wavelet/subband image coders', IEEE Transactions on Circuits and Systems for Video Technology, Vol.9, No.1, pp.95-108, February 1999.

[34] Michael W. Marcellin et al., 'An overview of quantization in JPEG 2000', Signal Processing: Image Communication, Vol.17, No.1, pp.73-84, Elsevier Science, January 2002.

[35] Detlev Marpe, Gabi Blattermann, Jens Ricke and Peter Maab, 'A two-layered wavelet-based algorithm for efficient lossless and lossy image compression', IEEE Transactions on Circuits and Systems for Video Technology, Vol.10, No.7, pp.1094-1102, October 2000.

[36] Detlev Marpe and Hans L. Cycon, 'Very low bit-rate video coding using wavelet-based techniques', IEEE Transactions on Circuits and Systems for Video Technology, Vol.9, No.1, pp.85-94, February 1999.

[37] Peter Meer, Doron Mints, Azriel Rosenfeld and Dong Yoon Kim, 'Robust regression methods for computer vision: A review', International Journal of Computer Vision, Vol.6, No.1, pp.59-70, 1991.

[38] Francois G. Meyer, Amir Z. Averbuch and Jan-Olov Stromberg, 'Fast adaptive wavelet packet image compression', IEEE Transaction on Image Processing, Vol.9, No.5, pp.792-800, May 2000.

[39] Adrian Munteanu, Jan Cornelis, Geert Van der Auwera and Paul Cristea, 'Wavelet image compression – The quadtree coding approach', IEEE Transactions

on Information Technology in Biomedicine, Vol.3, No.3, pp.176-85, September 1999.

[40] Adrian Munteanu, Jan Cornelis and Paul Cristea, 'Wavelet-based lossless compression of coronary angiographic images', IEEE Transactions on Medical Imaging, Vol.18, No.3, pp.272-81, March 1999.

[41] Torsten Palfner and Erika Muller, 'Effects of symmetric periodic extension for multiwavelet filter banks on image coding', IEEE Proceedings of International Conference on Image Processing (ICIP'1999), Vol.1, pp.628-32, 1999.

[42] Fernando Pereira, 'MPEG-4: Why, how and when?', Signal Processing: Image Communication, Vol.15, No.4-5, pp.271-9, Elsevier Science, January 2000.

[43] Niall C. Phelan and Joseph T. Ennis, 'Medical image compression based on a morphological representation of wavelet coefficients', Medical Physics, Vol.26, No.8, pp.1607-11, August 1999.

[44] See-May Phoong, Chai W. Kim, P.P.Vaidyanathan and Rashid Ansari, 'A new class of two-channel biorthogonal filter banks and wavelet bases', IEEE Transaction on Signal Processing, Vol.43, No.3, pp.649-65, March 1995.

[45] Nikolay Polyak and William A. Pearlman, 'A new flexible bi-orthogonal filter design for multiresolution filterbanks with application to image compression', IEEE Transactions on Signal Processing, Vol.48, No.8, pp.2279-88, August 2000.

[46] Richard Qian, Niels Haering and Ibrahim Sezan, 'A computational approach to semantic event detection', IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol.1, pp.200-6, 1999.

[47] Majid Rabbani and Rajan Joshi, 'An overview of the JPEG 2000 still image compression standard', Signal Processing: Image Communication, Vol.17, No.1, pp.3-48, Elsevier Science, January 2002.

[48] Injong Rhee, Graham R. Martin, S. muthukrishnan and Roger A. Packwood, 'Quadtree-structured variable-size block-matching motion estimation with minimal error', IEEE Transactions on Circuits and Systems for Video Technology, Vol.10, No.1, pp.42-50, February 2000.

[49] Diego Santa-Cruz, Raphael Grosbois and Touradj Ebrahimi, 'JPEG 2000 performance evaluation and assessment', Signal Processing: Image Communication, Vol.17, No.1, pp.113-130, Elsevier Science, January 2002.

[50] Harpreet S. Sawhney and Serge Ayer, 'Compact representations of videos through dominant and multiple motion estimation', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.18, No.8, pp.814-30, August 1996.

[51] Donald F. Schomer, Almos A. Elekes, John D. Hazle, John C. Huffman, Stephen K. Thompson, Charles K. Chui and William A. Murphy, 'Introduction to wavelet-based compression of medical images', Imaging & Therapeutic Technology, Vol.18, No.2, pp.469-81, March-April 1998.

[52] A.Sharaf and F.Marvasti, 'Motion compensation using spatial transformations with forward mapping', Signal Processing: Image Communication, Vol.14, pp.209-27, Elsevier Science, 1999.

[53] Ke Shen and Edward J. Delp, 'Wavelet based rate scalable video compression', IEEE Transactions on Circuits and Systems for Video Technology, Vol.9, No.1, pp.109-122, February 1999.

[54] Christoph Stiller and Janusz Konrad, 'Estimating motion in image sequences', IEEE Signal Processing Magazine, Vol.16, No.4, pp.70-91, July 1999.

[55] Shen-Chuan Tai, Yung-Gi Wu and Chang-Wei Lin, 'An adaptive 3-D discrete cosine transform coder for medical image compression', IEEE Transactions on Information Technology in Biomedicine, Vol.4, No.3, pp.259-63, September 2000.

[56] R.H.G. Tan, J.F.Zhang, R.Morgan and A.Greenwood, 'Still image compression based on 2D discrete wavelet transform', IEE Electronics Letters, Vol.35, No.22, pp.1934-5, 28 October 1999.

[57] David Taubman and Michael Marcellin. JPEG2000: Image compression fundamentals, standards, and practice. Kluwer Academic Publishers, Boston and London, 2002.

[58] David Taubman, Erik Ordentlich, Marcelo Weinberger and Gadiel Seroussi, 'Embedded block coding in JPEG 2000', Signal Processing: Image Communication, Vol.17, No.1, pp.49-72, Elsevier Science, January 2002.

[59] David Taubman, Erik Ordentlich, Marcelo Weinberger, Gadiel Seroussi, Ikuro Ueno and Fumitaka Ono, 'Embedded block coding in JPEG2000', IEEE Proceedings of International Conference on Image Processing (ICIP'2000), Vol.2, pp.33-36, 2000.

[60] P.P.Vaidyanathan, 'Quadrature mirror filter banks, M-band extensions and perfect-reconstruction techniques', IEEE Acoustic, Speech, and Signal Processing Magazine, Vol.35, No.7, pp.4-20, July 987.

[61] Jozsef Vass, Bing-Bing Chai, Kannappan Palaniappan and Xinhua Zhuang, 'Significance-linked connected component analysis for very low bit-rate wavelet video coding', IEEE Transactions on Circuits and Systems for Video Technology, Vol.9, No.4, pp.630-47, June 1999.

[62] C. De Vleeschouwer and B. Macq, 'Subband dictionaries for low-cost matching pursuits of video residues', IEEE Transactions on Circuits and Systems for Video Technology, Vol.9, No.7, pp.984-93, October 1999.

[63] James S. Walker, 'Lossy image codec based on adaptively scanned wavelet difference reduction', Optical Engineering, Vol.39, No.7, pp.1891-7, SPIE, July 2000.

[64] Qi Wang and Mohammed Chanbari, 'Scalable coding of very high resolution video using the virtual zerotree', IEEE Transactions on Circuits and Systems for Video Technology, Vol.7, No.5, pp.719-27, October 1997.

[65] Lora G. Weiss, 'Wavelets and wideband correlation processing', IEEE Signal Processing Magazine, Vol.11, No.1, pp.13-32, January 1994.

[66] Roland Wilson, 'Image analysis and segmentation using mixture models', IEE Colloquium on Time-Scale and Time-Frequency Analysis and Applications, pp.11/1-6, London , 29 February 2000.

[67] Zixiang Xiong, Kannan Ramchandran, Michael T. Orchard and Ya-Qin Zhang, 'A comparative study of DCT and wavelet-based image coding', IEEE Transactions on Circuits and Systems for Video Technology, Vol.9, No.5, pp.692-5, August 1999.

[68] Zixiang Xiong, Xiaolin Wu, D.Y.Yun and W.A.Pearlman, 'Progressive coding of medical volumetric data using three-dimensional integer wavelet packet transform', IEEE Second Workshop on Multimedia Signal Processing, pp.553-8, Piscataway, NJ, USA, 1998.

[69] Jizheng Xu, Zixiang Xiong, Shipeng Li and Ya-Qin Zhang, 'Memory-Constraint 3-D Wavelet Transform for Video Coding Without Boundary effects', IEEE Transactions on Circuit and Systems for Video Technology, Vol.12, No.9, pp.812-18, September 2002.

[70] Xuguang Yang and Kannan Ramchandran, 'Scalable wavelet video coding using aliasing-reduced hierarchical motion compensation', IEEE Transactions on Image Processing, Vol.9, No.5, pp.778-91, May 2000.

[71] Minerva M. Yeung and Boon-Lock Yeo, 'Video visualization for compact presentation and fast browsing of pictorial content', IEEE Transactions on Circuits and Systems for Video Technology, Vol.7, No.5, pp.771-85, October 1997.

[72] Yufei Yuan and Choong Wah Chan, 'Coding of arbitrarily shaped video objects based on SPIHT', IEE Electronics Letters, Vol.36, No.13, pp.1105-6, 22 June 2000.

[73] Wenjun Zeng, Scott Daly and Shawmin Lei, 'An overview of the visual optimization tools in JPEG 2000', Signal Processing: Image Communication, Vol.17, No.1, pp.85-104, Elsevier Science, January 2002.

[74] Kui Zhang and Josef Kittler, 'Global motion estimation and robust regression for video coding', Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, Vol.5, pp.2589-92, 1998.

[75] Zongping Zhang, Guizhong Liu, and Yiwen Yang, 'High performance full scalable video compression with embedded multiresolution MC-3DSPIHT', IEEE Proceedings of International Conference on Image Processing (ICIP'2002), Vol.3, pp.721-24, 2002.