

---

# Hierarchical Lattice Vector Quantisation Of Wavelet Transformed Images

---

Madhav Vij

*Trinity College*

A dissertation submitted to the University of Cambridge for the degree of  
Doctor of Philosophy

**To the memory of my grandfather**

## **Declaration**

---

The research described in this dissertation is entirely original, except as indicated in the text, and is not the result of work done in collaboration. No part of this dissertation has been submitted to any other University. The length of the dissertation is approximately 53000 words.

Signed,

Madhav Vij

## **Acknowledgements**

---

I would sincerely like to thank my supervisor, Dr. N. G. Kingsbury, for his endless encouragement and advice during the preparation of this thesis. I would also like to thank him for his valued suggestions and comments during my time at Cambridge. I would also like to thank various members of the Signal Processing Group who have helped me in different ways. The financial support was provided by a Robert Gardiner scholarship, the Science and Engineering Research Council (SERC), Trinity College Cambridge (for providing me with subsidised accommodation) and my very generous parents.

## **Keywords**

---

The following keywords may be useful for indexing purposes:

Image coding; discrete wavelet transform; arithmetic coding; context-based coding; lattice vector quantisation; pyramid vector quantisation; successive approximation coding; inter-band vector; embedded coding.

## **Abstract**

The objectives of the research were to develop embedded and non-embedded lossy coding algorithms for images based on lattice vector quantisation and the discrete wavelet transform. We also wanted to develop context-based entropy coding methods (as opposed to simple first order entropy coding).

The main objectives can therefore be summarised as follows: (1) To develop algorithms for intra and inter-band formed vectors (vectors with coefficients from the same sub-band or across different sub-bands) which compare favourably with current high performance wavelet based coders both in terms of rate/distortion performance of the decoded image and also subjective quality; (2) To develop new context-based coding methods (based on vector quantisation).

The alternative algorithms we have developed fall into two categories: (a) Entropy coded and Binary uncoded successive approximation lattice vector quantisation (SA-LVQ-E and SA-LVQ-B) based on quantising vectors formed intra-band. This is an embedded coding algorithm where truncating the received bit-stream at any point can produce reconstructed images at a series of lower bit-rates; (b) Entropy coded pyramid vector quantisation (ECPVQ) algorithm based on forming vectors inter-band. This is a non-embedded coding algorithm where the bit-rate and therefore the distortion of the decoded image can be adjusted only at the encoder (by adjusting the resolution of the lattice quantiser).

We have found some mixed results for the category (a) algorithms which we have developed. The binary uncoded version (no entropy coding) outperforms an equivalent scalar quantisation based method but the reverse is true for the respective entropy coded versions. This leads us to conclude that optimal scalar quantisation based embedded coding algorithms are likely to be superior to their vector quantisation based equivalents.

This development lead us to concentrate on non-embedded lattice VQ algorithms (category (b)). We developed a new algorithm called ECPVQ which codes vectors formed in a hierarchical way (based on forming vectors by grouping coefficients from different scales). The main original contribution was in grouping large numbers of equiprobable lattice code-vectors into a *few* groups known as *sub-classes/super-classes* on pyramidally shaped shells. This enabled *efficient* codes to be designed from training to entropy code the class indices. We found that the optimal quantiser, which we have called the  $Z_n/D_n$  *augmented lattice*, in terms of maximising rate vs. PSNR performance, was in fact a combination of the well known  $Z_n$  and  $D_n$  lattices. This actually involves using finer quantisation near the origin which is contrary to the approach adopted by many researchers using scalar quantisation based coders. We also developed an efficient context-based code for one of the three entropy codes which we needed to design for the ECPVQ algorithm.

Our results, using a huffman and an arithmetic coder, show that ECPVQ is comparable in terms of rate vs. PSNR performance, particularly at low bit rates, with the very best current state of the art wavelet based coders [55] and certainly superior to all lattice quantisation based coders as far as the author is aware. More importantly, we found that the decoded images from ECPVQ tend to better preserve subtle texture detail than the state of the art coders with which we compared our results visually (methods of Said and Pearlman [50] and the context based scalar quantisation algorithm of Chrysafis and Ortega [52]). The reason for this appears to be because of the finer quantisation of low energy wavelet coefficients that occurs with the augmented lattice.

---

# CONTENTS

---

**Declaration**

**Abstract**

**Contents**

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Aims and summary of the research	1
1.2	Layout of thesis	4
<b>2</b>	<b>Review of transform coding and filter design</b>	<b>6</b>
2.1	Transform coding	6
2.1.1	Basic Theory	6
2.1.2	Two dimensional transforms	7
2.1.3	Performance measure for transforms	8
2.1.4	The discrete cosine transform	8
2.2	Problems with the DCT	8
2.3	Subband decompositions	9
2.3.1	Decimation and interpolation	10
2.3.2	Two channel PR-QMF bank	10
2.4	Non-uniform subband decompositions	13
2.4.1	The wavelet transform	14
2.4.2	Finite resolution decompositions	15

2.4.3	Implementation using two-band filterbanks	16
2.5	The choice of “Wavelet” filters	17
2.6	Proper signal extension for perfect reconstruction	18
2.6.1	Periodic extension	19
2.6.2	Symmetric extension	19
2.7	Design of linear phase wavelet filters	20
2.7.1	Disadvantage of longer wavelet filters	21
<b>3</b>	<b>Scalar/Vector quantisation and entropy coding</b>	<b>23</b>
3.1	Quantisation techniques	23
3.1.1	Scalar quantisation	24
3.1.2	Vector quantisation	25
3.1.2.1	Training based vector quantisation	25
3.1.2.1.1	LBG algorithm	26
3.1.2.1.2	Complexity issues	30
3.1.2.1.2.1	Searching algorithms	30
3.1.2.1.2.2	Alternative VQ systems	31
3.1.2.1.3	Advanced VQ systems	33
3.1.2.2	Introduction to lattice VQ	35
3.1.2.2.1	Lattice VQ	35
3.1.2.2.1.1	Particular lattices	36
3.2	Review of entropy coding	37
3.2.1	Higher order entropy coding	38
3.2.1.1	Complexity of higher order entropy systems	39
3.2.1.1.1	Simplifications which are used	39
3.2.1.2	Conditioning tree approach	40
3.2.1.3	Choosing conditional pixel sequence	40

3.2.1.4	Implementation of conditional entropy coding schemes	41
3.3.	Brief description of arithmetic coding	41
<b>4</b>	<b>A successive approximation coder based on vector quantisation</b>	<b>44</b>
4.1	Review of previous work on scalar and vector based successive approximation algorithms	44
4.1.1	Embedded coding algorithms based on scalar quantisation	45
4.1.1.1	Shapiro's zero-tree algorithm for transmitting the ordering information	47
4.1.1.2	Set partitioning algorithm of Said and Pearlman	48
4.1.2	Successive approximation vector quantisation	49
4.1.2.1	An algorithm for the successive approximation of vectors	49
4.1.2.1.1	Definition of the problem	49
4.1.2.1.2	Conditions for convergence	51
4.1.2.1.3	Algorithm of Da Silva and Ghanbari	52
4.2	New algorithms for successive approximation coding of vectors	55
4.2.1	Implementation of SA-LVQ algorithm	55
4.2.1.1	Coding algorithm	56
4.3	Comparisons in performance between various SA-LVQ algorithms	59
4.3.1	Some results for SA-LVQ algorithms	62
4.4	Subjective performance of SA-LVQ algorithms	66
4.5	Conclusions	69
4.6	Future research	69

<b>5</b>	<b>A novel class based method for coding hierarchical vectors using entropy coded pyramid vector quantisation (ECPVQ)</b>	<b>71</b>
5.1	Review of previous entropy coded lattice VQ work	71
5.1.1	Review of Fisher's pyramid vector quantiser for inter and intra-band vectors	75
5.2	Motivation for developing new ECPVQ method	79
5.3	Choice of lattice for ECPVQ method	80
5.4	Choice of vectors/sub-band decomposition for new ECPVQ method	81
5.5	Class structures in the ECPVQ method (including block diagrams of encoder and decoder)	84
5.5.1	Introduction of the <i>class</i> structure on pyramidal shells	92
5.5.1.1	Enumeration encoding/decoding algorithms for indexing of classes	95
5.5.2	Introduction of the <i>sub-class</i> structure for hierarchical (or inter-band) vectors	96
5.5.2.1	Finding index of sub-class given class and $K$	98
5.5.3	Introduction of the <i>super-class</i> structure (equivalent of merging <i>sub-classes</i> )	102
5.5.3.1	Definition of <i>super-classes</i> for 21-D vectors	103
5.5.3.2	Enumeration encoding/decoding algorithms for determining the index of a <i>super-class</i>	104
5.5.3.3	Enumeration encoding/decoding algorithms for determining index of specific code-vector in a particular <i>super-class</i>	105
5.5.3.4	Method for determining index of code-vector of dimension $L$ on a pyramidal shell indexed by $K$	105

5.5.4	Introduction of the <i>super-super-class</i> (equivalent to merging of <i>super-classes</i> ) structure	106
5.5.4.1	Heuristic approach to merging <i>super-classes</i>	107
5.5.4.1.1	Motivation for using heuristic formula	108
5.5.4.2	Examples using merging formula based on training set statistics	111
5.5.4.3	Overall merging strategy for outer shells	113
5.6	Additional stages of ECPVQ algorithm	115
5.6.1	Coding of DC band using differential coding	116
5.6.2	Determination of reproduction vectors at decoder	120
5.6.3	Coding of shell index for 21-D hierarchical vectors	128
5.6.3.1	Context state merging	133
5.7	Coding results for $256 \times 256$ and $512 \times 512$ monochrome images	140
5.7.1	Coding of $256 \times 256$ images	140
5.7.1.1	Justification for using variations of the <i>sub-class</i> , <i>super-class</i> and <i>super-super-class</i> structures	141
5.7.1.2	Performance comparison of ECPVQ using 4 different filters pairs in the wavelet transform	143
5.7.1.3	Coding results using ECPVQ method	146
5.7.1.3.1	Some comparisons between different lattices used for quantisation in ECPVQ method	156
5.7.2	Coding of $512 \times 512$ images	159
5.7.2.1	Coding of 5-D vectors	160
5.7.2.2	Coding results for images Lena and Goldhill	163
5.7.2.3	Bit allocation between sub-bands	174
5.7.3	Computational complexity of ECPVQ	177

<b>6</b>	<b>Conclusions and suggestions for future research</b>	<b>179</b>
6.1	Conclusions	179
6.2	Suggestions for future research	180
	<b>Appendices</b>	<b>184</b>
A.1	Implementation of simple scalar quantisation based benchmark coder	185
A.1.1	Scalar quantisation based coder	185
A.1.2	Some results for benchmark coder	190
A.1.3	Reasons for rate vs. PSNR performance of different filters	193
A.1.4	Reasons for subjective performance of different filters	198
A.2	Derivation of a formula for the merging of super-classes	200
A.3	Definition of contexts used in conditional entropy coding scheme	202
A.3.1	Method for selecting contexts within each block of vectors	202
A.3.2	Context selection using modified method	204
A.4	Arithmetic/Huffman coding of $N$ equally probable symbols where $N$ is not a power of 2	205
A.4.1	Approach by arithmetic coding	205
A.4.2	Approach by huffman coding	206
A.5	Training set images	208
A.6	Relationships between symbol alphabets used in design of a conditional entropy code in 5.6.3 and shell index for 3 different lattices	212
A.7	Enumeration algorithms for encoding/decoding <i>classes</i> and <i>super-classes</i>	214
A.7.1	Enumeration methods for partitioning an integer	215
A.7.2	Enumeration algorithms for the <i>super-class</i> structure	215

A.7.2.1	Algorithms for indexing a specific <i>super-class</i>	217
A.7.2.2	Enumeration algorithms for indexing a specific code-vector within a given <i>super-class</i>	219
A.8	New enumeration encoding/decoding algorithms for determining the index of a specific code-vector of dimension $L$ on a pyramidal shell of index $K$	221
A.9	Enumeration algorithms for permutations	227
A.9.1	Permutations with no repetitions	228
A.9.2	Permutations with repetitions	228
<b>Bibliography</b>		<b>229</b>

## **Chapter 1**

---

### **Introduction**

---

In this chapter we give a summary of the research which is outlined in this dissertation. We have also given the layout of the thesis and we briefly detail the contents of each chapter.

#### **1.1. Aims and summary of the research**

The main research area of this dissertation has been in developing embedded and non-embedded still image coding algorithms for the lossy compression of digital grayscale images. The three main components of a lossy compression system tend to be the transform phase, the quantisation phase and the entropy coding phase. Our work has concentrated on the latter two phases.

In the transform phase, we have used a 2-D discrete separable wavelet transform. We have briefly investigated the performance of different filters based on our coding algorithms.

In the quantisation phase we have used lattice vector quantisation (LVQ) and in the entropy coding phase we have used both simple first order entropy codes and have also developed a context-based coding scheme.

The first original contribution of this dissertation is the extension of an existing algorithm by Da Silva and Ghanbari [76] for the successive approximation coding of vectors. This is an embedded coding algorithm whereby the transmitted bit-stream can be truncated at any point and produce a series of decoded images at lower bit-rates. We have applied the ideas used in a the state of the art embedded coding algorithm by Said and

Pearlman based on scalar quantisation [50] to develop a more efficient method for one part of the overall algorithm (known as the sorting phase [50]). We have found some mixed results. One version of our method which uses no entropy coding (binary uncoded) seems to outperform the binary uncoded version of the scalar quantisation based algorithm but the entropy coded version of the scalar quantisation based algorithm seems to be superior to the entropy coded version of our method. We therefore suggest that scalar quantisation based successive approximation methods (i.e. [50]) seem to be superior in general to vector quantisation based algorithms.

This development has lead us to concentrate on our next original contribution which is in developing a non-embedded lattice VQ method which we have called ECPVQ (Entropy coded pyramid vector quantisation). The bit-rate can be controlled by adjusting a single parameter (equivalent to the quality factor parameter of baseline JPEG). We have formed the vectors out of wavelet coefficients in a hierarchical way (using inter-band vectors where coefficients are obtained from sub-bands of similar orientation on different scales). We have chosen to use 21-D vectors. A novel class based entropy coding algorithm based on these hierarchical vectors is developed after lattice VQ has been performed. We have developed a class based method for entropy coding of the positional information of each quantised vector on pyramidally shaped shells. Class structures have been developed whereby the vectors within these classes are likely to be equiprobable. As the number of these classes is *small, efficient* entropy codes based on training set data can be developed. Then a simple near-uniform code can be used to transmit the index of the vector within a particular class. Two main class based structures called the sub-class and super-class methods are developed. With the first of these structures it should be noted that the same overall results would be obtained if the widely used spherically shaped shells were used rather than pyramidally shaped ones (this would be strictly only true if the shell index (or integer index of the pyramid on which the quantised vector lies) were coded by a first order entropy code rather than the efficient context-based code which we

use; if a first order code were used, this code could be combined with the first order code used to encode the index of the sub-class within which the quantised vector lies, to create a single code). Enumeration encoding and decoding algorithms are realised so that a practical coder can be developed.

Our next novel contribution is in developing a quantiser (which we have termed an *augmented lattice* as it is not a lattice in the strictest sense) which is a combination of the two well known lattices,  $Z_n$  and  $D_n$ , which we find outperforms either of those two on a rate vs. distortion basis (and subjectively outperforms the  $D_n$  lattice). This actually involves using finer quantisation near the origin which is contrary to the typical approach adopted in coders using scalar quantisation (SQ) which get improved performance by increasing the width of the “dead-zone” of mid-tread SQ’s.

We next develop a context-based coding method for coding the index of the pyramidal shells (or the radial parameter). We find a 20-30% reduction in entropy over a first order entropy code. Recent developments in scalar quantisation based context-coding methods would suggest that this is an area for future research.

Finally our results show that the performance of ECPVQ is comparable with the very best current state of the art wavelet based coders [55], particularly at high compression ratios, in terms of bit-rate vs. PSNR (peak signal to noise ratio) performance and certainly superior to all lattice quantisation based sub-band coders as far as the author is aware.

But more importantly, we have found that the decoded images from our method tend to better preserve subtle texture detail than current state of the art coders which tend to be scalar quantisation based [55] (e.g method of Said and Pearlman [50], the space frequency quantisation method of Xiong et al [51], the context-based scalar quantisation method of Ortega et al. [52] and the estimation quantisation algorithm of Orchard et al. [53]). The reason for this appears to be because of the quantisation we use (finer quantisation of low energy high frequency wavelet coefficients than other methods).

It should be noted that in common with those other methods (e.g [50,51,52,53]), we attempt to optimise rate vs. distortion performance (as opposed to specifically minimising a subjective error measure).

## **1.2 Layout of thesis**

We shall now briefly give the layout of the thesis.

**Chapter one** is the introduction chapter outlining the main aims and a summary of the research and a layout of the thesis.

**Chapter two** is a review chapter on the DCT and other subband transforms including the Wavelet transform. Filter design methods are also discussed as is implementation of discrete versions of sub-band transforms using filter banks.

**Chapter three** is a review chapter on scalar and vector quantisation. We also review entropy coding techniques (i.e. first order and conditional entropy codes) and mention practical codes such as arithmetic coding.

**Chapter four** contains our first original contribution of this dissertation which is the extension of an existing successive approximation vector quantisation algorithm for vectors which are formed intra-band. We begin the chapter with a brief review of progressive transmission methods and current scalar quantisation based embedded coding algorithms.

**Chapter five** which contains the central and most important work of this dissertation outlines the development of our ECPVQ algorithm. We develop novel class based structures for hierarchically formed vectors to design efficient entropy codes based on training set data. We investigate the performance of ECPVQ using different lattices including a new one which is a combination of existing lattices. We develop a new context-based coding scheme to encode the radial parameter from ECPVQ. We also develop a new method for determining the re-production vectors at the decoder. Finally we do a subjective and rate vs. distortion evaluation of our algorithm with current state of the art wavelet based coders. This chapter begins with a brief review of lattice vector quantisation.

**Chapter six** summarises the main conclusions that can be drawn from the work of chapter five. We also give suggestions for future research.

The **Appendices** include the following topics: implementation of a simple scalar quantisation based benchmark coder (a graphical method is suggested for the subjective evaluation of differences between different filter pairs); a look at the training set images which we have used; more detail on the context-based coding scheme which we develop in chapter 5; description in pseudo-code of all the enumeration encoding/decoding algorithms which we needed to develop in order to implement the algorithms of chapter 5; details on the huffman coder which we use in chapter 5.

# Review of Transform Coding and Filter Design

---

The basic theory is presented, and then common transforms for image coding are discussed, including the discrete cosine transform (DCT) and the lapped orthogonal transform (LOT). *Subband* coding can be considered a generalisation of transform coding and is reviewed in section 2.3. In non-uniform subband decomposition's described in section 2.4., the signal is decomposed into subbands of non-uniform bandwidth which can be advantageous in image applications. *Wavelet* analysis provides a mathematical framework for describing this technique and is described in detail. Finally, we shall also consider the design of wavelet filters.

## 2.1 Transform coding

In a transform coder, the discrete data signal is segmented into non-overlapping blocks, and each block is expressed as a weighted sum of discrete *basis functions*. The purpose of transform coding is to decompose the correlated signal samples into a set of uncorrelated *transform coefficients*, such that the energy is concentrated into as few coefficients as possible. The encoding process can then allocate most of the available encoding bits to the more energetic transform coefficients.

### 2.1.1 Basic theory

Denote a block of  $N$  signal samples a column vector  $F = [f_0, f_1, f_2, \dots, f_{N-1}]^T$ . Now suppose  $\phi_r$ ,  $r=0, \dots, N-1$  represents a family of linearly independent column vectors of length  $N$ ; these form the decomposition basis.

The signal vector  $F$  can then be decomposed as follows:

$$F = \theta_0 \phi_0 + \theta_1 \phi_1 + \dots + \theta_{N-1} \phi_{N-1} \quad (2.1)$$

where  $\theta_r$  is the weight corresponding to  $\phi_r$ . In matrix notation, it is

$$F = \phi^T \theta \quad (2.2)$$

where  $\phi$  is an  $N \times N$  transform matrix with elements  $\phi(r,k) = \phi_r(k)$  (so the  $r^{\text{th}}$  row of  $\phi$  contains the basis function  $\phi_r$ ), and  $\theta = [\theta_0, \theta_1, \dots, \theta_{N-1}]^T$ . The transform coefficients,  $\theta$ , can be obtained from the signal samples by rearranging equation (2.1) as  $\theta = (\phi^T)^{-1} F$ . If the vector  $\phi_r$ , represents an orthonormal basis, then the matrix  $\phi$  has the property  $\phi^{-1} = \phi^H$  (where  $\phi^H$  is the conjugate transpose of  $\phi$ ). Hence the forward transform becomes:

$$\theta = \phi^* F \quad (2.3)$$

Note that block transforms may be viewed as filterbanks. The signal  $F(n)$  is passed through a bank of  $N$  filters with impulse responses  $h_r(n) = \phi_r^*(N-1-n)$  (by considering equation 2.3 as a set of  $N$  convolutions) and the filter output signals are sub-sampled by  $N$ .

### 2.1.2. Two dimensional transforms

For image coding, two dimensional transforms are normally applied to  $N \times N$  pel sub-blocks of the image as this allows correlations in both dimensions to be exploited. However, for computational reasons, it is desirable that the 2-D basis functions, denoted  $\phi_{rs}(k,l) = \phi_r(k) \phi_s(l)$ . 2-D transforms satisfying this constraint are termed *separable*. If  $F$  represents the 2-D data block and  $\theta$  is the 2-D coefficient block, then a separable 2-D transform can be expressed as

$$\theta = \phi^* F \phi^H ; F = \phi^T \theta \phi \quad (2.4)$$

This allows the transformation to be achieved in two stages. First each column of the data block is transformed using  $\phi^*$ , then the rows of results are transformed similarly.

### 2.1.3. Performance measure for transforms

The performance measure of a transform can be measured in two ways:

- By degrees of decorrelation of the signal achieved by the transform. An ideal transform decomposes the signal into uncorrelated coefficients (which may be efficiently transmitted using a first order entropy code)
- By amount of energy compaction provided by the transform. An ideal transform should put the maximum signal energy into any given fraction of the transform coefficients.

### 2.1.4. The discrete-cosine transform

The discrete cosine transform [1,2] is the most common transform for image coding applications. It is signal independent and transforms signal samples  $x(n)$  to coefficients

$X(m)$  according to :

$$X(m) = \sum_{n=0}^{N-1} C_m x(n) \cos \frac{(2n+1)m\pi}{2N}, 0 \leq m \leq N-1 \quad (2.5)$$

$$x(n) = \sum_{m=0}^{N-1} C_m X(m) \cos \frac{(2n+1)m\pi}{2N}, 0 \leq n \leq N-1 \quad (2.6)$$

$$\text{where } C_m = \begin{cases} \sqrt{1/N} & m=0 \\ \sqrt{2/N} & m \neq 0 \end{cases}$$

The DCT can be represented using equations (2.5),(2.6) where the transformation matrix  $D$  is real and has elements :

$$d(m,n) = C_m \cos \frac{(2n+1)m\pi}{2N} \quad (2.7)$$

The DCT basis functions and corresponding frequency responses are given in [89].

## 2.2. Problems with the DCT

The DCT, and other block transforms, segment a signal into non-overlapping blocks and process each block independently. At low encoding bit-rates, the transform coefficients must be coarsely quantised, and so there will be significant reconstruction error after decoding the signal. A particular problem with block transforms is that

discontinuities in the reconstructed signal will, in general, occur across the block boundaries used to segment the signal. Thus the position of the blocks used in the encoding process may be visible in reconstructed images. This is known as the blocking effect.

The main reason that blocking can occur when using the DCT is that the basis functions do not decay towards zero at their end-points. This means that large discontinuities arise in the reconstructed signal at the ends of the basis functions (following quantisation of transform coefficients). A solution to this is to use *lapped transforms* in which the basis functions applied to adjacent data blocks are allowed to overlap yet no redundancy is generated. We will not discuss lapped transforms [3] except to say that the wavelet transform, which we shall consider, is a special type of lapped transform.

### **2.3. Subband decompositions**

The basic objective of subband coding is to divide the signal frequency band into a set of uncorrelated frequency bands by filtering and then to encode each of the subbands.

The subband coder achieves energy compaction by filtering data serially whereas transform coding utilises block transformations. If the subbands have little spillover from adjacent bands, the quantisation noise in a given band is confined to that band.

Starting with the two channel filterbank, we first derive the conditions the filters must satisfy for zero aliasing and then the more stringent requirements for perfect reconstruction with emphasis on the orthonormal solution. Expanding this two-band structure recursively in a hierarchical subband tree generates a variety of multiband PR realisations with equal or unequal band splits, as desired.

### 2.3.1. Decimation and interpolation

Decimation is the process of reducing the sampling rate of a signal by an integer factor  $M$ . This process is achieved by passing the full-band signal  $\{x(n)\}$  through an antialiasing filter  $h(n)$  and then subsampling the filtered signal to give  $y(n)$ . The effect of these operations is to expand the frequency spectrum of the signal. Mathematically, the process is described by:

$$X'(z) = X(z)H(z) \quad \text{and} \quad Y(z) = \frac{1}{M} \sum_{m=0}^{M-1} X'(z^{1/m} w^k) \quad (2.8)$$

where  $w = e^{-j2\pi/M}$ . To avoid aliasing during the subsampling process, the original signal spectrum  $X(e^{j\omega})$  must initially be filtered to reduce its bandwidth to  $\pm\pi/M$ . This is achieved using a low pass antialiasing filter  $h(n)$

Interpolation is the process of increasing the sampling rate of a signal by an integer factor  $M$ . This process is achieved by the combination of up-samples and low-pass filter  $g(n)$  to give the output signal  $\hat{x}(n)$ . The process is described mathematically as:

$$\hat{Y}(z) = Y(z^M) \quad \text{and} \quad \hat{X}(z) = \hat{Y}(z)G(z) \quad (2.9)$$

Therefore, up-sampling has two effects. First stretching the time axis induces a compression in frequency; Second, forcing the "interpolated" signal to pass through zero between samples of  $y(n)$  generates high-frequency signals or images. The images can be removed by a low pass filter which has bandwidth  $\pm\pi/M$ .

### 2.3.2. Two- channel PR-QMF bank

Figure 2.1 (at the end of this chapter) shows a 2-band subband coder for a 1-D signal.

Let  $h_0(n)$  be a FIR low-pass filter with real coefficients. The *mirror* filter is defined as:

$$h_1(n) = (-1)^n h_0(n) \quad (2.10)$$

or equivalently, in the transform domain,

$$H_1(z) = H_0(-z) \quad (2.11)$$

$$H_1(e^{j\omega}) = H_0(e^{j(\omega-\pi)}) \quad (2.12)$$

It can be shown that  $H_0$  and  $H_1$  are mirror images of each other about  $w = \pi/2$ , hence the appellation *quadrature mirror filters*.

In this section, we derive the requirements and properties of a perfect reconstruction, two-channel subband system. Using the relationships for decimation and interpolation filters from equations 2.8 and 2.9., with  $M=2$ , it is straightforward to show that:

$$\hat{X}_i(z) = \frac{1}{2} G_i(z) [H_i(z) X(z) + H_i(-z) X(-z)] \quad (2.13)$$

where  $i=0,1,\dots$  correspond to the branches in the filterbank. Therefore the reconstruction signal,  $\hat{X}(z) = \hat{X}_0 + \hat{X}_1$ , is given by:

$$\begin{aligned} \hat{X}(z) &= \frac{1}{2} X(z) [H_0(z)G_0(z) + H_1(z)G_1(z)] \\ &\quad + \frac{1}{2} X(-z) [H_0(-z)G_0(z) + H_1(-z)G_1(z)] \\ &= T(z)X(z) + S(z)X(-z) \end{aligned} \quad (2.14)$$

In this expression,  $S(z)X(-z)$  represents the effect of aliasing, so this term must be set to zero for perfect reconstruction. The second requirement for PR is that  $T(z)$  represents a pure delay. Thus the constraints are

- (a)  $S(z) = 0$  for all  $z$
- (b)  $T(z) = \alpha z^{-\beta}$ , where  $\alpha$  and  $\beta$  are constants ( $\beta$  integral)

This can be achieved by the solution:

$$G_0(z) = -H_1(-z) \quad (2.15)$$

$$G_1(z) = H_0(-z) \quad (2.16)$$

leaving us with [4]

$$T(z) = \frac{1}{2} [H_0(-z)H_1(z) - H_0(z)H_1(-z)] \quad (2.17)$$

In the classic QMF solution [5],

$$H_1(z) = H_0(-z) \quad (2.18)$$

For this class of analysis/reconstruction system, exact reconstruction requires that

$$H_0^2(e^{j\omega}) - H_1^2(e^{j\omega}) = 2 \quad (2.19)$$

A number of authors [6] have designed FIR filters to approximate this condition. There are two simple cases that exist where the exact reconstruction conditions of (2.19) are exactly satisfied. The first is the case where the analysis filters are infinitely long, ideal half band filters. Obviously, the resulting system is distortionless but not very useful for real implementations. The second case occurs when  $H_0(z)$  and  $H_1(z)$  are of order one or less, for example,

$$H_0(z) = 1 + z^{-1} \quad (2.20)$$

$$H_1(z) = 1 - z^{-1} \quad (2.21)$$

These filters lack the frequency resolving power present in higher order filters.

However, PR can be attained by using the FIR paraunitary solution derived by Smith and Barnwell [4].

He defines  $H_1(z) = z^{-(N-1)} H_0(-z^{-1})$  where  $H_0(z)$  and  $H_1(z)$  are said to be *conjugate quadrature filters (CQF)* and both filters are FIR and  $N$  is even.

This choice forces  $H_1(z) = -G_0(z)$  so that:

$$T(z) = \frac{1}{2} z^{-(N-1)} \left[ H_0(z)H_0(z^{-1}) + H_0(-z)H_0(-z^{-1}) \right] \quad (2.22)$$

Therefore, the perfect reconstruction requirement reduces to finding an  $H(z) = H_0(z)$  such that  $Q(z) = H(z)H(z^{-1}) + H(-z)H(-z^{-1}) = \text{constant} = R(z) + R(-z)$ .

This selection implies that all four filters are causal whenever  $H_0(z)$  is causal. In summary, the two-band paraunitary PR FIR solution satisfies:

$$G_0(z) = -H_1(-z) \quad (2.23)$$

$$G_1(z) = H_0(-z) \quad (2.24)$$

$$H_1(z) = z^{-(N-1)} H_0(-z^{-1}) \quad (2.25)$$

$$H_0(z) = H(z) \quad (2.26)$$

and with

$$R(z) = H(z)H(z^{-1}) \Leftrightarrow \rho(n) = h(n)*h(-n) \quad (2.27)$$

$$R(z) + R(-z) = 1 \Leftrightarrow p(2n) = \delta(n) \quad (2.28)$$

It can be shown that these filters for the 2-band case can NOT be linear phase.

An alternative method can be used to obtain linear phase symmetric filters. If

$G_0(z) = -H_1(-z)$ ;  $G_1(z) = H_0(-z)$  then  $T(z)$  can be written as:

$$\begin{aligned} T(z) &= \frac{1}{2} [H_0(-z)H_1(z) - H_0(z)H_1(-z)] \\ &= \frac{1}{2} [P(z) - P(-z)] \end{aligned} \quad (2.29)$$

where  $P(z) = H_0(-z)H_1(z)$  is called the product filter to satisfy constraint (b) which requires  $T(z) = \alpha z^{-\beta}$ ,  $P(z)$  must contain only even powers of  $z$ , apart from the coefficient for a single odd power of  $z$ . For example, a valid power filter is  $P(z) = \frac{1}{16}(-1 + 9z^{-2} + 16z^{-3} + 9z^{-4} - z^{-6})$ . So  $P(z) - P(-z) = 2z^{-3}$ . Various factorisations of

$P(z)$  to give the synthesis filters are possible, for example the Le Gall filters [7],

$$H_0(z) = \frac{1}{8}(-1 + 2z^{-1} + 6z^{-2} + 2z^{-3} - z^{-4}) \quad (2.30)$$

$$H_1(z) = \frac{1}{2}(1 - 2z^{-1} + z^{-2}) \quad (2.31)$$

## 2.4 Non-uniform subband decompositions

Signal decompositions with the continuous Fourier transform provides a measure of the frequency content of a signal. However, the Fourier transform is not capable of determining when a signal had a particular frequency characteristic and so is only appropriate for stationary signals. For non-stationary signals, the short-time Fourier transform (STFT) [8] is often used. The STFT moves a fixed window over the data and analyses the frequency content in that interval. Thus, the time resolution is improved compared with the standard Fourier transform, but the frequency resolution is reduced due to the finite bandwidth of the filters used to perform the decomposition. This

illustrates a fundamental constraint of time-frequency decompositions: if the RMS "spreads" of the filters in the time and frequency domains are denoted by  $\sigma_T$  and  $\sigma_\Omega$  respectively, then  $\sigma_T\sigma_\Omega \geq \text{constant}$ . Hence, it is not possible to simultaneously achieve arbitrary time resolution AND frequency resolution.

In the case of the STFT, the compromise between time and frequency resolution is the same at all frequencies; However, it may be desirable to represent both short duration high frequency signal components and long duration low frequency components.

### **2.4.1. The wavelet transform**

It has been suggested [51] that natural images are well characterized as a linear combination of energy concentrated in both frequency and space, i.e. most of the energy of typical images is concentrated in low-frequency information, and of the remaining high-frequency components of an image, most energy is spatially concentrated around edges. Efficient transform coding of such a source model calls for a transform that compacts energy into a few low-frequency coefficients, while also representing high-frequency energy in a few, spatially-clustered high-frequency coefficients. The wavelet transform [9,10] provides these desired features. Xiong [51] suggests (based on comparison between rate vs. distortion optimised wavelet and DCT based coding schemes) that block-based DCT's are not as effective as the wavelet transform at compacting high-frequency energy around edges (i.e blocks containing edges tend to spread high-frequency energy among many coefficients).

The wavelet transform [9,10] obtains improved frequency resolution at low frequencies (at the expense of time resolution) and improved time resolution at high frequencies (at the expense of frequency resolution). This is achieved by defining the wavelet basis functions as dilations and translations of a prototype function  $\Psi(t)$ .

The wavelet basis functions are given by:

$$\Psi_{ab}(t) = \frac{1}{\sqrt{a}} \Psi\left(\frac{t-b}{a}\right) \quad (2.32)$$

where  $a$  represents the *degree of dilation* and  $b$  represents the *position of the wavelet in time*. If  $(a,b)$  are continuous parameters, then the continuous wavelet transform (CWT) decomposes a signal  $f(t)$  into wavelet coefficients  $F(a,b)$  using a convolution:

$$F(a,b) = \int_{-\infty}^{\infty} \Psi_{ab}(t) f(t) dt \quad (2.33)$$

where  $f(t)$  and  $\Psi(t)$  are assumed to be real. The signal can be reconstructed by integrating over all possible wavelet positions and dilations:

$$f(t) = k \int_{-\infty}^{\infty} \int_0^{\infty} \frac{1}{a^2} F(a,b) \Psi_{ab}(t) da db \quad (2.34)$$

where  $k$  is a constant. For the transform to be invertible,  $\Psi(t)$  must have zero mean and will be the impulse response of a bandpass filter [10].

- For practical applications, the discrete wavelet transform (DWT) is defined by sampling the parameters  $(a,b)$  on an appropriate grid. For example, the dyadic sampling grid is obtained using  $a = 2^m$  and  $b = n2^m$  where  $m,n$  are integers (the value of  $m$  for a particular wavelet is termed its *scale*). For this sampling grid, the discrete wavelet coefficients  $d_{m,n}$  are related to the signal  $f(t)$  according to:

$$f(t) = \sum_m \sum_n d_{m,n} \Psi_{m,n}(t) \quad (2.35)$$

$$d_{m,n} = 2^{-m/2} \int f(t) \Psi(2^{-m}t - n) dt \quad (2.36)$$

assuming that the wavelet set is complete and so is invertible.

## 2.4.2. Finite resolution decompositions

In the previous section, a signal  $f(t)$  was decomposed at an infinite range of different scales in order to allow PR. However, real signals have a maximum resolution beyond which no useful information is represented.

Consider a signal  $f(t)$  (which is assumed to be represented at a particular scale, denoted 0, as later discussed). The signal can be decomposed into the sum of a low resolution signal (approximation), denoted  $f_v^1(t)$  and a detail signal (approximation error),

$$f_w^1(t): \quad f(t) = f_v^1(t) + f_w^1(t) = \sum_n c_{1,n} \theta_{1,n}(t) + \sum_n d_{1,n} \Psi_{1,n}(t) \quad (2.37)$$

In this expression,  $\Psi_{1,n}(t)$  and  $d_{1,n}$  represent the wavelet function and wavelet coefficients at scale 1. However the low resolution signal is defined in terms of a *scaling* function  $\theta_{1,n}(t)$  and scale coefficients  $c_{1,n}$ .

These define the components of the signal not represented by the wavelet coefficients, so  $\theta(t)$  is a low pass function. If desired, the coarse approximation  $f_v^1(t)$  can be decomposed further:

$$f(t) = f_v^L(t) + f_w^L(t) + f_w^{L-1}(t) + \dots + f_w^1(t) \quad (2.38)$$

The signal is now represented by a low-pass approximation at scale  $L$ ,  $f_v^L(t)$ , (defined by scale coefficients  $c_{L,n}$ ), plus the sum of  $L$  detail components at higher resolutions (defined by wavelet coefficients  $d_{k,n}$ ,  $k=1..L$ ). For discrete time signals, it is assumed that the scale coefficients at scale 0, denoted  $c_{0,n}$  are the original signal samples.

### 2.4.3. Implementation using two-band filterbanks

In this review, the derivation of suitable wavelet and scaling functions is not considered; instead we describe the relationships between the discrete wavelet transform and a subband decomposition using a dyadic tree structure. In particular, note that the scale coefficients are formed from the low pass leaf of the tree, while the wavelet coefficients at each scale are produced at the other leaves. Thus the dyadic subband tree [9] is a fast algorithm for computing the DWT. In terms of equivalent low and high pass filters, it is found [11] that an orthonormal wavelet basis can be formed by using the FIR conjugate quadrature filter solution described earlier providing that  $H_0(z)$  has at least one

zero at  $z=-1$ . Other two-band PR filters can be used (i.e. the LeGall pair) to provide a non-redundant decomposition but the basis will be non-orthogonal [11]. It will be a bi-orthogonal basis[11]. Daubechies filters are the most widely used CQF's [10]. An important consideration in the design of a wavelet based decomposition is the regularity of the wavelet. Regularity is a measure of the smoothness of the wavelet and scaling functions. Daubechies [10] has shown that the degree of regularity of a wavelet increases with the number of zeroes possessed by the low-pass filter  $H_0(z)$  at  $z= -1$ . The design criteria of wavelets are discussed in more detail in a later section.

Alternatively, another non-uniform subband decomposition method other than the DWT is the Laplacian transform [12] which is computed using a Laplacian pyramid type structure----This transform is NOT a critically decimated decomposition technique and is not considered in more detail here.

One advantage of an orthonormal filter basis (i.e. non-linear phase) is that the wavelet function of the analysis and synthesis stage is identical.

## **2.5. The choice of "wavelet" filters**

"Regularity" is a new criterion brought by wavelet theory. It is important to know whether this criterion is relevant for applications such as image compression. How do regularity, frequency, selectivity and phase act upon the performance of a still image compression scheme using the wavelet decomposition?

Rioul [13] uses a separable wavelet transform. He considers orthonormal filters, because of ease of design (however, they are by definition non-linear phase). He suggests that filter banks should not deviate far from orthonormality although he concludes that this is still an open question.

He concludes that "good" selectivity in frequency is not essential for coding performance. He also says that regularity may be relevant for short filters ( $L \leq 12$ ), for which the regularity order is relatively small. He feels that using more regular filters is

useless, as compression performance does not improve greatly for longer filters. Moreover, he states that the effect of phase seems negligible to the PSNR of the reconstructed image for orthonormal filters. However, he uses the PSNR as his criterion for comparing different filters. He does not investigate separable linear phase systems.

One reason given for not using non-linear phase filters, is that these filters have to use the periodic extension method to alleviate the filtering problem at the boundaries [75]. This can result [75] in artifacts at the image boundaries at low bit rates.

Various authors (e.g [71,72,75]) have investigated the performance of linear phase bi-orthogonal filters in wavelet transforms. Ghanbari and Da Silva [75] found out of 22 bi-orthogonal filter pairs (tested on a  $256 \times 256$  Lena coded at 0.5 bpp with a product lattice quantiser), a 10/14 filter pair gave the best subjective performance. They found the PSNR measure is not always a reliable guide to the subjective quality. Karunasekera [14] has developed a subjective error measure with which different decoded images can be compared. Ghanbari et al [75] found that the product of a measure called PPR (peak-to-peak value) with the coding gain [83] (which is correlated with the PSNR measure, correlation coefficient of 0.72) is correlated to their subjective index (with correlation coefficient of 0.87). Villasenor [72] has adopted a more analytical evaluation criterion based on impulse and step response of a linear shift invariant system.

## **2.6. Proper signal extension for perfect reconstruction**

In this section, we review the methods of signal extension used in the filtering of finite length signals. These methods depend on the signal to be of even length  $N$ . Filters of finite length signals cause problems at the boundaries of the signal due to the filter crossing the signal border. Two main methods of signal extension are used to enable PR.

- (a) making the signal periodic (circular convolution)
- (b) mirroring the value of the signal with respect to the boundary.

The criterion whether a boundary extension allows PR can be summarised as follows:

*Suppose, the length of  $N$  is extended to infinity at both boundaries. If after filtering in both the lowpass and the highpass subband all coefficients can be determined from a subset of  $N/2$  samples, then the extension enables PR.*

### **2.6.1. Periodic extension**

The periodic extension method is most often used, because the method allows PR for any type of filter. When filtering is implemented in the frequency domain with the FFT, the periodic method is used implicitly.

Although, periodic extension is generally applicable, it has a drawback for image coding. In general, there is a dissimilarity between the right and left hand side of the signal. This causes a sharp transition in the extended signal and results in large subband coefficients. These large values will obstruct the coding of the subbands as they require a lot of bits in order to be represented with enough accuracy. Getz [17] suggests a method which incorporates periodic implicitly into a DWT. He calls the resulting transform, the Discrete periodic wavelet transform.

### **2.6.2. Symmetric extension**

Symmetric extension was first proposed by Smith and Eddins [15]. This technique is only possible for symmetric or antisymmetric filters and depends upon the filter length being even or odd. The input signal is mirrored at all sides and this results in a smooth extension so no sharp transitions are introduced. For even filters, the symmetry axis must be halfway between two samples. Only for the synthesis highpass filtering is an antisymmetric extension required. For odd length filters the symmetric extension must be performed in a different way to obtain  $N/2$  different samples. In the analysis section, the extension depends upon the channel. For one channel, the left side is extended with odd symmetry and the right side with even symmetry. For the other channel, the reverse holds. Thus, the synthesis extension is different from the analysis extension [16].

It can be verified easily why even symmetric extension is used for even length filters fails here: the sample values of the subbands outside the  $N/2$  values would have no relationship with the ones inside. Therefore, they contain information about the signal and this is lost since these samples are not transmitted. Barnard [16] suggests an efficient signal extension scheme for filtering of arbitrary length signals.

## 2.7. Design of linear phase wavelet filters

We have already described a method where non-linear phase filters can be obtained for a wavelet based system (i.e. Conjugate Filter solution, one solution results in Daubechies even length filters whose phase is closest to being linear). However, it may be desirable to use linear phase filters other than the simple LeGall type filters.

Kingsbury and Tay[18] suggest a technique to obtain such filters. Firstly, let  $H_0(z)$  and  $H_1(z)$  be the analysis low pass and high pass filters respectively. Also, let  $F_0(z)$  and  $F_1(z)$  be the synthesis filters. Then they suggest choosing  $H_1(z)$  and  $F_1(z)$  to be:

$$H_1(z) = z^{-k} F_0(-z) \text{ and } F_1(z) = -z^{-k} H_0(-z) \quad (2.39)$$

where  $k$  must be an odd integer. This ensures aliasing is not present. They suggest using the functions:

$$H_T(Z) = (1 + Z)(1 + aZ + bZ^2) \text{ and } F_T(Z) = (1 + Z)(1 + cZ) \quad (2.40)$$

produced a closely matched pair (i.e. that the scaling and wavelet functions of the analysis and synthesis filters are almost identical) of filters where

$$Z = \frac{1}{2}(z + z^{-1}) = \cos(\omega) \quad (2.41)$$

(2.41) is the equation for the McClellan transformation. They suggest that one chooses the parameter  $c$  and then the two ( $a$  and  $b$ ) are determined by the need for the coefficients of  $Z^2$  and  $Z^4$  in  $D_T(Z) = H_T(Z)F_T(Z)$  to be zero to ensure perfect reconstruction. They show that  $a$  and  $b$  must satisfy the following:

$$a = -\frac{(1 + 2c)^2}{2(1 + c)^2} \text{ and } b = \frac{c(1 + 2c)}{2(1 + c)^2} \quad (2.42)$$

They suggest that in order to produce approximate symmetry between the filter pairs,  $H_T(Z)$  should equal  $F_T(Z)$  at  $Z = -1, 0$  and  $+1$ .

One solution is for  $c=-2/7$  giving the following filter polynomials:

$$H_T(Z) = \frac{1}{50}(50 + 41Z - 15Z^2 - 6Z^3) \quad (2.43)$$

$$F_T(Z) = \frac{1}{7}(7 + 5Z - 2Z^2) \quad (2.44)$$

Alternatively, longer transforms can be used instead of the simple 2-tap function in equation (2.41). They suggest symmetrical functions with 4 and 6 taps of the form:

$$Z = M(Z) = qz^5 + pz^3 + \left(\frac{1}{2} - p - q\right)(z + z^{-1}) + pz^{-3} + q^{-5} \quad (2.45)$$

may be used to give a sharper transition from  $Z=+1$  to  $Z=-1$ .

They have found that the smoothness of the wavelet and scaling functions improves, as the transition from  $Z=+1$  to  $Z=-1$  in  $M(z)$  becomes sharper (i.e.  $M(z)$  becomes longer in terms of filter coefficients)

They say that the even order coefficients of  $M(z)$  must remain zero. However, as the length of  $M(z)$  increases, the filters  $H_0(z)$  and  $F_0(z)$  becomes progressively longer.

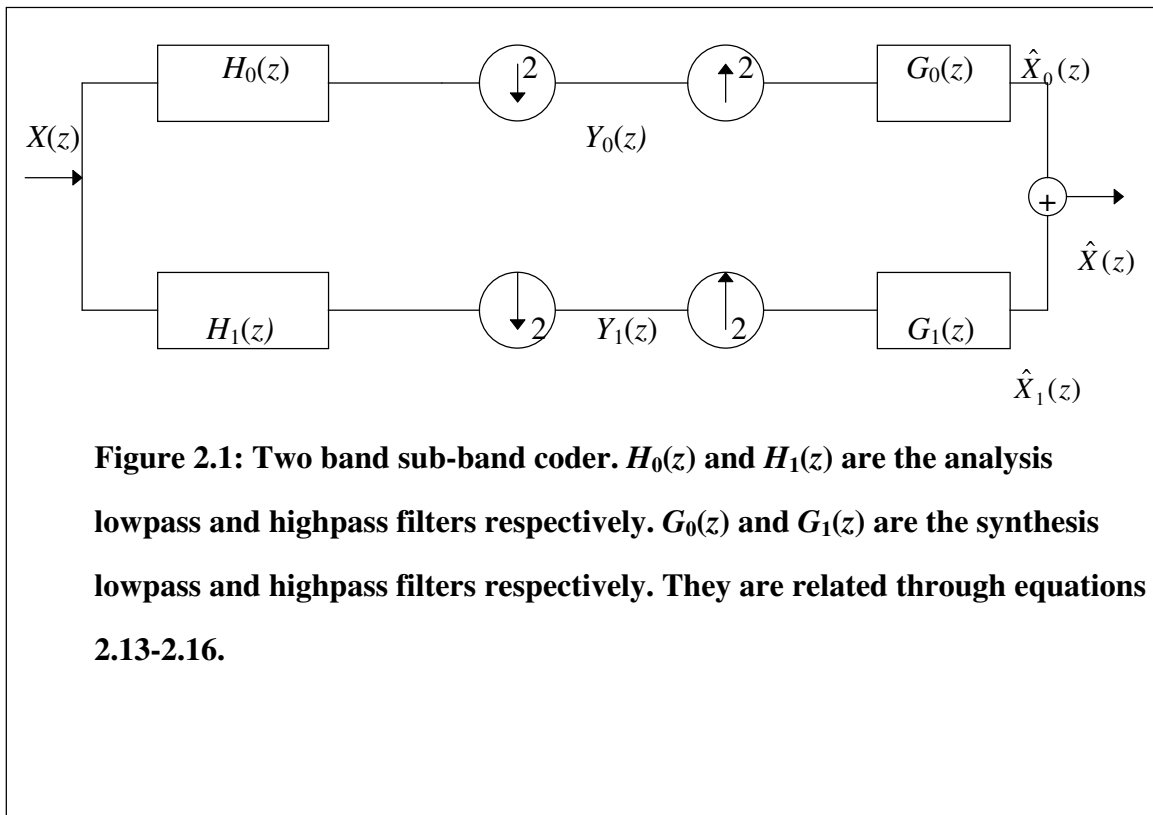
Alternative linear phase design methods are described in [19,75].

### 2.7.1. Disadvantage of longer wavelet filters

Although wavelet transforms avoid blocking artefacts, as the basis functions decay towards zero, they can potentially produce a different form of coding artefact known as *ringing*. Ringing is caused by coarse quantisation of coefficients corresponding to higher frequency basis functions. At sharp edges in an image, the impression of decaying repetitions of the edge may be visible propagating perpendicularly to the edge in the reconstructed signal.

However, for wavelet transforms which use longer basis functions, the ringing extends further, possibly into low activity regions and can be visually annoying when coding at low bit rates.

One problem suggested with the hypothesis that longer synthesis filters possibly increase the amount of visible ringing distortion is that this does not take into consideration the behaviour associated with the shape of the wavelet itself, such as how fast its amplitude decays toward its tails. Da Silva [75] proposed a measure called PPR (peak-to-peak ratio) which is the ratio of the maximum peak-to-peak value of the wavelet to the arithmetic mean of the second and third maximum peak-to-peak values. The larger the PPR is, the more “concentrated” the wavelet becomes, hence the spread of the error becomes smaller and less visible.



# Scalar/Vector Quantisation and Entropy Coding

---

In this chapter, we shall initially review scalar quantisation. We will then show that vector quantisation (VQ) can be more optimal in terms of coding gain. We will review the training algorithms used in VQ and then mention lattice VQ which has the advantage of reducing the computational complexity associated with standard VQ algorithms. Finally, we shall look at entropy coding when used in conjunction with scalar quantisation. It should be noted that entropy coding can equally be applied to VQ. Finally, we examine the coding gain achieved and the subsequent increase in complexity of higher-order entropy coding methods. We shall briefly mention arithmetic codes which can efficiently implement entropy coding and are particularly useful for implementing conditional entropy codes.

### 3.1. Quantisation techniques

For high levels of data compression, it is normally necessary to accept some degree of degradation of the signal, i.e. *lossy* coding must be used. This is achieved by *quantising* the amplitudes of the data samples. In the context of image coding, the data may correspond to the image pels themselves or the coefficients produced by a subband decomposition.

This technique is known as *scalar quantisation* and is reviewed in section 3.1.1. However, an alternative technique is *vector quantisation (VQ)*, reviewed in section 3.1.2. VQ can provide improved performance by exploiting correlations in the data that have not previously been removed by subband decomposition methods.

### 3.1.1. Scalar quantisation

Scalar quantisation methods apply a quantiser individually to each data sample and then entropy code the quantised amplitude levels. The data samples could be transform coefficients. The quantiser step size may be adapted with spatial location to take account of non-stationarities in the image.

Furthermore, for subband coefficients, the step size may be adapted according to the spatial frequency of the subband. In each case, adaptation of the step size can allow properties of the human visual system to be exploited.

The Lloyd-max Scalar Quantiser is an optimal, non-uniform quantiser (for fixed-length codewords) which can be determined using the LBG algorithm (for 1-D vectors). However, if the reconstruction values  $y_i$  are not equally probable, we can use entropy coding to reduce the rate below  $\log_2(L)$  to  $H(y)$  where  $L$  is the fixed number of levels. However, one can go further and restate the optimisation problem to minimize the distortion subject to a given entropy  $H(y)=R$ . The number of levels  $L$  can now be any desired value. The resulting quantiser is now a constrained-entropy quantiser. It has been shown that (with certain restrictions), at high bitrates, the uniform quantiser is the optimum constrained-entropy quantiser.

The obvious conclusion is that if one is willing to perform variable-rate entropy coding, then the uniform quantiser is always the scalar quantiser of choice. However, various researchers [51,53,56] have modified this when quantising transform coefficients. Typically, the quantising region or “dead-zone” near the origin is increased in width to efficiently entropy code the many low magnitude coefficients which arise from applying a wavelet/lapped or discrete cosine transform to image data. So for example, if  $\Delta$  is the uniform quantiser step size, then doubling the width of the “dead-zone” results in all coefficients that are in the interval  $-\Delta < c_{i,j} \leq \Delta$  being set to zero. These quantisers are known as dead-zone quantisers (they have two parameters, the zero bin width and the bin width of the non-zero bins [53]).

### 3.1.2. Vector quantisation

If significant correlations are present in the data, then vector quantisation (VQ) can provide improved performance compared with scalar quantisation (in fact, it can be shown that some improvement can be obtained even if no correlations exist [84]). We will review full-search VQ which requires an initial codebook design phase and also briefly, lattice vector quantisation

#### 3.1.2.1. Training-based vector quantisation

A vector quantiser (block quantiser,  $k$ -dimensional quantiser, block source code or matrix quantiser) consists of a reproduction alphabet or codebook  $C = \{y_i, i=1,2, \dots,N\}$  of  $N$  codewords together with a mapping  $q(\cdot)$  that assigns to each  $k$ -dimensional real-valued input vector  $\vec{x}$  a codeword  $y_i \in C$ . For image coding, the codewords are similar to the source vectors  $X$  ---that is, the codewords are also  $k$ -dimensional real-valued vectors.

A vector quantiser can also be seen as a combination of two functions: an *encoder*, which observes a particular input vector  $\vec{x}$  and generates the address  $i$  of the codeword  $y_i$  specified by  $q(x)$ , and a matching *decoder*, which uses this address to generate the reproduction vector  $y_i$ . When  $\vec{x}$  is quantised as  $y_i$ , a quantisation error results with respect to a distortion measure  $d(x, y_i)$ . The distortion measure represents the penalty or the cost associated with reproducing vector  $x$  by  $y$ . The best mapping  $Q(\cdot)$  is the one that minimises  $d(x,y_i)$ . Given a distortion measure, the performance of a vector quantiser is measured by the average distortion per sample. To design a codebook, the  $k$ -dimensional space of the vector  $x$  is partitioned into  $N$  disjoint cells. Associated with each cell  $S_i$ , the quantiser assigns a vector  $y_i$  as the codeword if  $x$  belongs to  $S_i$ .

We next summarize the LBG algorithm [20] which performs the codebook design.

### 3.1.2.1.1. LBG algorithm

We first define some notation. An  $N$ -level  $k$ -dimensional quantiser is a mapping  $q$ , that assigns to each input vector  $x = (x_0, \dots, x_{k-1})$ , a reproduction vector,  $\hat{x} = q(x)$  drawn from a finite reproduction alphabet,  $\hat{A} = \{y_i, i=1, \dots, N\}$ . The quantiser  $q$  is completely described by the reproduction alphabet (or codebook)  $\hat{A}$  together with the partition,  $S = \{S_i : i=1, \dots, N\}$  of the input vector space into the sets  $S_i = \{x: q(x) = y_i\}$  of input vectors mapping into the  $i^{\text{th}}$  reproduction vector.

We assume the distortion caused by reproducing an input vector  $x$  by a reproduction vector  $\hat{x}$  is given by a nonnegative distortion measure  $d(x, \hat{x})$ . Many such distortion measures have been proposed in the literature. The most common is the squared-error distortion:

$$d(x, \hat{x}) = \sum_{k=0}^{k-1} |x_k - \hat{x}_k|^2 \quad (3.1)$$

Another measure is the Holder norm:

$$d(x, \hat{x}) = \left\{ \sum_{i=0}^{k-1} |x_i - \hat{x}_i|^v \right\}^{1/v} \quad (3.2)$$

An  $N$ -level quantiser will be said to be optimal (or globally optimal) if it minimises the expected distortion, that is  $q^*$  is optimal if for all other quantisers  $q$  having  $N$  reproduction vectors  $D(q^*) \leq D(q)$  where:

$$D(q) = E\{d(X, q(X))\} \quad (3.3)$$

$X = (x_0, \dots, x_{k-1})$  is a real random vector. This performance measure (3.3) is useful if the quantiser  $q$  is to be used to quantise a sequence of  $n$  vectors that are stationary and ergodic. Therefore the averaged distortion:

$$n^{-1} \sum_{i=0}^{n-1} d(X_i, q(X_i)) \quad (3.4)$$

converges with probability one to  $D(q)$  as  $n \rightarrow \infty$ .

A quantiser is said to be locally optimum if  $D(q)$  is only a local minimum, that is, slight changes in  $q$  cause an increase in distortion.

A summary of the algorithm is:

- 1) Initialisation: Given  $N =$  number of levels, distortion threshold  $\varepsilon > 0$ , an initial  $N$ -level reproduction alphabet  $\hat{A}_0$ , and a training sequence  $\{x_j, j=0, \dots, n-1\}$ . Set  $m=0$  and  $D_{-1} = \infty$ .
- 2) Given  $\hat{A}_m = \{y_i : i=1, \dots, N\}$ , find the minimum distortion partition  $\rho(\hat{A}_m) = \{S_i : 1, \dots, N\}$  of the training sequence,  $x_j \in S_l$  if  $d(x_j, y_l) \leq d(x_j, y_i)$  for all  $l$ . Compute the average distortion  $D_m = D(\{\hat{A}_m, \rho(\hat{A}_m)\}) = n^{-1} \sum_{i=0}^{n-1} \min_{y \in \hat{A}_m} d(x_j, y)$  (3.5)
- 3) If  $(D_{m-1} - D_m)/D_m \leq \varepsilon$ , halt with  $\hat{A}_m$  as the final reproduction alphabet. Otherwise continue.
- 4) Find the optimal reproduction alphabet  $\hat{x}(\rho(\hat{A}_m)) = \{\hat{x}(S_i); i = 1, \dots, N\}$  for  $\rho(\hat{A}_m)$ . Set  $\hat{A}_{m+1} = \hat{x}(\rho(\hat{A}_m))$

Replace  $m$  by  $m+1$  and go to (1)

Once the final codebook  $\hat{A}_m$  is obtained, it is used on new data outside the training sequence with the optimum neighbour rule, that is an optimum partition of  $k$ -dimensional Euclidean Space.

For the squared-error criterion,  $\hat{x}(S_i)$  is the centroid given by:

$$x(S_i) = \frac{1}{\|S_i\|} \sum_{j: x_j \in S_i} x_j \quad (3.6)$$

where  $\|S_i\|$  denotes the number of training vectors in the cell  $S_i$ . If  $\|S_i\|=0$ , set  $\hat{x}(S_i) = y_i$  the old codevector (i.e. the  $i^{\text{th}}$  centroid from the previous iteration). Alternatively, one can simply remove the cell  $S_i$  and the corresponding reproduction symbol from the quantiser without affecting performance, and then continue with an  $(N-1)$  level quantiser. One could also reassign the reproduction vector corresponding to  $S_i$  to another cell  $S_j$  and continue the algorithm.

Now, consider the quantisation of blocking a discrete-time stationary ergodic source into contiguous vectors of length  $n$ , and representing each vector by one of  $m$

reproduction vectors. For transmission of the source across a binary channel, the index of each reproduction vector can be encoded into binary in a straightforward manner requiring  $(\log_2 m)/n$  bits per sample. However, for a fixed block length of practical size, a scheme that achieves the same distortion with lower average transmission rate than straight-forward binary coding of the indexes is one that “entropy encodes” its index sequence. Entropy encoding can reduce the average transmission rate from  $(\log_2 m)/n$  bits per sample to the (index entropy)/ $n$  bits per sample (where the index entropy is the average number of bits/vector required to transmit the index of a codevector).

Gray, Chou and Lookabaugh [21] developed an Entropy Constrained LBG algorithm where the standard LBG algorithm was a special case. They essentially tried to minimize the following function:

$$J_\lambda = D(q) + \lambda R(q) \quad (3.7)$$

where  $D(q) = E(\rho_n(X^n, \beta(q(X^n))))$  is the average distortion as computed using the standard LBG algorithm and  $R(\cdot)$  is the average codeword length. Here  $q$  is the quantiser,  $\rho_n$  is the distortion measure,  $\beta(\cdot)$  is the centroid of a partition, where  $X^n$  is the ensemble of source vectors and  $\lambda$  is a Lagrangian parameter. When  $\lambda=0$ , the algorithm reduces to the LBG algorithm. A summary of the algorithm follows:

- 1) Initialisation: Given  $N$ -number of levels, distortion threshold  $\varepsilon > 0$ , an initial  $N$ -level reproduction alphabet  $\hat{A}_0$ , a training sequence  $\{x_j, j=0, \dots, n-1\}$ , a Lagrange multiplier  $\lambda$ , a set of initial codeword lengths and associated probabilities,  $\{\gamma^{(0)}(j) : j=0, \dots, N-1\}$  and  $\{P^{(0)}(j), j=0, \dots, N-1\}$ . Let  $m=0$  and  $D_{-1} = \infty$
- 2) Given  $\hat{A}_m = \{y_i : i=1, \dots, N\}$ , find the minimum distortion partition  $\rho(\hat{A}_m) = \{S_i ; i = 1, \dots, N\}$  of the training sequence :  $x_j \in S_i$  if  $d(x_j, y_i) + \lambda |\gamma^{(m)}(i)| \leq d(x_j, y_l) + \lambda |\gamma^{(m)}(l)| \quad \forall l$  (3.8)

- 3) Compute the average distortion :

$$D_m = n^{-1} \sum_{j=0}^{n-1} \min_{y \in \hat{A}_m} d(x_j, y) + \lambda R \quad (3.9)$$

where  $R = \sum_{i=0}^{N-1} P^{(m)}(i) |\gamma^{(m)}(i)|$  and  $P(i)$  is the probability (frequency of occurrence) of

a code vector and index  $i$  occurring and  $\gamma(i) = \log_2(1/P(i))$  is the self-information in bits of a codevector with index  $i$ .

- 4) If  $(D_{m-1} - D_m)/D_m \leq \varepsilon$ , halt with  $\hat{A}_m$  as the final reproduction alphabet. Otherwise continue
- 5) Calculate  $P^{(m+1)}(i)$  and  $\gamma^{(m+1)}(i) = \log_2(1/P^{(m+1)}(i))$  for  $i \in 0, \dots, N-1$
- 6) Find the optimal re-production alphabet,  $\hat{x}(\rho(\hat{A}_m)) = \{\hat{x}(S_i); i = 1, \dots, N\}$  for  $\rho(\hat{A}_m)$ . Set  $\hat{A}_{m+1} = \hat{x}(\rho(\hat{A}_m))$ . Set  $m=m+1$  and return to 2).

The entropy rate of the resulting codebook decreases as  $\lambda$  increases from  $0 \rightarrow \infty$ . For very large  $\lambda$ , we approach a rate-zero codebook where only one codevector from the codebook is utilized. Note that the initial codeword lengths and associated probabilities can be obtained by partitioning the initial vector space using only the distortion criterion  $d(\cdot)$  and the initial reproduction alphabet.

There are several ways to choose the initial reproduction alphabet  $\hat{A}_0$  required by the algorithm. One method is that of the  $k$ -means method, namely choosing the first  $N$  vectors (where  $N$  is the codebook size) in the training sequence.

A recently suggested alternative to the LBG algorithm or as an initialisation for the LBG was proposed by Equitz [22]. He proposes an efficient algorithm known as the Pairwise Nearest Neighbour (PNN) algorithm.

A commonly used alternative initialization for the LBG algorithm is the ‘‘splitting’’ process [20].

### **3.1.2.1.2. Complexity issues**

We will briefly address the complexity of full-search VQ and alternatives such as Tree-based VQ. We will also mention two alternative VQ Systems that enable a reduction in the codebook size, namely Gain/Shape VQ and Multi-Stage VQ. We will also consider a more complex LBG VQ system, namely using a simulated annealing type approach which can obtain a global minimum on the distortion surface rather than the local minimum which the standard LBG algorithm usually finds. Finally, we shall mention a type of VQ method called trellis-coded quantisation which does not require explicit codebook design.

#### **3.1.2.1.2.1. Searching algorithms**

Tree-searched vector quantisers were first proposed by Buzo et al. [23] and are a natural byproduct of the “splitting” algorithm for generating initial code guesses. We focus on the case of a binary tree for simplicity, but more general trees will provide better performance whilst retaining a significant reduction in complexity.

Say that we have a good rate 1 code (2 codewords) and we form a new rate two code by splitting the two codewords. Instead of running a full search VQ design on the resulting 4-word codebook, however, we divide the training sequence into two pieces, collecting together all those vectors encoded into a common word in the 1-bit codebook. That is, all of the training sequence vectors in a common cell of the Voronoi partition [34]. For each of these subsequences of training vectors, we then find a good 1-bit code using the splitting+LBG algorithms. The final codebook (so far) consists of the four codewords in the two 1-bit codebooks designed for the two subsequences. A tree-searched encoder selects one of the words not by an ordinary full search of this codebook, but instead it uses the first one bit codebook designed on the whole sequence to select a second code and it then picks the best word in the second code. This encoder can then be

used to further subdivide the training sequence and construct even better codebooks for the subsequence.

Observe that at the end, there are  $2^R$  possible reproduction vectors as in a full search VQ, but now  $R$  binary searches are made instead of a single  $2^R$ -ary search. However, the codebook is no longer optimal in the minimum average distortion sense over the training data. Thus, one may trade performance for efficiency of implementation.

Ndifer [25] and more recently Feng [24] have proposed algorithms which reduce the computational complexity of standard full-search VQ systems *without* compromising the optimality of the LBG-designed codebook (or reproduction alphabet). Feng's algorithm initially sorts the codewords in the codebook according to the increasing value of the sum of the elements in the vector. He states that the minimum distortion codeword should have a similar mean value to the input vector, otherwise there would be large reconstruction errors. Therefore, he proposes that for any input vector  $X$ , one first finds the codeword having the closest mean value to  $X$ . However, this algorithm requires extra memory storage in both the encoder and decoder.

### **3.1.2.1.2.2. Alternative VQ systems**

An alternative way to reduce both memory and encoding complexity in image coding is to normalise the energy (“gain”) of a vector prior to encoding and to separately encode the gain with a scalar quantiser. The energy-normalised vector is called the shape. This technique was studied by Sabin and Gray [26], who found an optimal way to jointly design the gain and shape quantisers. The effectiveness of Gain-Shape VQ depends upon the degree to which the gain of a randomly selected vector is statistically independent of its shape. With this technique, higher dimensional vectors can be used with tolerable complexity but the optimality is compromised.

Multi-stage VQ [27] is a technique that reduces memory as well as encoding complexity. Here, the encoding error of a VQ is formed by taking the difference between

the original and quantised vectors and then feeding it into a second-stage VQ. Note that the actual binary word that must be transmitted consists of the concatenation of the binary words needed to identify the correct codevector in each codebook. Note also that this technique is equivalent to a single stage VQ whose codevectors are specially structured.

In summary, tree-structured and multi-stage codebooks make it possible to implement higher dimensional vector quantisers at relatively low complexity. However, because of the constraints imposed on the codebook structures they achieve only very limited improvements in performance for a given level of complexity.

A computationally expensive method of trying to obtain a global minimum for the LBG algorithm is to introduce noise into the system. This approach is known as Simulated Annealing [28].

We shall finally mention a type of VQ method called trellis coded quantisation (TCQ) [61,62,63]. The basic idea of TCQ is to allow an expanded signal set to use coded modulation for set partitioning. For encoding a memoryless source using TCQ at  $R$  bits per sample, a scalar codebook with  $2^{R+1}$  codewords is designed. This codebook is then partitioned into four disjoint subsets, each with  $2^{R-1}$  codewords. These subsets are used to label the branches of a trellis. Given a data sequence, the Viterbi algorithm [63] is used to choose the trellis path (sequence of codewords) that minimises the mean-squared error between the input sequence and output codewords. Each  $R$ -bit codeword consists of one bit specifying the chosen subset (trellis path) and an  $(R-1)$  bit codeword necessary to specify codewords from the chosen subsets. TCQ can be thought of as being a type of vector quantisation because of the expanded signal set it uses. TCQ is known to be particularly advantageous over scalar quantisation at higher bit-rates (the converse is true at low rates) [61].

### 3.1.2.1.3. Advanced VQ systems

Incorporating memory into an ordinary Vector Quantiser (VQ), we could exploit the interblock correlation to improve the performance of an ordinary VQ [29]. This high interblock correlation that exists in VQ is due to the small block size.

Recently, Nasrabadi and Feng [29,30] proposed a finite memory VQ encoder called the Dynamic Finite State Vector Quantizer (DFSVQ). In DFSVQ, for each input vector to be encoded, a sub-codebook is created on-line by direct manipulation of a fixed super-codebook. The Sub-codebook is built by a next-state function (re-ordering procedure) that uses the local statistics of the previously encoded blocks to dynamically select a number of codevectors from the super-codebook. If the reordering procedure can select the best match, in Euclidean sense, from the super-codebook, then only the sub-codebook has to be searched. Therefore, the computational complexity and the bit rate is reduced.

The major component of a DFSVQ system is the reordering procedure. Nasrabadi and Feng [29] suggest four main techniques which include the conditional histogram, address prediction, input vector prediction and nearest neighbour designs. The conditional histogram approach depends upon calculating a score function (which is a conditional probability) which is calculated for each codevector in the super-codebook given the VQ indices of the neighbouring blocks. The *score function* is computed as a product of four conditional probabilities and is given by:

$$s(I_n) = \prod_{i=A}^D P(I_n/I_i) \quad (3.10)$$

where  $I_n$  represents the index of each codevector,  $I_i$  for  $i=A, B, C, D$  are the corresponding indices representing the previously VQ encoded blocks  $\hat{A}, \hat{B}, \hat{C}$  and  $\hat{D}$ . Each conditional probability  $P(I_n/I_i)$  represents the directional probability or frequency of occurrence of getting a codevector (represented by an index  $I_n$ ) given one of the neighbouring encoded blocks at location A, B, C or D; each block is represented by its corresponding index  $I_i$  of the codevector in the super-codebook. These probabilities are obtained from four

directional block-transition probability matrices where an entry in each matrix represents the frequency of occurrence of two neighbouring codevectors (indices). Four directional block-transition matrices are needed to represent the frequency of occurrence of two blocks occurring in vertical, horizontal,  $45^{\circ}$  diagonal, or  $135^{\circ}$  diagonal direction of each other. Each directional block-transition probability matrix is obtained by the frequency of occurrence of two addresses in a particular direction.

All the codevectors in the super-codebook are reordered with respect to their corresponding score value obtained from (3.10). Therefore, the most probable reproduction codevectors approximating the input vector  $X$  would be moved to the top of the super-codebook.

An alternative technique to obtain a sub-codebook was proposed by Nasrabadi [29], known as Address Prediction. This technique relies on the fact that VQ addresses would be somehow correlated if the codevectors in the super-codebook are rearranged in such a manner that the neighbouring codevectors in the super-codebook could be very similar in a Euclidean sense. This could be achieved by re-arranging the codevectors with respect to their vector variances. Nasrabadi used a neural network to predict the address of the codevector for an input vector  $X$  based on the addresses of the codevectors for neighbouring input blocks. Then he defines the sub-codebook as all the codevectors within an offset parameter from the predicted address.

The simplest but most computationally complex technique is to find the best  $M$  matching codevectors in the super-codebook to the neighbouring blocks. However, this involves several searches of the super-codebook for every input vector encoded.

An alternative coding scheme for VQ was developed by Fioravanti et al. [31]. Their novel approach is known as Dynamic Codebook Reordering VQ (DCRVQ). Here, residual correlations between neighbouring codevectors are exploited by a non-linear predictor, using a neural network. Here the predictor predicts the current codevector  $X_{ij}$  on the basis of the four neighbouring previously decoded codevectors.

### 3.1.2.2 Introduction to lattice VQ

Lattice vector quantisation has attracted some interest in recent years as an alternative to full-search VQ for image coding.

Although the LBG approach is quite general, it suffers from two basic drawbacks. First, the training sequence approach requires considerable computer time for the design of each vector quantiser. Second and more importantly, the design that is produced has no general structure, and the implementation of the VQ encoder requires considerable computation. In the worst case, the implementation requires a distance calculation between the input vector and every output vector so that the best representation vector can be determined. Such a brute force search for the nearest representation point becomes infeasible for modest rates and dimensions.

The principal alternative to quantisers based on the LBG algorithm is the lattice quantiser [32,33,37,38,60,65,66,67,68,69,70,77]. By using a regular set of points in space, lattice quantisers offer the potential of rapid encoding and decoding algorithms [34], [35].

Furthermore, it has been noted [36] that the optimal high-resolution *entropy constrained vector quantiser* will approximate a lattice (true specifically when the quantisation used is arbitrarily fine).

Woolf and Rogers [32] suggest that for lattice VQ to be competitive, the quantised vector needs to be entropy coded. This has traditionally been performed on a *per image* basis [33] ---a complex and inefficient process.

#### 3.1.2.2.1. Lattice VQ

The input vectors generally have some maximum magnitude. The lattice points falling inside the hypersphere of that radius can be considered as comprising a large VQ codebook. The size of that codebook can be of the order of  $10^{20}$ . If a code of length

$\log_2(N)$  (where  $N$  is the codebook size) were assigned to each lattice point, the resultant coding gain would be small relative to uniform scalar quantisation.

One solution which is used, is to limit the codebook size by truncating the lattice [38]. The disadvantage of this approach is that vectors falling outside the truncated lattice have to be rescaled so that they quantise to a valid lattice point. The scaling factor then also needs to be transmitted to the decoder for optimal decoding [60].

One can also entropy code the quantised vectors. However, the number of lattice points in the codebook is very many orders of magnitude greater than the number of vectors which typically arise from a single image ---so it is impossible to accumulate reliable statistics in order to design an entropy code.

A more detailed discussion on lattice VQ (LVQ) can be found at the start of Chapter 5.

### 3.1.2.2.1.1 Particular lattices

An  $n$ -dimensional *lattice*,  $\Lambda$ , is an infinite set of integral combination of linearly independent vectors  $\{a_i\}$ . Its dual,  $\Lambda^*$ , is also defined.

$$\Lambda = \left\{ x: x = \sum_{i=1}^n u_i a_i, u \in Z \right\}$$

$$\Lambda^* = \left\{ x \in R^n: x \cdot u \in Z \quad \forall u \in \Lambda \right\} \quad (3.11)$$

Conway and Sloane [34] have developed fast quantisation algorithms for a number of important lattices, the  $\Lambda_{16}$ ,  $D_n$ ,  $E_6$ ,  $E_7$  and  $E_8$  lattices. The fast algorithms can be used to find the closest lattice point to an arbitrary point. They give a tabulation of the main lattices. The  $D_n$  lattice, for  $n > 1$ , consists of the points  $(x_1, x_2, \dots, x_n)$  having integer coordinates with an even sum. The  $E_8$  lattice is the union of the  $D_8$  lattice with the coset  $D_8 + \left( \frac{1}{2}, \frac{1}{2}, \dots, \frac{1}{2} \right)$ . The  $\Lambda_n$  lattice with  $n=16$  is known as the Barnes-Walls lattice and can be defined as the union of the lattice  $2D_{16}$  with 32 cosets, each being binary code-words from the first order *reed-muller* code of length 16 [60].

It is known that for independent uniformly distributed data quantised with a uniform scalar quantiser (with step size of unity), the mean-squared error per dimension is 1/12 (0.0833). This can be reduced [34] by quantising with a multi-dimensional lattice. It has been found [34] (for lattices with dimension less than 17) that the dual of the densest lattice for the sphere packing problem is the optimal lattice for quantisation. For example in 4 dimensions the densest sphere packing lattice is the  $D_4$  lattice (the MSE for both the dual  $D_4^*$  and  $D_4$  is approximately the same, 0.076603). In 5 dimensions, it is the  $D_5$  lattice (the MSE for  $D_5^* = 0.07563$  and for  $D_5 = 0.07579$ ). In 8 dimensions, it is the  $E_8$  lattice (MSE for both  $E_8^*$  and  $E_8$  is 0.071682). And finally in 16 dimensions, it is the  $\Lambda_{16}$  lattice (MSE for both  $\Lambda_{16}^*$  and  $\Lambda_{16}$  is 0.0683).

### 3.2. Review of entropy coding

The image entropy coding described here assumes scalar quantisation of coefficients obtained using transform coding. The arguments are equally valid when using VQ instead of SQ (except, that one is coding the address of the lattice point or of the reproduction code-vector in the LBG designed code-book). For a data source whose output symbols are independent, the optimal compression can be achieved by allocating code bits to the data symbols according to their self-information content. From Shannon, the self-information content of a symbol  $x$  with probability  $P(X=x)$  is given by:

$$I = -\log_2(P(X = x)) \text{ bits} \quad (3.12)$$

which gives the optimal allocation of code bits to symbol  $x$ . If the data symbols  $X$  can take  $N$  output values  $x_k$ ,  $k=0, \dots, N-1$ , then the first-order Source Entropy is:

$$H_{1st} = -\sum_{k=0}^{N-1} P(x_k) \log_2 P(x_k) \quad (3.13)$$

For the simple binary case ( $N=2$ ), the source entropy becomes:

$$H_{1st} = -\left[ P(x_0) \log_2(P(x_0)) + P(x_1) \log_2(P(x_1)) \right] \quad (3.14)$$

First-order entropy coding only gives optimal coding when all the data symbols are independent. In the following sections, we shall assume significant correlations between data symbols.

### 3.2.1. Higher-order entropy coding

When a source is not independent, high-order statistics of correlated samples can be used to improve the coding efficiency. The high-order entropy of a source can be effectively exploited by using either the joint probability of  $L$  symbols or the conditional probability. Gallacher [39] showed that the conditional probability approach, which is reviewed here, potentially has a lower rate than the joint probability approach. In this notation, the optimal word length to code a sample  $x$  is the higher-order conditional self-information given by  $\log_2 P(x|S^L(x))$  where  $P(x|S^L(x))$  is an  $L^{\text{th}}$  order conditional probability of  $x$  given state  $S^L(x)$  (which is the context information) which is formed by taking  $L$  samples (previously coded). Theoretically, the coding gain can be reduced from  $H_{1st}(X)$  to :

$$H_{ctx} = - \sum_{j=0}^{M-1} \left[ P(C = c_j) \sum_{k=0}^{N-1} P(X = x_k | C = c_j) \log_2 \left( P(X = x_k | C = c_j) \right) \right] \quad (3.15)$$

where  $C$  is the previous context information and  $X$  is the current input symbol. Here the output symbol and context data can take  $N$  and  $M$  values respectively. We are also assuming that the context states  $c_j$  are mutually exclusive.

We note that the potential rate reduction is equal to the average mutual information  $I(x; S^L(x))$ . The higher the dependence between samples, the more coding gain we can get. Also, as  $L$  increases the coding gain should also increase, in theory. However, the use of high-order statistics may become intractable due to the larger number of context states..

The benefit of using high-order statistics has been clearly demonstrated in Langdan and Rissanen's work on black-white document compression [46]. In [46], the conditioning state was defined as the combination of 10 pixels in the causal region and

1024 states were generated from all possible combinations of these 10 pixels. Also Young [41] has shown that significant reductions in bit-rate can be achieved when using 3 previously coded coefficients in coding the current coefficient value. He uses the hierarchical modified lapped transform in conjunction with SQ/conditional entropy coding and achieves up to 20% bit-rate reductions for coding video sequences.

### **3.2.1.1 Complexity of higher-order entropy systems**

In order to obtain significant reductions in bit-rate, it is desirable to use several previously coded coefficients as conditional/context information. We can illustrate the system complexity if say 3 context coefficients are chosen. This gives a total of  $2^3=8$  possible context states, (if the data is binary). If the contexts are quantised to 4 amplitude levels, the number of states increases to  $4^3=64$ . One can see that the number of states increases dramatically if either the data is more finely quantised or the number of context coefficients is increased.

#### **3.2.1.1.1. Simplifications which are used**

In conjunction with his HMFLT scheme, Young [41] suggests that all coefficients in a particular subband must have contexts in ‘form’ (for example, the contexts must all be derived from the same three subbands), although he suggests that the specific contexts may be different for each coefficient.

He also suggests that as the transform coefficients to be encoded can take a wide range of possible quantised amplitude values, the conditional entropy coding scheme may be used to transmit initial amplitude information only, with a separate first order entropy coder being used to transmit amplitude data that falls outside the initial range [42]. In the simplest case, only zero and non-zero coefficients may be distinguished by the context based code.

Finally, Tzou [43] suggests that all previous work on conditional entropy coding assumes that different conditioning states use different code tables. For high-order coding schemes, the number of states is larger and so is the number of code tables. He suggests that the main idea of code table reduction is to use one common code table for each group of similar code tables. He merges two code tables together either if they are similar or if one or both of the states occur rarely, as long as the entropy increase is tolerably small. They use as a distance measure between code tables, the rate increase caused by the merging of the two tables. Then the merging can be optimally performed using the LBG algorithm with a suitable initial guess. They provide a formula for obtaining the centroid of each partition.

### **3.2.1.2. Conditioning tree approach**

In Young's approach [41], the 3-pixel states can be considered as the leaves of a 3-bit full binary tree, if pixels are quantised to 2 levels. Due to characteristics of the image, some states occur rarely and the conditioning tree can be pruned (i.e. code table merging) with a slight decrease in efficiency. However, the pruning approach may not be practical since a full tree is often too complicated to obtain in the first place. An incremental tree extension method was introduced in [42] to extend the conditioning tree from the root by growing branches from the most profitable node iteratively.

### **3.2.1.3. Choosing conditional pixel sequence**

It is not straightforward to choose the best  $n$  context coefficients for each coefficient to be encoded based on an analysis of the entropy reductions achieved with single contexts.

If  $X$  is the current pixel and  $Y_i$  is one of the neighbouring pixels, then one may select the pixels according to which has the largest  $I(X;Y_i)$ , then the next largest and so on. However, a larger first-order average mutual information does not directly imply a

larger high-order mutual information. For example,  $I(X;Y_1) > I(X;Y_2) > I(X;Y_3)$  does not imply  $I(X;Y_1Y_2) > I(X;Y_1Y_3)$ . The optimal measure is the high-order mutual information  $I(X;S^L(X))$  where  $S^L(X)$  consists of  $L$  neighbouring pixels of  $X$ . The computational complexity of  $I(X;S^L(X))$  grows exponentially with the conditioning order. Furthermore,  $I(X;S^L(X))$  has to be computed for different  $S^L(X)$  which contains different pixels. This computational complexity makes it virtually impossible for high-order cases.

Young [41] suggests selecting 4 or 5 candidate contexts based on the first-order average mutual information criterion and then performing an exhaustive evaluation of the entropy reductions achieved with each possible group. Tzou [43] gives an alternative method.

#### **3.2.1.4. Implementation of conditional entropy coding schemes**

Conditional entropy coding schemes may be implemented using standard Huffman coding methods. This would require separate code tables to be designed for each subband with distinct statistics and for each possible context state. Such a scheme would be complex. Consequently, conditional entropy coding schemes are normally implemented using an arithmetic coder [41], [40,47] which allows data to be coded at a rate close to the entropy bound, irrespective of the probabilities of the data symbols. Arithmetic coding allows a data string to be coded close to the entropy of that string with respect to the *probability model* used for coding. Thus, there is a separation between the *coder* and the *probability model* which the coder uses.

### **3.3. Brief description of arithmetic coding**

Arithmetic coding is a data compression technique that encodes data (the data string) by creating a code string which represents a fractional value on the number line between 0 and 1. The coding algorithm is symbolwise recursive, i.e. it operates upon and encodes one data symbol per iteration. On each recursion, the algorithm successively

partitions an interval of the number line between 0 and 1, and retains one of the partitions as the new interval. Thus, the algorithm successively deals with smaller intervals and the code string, viewed as a magnitude, lies in each of the nested intervals. The data string is recovered by using magnitude comparisons on the code string to recreate how the encoder must have successively partitioned and retained each nested subinterval.

Arithmetic coding differs considerably from Huffman coding. The following coding algorithm can be used [41]:

Define the following notation:

*LPS* =Low probability Symbol,  $P_{LPS} \leq 0.5$

*HPS* =High probability Symbol

*I* = Interval register ( $0 \leq I \leq 1$ )

*C* =Codeword register ( $0 \leq C \leq 1$ )

*C\** =Transmitter codeword ( $C \leq C^* < C+I$ )

### Encoding Algorithm

Initialize:  $C:=0, I:=1$

Repeat for all data symbols {

  If symbol =*HPS* {

    code *HPS*

$C := C + I * (1 - P_{HPS})$

$I := I * P_{HPS}$

  } Else {

    code *LPS*

$C := C$

$I := I * P_{LPS}$

  }

}

Transmit:  $C^*$  ( $C \leq C^* < C+I$ )

### Decoding Algorithm

Initialize:  $C:=C^*, I:=1$

Repeat for all data sym. {

  If  $C \geq I * P_{LPS}$  {

*HPS* decoded

$C := C - I * (1 - P_{HPS})$

$I := I * P_{HPS}$

  }

  else {

*LPS* decoded

$C := C$

$I := I * P_{LPS}$

  }

}

At each stage, the interval register  $I$  represents the current interval value, while the codeword register points to the bottom of the interval. The generalisation to alphabets with  $M$  possible data symbols is quite straightforward; for each new data symbol, the current interval is split into  $M$  sub-intervals, one of which will correspond to the new interval for the data string.

In a practical arithmetic coder, a floating point arithmetic must be used since only finite precision registers are available [40,44,45]. Typically, the codeword and interval registers are both of length  $w$  bits, where  $w = 8$  or  $16$ . The value of  $w$  determines the accuracy with which probabilities can be stored. At each stage, the interval register  $I^*$  is shifted left such that  $0.75 < I^* \leq 1.5$  [45]. The code register is shifted left by the same number of places, and overflow bits are temporarily buffered (in case of subsequent carry over [45]) before being transmitted. Rubin [85] and Cleary et al. [80] give complete algorithms for the efficient implementation of practical arithmetic coders.

Normally, multiplication in a hardware implementation of the code is relatively expensive. In the special case with a binary alphabet, the multiplication was replaced in Langdon and Rissanen [40] by the much simpler shift operation, which was made possible by an approximation of the smaller symbol probability with an integral power of  $1/2$  (known as the Skew code). However, this does not generalise to non-binary alphabets for the reason that the symbol probabilities cannot be approximated well enough as powers of  $1/2$ .

Mohuddin [45] suggests a multiplication-free arithmetic code which is valid for both binary and multiple symbol alphabets.

## Chapter 4

---

### **A successive approximation coder based on vector quantisation**

---

In this chapter we will develop a successive approximation (or embedded) coder which progressively transmits quantised vectors. Embedded coding whereby a single compressed image file for an image at a given code rate can be truncated at various points and decoded to give a series of reconstructed images at lower rates was first developed by Shapiro [81]. This work was extended by Said and Pearlman [50]. Da Silva and Ghanbari [76] developed a successive approximation coder which extended Shapiro's work from scalar quantisation to vector quantisation. Our work is intended to extend the work of Da Silva and Ghanbari to amongst other things incorporate the ideas of Said and Pearlman. We develop two coding schemes, one which requires explicit entropy coding and the other which requires no entropy coding and in both cases we compare our results with the methods of Said and Pearlman [50] and Da Silva and Ghanbari [76]. Finally, we suggest some ideas for possible future research.

#### **4.1. Review of previous work on scalar and vector based successive approximation algorithms**

We will very briefly review the algorithms of Shapiro [81] and Said and Pearlman [50] for embedded coding of scalar quantised coefficients. We shall then review the vector quantisation method of Da Silva and Ghanbari [76].

### 4.1.1 Embedded coding algorithms based on scalar quantisation

Embedded zero-tree wavelet (EZW) coding is an effective and computationally simple technique for image compression.

One can assume that the original image is defined by a set of pixels values  $p_{i,j}$  where  $(i, j)$  is the pixel coordinate. To simplify the notation one can represent two-dimensional arrays with italic letters. The coding is actually done to the array  $c = \Omega(p)$  where  $\Omega(\cdot)$  represents a unitary hierarchical subband transformation (e.g [86]). Each element  $c_{i,j}$  is called a *transform coefficient* at coordinate  $(i, j)$ .

In a progressive transmission scheme, the decoder initially sets the reconstruction vector  $\hat{c}$  to zero and updates its components according to the coded message. After receiving the value (approximate or exact) of some coefficients, the decoder can obtain a reconstructed image  $\hat{p} = \Omega^{-1}(\hat{c})$ . A major objective in a progressive transmission scheme is to select the most important information ---which yields the largest distortion reduction ---to be transmitted first. For this selection one can use the mean squared error (MSE) distortion measure,

$$D_{mse}(p - \hat{p}) = \frac{\|p - \hat{p}\|^2}{N} = \frac{1}{N} \sum_i \sum_j (p_{i,j} - \hat{p}_{i,j})^2, \quad (4.1)$$

where  $N$  is the number of image pixels. Furthermore, one can use the fact that the Euclidean norm is invariant to the unitary transformation  $\Omega$ , i.e.,

$$D_{mse}(p - \hat{p}) = D_{mse}(c - \hat{c}) = \frac{1}{N} \sum_i \sum_j (c_{i,j} - \hat{c}_{i,j})^2 \quad (4.2)$$

It is clear from the above equation that if the exact value of the transform coefficient  $c_{i,j}$  is sent to the decoder, then the MSE decreases by  $|c_{i,j}|^2/N$ . This means that the coefficients with larger magnitude should be transmitted first because they have a larger content of information (in this context, information is used to indicate how much the distortion can decrease after receiving that part of the coded message). The information in the value of  $|c_{i,j}|$  can also be ranked according to its binary representation

and the most significant bits should be transmitted first. That idea is used in the bit-plane method for progressive transmission [87].

The coefficients are ordered according to the minimum number of bits required for its magnitude binary representation, that is, ordered according to a one-to-one mapping  $\eta: I \mapsto I^2$  such that

$$\left\lfloor \log_2 |c_{\eta(k)}| \right\rfloor \geq \left\lfloor \log_2 |c_{\eta(k+1)}| \right\rfloor, k=1, \dots, N \quad (4.3)$$

The coefficients are ordered by magnitude and the most significant bits are transmitted first.

There are two stages involved with EZW coding at each pass, known as the *sorting phase* and the *refinement phase*. During the sorting phase, the ordering information about which coefficients satisfy  $2^n \leq |c_{i,j}| < 2^{n+1}$  for a particular  $n$  is transmitted. This also includes the sign of the coefficients so that the decoder can change the value of that coefficient from zero to  $\pm 1.5 \times 2^n$  which is at the centre of the uncertainty interval (this is equivalent to transmitting the leading zeros to the left of the MSBD (most significant binary digit) in the binary representation of  $c_{i,j}$ ). Given  $n$ , if  $|c_{i,j}| \geq 2^n$  then one says that a coefficient is said to be *significant*, otherwise it is called *insignificant*. Here  $c_{i,j}$  is the value of the transform coefficient at location  $(i, j)$ . During the refinement phase, the bits to the right of the MSBD are transmitted (the position of the MSBD can be inferred from value of  $n$ ). The value of  $n$  is decremented by one and the coding process begins again with the next sorting phase.

It has been shown [81] that typically over 70% of the total bit-rate is needed to transmit the ordering information (or the significance map as it is known [81]) which is a series of binary decisions on whether a coefficient has a zero or a non-zero quantised value.

Shapiro developed a zero-tree based approach to this problem and Said and Pearlman have developed a set partitioning based algorithm which has been found to

improve considerably on Shapiro's method. Both these methods can be described as depth first algorithms [73]. A recent breadth first algorithm where a significant coefficient is visited just once has been suggested by Andrew [73] (however, this method loses the embedded quality and progressive transmission nature of Shapiro and Said and Pearlman's algorithms).

#### **4.1.1.1. Shapiro's zero-tree algorithm for transmitting the ordering information**

Shapiro encodes the significance map using *zerotree* coding. A wavelet coefficient  $x$  is said to be *insignificant* with respect to a given threshold  $T$  if  $|x| < T$ . The zerotree is based on the hypothesis that if a wavelet coefficient at a coarse scale is insignificant with respect to a given threshold  $T$ , then *all* wavelet coefficients of the same orientation in the same spatial location at finer scales are likely to be insignificant with respect to  $T$ . Empirical evidence suggests that this hypothesis is often true.

More specifically, in a hierarchical subband system, with the exception of the highest frequency subbands, every coefficient at a given scale can be related to a set of coefficients at the next finer scale of similar orientation. The coefficient at the coarse scale is called the "parent" and all the coefficients corresponding to the same spatial location at the next finer scale of similar orientation corresponding to the same location are called "offspring" or "children". Similarly, for a given parent, the set of coefficients at all coarser scales of similar orientation corresponding to the same location are called "descendants".

A scanning of the coefficients is performed in such a way that no child node is scanned before its parent. Given a threshold level  $T$  to determine whether or not a coefficient is significant, a coefficient  $x$  is said to be an element of a *zerotree* for threshold  $T$  if itself and *all* of its descendants are insignificant with respect to  $T$ . An element of a zerotree for threshold  $T$  is a *zerotree root* if it is not the descendant of a

previously found zerotree root for threshold  $T$  i.e. if it is not *predictable insignificant* from the discovery of a zerotree root at a coarser scale at the same threshold. A zerotree root is encoded with a special symbol indicating that the insignificance of the coefficient at finer scales is completely predictable. The significance map can be represented as a string of symbols from a 3-symbol alphabet which is then entropy-coded. The three symbols used are: 1) zerotree root; 2) isolated zero, which means that the coefficient is insignificant but has some significant descendant; and 3) significant. When encoding the finest scale coefficients, since these coefficients have no children, the symbols in the string come from a 2-symbol alphabet, whereby the zerotree symbol is not used.

Also the sign of the significant coefficient can be encoded along with the significance map. Thus the four symbols that are encoded in Shapiro's algorithm are the *zerotree root, isolated zero, positive significant* and *negative significant*.

#### 4.1.1.2 Set partitioning algorithm of Said and Pearlman

One of the main features of this scheme in transmitting the ordering data is that it is based on the fact that the execution path of any algorithm is defined by the results of the comparisons of its branching points. So, if the encoder and decoder have the same sorting algorithm, then the decoder can duplicate the encoder's execution path if it receives the results of the magnitude comparisons, and the ordering information can be recovered from the execution path.

The sorting algorithm divides the set of pixels into partitioning subsets  $T_m$  and performs the magnitude test:

$$\max_{(i,j) \in T_m} \{ |c_{i,j}| \} \geq 2^n ? \quad (4.4)$$

If the decoder receives a “no” to that answer (the *subset* is insignificant), then it knows that all coefficients in  $T_m$  are insignificant. If the answer is “yes” (the subset is significant), then a certain rule shared by the encoder and the decoder is used to partition  $T_m$  in new subsets  $T_{m,l}$  and the significance test is then applied to the new subsets. This

set division process continues until the magnitude test is done to all single co-ordinate significant subsets in order to identify each significant coefficient.

To reduce the number of magnitude comparisons (message bits) a set partitioning rule that uses an expected ordering in the hierarchy defined by the subband pyramid is defined (similar to that used by Shapiro in his zero-tree coding method). The objective is to create new partitions such that subsets expected to be insignificant contain a huge number of elements and subsets expected to be significant contain only one element.

To make clear the relationship between magnitude comparisons and message bits, the following function is used:

$$S_n(T) = \begin{cases} 1, & \max_{(i,j) \in T} \{c_{i,j}\} \geq 2^n, \\ 0, & \text{otherwise,} \end{cases} \quad (4.5)$$

to indicate the significance of a set of co-ordinates  $T$ . To simplify the notation of single pixel sets, one can write  $S_n(\{(i, j)\})$  as  $S_n(i, j)$ .

## 4.1.2 Successive approximation vector quantisation

Da Silva and Ghanbari [76] first introduced a successive approximation vector quantiser.

### 4.1.2.1 An algorithm for the successive approximation of vectors

In this method, each vector is coded by a series of vectors of decreasing magnitudes until a certain distortion level is reached. Da Silva and Ghanbari [76] have showed that lattice codebooks are an efficient tool for quantisation without the need for very large codebooks.

#### 4.1.2.1.1 Definition of the problem

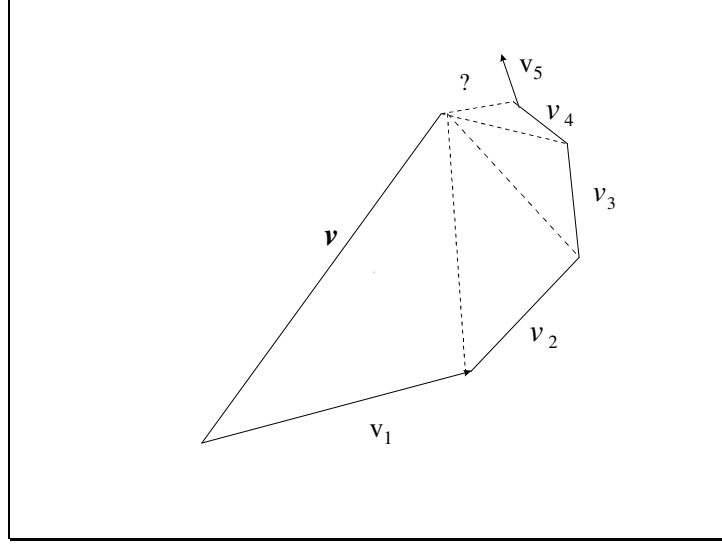
In order to state the problem clearly, one has first to consider the process of successive approximation in the scalar case. This is equivalent to approximating a given

length  $L$  by using at each pass yardsticks of increasingly smaller lengths until a certain level of error is obtained. The length  $l$  of the yardstick is always halved after each pass. The process begins by an initial yardstick length  $l$  such that  $l > L$ . Equation (4.6) shows an example where  $L = 0.7l$ . The length  $L$  can be expressed as:

$$L = +l - \frac{l}{2} + \frac{l}{4} - \frac{l}{8} + \frac{l}{16} + \dots \quad (4.6)$$

Therefore, the length  $L$  can be represented as a string of “+” and “-” symbols. As each symbol “+” or “-“ is added, the precision in the representation of  $L$  increases, and the distortion level decreases. Generalisation of this process of successive approximation of  $k$ -dimensional space is not a straightforward task. In general, a  $k$ -dimensional vector is defined by two parameters: its length, which is a scalar value that corresponds to the norm of the vector; and its orientation, which is a  $k$ -dimensional vector with unit energy. Successive approximation of vectors should consider both of these components (length and orientation) rather than only length as it is in the scalar case ( $k=1$ ).

Thus, instead of yardsticks of decreasing lengths, one can deal with “vector yardsticks” having decreasing lengths and given orientations in a  $k$ -dimensional space. In practice, these vector yardsticks will be chosen from a finite codebook, and therefore, the set of possible orientations is finite. At each pass the vector yardsticks will aim to approximate the residual vector formed as the difference between the original vector  $\vec{V}$  and its approximation so far. The question that arises is how one can guarantee that the vector approximation process converges ---that is, if the number of passes is sufficiently large, that the magnitude of the residual vector will be always smaller than an arbitrary value. Figure 4.1 shows that convergence is not necessarily guaranteed. Referring to Figure 4.1, we see that during the fifth pass the approximated vector is actually diverging from the actual vector  $\mathbf{v}$ . This is because the orientation code-book is fixed and contains vectors in certain directions only.



**Figure 4.1: Shows that convergence is not always guaranteed in the vector case.**

#### 4.1.2.1.2 Conditions for convergence

The following suppositions have been made:

1.) The orientation codebook  $Y$  is a finite set of  $k$ -dimensional vectors with unit energy such as,

$$Y = \{ \vec{y}_i : \|\vec{y}_i\| = 1; i = 1, 2, \dots, N \} \quad (4.7)$$

At each pass, a new vector yardstick is formed as the product of the current yardstick length and one of the unit orientation code vectors  $\vec{y}_i$

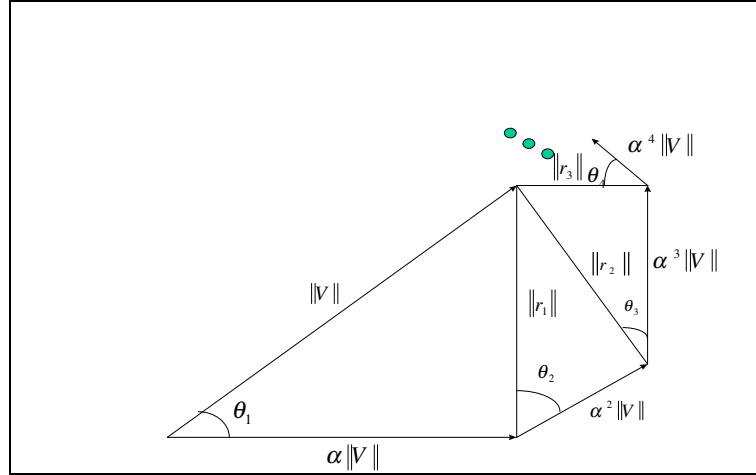
2.) The yardstick length at each pass will be scaled by a constant factor  $\alpha$ , the so-called approximation scaling factor, in the range of  $0.5 \leq \alpha < 1.0$

3.) For a given vector  $\vec{V}$ , the approximation process is activated for the first time at a certain pass indexed by  $i$ , if and only if the condition

$$l_i < \|\vec{V}\| \leq l_i / \alpha \quad (4.8)$$

is satisfied, where  $l_i$  denotes the yardstick length at pass  $i$  and  $\|\vec{V}\|$  is the norm of the vector  $\vec{V}$ . Therefore, the maximum error introduced by the first pass will occur in the

case that  $\|\vec{V}\| = l_{i-1} = l_i/\alpha$ . Figure 4.2 shows a diagram of the successive-approximation approach to quantising vectors.



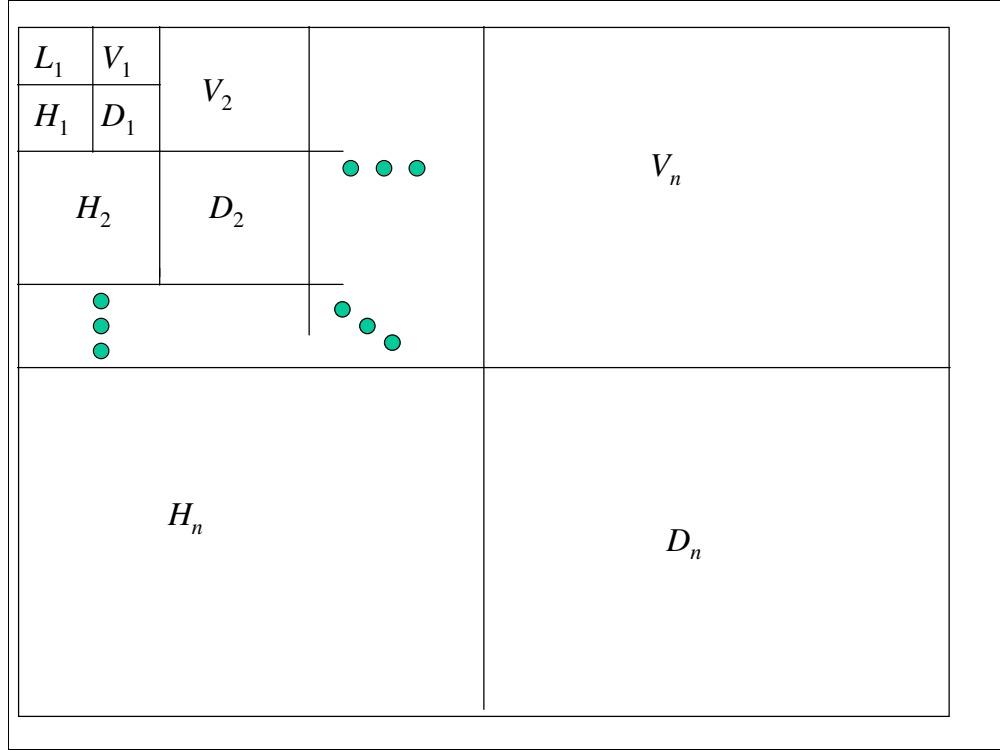
**Figure 4.2: Analysis of convergence for the multi-dimensional vector case. The notation is as given by Da Silva and Ghanbari [76]**

Da Silva and Ghanbari have found that for coding typical images, the value of  $\alpha$  for optimal rate vs. PSNR performance is greater than 0.5. They found that the best orientation code-books were based on the first spherical shells of the  $D_4$ ,  $E_8$  and  $\Lambda_{16}$  lattices, all of which exhibit the densest packing properties [34] in their respective dimensions.

#### 4.1.2.1.3 Algorithm of Da Silva and Ghanbari

Da Silva and Ghanbari use Shapiro's approach for exploiting the similarities among the bands of the same orientation to generate zero-tree roots.

In coding images, the image mean is extracted. A 5-stage biorthogonal wavelet transform is then applied to the zero-mean image yielding an image decomposition shown in Fig. 4.3.



**Figure 4.3: Hierarchical sub-band transform where  $V_n$ ,  $D_n$  and  $H_n$  are respectively the vertical, diagonal and horizontally oriented sub-bands.  $L_1$  is the DC band.**

The image is then divided into blocks of size  $M \times N$ , forming vectors of dimension  $MN$ . Depending on the band considered, scanning of the blocks to form vectors is different. The scanning is vertical in the vertical bands, horizontal in the horizontal bands and zigzag in the diagonal bands. The maximum value of all the vectors,  $V$ , is then computed. Initially, the yardstick length  $l$  is set to  $\alpha V$  where the value of  $\alpha$  is based on coding a  $256 \times 256$  Lena training set image at 0.5 bits/pixel (the value for  $\alpha$  then depends on which of the three different orientation code-books at dimensions 4, 8 or 16 is used). All the vectors are scanned, and the ones with magnitude smaller than  $l$  are set to zero (the vector is known as *insignificant*). Each of the remaining vectors is replaced by its closest orientation code vector scaled with magnitude  $l$ . After this pass, the locations of the zero vectors are transmitted. This is done via three symbols: zero( $Z$ ), zero tree root( $ZT$ ) and coded value ( $C$ ). If a vector is zero and all of its corresponding vectors in the

higher bands of the same orientation are also zero, this vector is replaced by a  $ZT$ , so that it is not necessary to transmit its corresponding vectors. For the lowest frequency band, a  $ZT$  implies that the corresponding vectors in all bands are zero. In case a vector is zero but not a  $ZT$ , it is marked as  $Z$ , and no information can be inferred about its corresponding vectors. On the other hand, a nonzero vector is replaced by a coded value symbol ( $C$ ).

The string generated by the three symbols is coded by an arithmetic coder. In the higher frequency bands, since there are no  $ZT$ 's the arithmetic coder uses a model with only two symbols ( $Z$  and  $C$ ). After encoding this string, which indicates the location of the zero vectors, the orientation code vectors of the nonzero vectors (marked as  $C$ ) are encoded. For this purpose, the model of the arithmetic coder is reinitialized to have as many symbols as the population of the orientation codebook. The yardstick length  $l$  is then updated through the multiplication by  $\alpha$ . The difference between the original and the nonzero reconstructed vectors is coded using the new yardstick length. The new orientation code vectors are also encoded into the bitstream via the arithmetic coder.

In the next pass the vectors that were previously found to be zero are scanned again. A new string of  $Z$ ,  $ZT$  and  $C$  is encoded into the bitstream. In order to reduce the number of symbols in this string, it is beneficial to obtain as many  $ZT$ 's as possible. To achieve this, the vectors that have been found nonzero so far are assumed to be zero during the  $ZT$  generation, although they are not encoded as  $Z$ . As in the previous pass, the indices of the  $C$  vectors are encoded and the whole process is repeated until a certain bit rate is achieved.

## 4.2. New algorithms for successive approximation coding of vectors

In this section, for our first original contribution, we shall apply the Said/Pearlman approach to transmitting the ordering information during each pass (i.e. whether a vector not previously found to be significant in a previous pass is significant or insignificant). It was shown by Said/Pearlman that their set partitioning scheme was more efficient than the zero-tree approach of Shapiro in the scalar case.

The new algorithm shall be called SA-LVQ (successive approximation lattice vector quantisation).

### 4.2.1. Implementation of SA-LVQ algorithm

The following notation is used where  $mn$  is the dimensionality of the vectors.  $\vec{O}(\vec{N})$  means the 4 vectors at the next finer level of the wavelet transform at the same orientation that contain the leaf nodes of each coefficient of the vector  $\vec{N}$ ,  $\vec{D}(\vec{N})$  means all the direct descendants of  $\vec{N}$ ,  $\vec{L}(\vec{N})$  means all the direct descendants of  $\vec{N}$  except for the immediate 4 offspring (that is  $\vec{L}(\vec{N}) = \vec{D}(\vec{N}) - \vec{O}(\vec{N})$ ). In the following algorithm, two symbols are transmitted, *NULL* (0 symbol) or *NOT NULL* (1 symbol). A set is deemed to be null if all the vectors within the set are all *insignificant* (as defined in 4.1.2.1.3). If, even a single vector is not *insignificant*, the set is deemed to be not null and the appropriate symbol is transmitted.

The vectors are formed from the subbands as outlined in the previous section. The algorithm is initialised as follows :

- ◆ The initial partition is formed with the set  $\vec{D}_{a,b}$ , where  $(a,b)$  is the co-ordinate of the first coefficient of an  $mn$ -dimensional vector, for all  $(a,b)$  in the highest pyramid level ( $L_1$  band of Figure 4.3). If the highest pyramid level is of size  $I \times I$ , then there will be  $I \times I / m \times n$  different values of  $(a,b)$

- ◆ If the set  $\vec{D}_{a,b}$  for all  $(a,b)$  in the highest pyramid level is not null then it is partitioned into the set  $\vec{L}_{a,b}$  for all  $(a,b)$  plus the four single vector sets with  $(k,l) \in \vec{O}_{a,b}$
- ◆ If the set  $\vec{L}_{a,b}$  is not null then it is partitioned into the four sets  $\vec{D}_{k,l}$ , with  $(k,l) \in \vec{O}_{a,b}$
- ◆ Each of the four sets now has the format of the original set and the same partitioning rules can be used recursively.

#### 4.2.1.1 Coding Algorithm

Since the order in which the subsets are tested for significance is important, in a practical implementation the significance information is stored in three ordered lists, called the *list of insignificant sets (LIS)*, *list of insignificant vectors (LIV)*, and *list of significant vectors (LSV)*. In all lists each entry is identified by a coordinate  $(a,b)$ , which in the *LIV* and *LSV* represents the co-ordinates of the first element of a vector and in the *LIS* represents either the set  $\vec{D}_{a,b}$  or the set  $\vec{L}_{a,b}$ . To differentiate between them we can say that a *LIS* entry is of type **A** if it represents  $\vec{D}_{a,b}$  and of type **B** if it represents  $\vec{L}_{a,b}$ .

During the zero vector generation process, the vectors in the *LIV* ----which were insignificant in the previous pass ---are tested, and those that become significant are moved to the *LSV*. Similarly, sets are sequentially evaluated following the *LIS* order, and when a set is found to be significant it is removed from the list and partitioned. The new subsets with more than one element (i.e. vector) are added back to the *LIV* while the single-coordinate sets are added to the end of the *LIV* or the *LSV* depending whether they are insignificant or significant, respectively. The *LSV* contains the coordinates of the vectors that are visited in the refinement pass.

## ALGORITHM:

**1. Initialisation:** Set the *LSV* as an empty list and add the coordinates  $(a,b)$  (where  $(a,b)$  are in the highest (coarsest) pyramid level to both the *LIV* and to the *LIS* as type **A** entries. Let the dimensions of the  $L_1$  Band (Figure 4.3) be  $I \times I$ . Let the vector dimension be  $m \times n$ .

for  $a = 1:m:I$  do:

for  $b = 1:n:I$  do:

Add each  $(a,b)$  to both the *LIV* and the *LIS*

end;

end;

Let the initial yardstick be equal to  $\alpha$  multiplied by the  $l_2$  norm of the vector with the largest magnitude.

## 2. Sorting Pass:

for each entry  $(a,b)$  in the *LIV* do:

1) output whether the vector is NULL or NOT NULL.

2) if NOT NULL, then move  $(a,b)$  to the *LSV*

3) Also, output symbol for nearest orientation code-vector (where the code-book consists of  $K$  unit energy code-vectors)

4.) for each entry  $(a,b)$  in the *LIS* do:

5) if the entry is of type **A** then:

◆ output whether  $\vec{D}_{a,b}$  is NULL or NOT NULL

◆ if NOT NULL then

• for each  $(k,l) \in \vec{O}_{a,b}$  do:

• output whether vector given by co-ordinates  $(k,l)$  is Null or not;

• if not null, then add  $(k,l)$  to the *LSV* and

• Also, output symbol for nearest orientation code-vector

• if null, then add  $(k,l)$  to the end of the *LIV*

- if  $\vec{L}_{a,b}$  is NOT NULL then move  $(a,b)$  to the end of the *LIS* as an entry of type **B**
  - else remove  $(a,b)$  from the *LIS*
- end;

2) if the entry is of type **B** then

- ◆ output whether  $\vec{L}_{a,b}$  is NULL or NOT NULL
- ◆ if NOT NULL then
  - add each  $(k,l) \in \vec{O}_{a,b}$  to the end of the *LIS* as an entry of type **A**;
  - remove  $(a,b)$  from the *LIS*

### 3. Refinement pass

For each  $(a,b)$  in the *LSV* do:

Let the original vector given by co-ordinates  $(a,b)$  be  $\vec{V}$ . Let its current reconstructed value be  $\vec{E}$  and the value of the current yardstick be  $Y$ .

Then find the nearest orientation code-vector to  $\vec{V} - \vec{E}$  and let it be  $\vec{U}$ .

Then the new reconstructed value is  $Y\alpha\vec{U} + \vec{E}$

### 4. Yardstick update

Update yardstick by multiplying it by  $\alpha$ . Go to step 2.

One important characteristic of the algorithm is that the entries added to the end of the *LIS* are evaluated before that same sorting pass ends. So, when we say “for each entry in the *LIS*”, we also mean those being added to its end. Note that in the Algorithm, all branching conditions based on the outcome of significance tests ---which can only be calculated based on knowledge of the values of the wavelet coefficients are output by the encoder. Thus, to obtain the desired decoder’s algorithm, which duplicates the encoder’s execution path as it sorts the significant vectors, we simply replace the words *output* with *input*. The ordering information is recovered when the co-ordinates of the significant coefficients are added to the end of the *LSV*. But note that whenever the decoder inputs

data, its three control lists (*LIS*, *LIV* and *LSV*) are identical to the ones used by the encoder at the moment it outputs that data, which means that the decoder indeed recovers the ordering from the execution path.

During the **initialisation phase**, all vectors in the DC band are added to the *LIS* as they all have immediate offspring. Normally vectors have 4 offspring but vectors in the DC band have 3 immediate offspring given by  $\{(a, b + I), (a + I, b), (a + I, b + I)\}$  where  $(a, b)$  are the co-ordinates of the vector (actually, co-ordinates of the top left pixel of an  $m \times n$  block from which an  $mn$  dimensional vector is formed) and the dimensions of the DC band are  $I \times I$ . The co-ordinates of the 4 offspring when the vector is not in the DC band or in the 3 lowest level bands are as follows: (It is assumed the co-ordinates of the top left element of a matrix are (1,1) as in Matlab)

$$\{(2a - 1, 2b - 1), (2a - 1, 2b - 1 + n), (2a - 1 + m, 2b - 1), (2a - 1 + m, 2b - 1 + n)\} \quad (4.9)$$

During the **refinement phase**, each of the vectors in the *LSV* is further refined using the current yardstick length multiplied by  $\alpha$  (i.e. the yardstick length of the next pass) and the appropriate symbol transmitted. When significant vectors are discovered during the sorting phase, the symbol which represents the nearest orientation codevector (i.e. equivalent to the sign being output for scalars) is immediately transmitted so that during **both** the sorting and refinement phases, as the bit-rate is increasing, the distortion of the reconstructed image is likely to be reducing. This is in contrast to the approach used by Da Silva and Ghanbari [76] which we shall mention in the next section.

### 4.3. Comparisons in performance between various SA-LVQ algorithms

In this section we shall be giving some results for two versions of our coding scheme, SA-LVQ, namely SA-LVQ-E and SA-LVQ-B. In order to define these two schemes, we first need to state the orientation code-book used in our simulations.

Da Silva and Ghanbari analysed the performance of various lattice based code-books. They found the best performance in general was by using vectors on the first spherical shells of the  $\Lambda_{16}$ ,  $E_8$  and  $D_4$  lattices. The performance of the  $\Lambda_{16}$  lattice was in general about 0.2dB better than that for the  $E_8$  lattice which was in turn about 0.7dB better than for the  $D_4$  lattice.

In this section we have used the code-vectors on the first spherical shell of the  $E_8$  lattice to be in the orientation code-book. 8-D vectors are formed by dividing the image into  $4 \times 2$  or  $2 \times 4$  blocks of coefficients depending on the orientation of the sub-band [76]. There are 240 code-vectors on this shell of the form: 128 of the form  $(1/2, 1/2, 1/2, 1/2, 1/2, 1/2, 1/2, 1/2)$  with there being an even number of minus signs and 112 of the form  $(1, 1, 0, 0, 0, 0, 0, 0)$  where all permutations including any number of minus/plus signs are permissible. We have used the fast encoding quantisation algorithm, originally suggested by Conway and Sloane and then given in a modified form by Ghanbari et al. for the truncated and scaled version [35,79], for this lattice.

We have found in practical experiments that the contribution to the ordering information (i.e. those bits output during the sorting phase) when using the first shell of the  $D_4$  lattice is in general about 27-30% of the overall bit-rate. The contribution reduces to between 16-19% of the overall rate when using the first shell of the  $E_8$  lattice. It is therefore likely that the contribution to the overall bit-rate of the ordering information, if using the first shell of the  $\Lambda_{16}$  lattice (where there are 4320 vectors on the first shell), is likely to be considerably lower than for the  $E_8$  lattice. This would be likely to reduce any benefit of using a more efficient algorithm than that used by Da Silva and Ghanbari during the sorting phase as we have attempted to do.

The algorithms which we have denoted as SA-LVQ-E and SA-LVQ-B can now be defined. We first give a definition of the method Silva-Ghan-Mod which we shall use for comparison purposes and is a modification of the method outlined by Da Silva and Ghanbari.

#### **Algorithm Silva-Ghan-Mod**

- \* A 5-level bi-orthogonal wavelet transform with the 7/9 tap filters (see A.1 for coefficients etc.) is used (rather than the 14/10 tap filters used by Da Silva and Ghanbari)

- \* We have found that the 240 orientation code-vectors are typically approximately equiprobable. An arithmetic coder [45] is used to encode both the symbol string ( $C$ ,  $Z$  and  $ZT$ ) generated during the sorting phase and the index of the orientation code-vectors during the refinement phase.

- \* After finding a significant vector during the sorting phase, we immediately transmit the index of the nearest (based on  $l_2$  norm) orientation code-vector (after transmitting the  $C$  symbol of course!) rather than wait until all previously insignificant vectors (from previous passes) are tested and then transmit this information during the refinement phase as is done by Da Silva and Ghanbari. This is to ensure that during the sorting phase, as the bit-rate is increasing the distortion of the decoded image is reducing.

#### **Algorithm SA-LVQ-E**

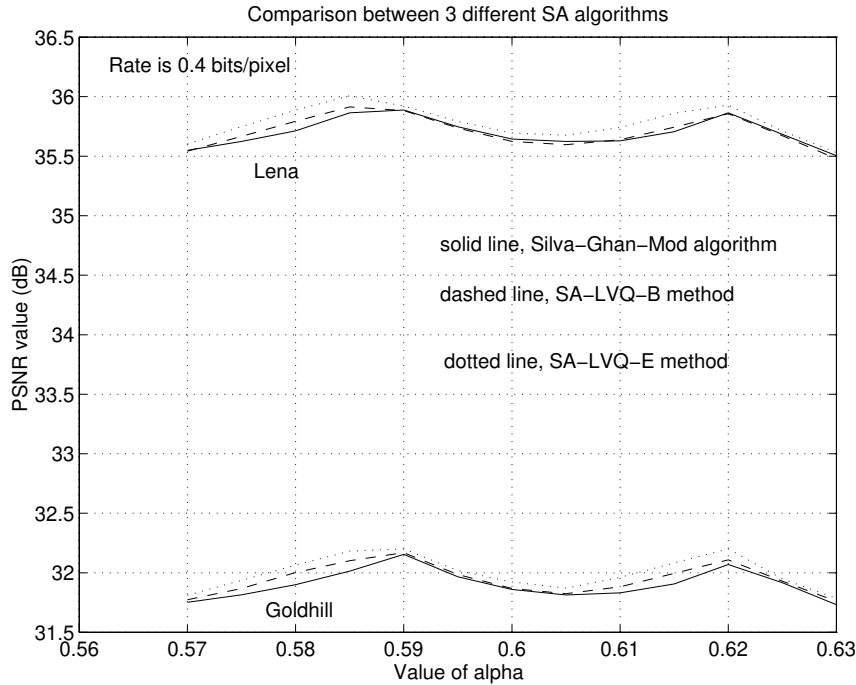
- \* The output symbols from the sorting phase (from a two symbol alphabet) and the orientation code-vectors are arithmetically coded [45] as previously described in the Silva-Ghan-Mod algorithm.

### **Algorithm SA-LVQ-B**

\* The output symbols from the sorting phase are not entropy coded (i.e. they are transmitted binary uncoded (as described by Said/Pearlman [50])). The orientation code-vectors are also transmitted without entropy coding using 8 bits per orientation vector (i.e.  $\lceil \log_2 240 \rceil$ ). One version of Said and Pearlman's [50] SPIHT coder uses no entropy coding (S-P-B) and the other version uses entropy coding (S-P-E) on the bits output during the sorting phase. They suggest that the former method may be desirable where computational considerations are important. Some differences between the binary uncoded version of Said and Pearlman and our SA-LVQ-B method from a computational point of view are: 1) Typically because the scaling parameter  $\alpha$  is greater than 0.5 with the SA-LVQ-B method (as opposed to 0.5 with the S-P-B method) more passes are required to achieve the same distortion which imposes a greater computational burden on the SA-LVQ-B method); 2) The quantisation algorithm for the  $E_8$  lattice [35] requires 2 distance (quadratic norm) computations to determine the nearest lattice point whereas in the S-P-B method no such distance computation is required; 3) However, in the S-P-B method the vast majority of the bits are output during the sorting phase which requires 3 linked lists to be maintained, namely the LIP (list of insignificant pixels), LIS (list of insignificant sets) and LSP (list of significant pixels). Whereas the S-P-B algorithm is dealing in single wavelet coefficients, the SA-LVQ-B method is *more efficiently* determining the ordering information of blocks of eight wavelet coefficients at a time.

#### **4.3.1. Some results for SA-LVQ algorithms**

In this section we shall be comparing the performance of the SA-LVQ algorithms on the two well known  $512 \times 512$  test images **Lena** and **Goldhill**. Results on other images have shown that the following results are indicative of the performance of the SA-LVQ methods.



**Figure 4.4: Plot of PSNR performance using different values of  $\alpha$  for the three algorithms, SA-LVQ-E, SA-LVQ-B and Silva-Ghan-Mod. The bit-rate is 0.4 bits/pixel and the two images are of size 512×512.**

Figure 4.4 shows the relative performance of the SA-LVQ-E, SA-LVQ-B and Silva-Ghan-Mod algorithms at a bit-rate of 0.4 bits/pixel for different values of the scaling parameter  $\alpha$  (it should be remembered that for scalar quantisation the minimum value for  $\alpha$  to achieve convergence is 0.5). We find that there tends to be two peaks at which the PSNR is maximised which we shall call  $\alpha_1$  and  $\alpha_2$  where  $\alpha_2 > \alpha_1$ . Because the number of passes to achieve a desired distortion level would tend to be greater for  $\alpha_2$  compared with  $\alpha_1$ , it would be suitable to select as our training set value for  $\alpha$ , the smaller of the two values. Based on an analysis of these and other images at various bit-rates we have selected a training set value for  $\alpha=0.585$ .

Figure 4.4 also shows that there is only a small improvement in rate vs. PSNR performance from using a more efficient algorithm during the sorting phase (SA-LVQ-E vs. Silva-Ghan-Mod). In the scalar case for example, Said and Pearlman found an almost

1 dB improvement in rate vs PSNR performance over the method of Shapiro. The reasons why a much smaller improvement is observed in the vector case are as follows: 1) The contribution of the ordering information as previously mentioned is only about 16-19% of the overall bit-rate as opposed to over 70% in the scalar case; 2) In the scalar case, the yardstick value (which determines significance/insignificance) is being successively reduced by  $\alpha=0.5$  whereas for convergence reasons in the vector case,  $\alpha$  is nearer 0.6. This means many more sorting phases are needed to achieve the same distortion. The advantage of the set partitioning algorithm over the zero-tree method of Shapiro would appear to diminish in this situation.

*Table 4.5 : Tabulation of PSNR results for various algorithms for the images Lena (4.5.1) and Goldhill (4.5.2.) respectively, at 5 different bit-rates. The bit-rates are in units of bits/pixel and the numbers in the columns below the bit-rates are the PSNR values of the decoded image in units of dB.*

**Table 4.5.1: For the Lena image**

<b>bit-rate</b>	<b>0.15</b>	<b>0.25</b>	<b>0.4</b>	<b>0.5</b>	<b>0.75</b>
<b>Silva-Ghan-mod</b>	31.45	33.78	35.86	36.90	38.68
<b>SA-LVQ-E</b>	31.48	33.80	36.00	36.98	38.74
<b>SA-LVQ-B</b>	31.41	33.76	35.91	36.93	38.70
<b>S-P-E</b>	31.90	34.12	36.25	37.22	39.05
<b>S-P-B</b>	31.48	33.69	35.83	36.84	38.56

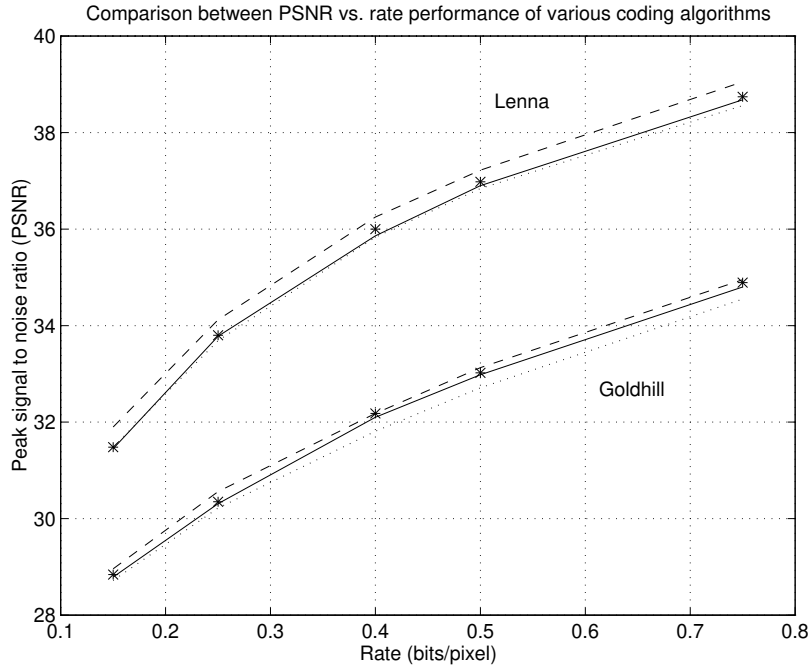
**Table 4.5.2 For the Goldhill image**

<b>bit-rate</b>	<b>0.15</b>	<b>0.25</b>	<b>0.4</b>	<b>0.5</b>	<b>0.75</b>
<b>Silva-Ghan-mod</b>	28.76	30.31	32.01	32.96	34.62
<b>SA-LVQ-E</b>	28.84	30.35	32.18	33.02	34.89
<b>SA-LVQ-B</b>	28.79	30.31	32.10	32.98	34.80
<b>S-P-E</b>	28.96	30.56	32.18	33.13	34.95
<b>S-P-B</b>	28.72	30.22	31.81	32.71	34.55

Table 4.5 shows results for the various algorithms at different bit-rates for the images Lena and Goldhill with  $\alpha = 0.585$ . The S-P-E and S-P-B methods which table 4.5 refers to are the entropy coded and binary uncoded versions respectively of the Said/Pearlman algorithm.

We find that for the **Lena** image, SA-LVQ-E outperforms the Silva-Ghan-Mod method by on average 0.07dB. The SA-LVQ-B algorithms which uses no entropy coding is comparable in performance to the Silva-Ghan-Mod method. Also at bit-rates above and including 0.25 bpp, *the SA-LVQ-B algorithm modestly outperforms the S-P-B method by about 0.08dB on average.* However *SA-LVQ-E is inferior to the best Said/Pearlman method for scalars (S-P-E) by on average 0.31dB.*

For the **Goldhill** image, SA-LVQ-E outperforms the Silva-Ghan-Mod method by on average 0.13dB. The SA-LVQ-B algorithm which uses no entropy coding also betters the Silva-Ghan-Mod method by about 0.07dB. *The SA-LVQ-B algorithm outperforms the S-P-B method by about 0.2dB on average.* However *SA-LVQ-E is inferior to the best Said/Pearlman method for scalars (S-P-E) by on average 0.1dB.*



**Figure 4.6: A visual format for the results in table 4.5. The various algorithms referred to are as follows: (dotted line:S-P-B, solid line:SA-LVQ-B, ‘\*’:SA-LVQ-E and dashed line:S-P-E)**

Figure 4.6 shows the results of table 4.5. in a visual format.

#### 4.4 Subjective performance of SA-LVQ algorithms

We have compared the subjective quality of the decoded images from the SA-LVQ-B, S-P-B and S-P-E algorithms (the latter two referred to as S+P methods) for the original image **Lena** at 0.25 bits/pixel. The respective PSNR values for the decoded images are 33.75dB, 33.69dB and 34.12dB. We found for example that in the hat region, (of Lena) in quite a lot of areas, there was more texture detail using the SA-LVQ-B method than either of the Said/Pearlman algorithms. However, in at least a few areas of the hat and other regions, the S+P methods seemed to produce decoded images closer to the original. We can attribute this mixed performance of the SA-LVQ-B method to the differences in the

quantisation used compared to the S+P methods. These differences can be highlighted for the present example of Lena coded at 0.25 bpp and are as follows:

1) The SA-LVQ-B method terminated during a refinement phase with the current yardstick value=40.758. The S-P-B method terminated during a sorting phase with the yardstick value=16. (Here 48% of the coefficients with magnitude between 16 and 32 were set to  $\pm 24$  and the rest were still zero)

2) In the S-P-B method, the absolute values at the decoder of about 4000 of the 8100 coefficients (the figure of 8100 is based on the wavelet transform of the original image) with magnitude between 16 and 32 were set to 24 and the rest were zero. In the SA-LVQ-B method, of those 8100 coefficients, 4300 at the decoder had magnitude greater than 14, 4500 greater than 12 and 4800 greater than 9.

3) Of the approximately 16300 coefficients between magnitudes of 8 and 16, S-P-B sets these all to zero at the decoder whereas 2300 of those have magnitudes greater than 8 with the SA-LVQ-B algorithm.

4) However of the approx. 1850 coefficients between 32 and 40.758, the S-P-B method sets the magnitudes of these ones to 48 at the decoder but only 1500 of them have magnitude greater than 24 with the SA-LVQ-B algorithm (1800 have magnitude greater than 10).

In Figure 4.7 we can see sections of the decoded images from the three algorithms, SA-LVQ-B, S-P-E and S-P-B. For example in the little bit of the hat area which is visible, there is slightly more texture in the SA-LVQ-B image but the white part of the lower right eye is more visible in the S+P images. In the original image, we can clearly see distinct eyelashes below the bottom two eyelids. These distinct eyelashes have “merged” in the three decoded images. We shall see in the next chapter that our major algorithm of this dissertation called ECPVQ can in fact show these eyelashes distinctly.



**Figure 4.7: Shows  $128 \times 128$  sections of the decoded images of three algorithms where the original image coded is Lena at 0.25 bpp. (Top left: Original Lena. Top right: SA-LVQ-B algorithm with PSNR =33.75dB. Bottom left: S-P-B with PSNR =33.69dB. Bottom right: S-P-E with PSNR =34.12dB)**

## 4.5 Conclusions

We have in this chapter developed an extension to an existing successive approximation vector quantisation algorithm using ideas suggested by Said and Pearlman [50] when using scalar quantisation. We found only a modest improvement in rate vs. PSNR performance over the existing VQ algorithm of Da Silva and Ghanbari. On further investigation, we found that the optimum embedded scalar quantisation method which uses entropy coding outperforms (from a rate vs. PSNR viewpoint) our entropy coded successive approximation (SA) method. However a binary uncoded version of our SA method outperforms a binary uncoded version of the embedded scalar quantisation method.

We feel it is therefore likely that the best rate/distortion performance is likely to be realised by entropy coded scalar quantisation embedded algorithms rather than embedded vector quantisation methods.

## 4.6 Future research

We used the  $E_8$  lattice as the basis for our orientation code-book. Da Silva and Ghanbari found that the  $\Lambda_{16}$  lattice performed on average 0.2dB better. However, any advantage in using the algorithm we employed during the sorting phase is likely to be even less than we found because the contribution of the ordering bits to the overall bit-rate is likely to be further reduced due to the increase in dimensionality (from 8 to 16) and in the number of orientation code-vectors (from 240 to 4320).

Therefore any future research in this area should concentrate on quantising vectors formed inter-band (i.e. say 20-D vectors where 4 coefficients are from one sub-band on a particular scale and 16 from the sub-band at the next finer scale of the same orientation in the spatial location). A fast quantising algorithm for the  $\Lambda_{20}$  lattice (this is the closest packing lattice in 20 dimensions [34] and has 19400 code-vectors on its first spherical shell) would need to be written. In the sorting phase, because each 20-D vector would

have 16 children rather than 4 as previously, a more efficient method than the set partitioning approach may be a modification of the contextual bit-plane coding approach of the significance plane which has been successfully used recently [74].

## Chapter 5

---

### A novel class based method for coding hierarchical vectors using entropy coded pyramid vector quantisation

---

We will introduce in this chapter a new way of coding hierarchical vectors, quantised by a lattice (which we have termed ECPVQ, or entropy coded pyramid vector quantisation), based on a 2-D separable wavelet transform. This will exploit the correlation between neighbouring vectors and that between coefficients in each vector. A novel approach will be taken in entropy coding the lattice quantised vectors. One of the lattices (the  $Z_n$  or cubic lattice) considered for quantisation is also equivalent to scalar quantisation by a uniform mid-tread quantiser. A new quantiser called the  $Z_n/D_n$  augmented lattice is also developed.

We shall initially review the previous work in this area. Then we shall describe the motivation behind our new entropy coding scheme and subsequently its details.

Finally arithmetically coded results using ECPVQ are presented, and comparisons with current state-of-the-art coders are given for the peak signal-to-noise ratio criterion (the subjective quality is also visually analysed), which illustrates the good rate vs. distortion, subjective and computational performance of the ECPVQ technique.

#### 5.1. Review of previous entropy-coded lattice vector quantisation work

Using the well-known LBG method for vector quantisation [20] is computationally expensive and results in blur artifacts at low bit-rates [60]. For a fixed vector length of practical size, the scheme that entropy-codes its index sequence [21] is the one that achieves the same distortion  $d_n$ , with a lower average transmission rate. In fact, the ECVQ

algorithm [21], which is a generalisation of the LBG algorithm, is used to design vector quantisers subject to an entropy constraint.

For high bit rate  $R$  and high dimension space  $n$ , the number  $2^{nR}$  (the total number of code-vectors required) is not achievable by vector quantisers generated by the LBG or ECVQ algorithms. In fact, this optimal algorithm is computationally expensive.

In 1979 Gersho [36] conjectured that the optimal high-resolution ECVQ should have the form of a *lattice*. Thus lattice quantiser methods have been investigated in image coding; see [32,33,37,38,60,65,66,67,68,69,70,77,78].

The “code-book” of a lattice quantiser is a particular subset of a regular arrangement of points in an  $n$ -dimensional space centred on zero.

Quantisation algorithms have been developed by Conway and Sloane for the different types of lattice. Unlike LBG-type algorithms, there is usually no need to compute a norm to find (among all of the vectors in the codebook) the best reproduction vector.

Antonini amongst others [33,37,38,60,68,70] found that to obtain a suitable and implementable coding scheme, it was necessary to scale and truncate the lattice suitably. Various authors [33,60,67] have considered the question of the shape of the truncated area. When the signal to be compressed has an i.i.d. multivariate Gaussian distribution, the surfaces of equal probability are ordinary hyper-spheres. The truncated area should then be spherical. However, motivated by image coding applications, Fisher in 1986 [66] investigated the case of Laplacian sources (for cubic lattices) where surfaces of equal probability are planar and form shapes called *pyramids*. In his approach (for a fixed-rate scheme as opposed to variable-rate algorithms), the radius and codebook index of vectors lying on the pyramid are coded.

Antonini and others [33,60,67,68,70] have applied lattice vector quantisation (LVQ) to subband image coefficients using both the Gaussian and Laplacian models for

the data. However experiments have shown that the Laplacian pdf is more suitable for modelling the pdf of wavelet coefficients [60].

The general technique employed by researchers using either model for sub-band/wavelet coefficients is similar:

- Firstly, the sub-bands are split into blocks of coefficients to form  $n$  dimensional vectors. The dimensionality of the vectors may vary from sub-band to sub-band depending on the size of the sub-band and the statistics of the coefficients within it (determining the sub-bands' contribution in terms of bits to the overall desired bit-rate)

- A lattice is chosen for quantisation of the source vectors (normally a  $D_n$ ,  $Z_n$ ,  $E_8$  or  $\Lambda_{16}$ )

- For each sub-band, a dictionary size for the truncated lattice (i.e. number of code-words within the truncated region) is selected. The number of points within the truncated area can be calculated for all the main lattices (i.e  $D_n$ ,  $Z_n$ ,  $E_8$  using either the well known theta series [34] (for spherical shaped shells) or the more recently developed  $Nu$  functions [60] (for pyramidally shaped shells). This determines the energy of the largest shell (whether pyramidal or spherical in shape).

- If the scaled source vectors lie within the truncated area, they are quantised using one of the fast algorithms in [35] and the index of the nearest lattice vector is encoded using entropy coding. The probabilities of using each lattice point are pre-stored from a training phase. For any scaled source vector which lies outside the truncated region, it was found [60] that the optimum solution (in terms of rate vs. distortion) is to project this “high energy” vector onto or within the largest sphere or pyramid using a different scaling factor  $\alpha$  -Then quantisation and entropy coding can take place as usual. In addition  $\alpha$  is quantised to  $\text{round}(\alpha)$ , binary encoded and transmitted. This enables this different scaling factor to be used at the decoder to re-scale the lattice vector.

One of main limitations of this approach is that as the dimensionality of the vectors increases, and for a particular dimensionality the lattice truncation energy is increased

(energy of the largest possible pyramid or sphere), the dictionary size also increases. As each lattice point within the dictionary is assumed to have a separate length entropy code (different probabilities of occurrence) associated with it, these probabilities must be determined from a training phase. However the accuracy with which these probabilities can be determined diminishes as the dictionary size increases. For example if the dimensionality of the vectors is 4, and the lattice truncation energy is 10 (for a  $D_4$  lattice), then the number of vectors within the truncated pyramidal region is 4961. For a dimensionality of 16, and a lattice truncation energy of 10 (for the  $\Lambda_{16}$  lattice), the number of vectors is 92160. Therefore, to obtain accurate probabilities, the dimensionality is likely to be restricted to around 4 or 8 dimensions and the energy of the largest pyramid or sphere is also limited.

One of the reasons why lattice vector quantisation is preferred over standard LBG type VQ algorithms is the ability to use vectors of large dimensionality. However when entropy coding is used, the dimensionality must be limited by the standard approach which has been used by various researchers (e.g [33,60,67,70]).

Recently Rogers et al. [32] and Fisher and Yusof [65] have tried to overcome this restriction by assuming that all possible code-vectors on a particular pyramid or sphere are equally probable. In addition, instead of just considering intra-band vectors (vectors composed of sub-band coefficients from the same sub-band), inter-band vectors (where coefficients of different sub-bands are grouped together) have also been looked at. When inter-band vectors contain coefficients from the same sub-band (as well of course as coefficients from other sub-bands), one can potentially exploit different correlations, for example correlations across scales in the case of a dyadic wavelet transform. Rogers [32] coded 63-D inter-band vectors (from a 2-D separable wavelet transform) on spherical shells. No truncation of the lattice was necessary. The energy of each spherical shell was entropy coded. A fixed-length code was then used to encode the position of the code-vector on the appropriate shell. As the number of shells is modest (less than 200), he

found that accurate probability estimates from a training phase could be determined, enabling good design of a huffman code for the “radial” parameter. His results were not good, in terms of rate vs. PSNR performance (in comparison with other coders), but this is likely to be because all code-vectors on any spherical shell **should not** be assumed to be equally probable in the case of **inter-band** vectors (as the coefficients may have different **variances** even assuming that the joint distribution can be described as multivariate Gaussian and the coefficients are independent).

Fisher and Yusof in their 1996 paper [65] explored the use of both inter and intra-band vectors for the case where pyramidally-shaped shells are used. He introduced [65,69] enumeration methods, for the first time, for his “weighted” pyramid which is used to encode inter-band vectors (from a DCT, the 63 a.c. coefficients are grouped together in vectors). We shall briefly review his method and then propose a new method, specifically adapted to code inter-band vectors from a 2-D separable wavelet transform.

### **5.1.1. Review of Fisher’s [65] pyramid vector quantiser for inter and intra-band vectors**

Fisher’s lossless coding scheme for quantised vectors is motivated by a laplacian model for the transformed image data. The lossless code design is dependent on the quantised vector activity.

The lattice VQ and variable-length lossless code presented are based on using a pyramid support region for a fixed-rate lattice VQ . No estimates of encoding rate are necessary and actual compressed file size is used to determine the encoding rate.

Efficient enumeration and decoding algorithms are used to map lattice codewords to binary strings and vice versa.

The VQ and lossless code may be viewed as variable-rate versions of the pyramid VQ.

Assume a vector  $x$  of independent laplacian random variables. Let the vector  $x = (x_1, x_2, \dots, x_L)^T$  have elements with variances  $2/\lambda_i^2, i=1\dots L$ , and mean zero.

The joint density of the vector  $x$  is then given by :

$$f(x) = \left( \prod_{i=1}^L \left( \frac{\lambda_i}{2} \right) \exp\left(-\sum_{i=1}^L \lambda_i |x_i|\right) \right) \quad (5.1)$$

and contours of constant density satisfy

$$\sum_{i=1}^L \lambda_i |x_i| = \text{constant} \quad (5.2)$$

We define the ‘weighted pyramid’ as

$$S(L, \underline{\lambda}, C) = \left\{ x: \frac{1}{L} \sum_{i=1}^L \lambda_i |x_i| = C \right\} \quad (5.3)$$

The  $1/L$  factor is to normalise per dimension.

The asymptotic equipartition principle of information theory [39] indicates that for large encoding rate and uniformly distributed VQ codewords all codewords close to  $S(L, \underline{\lambda}, C)$  are roughly equally likely. In principle, then, we can construct a VQ/lossless code by using a lattice to define the uniformly distributed VQ region in  $\mathfrak{R}^L$  and select a suitable enumeration code to assign equal-length binary codewords to index each lattice vector that lies on the surface  $S(L, \underline{\lambda}, C)$ .

However, suitable scaling and rounding must be introduced to make the enumeration code work. This is accomplished as follows:

Let  $\tilde{x} = x/\Delta$  so that the weighted pyramid becomes

$$S(L, w, K) = \left\{ y: \sum_{i=1}^L w_i |\tilde{y}_i| = K \right\} \quad (5.4)$$

where  $w_i = Rnd\left[\frac{\lambda_i}{\min_j \lambda_j}\right]$ ,  $i = 1, \dots, L$ , are positive integers,  $Rnd[x]$  denotes the closest

integer to  $x$  and  $\tilde{y}$  is the lattice point closest to  $\tilde{x}$ . *It should be noted that in Fisher’s method, each  $\lambda_i$  is obtained from an estimate of the variance  $\sigma_i^2$ . It would be equally valid for example to obtain  $\lambda_i$  based on an estimate of the mean absolute value. This*

would result in different values for  $w_i$  being obtained (except for the highest variance component whose  $w_i$  is unity). The only case where the same  $w_i$  values are obtained would be if all the coefficients came from the same sub-band.

The vector quantisation of an input  $x$  is simple.  $x$  is scaled to form  $\tilde{x} = x/\Delta$  and the closest lattice point to  $\tilde{x}$ ,  $\tilde{y}$  is computed.

We refer to  $\Delta$  as the lattice stepsize. For the cubic lattice ( $Z_n$ ),  $\tilde{y}$  is easily computed by rounding each element of  $\tilde{x}$  to the nearest integer.

Fisher treats only the cubic lattice in his paper as his enumeration algorithms were developed with this lattice in mind. This effectively uses the (scaled) midpoint of each quantisation region as its reproduction vector since  $\|x - y\|^2 = \Delta^2 \|\hat{x} - \hat{y}\|^2$  and as the possible  $\tilde{y}$  are fixed lattice points, the quantisation distortion is adjusted simply by selecting an appropriate value of  $\Delta$ .

The lossless code maps the lattice point  $\tilde{y}$  to a binary string for transmission or storage. This code has codewords formed as a concatenation of codewords from two uniquely decodable codes. The first code is a prefix code for the “size” of the weighted pyramid on which  $\tilde{y}$  lies. The second code is an enumeration code for the particular lattice point  $\tilde{y}$  on  $S(L, w, K)$ .

More specially, let the weighted pyramid “radius” be defined as the weighted  $l_1$  norm

$$r = \|\tilde{y}\|_{l,w} = \sum_{i=1}^L w_i |\tilde{y}_i| \quad (5.5)$$

Since  $\tilde{y}_i$  is an integer,  $r$  must be a nonnegative integer. Let  $C'$  be a code for  $r$ , with (non-integer length) codewords  $c'$ .

Clearly,  $\tilde{y} \in S(L, w, r)$ ; therefore given knowledge of  $r$ , it takes  $\lceil \log_2 N(L, w, r) \rceil$  bits to uniquely specify  $\tilde{y}$ , where  $\lceil x \rceil$  is the smallest integer no smaller than  $x$ . Let  $C''(r)$  be an enumeration code that maps each  $\tilde{y} \in (S(L, w, r))$  to a unique binary codeword, say  $c''$  of length  $\lceil \log_2 N(L, w, r) \rceil$

Then the lossless encoding of  $\tilde{y}$  consists of

- 1) computing  $r$  as the weighted  $l_1$  norm
- 2) losslessly encoding  $r$  as  $c'$  (by Huffman coding in Fisher's method)
- 3) losslessly encoding  $\tilde{y}$  using the enumeration code  $C''$  [69]
- 4) forming a concatenated codeword  $c=(c',c'')$

## IMPLEMENTATION

### Encoding

- 1) vector quantisation : The input vector  $x$  is scaled and the closest lattice point computed  $\tilde{y} = Rnd[x/\Delta]$
- 2) Enumeration: The value  $r = \|\tilde{y}\|_{1,w}$  is computed
- 3)  $r$  is encoded using a code  $C'$  as  $C'(r)$
- 4)  $\tilde{y}$  is encoded as a  $\lceil \log_2 N(L,w,r) \rceil$  bit string  $c''(\tilde{y})$  using an enumeration code  $C''(r)$  [69]
- 5) Concatenated codeword  $(c'(r),c''(\tilde{y}))$  is transmitted

The decoding is straightforward. A small amount of side information is used to represent the quantisation stepsize  $\Delta$  and the variances  $\sigma_i^2$ . From the variance size information and the Laplacian model, the reproduction centroids can be calculated

The transmitted or stored data consists of binary strings and those are decoded as follows:

### Decoding

- 1)  $C'(r)$  is parsed from  $(c',c'')$  and decoded as  $r = \|\tilde{y}\|_{1,w}$
- 2) The value of  $\lceil \log_2 N(L,w,r) \rceil$  is used to parse  $c''$  and the enumeration decoding algorithm is used to compute the lattice vector  $\tilde{y}$
- 3) Each dimension of  $\tilde{y}$  indexes a quantisation interval. The reproduction level is the respective centroid, computed based on the transmitted variances and the laplacian model.

## 5.2 Motivation for developing new ECPVQ method

We have coded quantised inter-band vectors by making use of pyramidal shells in such a way that no weighting is necessary even though some elements of our vectors have different variances to others. For entropy coding purposes, on each shell we shall be grouping equiprobable (based on ensemble *statistics*) groups of code-vectors in what we will term *sub-classes* or *super-classes* (we will be considering a specific approach based on the sub-class structure and then testing whether the assumptions or simplifications used to derive the super-class structure are valid by comparing the performance of the sub-class method with the super-class method). Thus to identify a particular quantised vector, the decoder would need to know three things: 1) On which shell does the quantised vector lie (i.e. its “size” or  $l_1$  norm); 2) In which sub-class/super-class is it to be found; 3) Finally the index within the sub-class/super-class of the quantised vector. The first two indices would then be entropy coded based on ensemble (training set) statistics and the third index can be binary coded with  $\log_2(k)$  bits where  $k$  is the number of equiprobable code-vectors in the sub-class/super-class.

For this type of approach to be successful, we require a number of things to be true. First the number of sub-classes/super-classes on any shell should be modest in size (to allow accurate probabilities to be estimated from a limited training set), secondly the most probable sub-classes/super-classes (on any shell) should contain fewer equiprobable code-vectors than the less probable ones (to reduce the number of bits required for the third index) and finally there should be a high concentration of quantised vectors lying on shells close to the origin (as the total number of bits increases approximately linearly with increased pyramid radius ( $l_1$  norm) [65]).

This method has the advantage over the “weighted” pyramid approach in the following ways:

- ◆ Different quantisers other than the cubic lattice used by Fisher can be used by the algorithms that we are going to develop.
- ◆ In Fisher’s “weighted” pyramid approach, he had to estimate values for his weighting parameters ( $w_i$ ). As we mentioned previously, different estimators can be used

to calculate these parameters yielding different results. We shall not need to estimate parameters in this way.

◆ Although we shall be using pyramidally shaped shells, the first approach we adopt, when grouping vectors into structures known as *sub-classes*, should yield exactly the *same overall results whether the shape of the shells is pyramidal or spherical*. In fact, as we shall later see, it is even more general than this (this, though would only be strictly true if coding of the radial parameter was with a first-order entropy code; we in fact shall be using a much more efficient context-based coding scheme for this task).

Furthermore, for the first main class structure developed for inter-band vectors, known as sub-classes, no assumption is made about the shape of the probability distribution function from which the components of a vector are drawn.

We shall show that the new ECPVQ method for coding 512×512 images significantly outperforms Fisher’s coding scheme in his 1996 paper [65] where he uses a combination of his “weighted” pyramid to code 63-D inter-band vectors from a DCT of the DC band of a 16-band uniform sub-band decomposition, and a standard (“non-weighted”) pyramid to code 32 or 64-D (intra-band) vectors separately for each of the 15 non-DC sub-bands.

### **5.3 Choice of quantiser for ECPVQ method**

We have decided to use the two simplest lattices in our work, namely the well known  $Z_n$  (also called the *cubic lattice*) and  $D_n$  lattices which were defined in chapter 3 where  $n$  is the dimension of the vector. Note that we include the origin as part of either lattice. It should be noted that the  $Z_n$  lattice is also equivalent to a uniform mid-tread scalar quantiser. It is important to note that our new coding method can in principle be extended to the use of any lattice such as the well-known  $\Lambda_n$  lattice. When coding on pyramidal shells, we find that the  $D_n$  lattice is equivalent to coding on the even-valued ( $I_1$

norm is even/zero) shells whereas the  $Z_n$  lattice is equivalent to coding on the odd and even valued shells ( $l_1$  norm is an integer  $\geq 0$ )

We have also explored the use of another quantiser which we have termed the  $Z_n/D_n$  *augmented lattice*, and is a union of the  $D_n$  lattice with the innermost shell of the  $Z_n$  lattice (i.e. union of  $D_n$  lattice with  $2n$  quantising points which are all permutations of  $(\pm 1, 0, \dots, 0)$ ). It should be noted that this augmented lattice is not a lattice in the strictest sense [34]. The motivation for using this quantiser is discussed later in this chapter and will become apparent for the 21-D inter-band vectors which we are mainly quantising. The definition for the quantisation performed by this new lattice is as follows:

Let  $x_n$  be a vector of dimension  $n$  whose elements are real. Let  $y_n$  and  $p_n$  (whose elements are integers) be formed from applying the  $Z_n$  and  $D_n$  lattice respectively to  $x_n$ , then  $w_n = y_n$  if  $\sum_n |y_n| \leq 1$  otherwise  $w_n = p_n$  where  $w_n$  results from quantising  $x_n$  by the  $Z_n/D_n$  lattice. This is equivalent to using one odd shell ( $l_1$  norm of one) and all the even shells (including the origin) for quantising.

#### **5.4. Choice of vectors/sub-band decomposition for new ECPVQ method**

In this chapter we have coded  $256 \times 256$  and  $512 \times 512$  sized monochrome images which contain 8 bits per pixel. We have decided to use a 2-D separable wavelet transform (see chap. 2) using either the 7/9 (see appendix A.1 for coefficients) or 10/18 tap (developed by Villasenor [49]) bi-orthogonal linear phase filters. These filters are the most widely used in recent coding applications. Symmetric extension [15] has been used to reduce boundary effects (as described in Chapter 2). In section 5.7.1.2 we shall also give some coding results for other filter pairs.

For  $256 \times 256$  images, a 3-level decomposition is used. The DC band is then of size  $32 \times 32$  and we have 3 vertically ( $V_k, k=1..3$ ), 3 horizontally ( $H_k, k=1..3$ ) and 3 diagonally ( $D_k, k=1..3$ ) oriented sub-bands of sizes  $32 \times 32$ ,  $64 \times 64$  and  $128 \times 128$  respectively. We form 21 dimensional vectors from respectively the vertically, horizontally and diagonally oriented sub-bands.

Then the vectors  $x_n$ ,  $n= 1..21$ , formed from vertically oriented sub-bands are given by the following: (see Figure 5.1)

For each  $(i, j) \in V_1$ ,  $x_n = \{(i, j), \{O(i, j)\}, \{L(i, j)\}\}$  where  $O\{(i, j)\}$  are the four offspring of  $(i, j)$  as defined in section 4.2.1, and  $L\{(i, j)\}$  are the 16 descendants of  $(i, j)$  in sub-band  $V_3$  and is also defined in section 4.2.1. The vectors based on each  $(i, j) \in H_1$  and each  $(i, j) \in D_1$  are similarly formed. Thus we have three  $(32 \times 32)$  arrays of 21-D quantised vectors to code, containing horizontally oriented sub-band coefficients, vertically oriented coefficients and diagonally oriented coefficients.

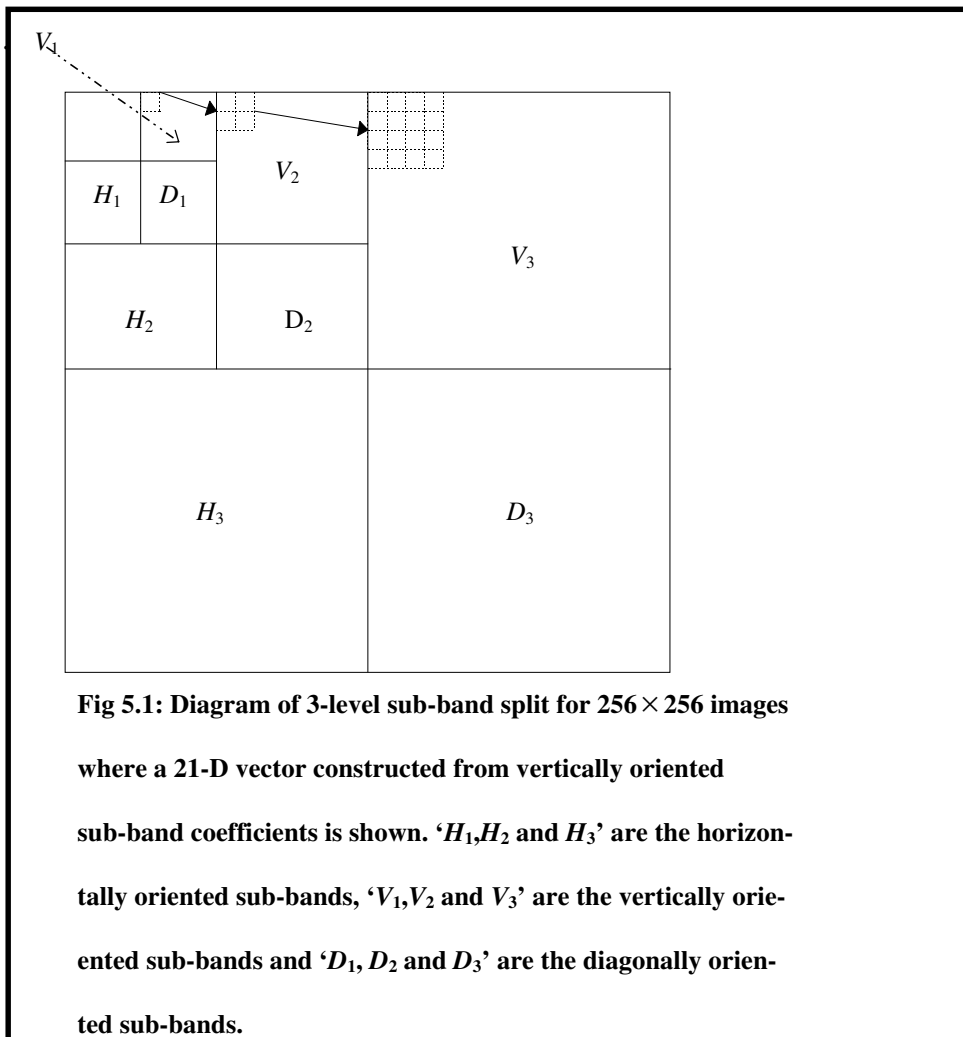
We have chosen 21 dimensions for the  $256 \times 256$  sized images (rather than higher dimensional vectors) to try to ensure that a high percentage of quantised vectors are concentrated on shells near the origin. Fisher [65,66] found that for memoryless sources, the performance of entropy coded pyramid vector quantisation cannot do better than entropy coded scalar quantisation when the integer lattice is used for quantisation. This is due, because for memoryless sources, the distribution of the radial parameter is quite different to that for sources with memory. Therefore although higher-dimensional vectors potentially can exploit greater correlation, as they are likely to lie on shells away from the origin (where it takes more bits to code them [65]), coding gains, if any, are likely to be modest (any loss of coding performance is likely to be due to inaccurate modelling of the histogram for the “radial” (or energy) parameter or inaccurate probability estimates for designing entropy codes for the outermost shells). Also choosing inter-band vectors allows an implicit global bit allocation strategy which coding intra-band vectors does not allow (e.g [60]). We are also exploiting correlations across scales (exploitation of the similarities among bands of the same orientation, see a detailed analysis by Shapiro [81]) as well as correlations between neighbouring coefficients within a particular sub-band.

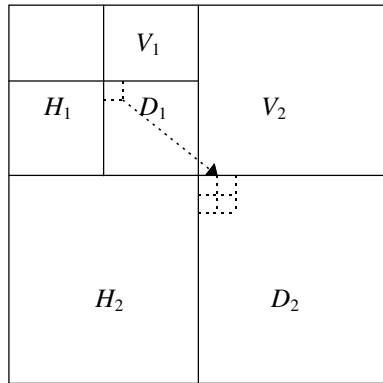
Finally for  $512 \times 512$  images, we use a 5-level decomposition with 5 vertically ( $V_k$ ,  $k=1..5$ ) 5 horizontally ( $H_k$ ,  $k=1..5$ ) and 5 diagonally ( $D_k$ ,  $k=1..5$ ) oriented sub-bands. The

size of the DC sub-band is  $16 \times 16$ . To exploit the algorithms developed for coding 21-D vectors, we group coefficients of levels 3 (coarse scale) to 5 (finest scale) ( $V_k, H_k$  and  $D_k, k=3..5$ ) in the same way as for  $256 \times 256$  images except we have 3 arrays of size  $(64 \times 64)$  of 21-D vectors instead. We code the A.C. sub-bands of the coarsest two levels using 5-D inter-band vectors ( $y_w, w=1..5$ ) defined by:

$$\text{For each } (i, j) \in V_1, y_w = \{(i, j), \{O(i, j)\}\} \quad (5.6)$$

The definitions for each  $(i, j) \in H_1$  and  $(i, j) \in D_1$  are similar (see Figure 5.2)





**Fig 5.2: Diagram of coarsest two levels of 5-level sub-band split for  $512 \times 512$  images. Also shows construction of a 5-D vector (diagonally oriented)**

Here we shall be coding 3 arrays of 5-D vectors of size  $(16 \times 16)$ . We use the 21-D vectors for the 3 finest scales because the higher dimensional vectors should be used to code the sub-bands which contribute the greatest number of bits to the overall bit-rate.

*In subsequent sections, when detailing the various stages of our ECPVQ scheme, we shall be assuming that  $256 \times 256$  size images are being coded unless explicitly stated.*

## 5.5. Class structures in the ECPVQ method

We shall be introducing new concepts such as classes, sub-classes, super-classes and super-super-classes. The main aim behind introducing them is to put equiprobable code-vectors, based on ensemble or training set statistics, on each pyramidal shell (indexed by  $K$  which is the  $l_1$  norm of the quantised vector), into groups. Then the index of each group can be entropy coded (provided the number of groups on each shell is small, accurate probabilities from training data can be estimated) and the index of the lattice point within that group can be coded using a fixed-length code (as all code-vectors within that group are approximately equiprobable).

The concept of **classes** is based on the coding of intra-band vectors (i.e. vectors whose coefficients have the same variance). No assumption need be made about the pdf

of each coefficient being laplacian for example. Now, each **class** can be defined as comprising vectors whose re-ordered element magnitudes are of the form  $(c_1, c_2, \dots, c_L)$  such that  $c_1 \geq c_2 \dots \geq c_{L-1} \geq c_L$  and  $c_1, c_2, \dots, c_L$  are all integers greater than or equal to zero (the sum of the components of the class vector is equal to  $K$ , the shell “radius”). Now we shall assume that all possible code-vectors which lie in any given class (which are all permutations of  $(\pm c_1, \pm c_2, \dots, \pm c_L)$ , where the  $\pm$  only applies to non-zero values) are equiprobable. This is a necessary assumption to make the problem tractable and is only strictly true if the components of a given class are *independent*. We shall explain further later.

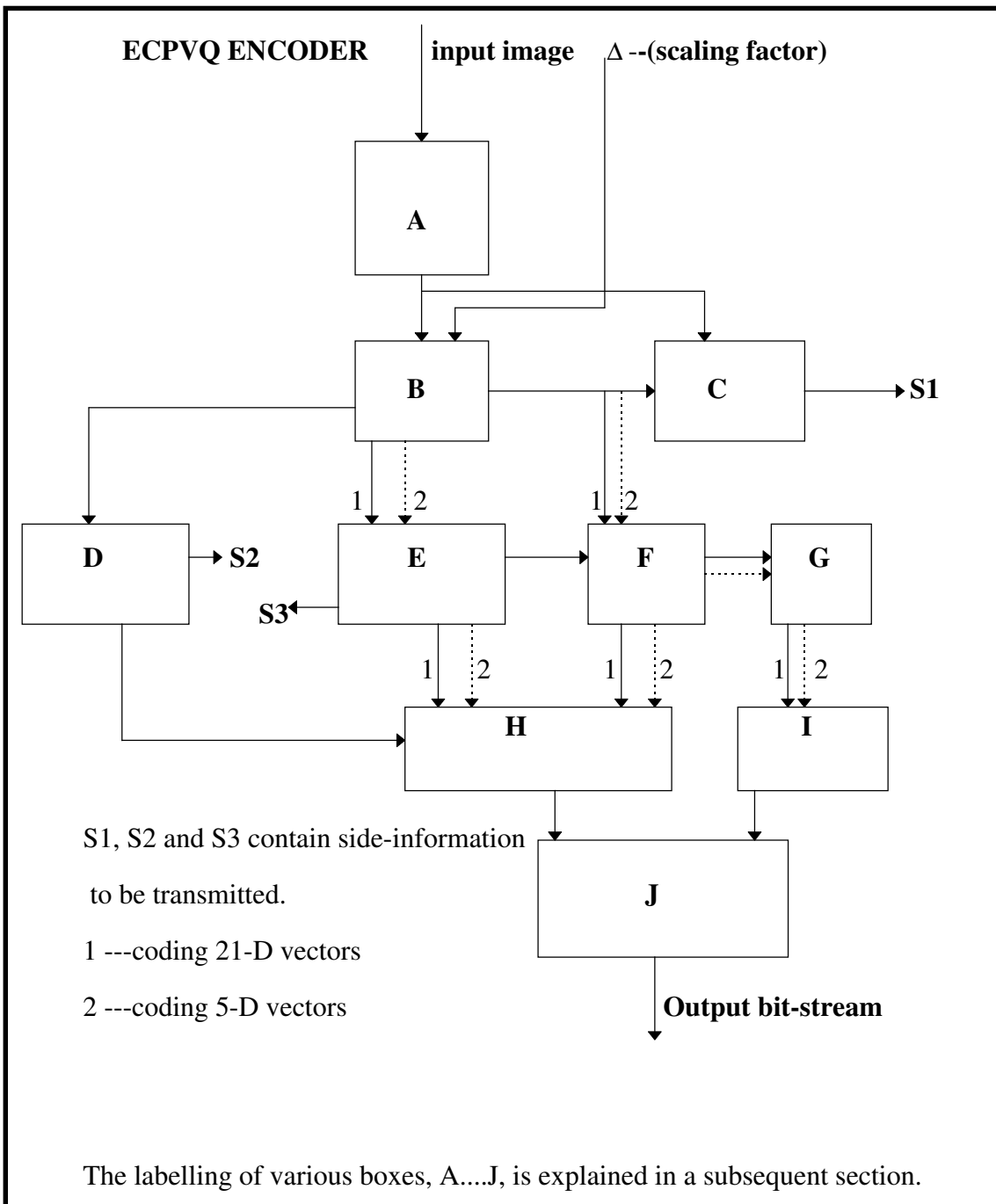
The concept of **sub-classes** is based on coding the 21-D inter-band vectors which we propose. Here each **class** contains a certain number of **sub-classes** to cater for the fact that the 21-D vectors are composed of 3 distinct groups of elements (i.e. coefficients of different variance). Thus, for example the **class**  $c=(1,0,0,0,0,0,\dots,0)$  would comprise 3 different **sub-classes**,  $(1,(0,0,0,0),(0,\dots,0))$ ,  $(0,(1,0,0,0),(0,0,\dots,0))$  and  $(0,(0,0,0,0),(1,0,0,\dots,0))$ . This is because the groups of elements  $c_1, c_2 \dots c_5$  and  $c_6 \dots c_{21}$  have distinct variances.

Because the total number of *sub-classes* increases as  $K$  increases (thus making it difficult to obtain accurate probability estimates from the training data to design entropy codes), we need to devise a strategy of merging sub-classes to form what we have termed **super-classes**. The concept of **super-classes** is based on an assumption that each of the 3 distinct groups of elements contains i.i.d. coefficients which form a multivariate Laplacian joint pdf. This idea is introduced to deal with the fact that the variance of the coefficients differs.

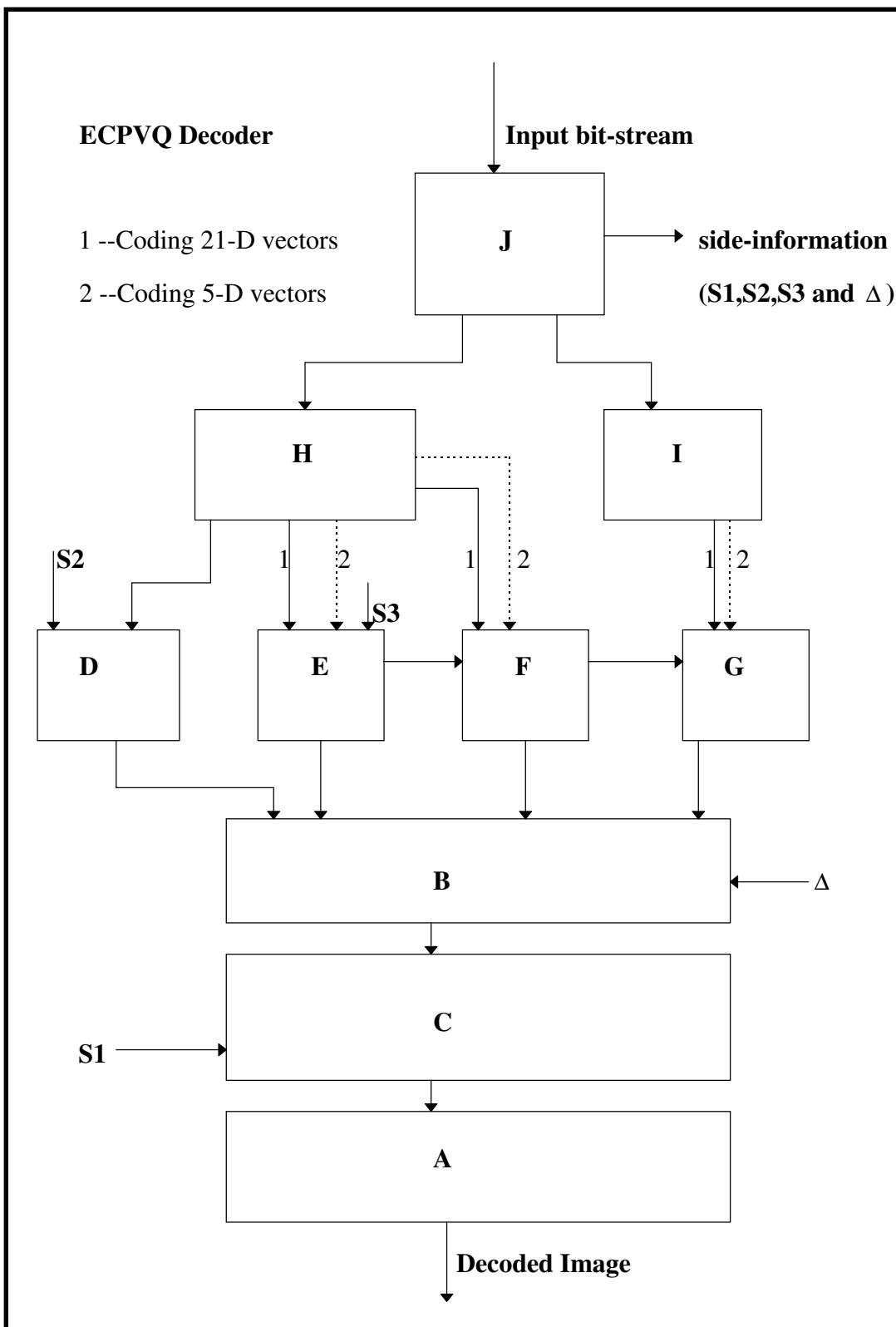
The assumption of an underlying Laplacian distribution is tested by comparing the performance of coding the test image  $256 \times 256$  **Lena** using **super-classes** to that using **sub-classes** for various values of  $K$  (the shell index).

To further reduce complexity and improve performance for medium to high bit-rate coding (approximately  $>0.5$  bits/pixel for both  $256 \times 256$  and  $512 \times 512$  images), it has been found that for large  $K$  (we have chosen  $K > 22$ ), the number of training vectors is not sufficient to produce accurate probability estimates for the **super-classes** on pyramidal shells. A heuristic method, which performs well, has been used to merge groups of mainly highly improbable **super-classes** into what we have called **super-super-classes**. Furthermore, we will show that as  $K$  increases the number of vectors lying upon the shell indexed by  $K$  decreases, thus further justifying a reduction in complexity for the latter/outer shells (i.e. a reduction in the number of probabilities that need to be stored at the encoder/decoder). It should be observed that merging all the **super-classes** on a shell to create one single **super-super-class** is equivalent to Fisher's "ordinary" (with weights  $w_i$  equal to unity) pyramid coding scheme where all the code-vectors on a pyramidal shell with  $l_1$  norm of  $K$  can equiprobable.

The complete block diagrams for the ECPVQ encoder/decoder algorithms are given in Figures 5.3-5.4. These diagrams should be referred back to when the sections in the chapter to which certain boxes (or blocks) refer have been read.



**Fig 5.3: Block Diagram of ECPVQ Encoder**



**Fig 5.4: ECPVQ Decoder block diagram**

## **Nomenclature used in block diagrams 5.3 and 5.4.**

### **A Transform Phase**

**Encoder** - $n$ -level 2-D separable wavelet transform (3-levels for  $256 \times 256$  images or 5-levels for  $512 \times 512$  images) using  $7/9$  tap filters (see A.1 for definition). The input image is an 8 bits per pixel grayscale image.

**Decoder** -  $n$ -level inverse 2-D separable wavelet transform (section 5.4). Reconstructed image pixel values are rounded to the nearest integer and any values less than zero are set to zero and any values greater than 255 are set to 255. This is because the range of pixel values for the original image is between zero and 255.

### **B Quantisation stage**

**Encoder** - For  $256 \times 256$  images, uniform mid-tread scalar quantisation of the  $32 \times 32$  sized DC band (with step size  $\Delta$ ) and quantisation of A.C. sub-bands by the  $Z_{21}/D_{21}$  lattice (using scaling factor  $\Delta$ ). For  $512 \times 512$  images, uniform quantisation of  $16 \times 16$  sized DC band (step size  $\Delta$ ), quantisation of coarsest 6 A.C. sub-bands by  $Z_5/D_5$  lattice, and the remaining 9 A.C. sub-bands by the  $Z_{21}/D_{21}$  lattice. **Decoder** - Construction of  $256 \times 256$  or  $512 \times 512$  matrix knowing quantised values in each of 10 or 16 sub-bands respectively, given value of  $\Delta$  from side-information. (sections 5.3-5.4)

### **C Minimisation of error stage**

**Encoder** -Transmission of 'mu'\* values as side-information (**S1**) to decoder for finest 6 A.C. sub-bands. The variance of the coarsest 3 or 9 sub-bands is also transmitted, depending on the image size being  $256 \times 256$  or  $512 \times 512$ . This is so the centroids can be calculated at the decoder.

**Decoder** -For the finest 6 sub-bands, the 'mu' values (**S1**), together with the value for  $\Delta$ , are used to obtain the reproduction vectors. For the remaining A.C. sub-bands, the values of  $\epsilon(\Delta)$  are calculated based on the transmitted variances and sub-bands values are obtained. (\*see section 5.6.2)

## **D DC band coding**

**Encoder** -Using DPCM method to code DC sub-band of size  $32 \times 32$  or  $16 \times 16$ . Transmission of side-information (**S2**) to decoder.

**Decoder** -Reconstructing DC sub-band from appropriate symbol stream and side-information **S2**. (see section 5.6.1)

## **E Radial parameter coding**

**Encoder** --Transmission of radial parameter information for 21-D vectors (1.), where the image size is  $256 \times 256$  or both 21-D and 5-D vectors (1. and 2.), where the image size is  $512 \times 512$ . Side-information **S3** is transmitted

**Decoder** --Reconstruction of this radial parameter information. (see section 5.6.3.)

## **F Position (1.) information coding**

**Encoder** ---Transmission of index of either sub-class/super-class or super-super-class in which each non-zero 21-D (or each 21-D and 5-D vector) vector lies.

**Decoder** --Reconstruction of 21-D (or 21-D and 5-D) vectors knowing this index. (see sections 5.5.2 -5.5.4..)

## **G Position (2.) information coding**

**Encoder** ---Transmission of index for specific code-vector within a given sub/super or super-super-class for each non-zero vector.

**Decoder** --Given this index, the index from **F** and the index from **E**, reconstruction of each non-zero 21-D (or 21-D and 5-D) vector. (see sections 5.5.2-5.5.4.)

## **H Arithmetic coder**

**Encoder** ---Arithmetic Encoder to produce binary bit-stream. Encoder used to entropy code symbol stream from **D**, **E** and **F**.

**Decoder** --Arithmetic Decoder to produce symbol stream from binary bit-stream.(see section 5.7)

## **I Huffman coder**

**Encoder** ----Huffman encoder to produce binary codes (for symbol stream from **G**) for  $N$  equiprobable symbols (see section 5.7 and A.4.2).

**Decoder** ----Huffman decoder which decodes indices from binary codes.

## **J De-/Multiplexer**

**Encoder** ---Multiplexer which adds side-information to be transmitted to a header which precedes the transmitted binary bit-stream which is a concatenation of bit-streams produced by **H** and **I**.

**Decoder** ---De-multiplexer which strips out the side-information.

### 5.5.1. Introduction of the *class* structure on pyramidal shells

The number of integer co-ordinate vectors on pyramid  $S(L,K)$  is denoted  $N(L,K)$  and derived by Fisher [66] as follows where  $L$  is the dimensionality of a vector  $\mathbf{x}$  and  $K$  is the  $l_1$  norm: All of the  $N(L,K)$  possible vectors on  $S(L,K)$  are of the form:

$$x = \begin{cases} i \\ x_2 \\ \vdots \\ x_L \end{cases}, \text{ for } i = 0, \pm 1, \dots, \pm K \quad (5.7)$$

The partition of  $\mathbf{x}$  corresponding to  $(x_2, \dots, x_L)$  has  $N(L-1, K-|i|)$  possible integer co-ordinate vectors (code-vectors). This is because the dimensionality of  $(x_2, \dots, x_L)$  is  $L-1$  and its  $l_1$  norm is  $K-|i|$ . This implies that :

$$\begin{aligned} N(L, K) &= 2(N(L-1, K-1) + N(L-1, K-2) + \dots + N(L-1, 1) + 1) + N(L-1, K) \\ N(L, K-1) &= 2(N(L-1, K-2) + N(L-1, K-3) + \dots + N(L-1, 1) + 1) \\ &+ N(L-1, K-1) \end{aligned} \quad (5.8)$$

Combining the two, one gets:

$$\begin{aligned} N(L, K) &= N(L, K-1) + N(L-1, K-1) + N(L-1, K) \\ N(L, 1) &= 2L \\ N(1, K) &= 2 \end{aligned} \quad (5.9)$$

Now we want to find a method to obtain the number of classes on a particular shell (given by  $K$ ). Each **class** comprises vectors whose re-ordered element magnitudes are of the form  $(c_1, c_2, \dots, c_L)$  where  $c_1 \geq c_2 \geq \dots \geq c_{L-1} \geq c_L$  and  $c_1 \dots c_L$  are all integers  $\geq 0$ . Also  $\sum_{i=1}^L c_i = K$ .

We can now assume that the code-vectors which lie in each class are all equally probable, because all the possible code-vectors in a particular class are all permutations of  $(\pm c_1, \pm c_2, \dots, \pm c_L)$ . This assumption is made because accurate probability estimates for all of the permutations are often not possible due to the huge disparity between the number of permutations and the number of training data vectors available. Where accurate probability estimates can be made for each of the permutations in a class, we have found that the equal probability assumption results in little loss of performance when the entropy codes being designed are based on ensemble statistics. Of course, based on the

image to be coded, the assumption of equal probability would be inaccurate (because many permutations would have zero probability) but it is not practical to design entropy codes on a ‘per image’ basis as too much side-information would need to be transmitted to the decoder. It should be noted that this equal probability assumption is more *specific* or less general than assuming all  $N(L,K)$  code-vectors on  $K$  are equally probable (as is the case when assuming each element of an input vector of dimension  $L$  is independent, has the same variance and comes from a laplacian pdf).

This can be accomplished by what is known in statistics as finding the number of partitions of an integer  $K$  into  $j$  parts [82] (non-zero elements such that the sum of the  $j$  elements is equal to  $K$ ).

### Example

Let  $K=4$  and  $L=4$ . It can be seen that there are 5 **classes** on this shell and they are given by:  $\{(4,0,0,0),(3,1,0,0),(2,2,0,0),(2,1,1,0),(1,1,1,1)\}$ . In partition theory, this is equivalent to finding the partitions of an integer 4 into 1,2,3 and 4 parts. (1,1,1,1) is the integer 4 partitioned into 4 parts. (2,1,1) is the integer 4 partitioned into 3 parts (we append a ‘0’ to make a 4-D vector) and (3,1) and (2,2) is the integer 4 partitioned into 2 parts (we append two zeros to end of (3,1) as before to make a 4-D vector).

Thus, in general to determine the number of **classes** lying on a shell indexed by  $K$ , all we need do is find the number of partitions of integer  $K$  into  $M$  or fewer parts,  $(M,M-1, ..1)$  where  $M$  is the minimum of  $L$  or  $K$ .

So,

$$N^*(L, K) = \sum_{j=1}^{\min(L,K)} Pa(K, j) \quad (5.10)$$

where  $Pa(K,j)$  is the number of partitions of integer  $K$  into  $j$  parts. It has been shown [82] that  $Pa(K,j)$  is given by the following recursive equation:

$$Pa(K, j) = Pa(K - 1, j - 1) + Pa(K - j, j) \quad (5.11)$$

And  $Pa(K,j)$  is also defined as [82],

$$Pa(K, j) = \sum_{i=1}^j Pa(K-j,i) \tag{5.12}$$

$$Pa(K,j) = 1 \text{ if } K=j$$

$$Pa(K,j) = 1 \text{ if } j=1$$

$$Pa(K,j) = 0 \text{ if } j>K$$

where  $Pa(K,j)$  is the number of partitions of integer  $K$  into  $j$  parts.

**Table 5.5. Comparison between number of code-vectors and number of classes on a pyramidal shell indexed by  $K$  for two vector dimensions,  $L=16$  and  $L=4$ .**

$K$	$L=16$		$L=4$	
	$N(16,K)$	$N^*(16,K)$	$N(4,K)$	$N^*(4,K)$
1	32	1	8	1
2	512	2	32	2
3	5472	3	88	3
4	44032	5	192	5
5	-----	7	360	6
6	-----	11	608	9
7	-----	15	952	11
8	-----	22	1408	15
9	-----	30	1992	18

Table 5.5 shows that for a certain shell indexed by  $K$ , one can arbitrarily increase the vector dimensionality without significantly increasing the number of classes on that particular shell.

The method whereby a quantised vector (lattice point) can be represented by:

- 1) Index of shell upon which it lies (i.e.  $K$ )
- 2) Index of the class in which it lies
- 3) Index of permutation within the class (i.e. ‘the exact ordering information’) given knowledge of class.
- 4) Transmission of the sign bits for the non-zero elements

is appropriate where all the elements in the input vector have the same statistics (i.e. are of the same variance).

#### **Example of class-based method and comparison with standard approach by Fisher[65]**

Let  $L=4$  and  $K=2$ . Let the quantised vector be given by  $(-1,0,1,0)$  where each coefficient has the same variance. The class-based coding scheme is as follows:

- ◆ Index of shell is transmitted by entropy coding, i.e.  $K=2$
- ◆ The 2 classes for  $K=2$  are  $\{(2,0,0,0),(1,1,0,0)\}$ . The index for the classes ranges from  $[0,1]$ . The class in which vector lies is  $(1,1,0,0)$ . This index is 1 and can be entropy coded. Accurate probabilities from a training phase can be obtained because of the relatively few classes on this shell (2 in this case)
- ◆ The index of the permutation  $(1,0,1,0)$  is determined (see A.9) (the range is  $0\dots5$ ) and transmitted using a “fixed-length” code. Each of the 6 permutations is assumed to be represented by a code-word of length  $\log_2(6)$  bits.
- ◆ The sign bits  $[-1,+1]$  are then transmitted.

Fisher’s method [65] for intra-band vectors would transmit  $K$  and assume that all possible code-vectors on  $K$  are equally probable and assign a code-word of length  $\log_2(N(L,K))$  bits to each. Of course, this makes the assumption that each element of an input vector is independent and has a laplacian pdf. The class-based algorithm can test the validity of the laplacian pdf assumption.

### 5.5.1.1. Enumeration encoding/decoding algorithms for indexing of classes

We will now briefly discuss methods of enumeration encoding/decoding algorithms for indexing a particular class on a given shell indexed by  $K$ . These are based on partitioning an integer  $K$  into 1 part, 2 parts,..., $M$  parts where  $M$  is the minimum of  $L$  (the dimensionality) and  $K$ . Thus the range of indices is partitioned according to the number of significant digits in the input  $L$ -dimensional vector.

#### Example

Let  $L=5$  and  $K=4$ . Thus the set of all classes is ordered as follows:

$\{(4,0,0,0,0),(3,1,0,0,0),(2,2,0,0,0),(2,1,1,0,0),(1,1,1,1,1)\}$  and this is equivalent to ordering classes as partitioning 4 into 1 part, 4 into 2 parts, 4 into 3 parts and 4 into 4 parts and appending appropriate number of zero elements to make up  $L$ -dimensional vectors. Thus, to find the index of class  $(2,2,0,0,0)$ , one would need to find the index of partition  $(2,2)$  out of all partitions of 4 into 2 parts and add to this the number of partitions of integer 4 into 1 part. Others are calculated similarly. The encoding and decoding enumeration algorithms for the partition of an integer  $K$  into  $j$  parts are given in appendix A.7.1.

### 5.5.2 Introduction of the *sub-class* structure for hierarchical (or inter-band) vectors

We now consider the question of coding the 21-D inter-band vectors described in 5.4, where the vector is of the format  $(1,4,16)$  (i.e. one coefficient from one coarse subband, 4 coefficients from another finer subband and the other 16 coefficients from yet another even finer subband).

We need to introduce a new concept called **sub-classes** which, on any particular shell indexed by  $K$ , contains code-vectors which are assumed to be equally probable. We

need to think of the 21-D vector as containing three distinct groups of elements (of sizes 1, 4 and 16 respectively).

We wish to allow any ordering within each group but **not to** allow an interchange between the groups. Thus we must find a method of sub-dividing each class into a number of **sub-classes**.

### Example

$L=21$  (as we are considering 21-D vectors!) and let  $K=2$ . We have found that there are 2 **classes** on this shell, namely  $(2,0,0,\dots,0)$  and  $(1,1,0,0,\dots,0)$ .

For the class  $(2,0,0,\dots,0)$  there are 3 **sub-classes** given by  $(2,0,0,0,\dots,0)$ ,  $(0,2,0,0,0,0,\dots)$  and  $(0,0,0,0,0,2,0,0,\dots,0)$ . The first sub-class has the integer 2, within the first group of elements, the second has it within the second group of elements and the third has the integer 2 within the third group of elements.

For the class  $(1,1,0,0,\dots,0)$  there are 5 **sub-classes** given by

- $(1,(1,0,0,0),(0,\dots,0))$
- $(1,(0,0,0,0),(1,0,0,\dots,0))$
- $(0,(1,1,0,0),(0,0,\dots,0))$
- $(0,(1,0,0,0),(1,0,\dots,0))$
- $(0,(0,0,0,0),(1,1,0,0,\dots,0))$ .

So on the shell indexed by  $K=2$  there are a total of 8 **sub-classes**.

For a given input quantised 21-D vector, we need to determine: 1) The value of  $K$  which can be entropy coded; 2) The index of the **class** in which the quantised vector lies; 3) The index of the **sub-class** in which the quantised vector lies given the **class** index.

We can then produce a single index (for the index of the sub-class out of all sub-classes on a particular shell indexed by  $K$ ) from the class and sub-class indices (i.e. (2) and (3)).

When  $K=2$  this single index would be in the range  $[0..7]$ .

Finally, 4) We would need to transmit the permutation information for the group of 4 elements and the group of 16 elements along with the signs of the non-zero elements. If we let  $p_4$  be the total number of permutations of the group of 4 elements, for a particular 21-D quantised vector, and  $p_{16}$  be the number of permutations of the group of 16 elements, then the total number of permutations is given by,  $tot = p_4 p_{16} 2^s$ , where  $s$  is the number of significant (non-zero) elements in the 21-D quantised vector. Based on our arguments on **classes**, we assume that each permutation can be represented using  $\log_2(tot)$  bits. The permutation index which is transmitted is obtained as follows (we assume that the  $s$  sign bits are transmitted separately).

#### **Method for obtaining ordering information (permutation index)**

1.) An index  $a$  which ranges from  $[0, p_4 - 1]$  is obtained using the permutation algorithm (A.9) for the 4 element group in the quantised vector being coded.

2.) Another index  $b$  is obtained for the ordering of the 16 digits in a similar fashion. It ranges from  $[0, p_{16} - 1]$

The indices are combined to form

$$p_{21} = p_{16}a + b \quad (5.13)$$

whose range is from  $[0, p_4 p_{16} - 1]$ . This is known as a **product enumeration index**

The decoding is straightforward and a similar decoding algorithm is given in A.7.2.2

#### **5.5.2.1 Finding index of sub-class given class and $K$**

Here we will briefly describe the problem of determining the number of sub-classes in a given class (and given  $K$ , the index of the shell) and an enumeration algorithm to determine the index of a particular sub-class again knowing the class.

This requires consideration of what is known in statistics as the **limited repetition problem [82]**:

The solution is obtained by using generating functions [82] :

**Example**

If we want to make a selection of size  $P$  from  $N$  distinct objects given a vector containing numbers of repetitions of each object  $(r_1 \dots r_N)$  where  $1 \leq r_i \leq P, i = 1..N$

**Solution** : Using generation functions, it can be shown [82] that the answer is the magnitude of the coefficient of  $x^P$  in  $(1+x+x^2+ \dots+ x^{r_1}) \dots\dots\dots(1+x+x^2+ \dots+ x^{r_N})$

**Algorithm for counting the number of sub-classes in a particular class**

We illustrate the algorithm by means of an example. We want to find the number of **sub-classes** in the class  $c = (2,1,1,0,0,0,\dots,0)$  with  $K=4$ . The algorithm is as follows:

- ◆ Find the unique elements of the particular class. In our case there are 3 unique elements, namely  $(2,1,0)$ . In general they are  $\{x_1, x_2, \dots, x_{nm}\}$  where  $nm$  is the number of unique elements.
- ◆ Remove one copy of each unique element in turn from  $c$  and find the number of selections, from the unique elements (of length  $N$ ) in the remaining 20, of size  $P=4$  (the second group of elements in 21-D vector is of size 4) given a vector of length  $N$  containing the number of repetitions of each such unique element. All these  $nm$  selections are then summed to give the number of sub-classes in that class.
- ◆ Let  $S(N,P,R)$  be the number of selections of size  $P$  from  $N$  distinct objects given the vector  $R$  of length  $N$  containing the number of repetitions of each object. Then for our example, the number of sub-classes is given by:

$$S(2,4,\{2,18\}) + S(3,4,\{1,1,18\}) + S(3,4,\{1,2,17\}) = 3+4+6=13$$

### Encoding enumeration algorithm

We illustrate the encoding enumeration algorithm to determine the index of a specific sub-class by means of an example.

**Problem:** To determine the index,  $i_s$ , of sub-class  $s = (0,(1,1,0,0),(2,0,0,\dots,0))$  where  $K=4$ .

**Solution:** From the previous example we know that  $i_s \in [0..12]$  as there are 13 sub-classes within the class  $(2,1,1,0,0,0,\dots,0)$ . We also know that if  $s_1=2$ ,  $i_s \in [0..2]$ , if  $s_1=1$ ,  $i_s \in [3..6]$  and if  $s_1=0$ ,  $i_s \in [7..12]$ . Thus we initially partition the range of indices according to the magnitude of the first element  $s_1$ . Let this range be denoted by  $[m_1\dots m_2]$

- The overall index,  $i_s = i_1 + i_{20}$  where  $i_1 = m_1$  and  $i_{20} \in [0..m_2-m_1]$ .
- For our example  $i_1 = 7$ . To determine  $i_{20}$ , we need to find the index of the selection  $\{1,1,0,0\}$  (or elements  $s_2\dots s_5$ ) from the unique elements  $\{2,1,0\}$  contained within elements  $s_2\dots s_{21}$  with the repetition vector,  $r = \{1,2,17\}$  (i.e. the element 2 is repeated once, the element 1 is repeated twice and the element 0 is repeated 17 times). This can be re-formulated [82] as a selection of  $P=4$  identical balls from  $N=3$  distinct boxes with restrictions on the maximum number of balls in each box. The variable  $P$  denotes the number of elements from  $s_2\dots s_5$  and  $N$  is the number of unique elements within  $s_2\dots s_{21}$ .
- In our example, our selection consists of zero balls in the first box, two balls in the second box and two balls in the third box. This is because the first box represents the element 2, the second box the element 1 and the third box the element 0 and our selection is  $\{1,1,0,0\}$  or ball in respectively box 2, box 2, box 3 and box 3. The repetition vector,  $r = \{1,2,4\}$  as each  $r_i \leq P$ ,  $i = 1..N$ .
- We can determine, using generating functions [82], the number of possible selections if (a) there is only *one ball* in the first box; (b) there are *no balls* in the first box. We find that if condition (a) is true that  $i_{20} \in [0..2]$ , otherwise if condition (b) is correct, then  $i_{20} \in [3..5]$  (i.e. the valid range for  $i_{20}$  is split according to the number of possible balls in the first box)
- This range is then sub-divided according to the number of balls in each subsequent box until we reach the final box.
- For our example, after the first box  $i_{20} \in [3..5]$ , because in our selection there are no balls in this first box. There are three possibilities for the number of balls in the second box: (a) 2 balls; (b) 1 ball; (c) no balls. The value of  $i_{20}$  would respectively be 3, 4 or 5. Therefore in our example, the index  $i_s = 7 + 3 = 10$

### Decoding enumeration algorithm

The decoding algorithm involves the following steps:

- 1) Determining value of element  $s_1$  from the sub-range that the index  $i_s$  lies in. This determines  $i_1$  and so  $i_{20} = i_s - i_1$ .
- 2) Based on the nomenclature of the encoding algorithm, determining the number of balls in the first box. From that, the number of balls in subsequent boxes can be determined and the elements  $s_2\dots s_5$  found. It is trivial then to obtain the elements  $s_6\dots s_{21}$ .

**Table 5.6: Total number of classes, sub-classes and super-classes on each pyramidal shell  $K$  for the  $Z_{21}/D_{21}$  augmented lattice**

$K$	<i>classes</i>	<i>sub-classes</i>	<i>super-classes</i>
1	1	3	3
2	2	8	6
4	5	38	15
6	11	135	28
8	22	404	45
10	42	1073	66
12	77	2606	91
14	135	5903	120
16	231	12641	153
18	385	25832	190
20	637	50734	231
22	1001	96283	276
24	1571	177318	325
26	2424	317978	378

Table 5.6. shows the total number of classes, sub-classes and super-classes (see next section) for various values of  $K$ . We see that the number of sub-classes rapidly increases as  $K$  increases. As the probabilities for the entropy codes, needed to encode the index of a sub-class, on a particular shell, are designed based on a set of training set images, for large  $K$ , a large number of the histogram bins would be zero. Thus, as the reader will see later, we confine ourselves to using the sub-class structure, for  $0 < K \leq 12$  and find an alternative method to code the position information on shells where  $K > 12$ .

**Table 5.7: Shows number of equally probable code-vectors in each sub-class for  $K=2$ .**

Sub-class	number of permutations (including sign info)
(2,0,0,0,0,0,0,.....)	2
(0,2,0,0,0,0,0,.....)	8
(0,0,0,0,0,2,0,.....)	32
(1,1,0,0,0,0,0,.....)	16
(1,0,0,0,0,1,0,.....)	64
(0,1,1,0,0,0,0,.....)	24
(0,1,0,0,0,1,0,.....)	256
(0,0,0,0,0,1,1,0,..)	480

Table 5.7 shows the number of equiprobable code-vectors contained within each sub-class for a (1,4,16)-D vector where  $K=2$ . This table shows that on the shell indexed by  $K=2$ , if the sub-class (1,(1,0,0,0),(0,0,0.....,0)) turns out to be very probable, the number of bits needed to transmit the ordering information are relatively few, i.e.  $\log_2(16)$  bits. This may be the case because the ‘1’ element in the first group is likely to have the highest variance (as it is from a low frequency sub-band) and the ‘1’ element in the second group of 4 elements is likely to have a higher variance than the 16 zero elements in the third group of elements for the same reason.

### **5.5.3 Introduction of the *super-class* structure (equivalent to merging of *sub-classes*)**

We have decided to implement a method which effectively merges groups of **sub-classes** on each pyramidal shell indexed by  $K$ . One of the reasons for doing this is that the total number of **sub-classes** increases rapidly as the shell index  $K$  increases (e.g. for

$K=12$ , the number of sub-classes is 2606). This means that accurate probabilities for each of the sub-classes on a shell where  $K$  is greater than 12 (see table 5.6) cannot be obtained from the training phase and thus good entropy codes cannot be designed. By merging groups of **sub-classes** into so called **super-classes**, where the number of **super-classes** on shells where  $K>12$  is considerably less than the number of **sub-classes**, more accurate probability estimates can be obtained and performance can actually improve with merging for this reason (as we will see in section 5.7.1.1).

### 5.5.3.1. Definition of *super-classes* for 21-D vectors

Let  $K$  be the  $l_1$  norm of the input quantised vector. This is the index of the shell upon which the quantised vector lies. Let the quantised vector be denoted by  $q_n$  where  $n=1..21$ . Then define three different radii,  $r_1, r_4$  and  $r_{16}$  such that

$$\begin{aligned} r_1 &= |q_1| \\ r_4 &= \sum_{n=2}^{n=5} |q_n| \\ r_{16} &= \sum_{n=6}^{n=21} |q_n| \end{aligned} \tag{5.14}$$

These three radii correspond to the  $l_1$  norm of the distinct groups of elements in our 21-D vector. We define the set of all super-classes on a shell indexed by  $K$  to be all combinations of  $(r_1, r_4, r_{16})$  where  $0 \leq r_1, r_4, r_{16} \leq K$  such that  $r_1 + r_4 + r_{16} = K$ .

#### Example

Let  $K=2$ . The 6 super-classes on the shell indexed by  $K=2$  are given by  $\{(2,0,0), (0,2,0), (0,0,2), (1,1,0), (1,0,1), (0,1,1)\}$ . We assume that all the possible code-vectors in each super-class are equally probable. So for example, for the super-class  $(0,2,0)$  the two sub-classes  $(0, (2,0,0,0), (0,0, \dots, 0,0))$  and  $(0, (1,1,0,0), (0,0, \dots, 0,0))$  are merged. The number of code-vectors in the former sub-class is 8 ( $4 \times 2^1$ ) and in the latter

is 24 ( $6 \times 2^2$ ). So the total number of code-vectors in the super-class (0,2,0) is 32 and we assume them to be equally probable.

*This is equivalent to assuming that the coefficients in the 4-element group are independent, have the same variance and come from a laplacian pdf.* For other super-classes, this concept is extended for the 16-element group also.

It can be easily shown that the total number of super-classes on a particular shell indexed by  $K$  (the  $l_1$  norm of the quantised vector) is given by  $(K+1)(K+2)/2$ .

Thus, even for large values of  $K$  the total number of super-classes is relatively modest enabling accurate probability estimates to be determined by a training phase. However in section 5.5.4. we extend the method further and employ an efficient heuristic method to merge groups of super-classes to form super-super-classes for large values of  $K$ .

It should be noted that the number of super-classes on any shell, indexed by  $K$ , is modest only because they are only 3 distinct groups of elements in the 21-D vector. For  $d$  distinct groups of elements the number of super-classes is given by  $(K+1)(K+2)...(K+d-1)/(d-1)$ . So for example if  $d=63$ , (as is the case if we are coding 63-D vectors formed from the 63 non-dc. coefficients of an  $8 \times 8$  DCT) and  $K=1$ , then the number of super-classes would be  $9.2 \times 10^{86}$ . So this approach is only really appropriate for the wavelet based 21-D vectors which we are considering.

### **5.5.3.2. Enumeration encoding/decoding algorithms for determining the index of a super-class**

The encoding/decoding algorithms for determining the index of a specific super-class where the index is in the range  $[0, (K+1)(K+2)/2-1]$  are given in pseudo-code with explanations in appendix A.7.2.1.

### **5.5.3.3. Enumeration encoding/decoding algorithms to determine index of specific code-vector in a particular *super-class***

One also needs enumeration algorithms to encode/decode the ordering information (including sign info.) within a specific **super-class**: The encoding/decoding algorithms are given below: This index is in the range  $0 \dots N(4,r_4)N(16,r_{16})-1$  if  $r_1=0$  or  $0 \dots 2N(4,r_4)N(16,r_{16})-1$ , otherwise (i.e. if  $r_1$  is non-zero, the sign of  $r_1$  would need to be included). Here  $N(L,K)$  is the number of integer code-vectors of dimension  $L$  on pyramidal shell indexed by  $K$  and  $r_1, r_4$  and  $r_{16}$  are the  $l_1$  norms respectively of the 3 groups of distinct elements in the 21-D vector (i.e. 1 element, 4 elements and 16 elements). The encoding and decoding algorithms are given in full in appendix A.7.2.2.

### **5.5.3.4. Methods for determining index of code-vector of dimension $L$ on a pyramidal shell indexed by $K$**

There are various methods which can be used to find the index of a codeword on a pyramidal shell given the vector,  $L$  and  $K$ . One method is described by Fisher [65] (which utilises the recursive definition for  $N(L,K)$  given in 5.9.) and another alternative method is proposed here which has some similarities with the product enumeration technique proposed by [78] to separate the sign information of the significant elements from the encoding index (storing them in the least significant bits of the encoding index). Meng [78] found that in a noisy channel environment, using a fixed-rate scheme, his enumeration methods out-performed Fisher's techniques. He surmised that this was primarily due to separating sign bits from the rest of the codeword used to transmit the encoding index. It would be interesting to compare performance of Fisher's enumeration methods vs. the new ones proposed here, for the entropy-coded scheme suggested in this chapter, for a noisy channel. The encoding/decoding algorithms are based firstly on an alternative non-recursive definition for  $N(L,K)$  and is as follows:

It has been shown that (e.g [77,78])

$$N(L, K) = \sum_{s=1}^m \binom{L}{s} \binom{K-1}{s-1} 2^s \quad (5.15)$$

where  $N(L, K)$  is the number of possible integer co-ordinate code-words on a pyramid of radius  $K$  and dimension  $L$ . They are also based on the enumeration algorithms developed for the partitioning of an integer  $K$  into  $j$  parts given in 5.5.1.1. and used for the enumeration algorithms developed for the **classes** groupings we developed in 5.5.1. The full pseudo-code for these enumeration algorithms is given in A.8.

#### **5.5.4 Introduction of the *super-super-class* (equivalent to merging of *super-classes*) structure**

We have found that for the image data coded in this chapter (see 5.4), to improve coding performance, for shells with  $K > 22$ , we need to merge groups of **super-classes** together to form **super-super-classes** so that from a limited training set, accurate probability estimates for each of the new super-super-classes, whose indices are entropy coded, can be obtained. It is also the case that at low bit-rates, the number of quantised vectors lying on the outer pyramidal shells (for  $K > 22$ ), is quite small. This means that very efficient position information coding for these quantised vectors is not necessary (as the effect to the overall bit-rate is tiny as we shall see later) and ideally we would desire a merging strategy such that the number of **super-super-classes** diminishes as  $K$  increases.

One method (given in Appendix A.2) for each shell (indexed by  $K$ ) based on training set data, is to select a pair of super-classes to be merged such that the increase in entropy as a consequence of this is minimised and continue doing this until the desired number of super-super-classes is reached. It should be noted that this is not optimal within the training set. Lei [43] devised a method based on the LBG technique to merge context states (in a higher-order entropy code) such that for a given number of merged states, the increase in entropy is minimised -a local minimum is found and in theory a similar type of approach could be adopted for merging our super-classes. However,

optimising performance within a training set does not allow one to assume optimality for data which is outside the training set.

So, we have devised an heuristic approach which actually improves performance after merging. This is because over 80% of the histogram bins, for the super-classes on the outer shells, based on training data are empty. This is due to the fact that a lot of super-classes on the outer shells ( $K > 22$ ) are highly improbable and a vast number of training vectors would be required to determine accurate probabilities.

#### 5.5.4.1. Heuristic approach to merging *Super-classes*

We shall first give the heuristic formula and then illustrate how it works by means of an example: Let  $S_i$  denote a super-class on a shell indexed by  $K$  ( $l_1$  norm) such that  $i$  is in the range  $[0, ((K+1)(K+2)/2)-1]$  and let  $S_i$  be of the form  $(a_1, a_2, a_3)$  where  $a_1 + a_2 + a_3 = K$  and  $a_1$ ,  $a_2$  and  $a_3$  are respectively the  $l_1$  norms of the first element, second to fifth elements and sixth to twenty-first elements respectively of the 21-D input quantised vectors. Then for each  $S_i$  calculate an associated merging index,  $jm_i$ , given by:

$$jm_i = \text{round} \left[ \lambda_2 \left( \max(|a_1 - a_2|, |a_1 - a_3|, |a_2 - a_3|) \right) + \lambda_1 \log_2 P_i \right] \text{ where } jm_i \geq 0 \quad (5.16)$$

where  $\max(x, y, z)$  is the maximum of integers  $x, y$  and  $z$ ,  $\text{round}(x)$  would be the nearest integer to the real positive number  $x$ ,  $P_i$  is the number of equally probable code-vectors in the super-class of index  $i$  and  $\lambda_1 \geq 1.0$ ,  $0 \leq \lambda_2 \leq 1.0$  are parameters assigned by the user (they determine how many super-super classes there are in total on a shell indexed by  $K$ ).

Let  $m_1$  be  $\min(jm_i)$  over all  $i$  and  $m_f$  be  $\max(jm_i)$ . Then the total number of super-super-classes,  $sst \leq m_f - m_1 + 1$  (as not all integers in the range  $[m_1, m_f]$  may be valid values for  $jm_i$ ). Let the vector  $x$  be  $[m_1, m_2, m_3, \dots, m_f]$  such  $x$  contains  $sst$  elements and each  $m_1, m_2, \dots, m_f$  is a valid value of  $jm_i$  and  $m_f > \dots > m_3 > m_2 > m_1$ . Then super-super-class,  $SS_j$  (where  $j$  is in the range  $[1, sst]$ ) is formed by combining all super-classes  $S_i$  such that  $x(j) = jm_i$ .

### Example

Let  $K=2$  and the super-classes are denoted by  $S_i$ ,  $i=0\dots5$ . The 6 super-classes are:

$\{(2,0,0),(1,0,1),(1,1,0),(0,0,2),(0,1,1),(0,2,0)\}$  and the associated  $P_i$  for each are:

$\{2,64,16,512,256,32\}$  where the sum of those values is 882 (or  $N(21,2)$ ).

let the number of super-classes,  $sst$  be 3 and the values of  $jm_i$  respectively be:

$\{0,2,3,2,0,0\}$ . Here  $m_1$  would equal 0,  $m_f$  would equal 3, and the vector  $x$  would contain

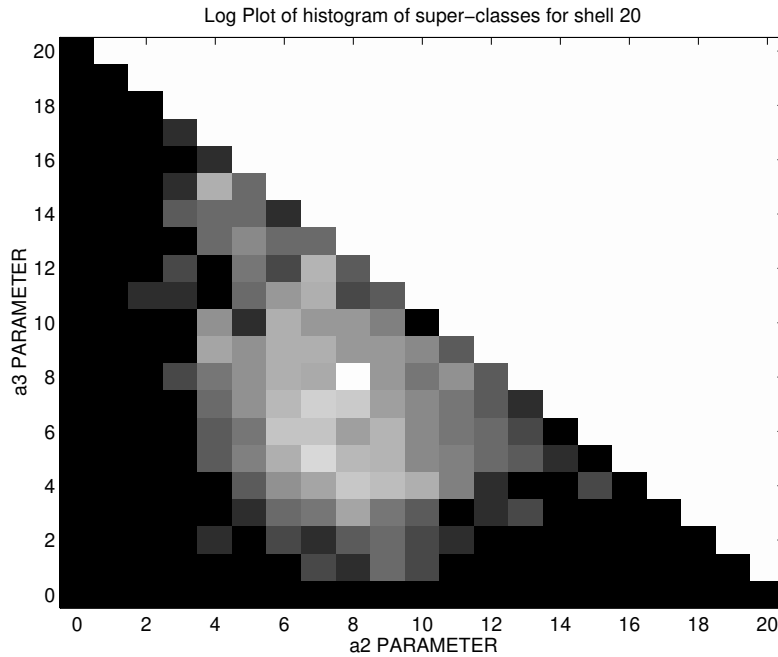
the following elements:  $\{0,2,3\}$ . Thus we need to form 3 new super-super-classes:

- ◆  $j$ , the super-class index, =1. Merge super-classes  $(2,0,0),(0,1,1)$  and  $(0,2,0)$
- ◆  $j=2$ . Merge super-classes  $(1,0,1)$  and  $(0,0,2)$
- ◆  $j=3$ . Contains super-class  $(1,1,0)$ .

When we merge for  $j=1$ , super-classes denoted by  $(2,0,0)$ ,  $(0,1,1)$  and  $(0,2,0)$  we are reducing by 2 the number of groups of equally probable code-vectors on the shell with  $l_1$  norm of  $K=2$ . For the new super-super-class given by  $SS_1$ , a total of  $2+256+32$  code-vectors are assumed to be equally probable.

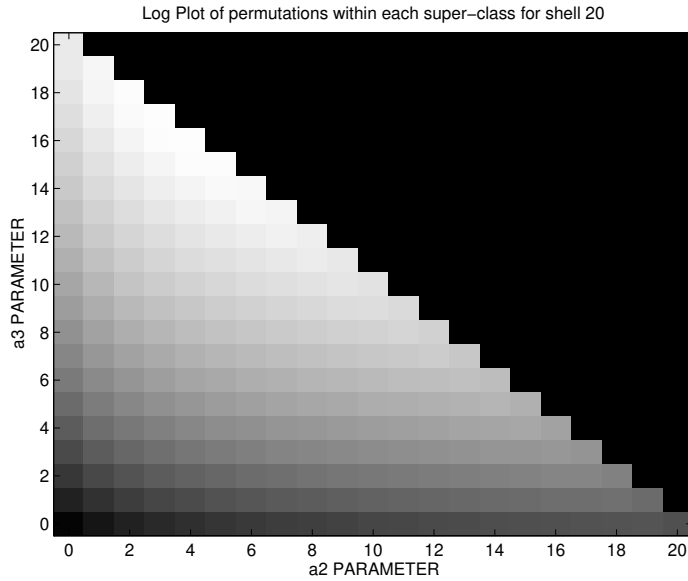
#### 5.5.4.1.1. Motivation for using heuristic formula

We will illustrate with some diagrams the operation of the formula  $\text{round}\left[\lambda_2\left(\max(|a_1 - a_2|, |a_1 - a_3|, |a_2 - a_3|) + \lambda_1 \log_2 P_i\right)\right]$  based on an example. Here  $K=20$ , the number of super-classes is  $(21 \times 22)/2 = 231$ . We have obtained the probability of occurrence of each super-class from a training set (see A.5) based on quantisation by a  $D_{21}$  lattice. There are a total of 882 training data vectors on this pyramidal shell with about 44% of the 231 histogram bins empty.



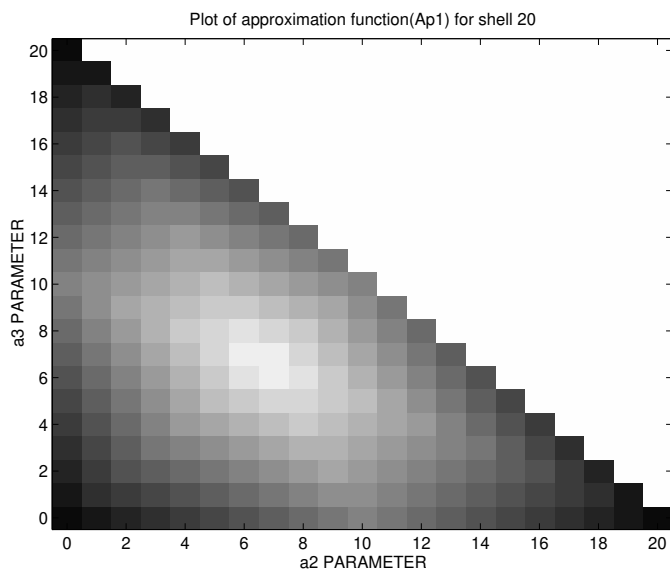
**Fig 5.8. Plot of  $\log_2$  of each histogram bin for each super-class**

Fig 5.8 shows a plot of  $\log_2(P(S_i))$  for  $i = 0 \dots 231$  where  $P(S_i)$  is the probability of occurrence of super-class  $S_i$ . This is equivalent to the information content of each histogram entry. The low gray-scale (dark) values indicate that those super-classes,  $(K-a_2-a_3, a_2, a_3)$  are highly improbable and the high gray-scale values indicate the super-classes which are most probable.

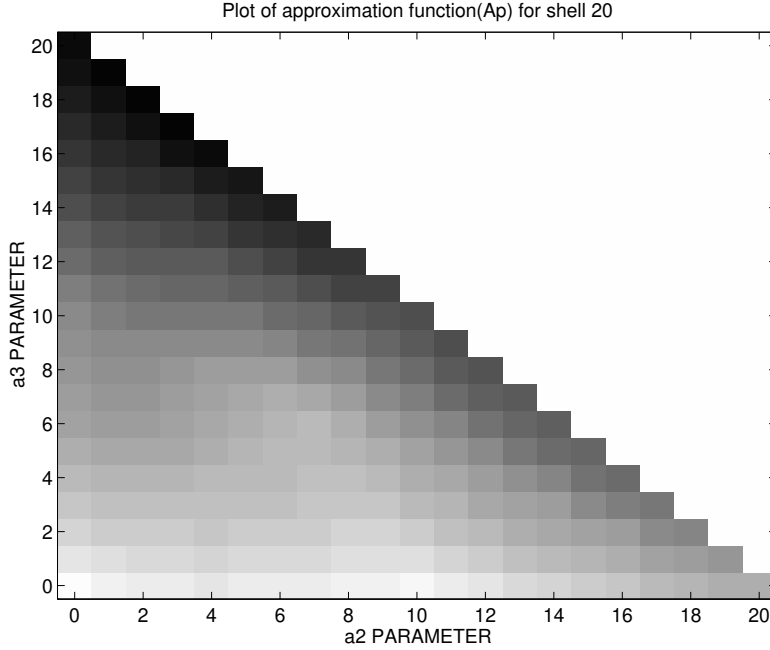


**Fig 5.9. Plot of  $\log_2$  of number of code-vectors in each super-class**

Figure 5.9. shows the values of each  $\log_2(P_i)$  in each super-class denoted by  $(K-a_2-a_3, a_2, a_3)$ . This gives the number of bits needed to code any particular code-vector in super-class  $S_i$ . We can see a slow variation from right to left and also a variation vertically.



**Fig 5.10: Plot of how super-classes are merged with  $\lambda_1 = 0, \lambda_2 = 1$  in merging formula 5.16**



**Fig 5.11: Plot of how super-classes are merged with  $\lambda_1 = 1, \lambda_2 = 1$  in merging formula 5.16**

Figures 5.10 and 5.11 show how the 231 super-classes are merged for the case 1) where the permutation function is ignored ( $\lambda_1 = 0$ ); and 2) where it is included in the merging equation. Here super-classes with the same gray-scale value are merged. We can see that improbable super-classes are being merged using this method. For the first case, we found that the percentage increase in the entropy after merging is 9.39% and the total number of super-super-classes is 20 and for the latter case, these figures are 1.42% and 42 super-super-classes respectively.

#### **5.5.4.2. Examples using merging formula based on training set statistics**

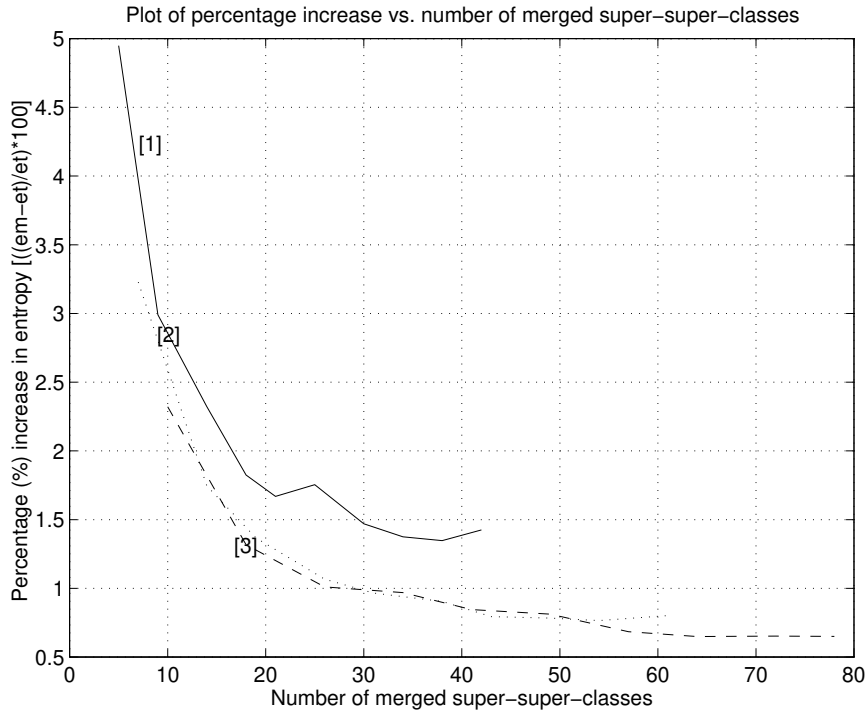
We will consider how the number of super-super-classes varies with the percentage increase in entropy after merging for various values of  $\lambda_1$  and  $\lambda_2$ . Here we define  $et$  as the entropy using the super-classes and  $em$  as the entropy using the super-super-classes.

Thus,

$$et = \sum_{i=0}^{i=231} \left( -P(S_i) \log_2(P(S_i)) + P(S_i) \log_2(P_i) \right) \quad (5.17)$$

$$em = \sum_{i=1}^{i=sst} \left( -P(SS_i) \log_2(P(SS_i)) + P(SS_i) \log_2(PP_i) \right) \quad (5.18)$$

where  $P(S_i)$  is the probability of occurrence of super-class  $S_i$ ,  $P_i$  is the number of equally probable code-vectors in super-class  $S_i$ ,  $sst$  is the total number of super-super-classes and  $P(SS_i)$  is the probability of occurrence of super-super-class  $SS_i$  and  $PP_i$  is the number of equally probable code-vectors in super-super-class  $SS_i$ . The sum over all super-super-classes of  $PP_i$  is the same as the sum of  $P_i$  over all super-classes.



**Fig 5.12: Plot of number of super-super-classes vs. increase in entropy for 3 different values of  $\lambda_1$  where we allow  $\lambda_2$  to vary from 0.1 to 1.0 in steps of 0.1. ([1]—solid line, with  $\lambda_1=1.0$ , [2]—dotted line, with  $\lambda_1=1.5$ , [3]—dashed line, with  $\lambda_1=2.0$ )**

The values for the super-super-classes on the  $x$ -axis of Figure 5.12 are generated by varying  $\lambda_2$  between 0.1 (extreme left) and 1.0 (extreme right) and using three different values for  $\lambda_1$  for the three different curves, [1]-  $\lambda_1=1.0$  , [2]-  $\lambda_1=1.5$  and [3]-  $\lambda_1=2.0$ . From the graph, one can determine the values of  $\lambda_1$  and  $\lambda_2$  which minimise the number of super-super-classes (so that when coding data outside the training data, the fewer the number of histogram bins, the more accurate the probabilities that are measured) for a given percentage increase.

### 5.5.4.3. Overall merging strategy for outer shells

We find (figure 5.24) that the probability of occurrence of  $K$ ,  $P(K)$ , reduces as  $K$  (pyramidal shell “radius”) increases but the number of super-classes increases. So we want to find a merging strategy such that the total number of super-super-classes reduces as  $K$  increases. We have adopted the following approach, which we have found to work well at low-bit rates where the maximum value of  $K < 60$ . It should be noted that the merging is done off-line using training data statistics.

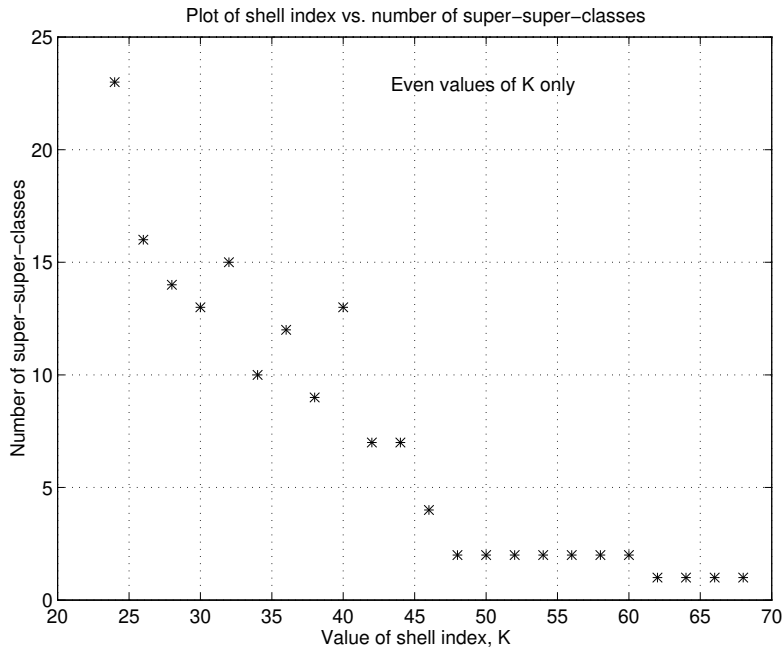
- ◆ We find that merging of super-classes is appropriate for values of  $K > 22$  (as over 50% of the histogram bins are empty and accurate probability estimation is difficult). We describe the strategy when the values of  $K$  are even (i.e. when a  $D_{21}$  or  $Z_{21}/D_{21}$  lattice is used for quantisation). A similar approach is taken for the  $Z_{21}$  lattice.
- ◆ Let  $et$  be the entropy using super-classes on shell  $K=24$ . Let  $P(K=24)$  be  $pr_{24}$  where  $P(K=24)$  is the probability of occurrence of shell 24 for the training set vectors. We allow a certain increase in the entropy of  $inc$  (where  $inc$  is a percentage) after merging. Thus the actual increase in bits/vector is  $target = \frac{inc}{100} \times et$ . We can then find the values for  $\lambda_1$  and  $\lambda_2$  which minimise the number of super-super-classes.

- ◆ For subsequent shells, the increase in entropy allowed is given by:

$$\frac{pr_{24}}{pr_K}(\text{target}) \text{ bits/vector} \quad (5.19)$$

- ◆ where  $pr_K$  is the probability of occurrence of shell  $K$ . Thus again one can find suitable  $\lambda_1$  and  $\lambda_2$  which minimise the total number of super-super-classes on shell  $K$ .

The reasoning behind this is that we want to allow the increase in entropy for each shell ,due to merging, to contribute equally to an overall increase so we must correspondingly allow the absolute increases on each shell (in bits/vector) to increase proportionately.



**Fig 5.13: Plot of number of shell index vs. number of super-super-classes**

Fig 5.13. has been obtained by setting the parameter  $inc=1.5\%$ . We see that the number of super-super-classes reduces as  $K$  increases in general (although there are one or two slight anomalies). The merging strategy for each  $K \geq 24$  is stored at the encoder/decoder and used when coding with either the  $D_{21}$  or the  $Z_{21}/D_{21}$  lattice. For the  $Z_{21}$  lattice a similar merging algorithm is performed (except we start at  $K=23$ ) with the parameter  $inc=2.15\%$

(this is shown to give good performance for images coded at low bit rates (i.e. maximum  $K$  tends to be  $<60$ )). We also see that at  $K=60$ , the total number of super-super-classes is one. This means that all possible code-vectors,  $N(21,60)$  (equation 5.9), are contained within this one super-super-class.

From Figure 5.13, we have reduced the total number of probabilities that need to be stored from about 20000 (for  $K$  between 24 and 60, in steps of 2) to 153. This is a reduction of over 99%.

If the value of  $inc$  is increased, the super-super-classes will reduce more rapidly to 1 and if the value of  $inc$  is reduced, the opposite is true. For optimality, this training process would need to be re-run, with reduced value of  $inc$ , if high bit-rate coding is desired

## **5.6. Additional stages of ECPVQ algorithm**

Here we shall be discussing the other stages in the ECPVQ algorithm. In section 5.6.1, we shall be looking at how to efficiently code the DC sub-band and introduce a weighting method which takes horizontal and vertical correlations into account when using DPCM. In section 5.6.2, we shall be looking at how the reproduction vectors at the decoder are formed from the quantised vectors and the input scaling factor (by which the vectors are scaled before being lattice quantised) and introduce a novel method called ‘multiple mu’ which is shown to improve performance over centroid estimation techniques. In section 5.6.3, we look at an efficient method for entropy coding the shell index ( $K$ ) which takes a wide range of correlation information into account.

### 5.6.1 Coding of DC band using differential coding

We initially have to code a DC sub-band of size  $32 \times 32$  (for  $256 \times 256$  images, as we are using a 3 level decomposition) or  $16 \times 16$  (for  $512 \times 512$  images, as we are using a 5-level decomposition in that case). Typically, there is large correlation between adjacent subband values and this can be exploited using differential encoding (similar to JPEG). We use a similar method to that used in JPEG [2] with slight modifications. Let  $pq(x,y)$  be the quantised predictor, and  $E(x,y)$  be the error between the actual value of a coefficient,  $c(x,y)$ , and its predictor estimate, i.e.  $E(x,y) = c(x,y) - pq(x,y)$ . Thus,  $Eq(x,y)$  which is the uniformly quantised prediction error is given by

$$Eq(x,y) = \text{round}(E(x,y)/\text{step\_size}) \quad (5.20)$$

The scanning order that is used to code error coefficients is done as follows:

Let  $len$  be 32 (for  $256 \times 256$  images) or 16 (for  $512 \times 512$  images).

- 1.) Scanning the first row from coeff.  $c(1,2)$  to  $c(1,len)$
- 2.) Scanning the first column from coeff.  $c(2,1)$  to  $c(len,1)$

Scanning the rest row-wise from  $c(2,2)$  to  $c(len,len)$

The predictor used is as follows:

```

if (x ∈ [2...len] and y ∈ [2...len])
    p(x,y) = w1cq(x,y-1) + w2cq(x-1,y) where w1+w2=1;
    pq(x,y) = round(p(x,y)/step_size)
elseif (y=1)
    pq(x,y) = cq(x-1,1)
else
    pq(x,y) = cq(x,y-1)
end

```

where  $w_1$  and  $w_2$  are the weighting factors. Experiments have shown that there is often strong correlation in one direction and weak correlation in the other, thus one weighting factor is often close to unity and the other close to zero. The different predictors used are

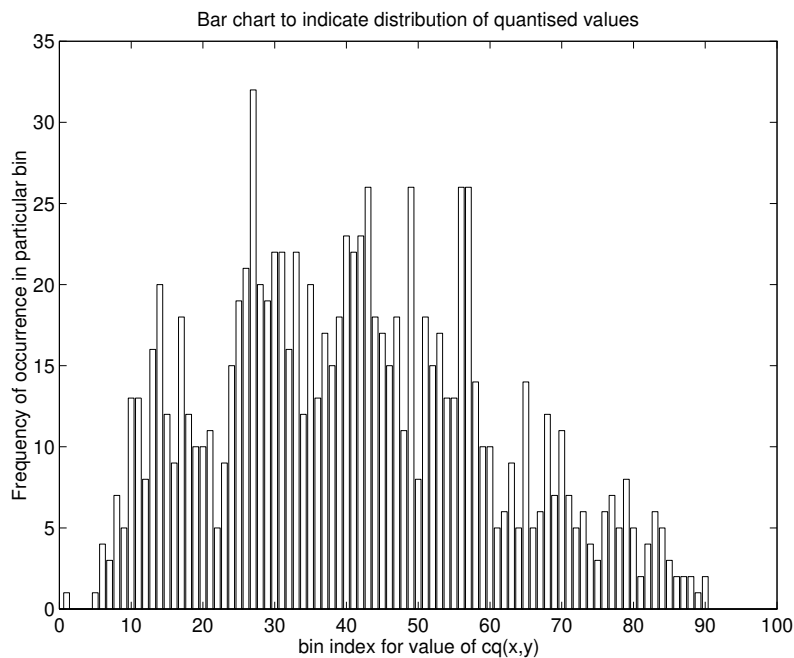
to ensure causality given the scanning order used. The weighting factors are estimated at the encoder to minimise the first-order entropy and are transmitted to the decoder. The quantised value  $cq(1,1)$  is also transmitted to the decoder. A total of 18\* bits are used for the transmission of the 3 words of side-information (\* 5 bits each for  $w_1$  and  $w_2$ , and 8 bits for  $cq(1,1)$ ).

A laplacian distribution is used to model the probabilities of the quantised error coefficients. The standard deviation and mean of the error coefficients (which is close to zero) are transmitted to the decoder so that the estimated probabilities need not be transmitted. The two extreme quantised values i.e.  $min(Eq(x,y))/step\_size$  and  $max(Eq(x,y))/step\_size$  are also transmitted. The estimated probabilities are stored to a precision of 10 bits ( $\log_2(\text{number of coefficients})$ ). The total amount of side-information to be transmitted is about 52 bits (including the 18 bits previously mentioned).

The coefficient values at the decoder are determined as follows:

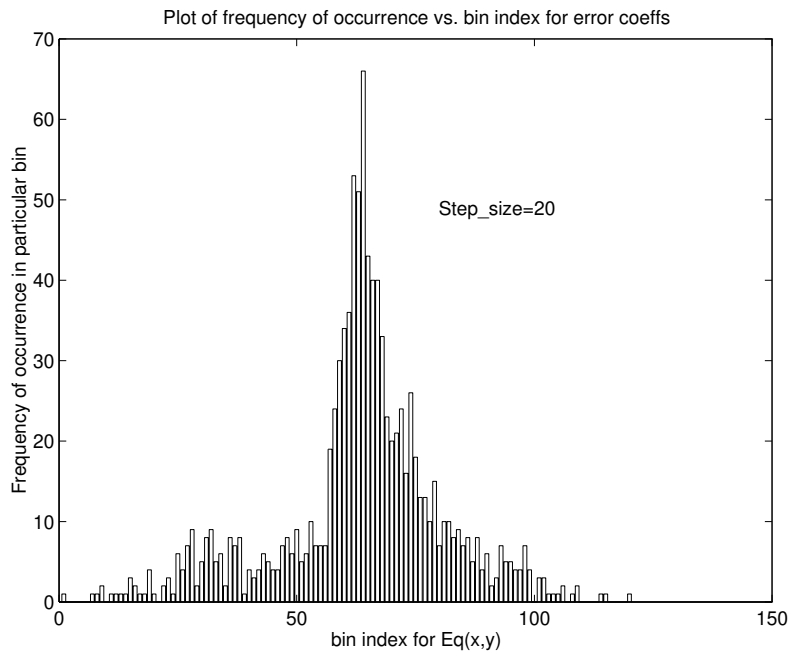
$$Dq(x,y) = Eq(x,y) + pq(x,y) \tag{5.21}$$

It is easily shown that  $Dq(x,y) = cq(x,y)$  for all  $x$  and  $y$ .



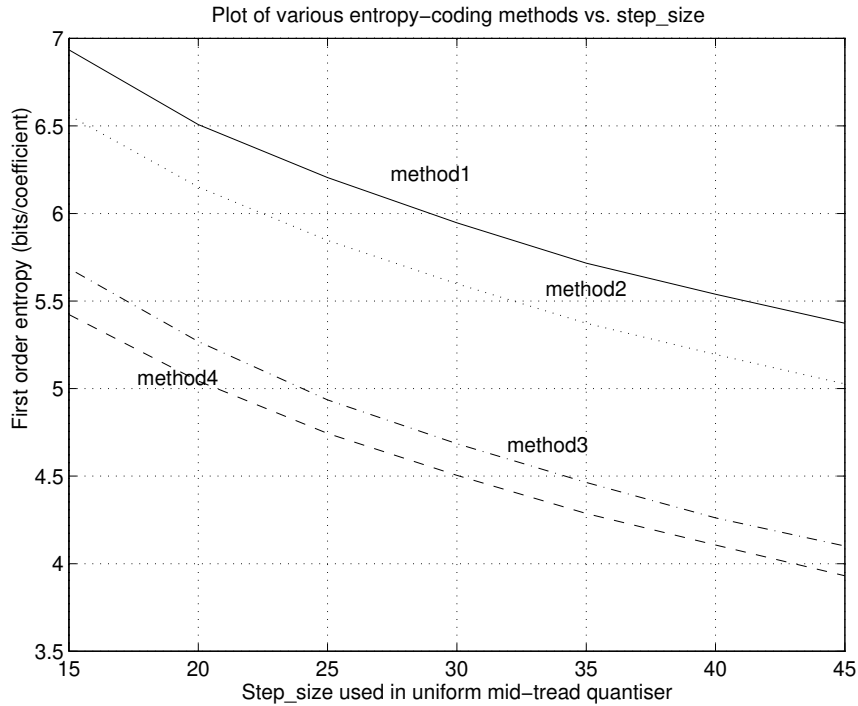
**Fig 5.14 :Plot of histogram of quantised coefficients in DC band**

Fig 5.14 shows the distribution of quantised coefficients for  $step\_size=20$  of the image **Lena**. It shows that entropy coding the raw quantised values would yield little reduction in the bits required to transmit the values (as opposed to a  $\log_2(\text{number of bins})$  coding method).



**Fig. 5.15-Plot of histogram of the errors after prediction in coefficients in DC band.**

Fig 5.15 shows a large concentration of error values in a few bins implying that entropy coding would mean substantial gains in bit-rate. The optimised values of the weighting factors are  $w_1=0$  and  $w_2=1$ . We have found that for overall bit-rates of about 0.6 bits/pixel, the optimised weights for the  $256 \times 256$  images *Peppers* and *Sailboat* (see A.5),  $w_1=0.6, w_2=0.4$  for the former and  $w_1=0.4, w_2=0.6$  for the latter, which indicates approximately equal correlation in both horizontal and vertical directions and shows the usefulness of this approach.



**Fig 5.16 Plot of various entropy coding methods for DC band**

Figure 5.16 shows the reduction in entropy when coding error coefficient values as opposed to actual coefficient values. Method1 assumes each quantising level from the minimum to the maximum level is equally probable. Method2 is the actual entropy of the quantised coeffs ( $cq(x,y)$ ). Method4 is the actual entropy of the error coefficients ( $Eq(x,y)$ ) and method3 is the entropy using probabilities based on a Laplacian model. **Method3** is the method used in this chapter and includes the bits transmitted as side-information. The gain of method3 over method1 is approx. 20%. It has been found that the increase in entropy in using the estimated probabilities (method3 over method4) is approx. 4-5%. However method4 is not practical as the probability values would also need to be transmitted to the decoder.

## 5.6.2 Determination of reproduction vectors at decoder

We will present an existing method that can be used to determine the reproduction vectors and will develop a modified method which has been shown to improve performance in terms of distortion for a given step-size  $\Delta$ .

The vector quantisation of an input  $\mathbf{x}$  (which is a 21-D vector, in our case) is simple.  $\mathbf{x}$  is scaled to form  $\mathbf{x}_s = \mathbf{x}/\Delta$  and the closest lattice point (the lattices that are used are discussed later) to  $\mathbf{x}_s$ , say  $\mathbf{y}_q$  is computed. Let  $y_i(x,y)$  be a single quantised coefficient.

The vector quantisation encoding regions are the Voronoi regions for the lattice vectors. The VQ reproduction vectors can be selected in one of two ways. The simplest method, which may be appropriate for large encoding rates, is to let the output codeword be  $\mathbf{y} = \Delta \mathbf{y}_q$ . This effectively uses the (scaled) midpoint of each quantisation region as the reproduction vector.

A better method is to select as reproduction vectors, the centroids (based on the source distribution) of the respective lattice quantisation regions. It has been shown by [69] that when using the cubic lattice and centroids for the reproduction levels (assuming independent laplacian data), the performance is equivalent to that of the uniform threshold quantiser (UTQ). This allows one to use the known expression for the centroid,[88], that if the UTQ intervals are  $[(k-0.5)\Delta, (k+0.5)\Delta]$  where  $k = 1, 2, 3, \dots$ , then the interval centroids are  $k\Delta + \epsilon(\Delta)$  and when  $k = -1, -2, -3, \dots$ , the intervals centroids are  $k\Delta - \epsilon(\Delta)$  where  $\epsilon(\Delta) = \frac{1}{\lambda} - \frac{\Delta}{2} \frac{(1 + e^{-\lambda\Delta})}{(1 - e^{-\lambda\Delta})}$ .

$$k\Delta - \epsilon(\Delta) \text{ where } \epsilon(\Delta) = \frac{1}{\lambda} - \frac{\Delta}{2} \frac{(1 + e^{-\lambda\Delta})}{(1 - e^{-\lambda\Delta})}. \quad (5.22)$$

However, an alternative method has been proposed for the 21-D vectors that are considered in this chapter. The method is as follows (called the ‘multiple mu method’): Here we shall consider the approach adopted for coding 256×256 images where 3 levels of decomposition are used. The sub-band labelling which we subsequently use relates to the sub-band labelling in Fig 5.1 as follows:  $V_1$ ,  $H_1$  and  $D_1$  correspond to B2, B3 and B4

respectively,  $V_2$ ,  $H_2$  and  $D_2$  correspond to B5, B6 and B7 and finally  $D_3$ ,  $H_3$  and  $D_3$  correspond to B8, B9 and B10 respectively.

- In the three sub-bands labelled (B2, B3 and B4), single coefficients  $y_i(x,y)$  are considered and thus the centroid method which has been described above is used.
- For the sub-bands labelled (B5, B6 and B7), coefficients,  $y_i(x,y)$ , are grouped together to form  $2 \times 2$  (4-D) vectors,  $y_{B_w}(n)$  where  $w$  is 5,6 or 7 and  $n = 1 \dots 1024$  where 1024 is the number of such vectors in sub-band  $B_w$ .
- For the sub-bands labelled (B8,B9 and B10), coefficients,  $y_i(x,y)$ , are grouped together to form  $4 \times 4$  (16-D) vectors,  $y_{B_w}(n)$  where  $w$  is 8,9 or 10.
- For each of the six sub-bands (B5....B10), the reproduction vectors are determined using the following,  $y(n) = \mu_{B_w,K} \Delta y_{B_w}(n)$  where  $\mu_K$  is the scaling factor, given  $K$  where  $K$  is the index of the pyramidal shell upon which the vector  $y_{B_w}(n)$  lies.

The method for  $512 \times 512$  images where 5 levels of decomposition are used is similar. For the 9 A.C sub-bands on the coarsest three levels, the approach adopted for bands B2, B3 and B4 would apply. For the 6 remaining sub-bands on the two finest levels, the approach adopted for bands B5, B6 and B7 and B8, B9 and B10 would apply.

It is easily argued that  $\mu_{B_w,K}$  must be  $\leq 1.0$ . Fisher [65] used this approach to find reproduction vectors in a fixed-length coding scheme when he used just one pyramidal shell on which to scale code-vectors.

For shells closer to the origin, where  $K$  is much less than  $L$  (the vector dimensionality), Fisher [66] said that all of the representation *scaled* code-vectors may lie along edges of the pyramid so the representation sphere centroid (For the mse criterion, the appropriate representation region is the sphere and the VQ representation regions should be adjusted to provide the maximum volume of intersection with the pyramid surface) should be located somewhat inside the pyramid. However, as  $K$  increases, he stated that the *scaled* code-vectors are more likely to lie in the middle of a face of the pyramid, and thus this point should be used as the sphere centroid. For, very large

encoding rates, where the scaling factor is small and a significant proportion of vectors lie on outer pyramid shells, an approximation allowing  $\mu_{Bw,K}=1.0$ , can be utilised.

In a practical implementation, the values of  $\mu_{Bw,K}$  need to be transmitted to the decoder, such that the mse is minimised in each sub-band. However, to limit the number of bits needing to be transmitted, we transmit values of  $\mu_{Bw,K}$  for  $K = 1 \dots K_{\max}$ . Three methods can be used for vectors which lie on shells beyond shell indexed by  $K_{\max}$ .

1.) Allow,  $\mu_{Bw,K}$  to be equal to unity for  $K > K_{\max}$  or obtain a single  $\mu_{Bw,K}$  which minimises the mse of the remaining vectors (i.e. those that lie on shells beyond shell indexed by  $K_{\max}$ )

2.) For individual coefficients in vectors which lie on such shells, use the centroid method with  $\varepsilon(\Delta)$  as described previously.

3.) An alternative heuristic formula is developed which given  $\mu_{Bw,K_{\max}}$ , allows  $\mu_{Bw,K} \rightarrow 1.0$  as  $K$  increases.

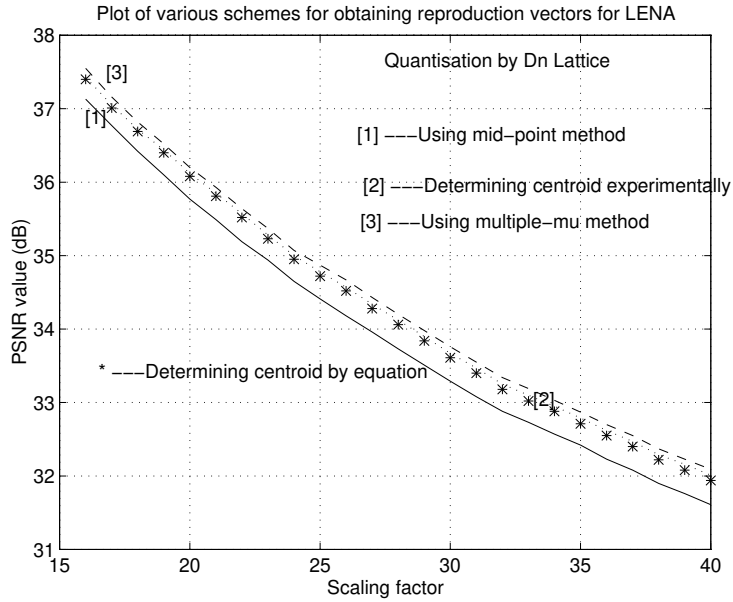
Let  $q = K_{\max}(1 - \mu_{Bw,K_{\max}})$  and then,  $\mu_{Bw,K} = (K - q) / K = 1 - q / K$

So,  $\mu_{Bw,K} = 1 - (K_{\max} / K) (1 - \mu_{Bw,K_{\max}})$ .

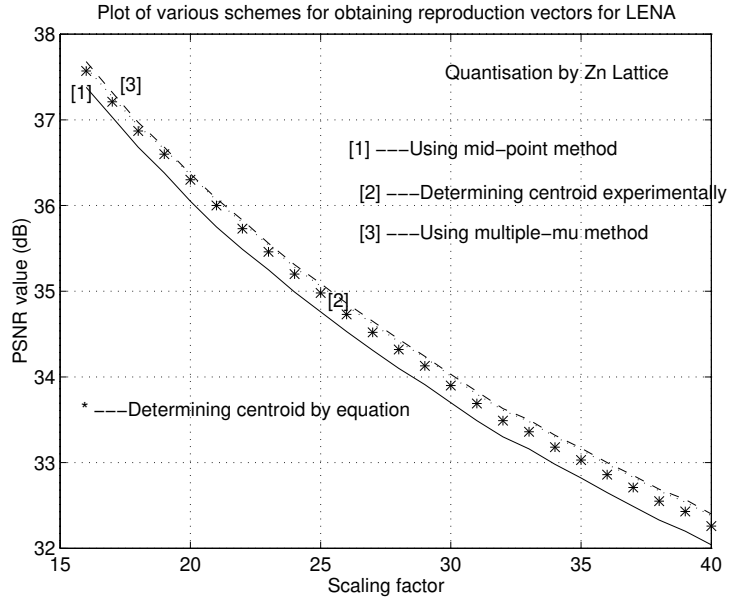
It can easily be seen that as  $K$  increases,  $\mu_{Bw,K} \rightarrow 1.0$ .

In our implementation, for sub-bands B5..B7,  $K_{\max}$  is set equal to 6 and for sub-bands B8...B10,  $K_{\max}$  is set equal to 9 (for a more complex but optimal implementation, when coding at very low bit-rates these values should be reduced as the scaling factor,  $\Delta$ , is quite large). Thus, the number of bits of side-information transmitted to the decoder is equal to  $5 \times (6 + 6 + 6 + 9 + 9 + 9) = 225$  bits where 5 bits are used for each  $\mu_{Bw,K}$ .

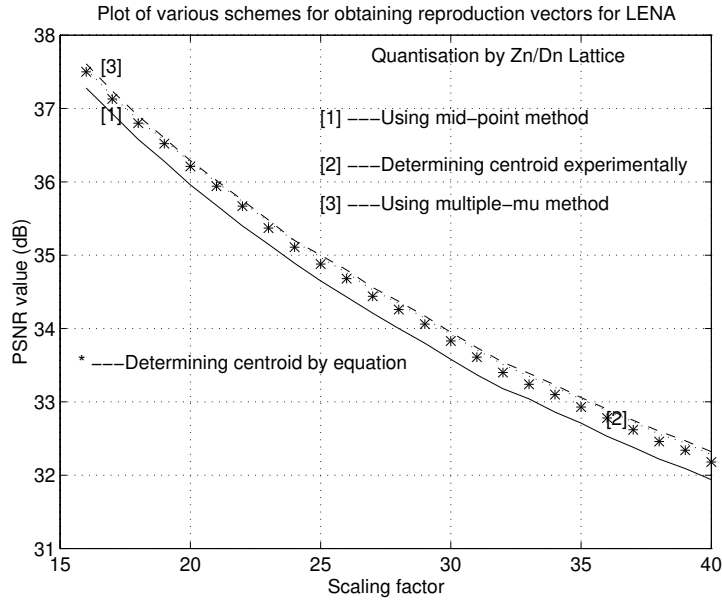
For  $256 \times 256$  images, this is quite a lot of side-information (for  $512 \times 512$  images, this is less significant). We shall suggest later a coding method for reducing this side-information by over 40%.



**Fig 5.17 Plot of 3 methods for obtaining reproduction vectors for various scaling factors with quantisation by  $D_{21}$  lattice ([1]-solid line, [2]-dotted line, [3]-dashed line)**



**Fig 5.18 Plot of 3 methods for obtaining reproduction vectors for various scaling factors with quantisation by  $Z_{21}$  lattice ([1]-solid line, [2]-dotted line, [3]-dashed line)**



**Fig 5.19 Plot of 3 methods for obtaining reproduction vectors for various scaling factors with quantisation by  $Z_{21}/D_{21}$  lattice ([1]-solid line, [2]-dotted line, [3]-dashed line)**

Figures 5.17-5.19 show the various methods of obtaining reproduction vectors when quantisation is by respectively the  $D_{21}$ ,  $Z_{21}$  and  $Z_{21}/D_{21}$  lattice and the image been coded is  $256 \times 256$  **Lena**. A scaling factor of 15 to 16 is equivalent to a overall bit-rate of about 1 bit/pixel.

Figures 5.17-5.19 show that the gain in using the ‘multiple mu’ method over using the mid-point of each quantisation region is about 0.39-0.49dB for the  $D_n$  lattice, 0.29-0.37dB for the  $Z_{21}$  lattice and 0.31-0.38dB for the  $Z_{21}/D_{21}$  lattice.

Summarising, there are four main methods for determining the reproduction vectors at the decoder: (1) Using the mid-point (scaled) of a quantisation region (2) Determining the centroid by calculating  $\varepsilon(\Delta)$  for each sub-band from equation 5.22. and the variance which is transmitted to the decoder (3) Determining the centroid by using the transmitted values for  $\varepsilon(\Delta)$  from the encoder, which are determined experimentally for each sub-band (by an exhaustive evaluation of 101 equally spaced values for  $\varepsilon(\Delta)$  between  $[0, \Delta/2]$ )

for each A.C. sub-band) and (4) using a combination of (2) for some sub-bands (on the coarser levels) and the ‘multiple-mu’ method for the 6 sub-bands on the finest two levels.

The advantage in using the method outlined in (4) over the method outlined in (3) is between 0.07-0.12dB, 0.02-0.08dB and 0.03-0.08dB for the  $D_{21}$ ,  $Z_{21}$  and  $Z_{21}/D_{21}$  lattices respectively. The advantage in using (4) over the method outlined in (2) increases to 0.12-0.17dB, 0.07dB-0.15dB and 0.08-0.16dB for each of the lattices respectively. Thus, we have found that when the centroids for each sub-band are effectively determined by equation 5.22 -with only the variances needing to be transmitted to the decoder- (as opposed to by an optimization process at the encoder which determines the “best” value of  $\epsilon(\Delta)$  for each sub-band), the cost is typically about 0.05dB, 0.1dB and 0.09dB respectively depending on the lattice but there is obviously an advantage in computational terms.

The results indicate that the ‘multiple-mu’ method is most useful for the  $D_{21}$  lattice but when the  $Z_{21}$  or  $Z_{21}/D_{21}$  lattice is used, the ‘multiple-mu’ method would give the most gain at medium-high bit-rates ( $\geq 0.95$  bits/pixel for **Lena**). At low bit-rates, the method where the centroids are determined experimentally is adequate and sufficiently close in terms of PSNR performance not to justify the additional complexity introduced by the ‘multiple-mu’ approach except when the  $D_{21}$  lattice is used for quantisation purposes. For simplification, in subsequent results that we present, we have used the ‘multiple-mu’ method irrespective of desired bit-rate or lattice used.

We present an example of the ‘multiple-mu’ method for the Lena image of size  $512 \times 512$  quantised by the  $Z_{21}/D_{21}$  lattice with  $\Delta=10.5$  (a bit-rate of approximately 0.95 bits/pixel). We have found that the advantage of the method described in (4) over that described in (3) is 0.11dB (40.35dB as against 40.24dB). The ‘mu’ values which are transmitted, for the 6 sub-bands on the finest two levels, are as follows:

sub-band of size  $128 \times 128$

(vertical orientation): **0.72 0.84 0.88 0.94 0.96 0.96**

sub-band of size  $128 \times 128$

(horizontal orientation): **0.72 0.82 0.88 0.94 0.96 0.98**

sub-band of size  $128 \times 128$

(diagonal orientation): **0.72 0.84 0.90 0.94 0.96 0.98**

sub-band of size  $256 \times 256$

(vertical orientation): **0.62 0.64 0.68 0.72 0.74 0.78 0.82 0.86 0.88**

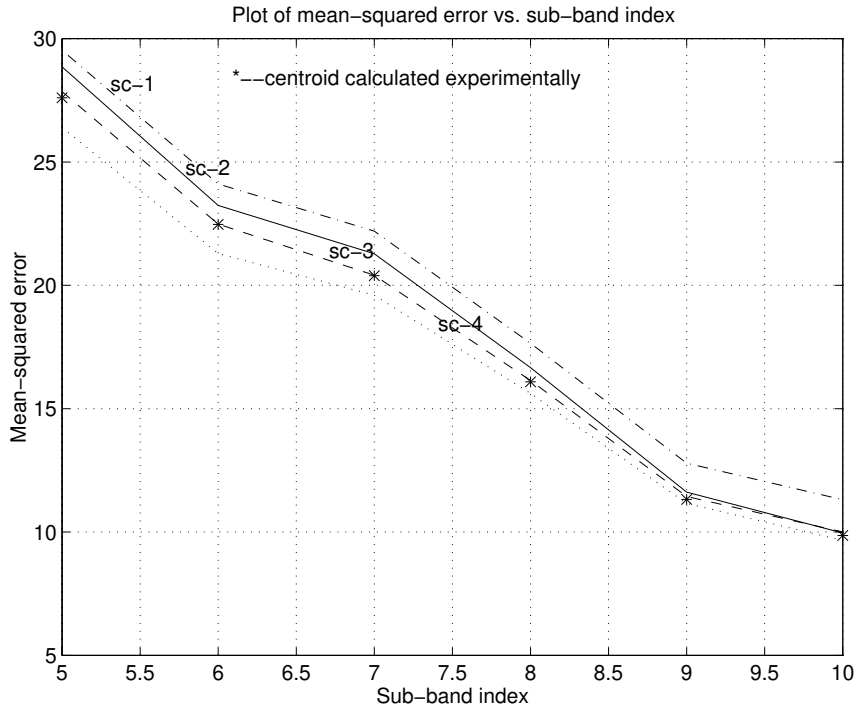
sub-band of size  $256 \times 256$

(horizontal orientation): **0.62 0.66 0.68 0.74 0.76 0.80 0.84 0.84 0.86**

sub-band of size  $256 \times 256$

(diagonal orientation): **0.60 0.64 0.68 0.70 0.76 0.78 0.82 0.82 0.84**

As one can see, there is a high degree of correlation between successive ‘mu’ values for each sub-band. We have found that, by 1) transmitting the first ‘mu’ value for each sub-band using 5 bits; and 2) transmitting the remaining ‘mu’ values for each sub-band by huffman encoding the difference between successive values, the total number of bits needed reduces from 225 to 135 bits (for given example). The huffman code-tables were obtained from training set statistics (the range of the difference values is  $[-0.5, +0.5]$  in steps of 0.02). However, in the version of the ECPVQ method used to generate final results we have used the simpler (i.e. 5 bits per ‘mu’ value) method for coding  $256 \times 256$  sized images. We have though used the differential entropy coding approach when presenting results for coded  $512 \times 512$  sized images.



**FIG 5.20 Plot of different methods for obtaining reproduction vectors vs. sub-band index for quantisation with the  $D_{21}$  lattice.**

**(sc-1: dot-dashed line, sc-2: solid line, sc-3: dashed line, sc-4: dotted line)**

The following notation in Fig. 5.20 has been used: ‘Sc-1’ is where the decoded vectors are chosen as  $\mathbf{y} = \Delta \mathbf{y} \mathbf{q}$ . ‘Sc-2’ is where  $\mu_{Bw,K}$  is constant for any  $K$  given  $Bw$ . ‘Sc-3’ is the scheme where the centroid formula (5.22) is used. The ‘\*’ indicates where the centroid has been calculated experimentally, rather than using the equation 5.22. ‘Sc-4’ is the ‘multiple mu’ scheme.

Fig 5.20 shows that methods ‘Sc-2’ and ‘Sc-3’ converge for very high frequency sub-bands. Also that the centroid equation 5.22 is very accurate in determining the optimal value of  $\epsilon(\Delta)$ .

### 5.6.3 Coding of shell index for 21-D hierarchical vectors

We shall consider here the coding of the shell index information for the 21-D hierarchical vectors which we need to code in our algorithm. The simplest method would be to use a first-order entropy code. However, due to likely correlations between the shell indices of neighbouring vectors, we shall use a conditional entropy code (context-based code or higher-order code) to exploit these correlations. To simplify a practical implementation (i.e. reduce the number of probabilities needing to be stored etc.), we shall employ a context-based code to code the first few shell indices and a simple first-order code for any remaining shell index information (as proposed by [42] for amplitude information). Thus it can be shown that  $M = T^{nc}$  where  $M$  is the number of context states,  $T$  is number of values each context can take and  $nc$  is the number of contexts. The number of probabilities that need to be stored is  $R \times nc$  where  $R$  is the number of values the output symbol  $X$  can take given any context state (the method of [42] simplifies this further by taking  $T = R$ ).

The contribution to the overall entropy, of the context-based code, is given by:

$$H_{cxt\_tr} = - \sum_{j=0}^{M-1} \left[ P(C = c_j) \sum_{i=0}^{R-1} P(X = x_i | C = c_j) \log_2 \left( P(X = x_i | C = c_j) \right) \right] \quad (5.23)$$

where  $C$  is the previous context information and  $X$  is the current output symbol and context data can take  $R$  and  $M$  values respectively. The total symbol alphabet for  $X$  consists of  $\{x_0, x_1, \dots, x_{N_{max}}\}$  but we constrain ourselves to selecting just the first  $R$  values where typically  $R \ll N_{max}$  (for complexity reasons) and the total number of symbols in the alphabet is  $N_{max} + 1$ . The way that the shell index is related to the symbols  $\{x_0, x_1, \dots, x_{N_{max}}\}$  depends on the type of lattice used and is described in Appendix A.6.  $N_{max}$  is also related to the maximum shell index and this definition is also in A.6. We are also assuming that the context states  $c_j$  are mutually exclusive. Here  $j = 0 \dots T^{nc} - 1$ . Each context state  $c_j$  is of the form,  $((C_1 = t_1) \& (C_2 = t_2) \& (C_3 = t_3) \& \dots (C_{nc} = t_{nc}))$  where  $nc$  is the number of

contexts,  $C_1 \dots C_{nc}$  are the  $nc$  contexts and  $t_1 \dots t_{nc}$  are the values of those contexts and range from  $0 \dots T-1$ . Here again, the values each context  $C_n$  can take,  $0 \dots T-1$ , can be related to the shell index and are defined in A.6

The contribution to the overall entropy, of the first-order code, can be easily derived as follows where  $F$  is a normalising factor:

$$P_n = \sum_{i=0}^{R-2} P_i \quad \text{and let } P_i = P(X = x_i) \quad (5.24)$$

$$F = \frac{1}{(1 - P_n)} \quad (5.25)$$

Here the sum of the probabilities from  $i=R-1$  to  $N_{max}$  is  $(1-P_n)$  and a separate first order entropy code needs to be designed for the probabilities,  $P_i, i=R-1 \dots N_{max}$

$$H_{1st} = \frac{1}{F} \sum_{i=R-1}^{N_{max}} -P_i F \log_2(P_i F) = \sum_{i=R-1}^{N_{max}} -P_i \log_2(P_i F) \quad (5.26)$$

$$H_{1st\_tr} = \sum_{i=R-1}^{N_{max}} -P_i \log_2(c e^{-ki}) \quad (5.27)$$

where the decaying exponential,  $c e^{-ki}$  is fitted to the probability curve,  $F P_i, i \geq R-1$  such that  $k$  is selected to minimise  $\sum_{i=R-1}^{N_{max}} -P_i F \log_2(c e^{-ki})$  where  $c$  is obtained such that

$c = \frac{1}{\sum_{i=R-1}^{N_{max}} e^{-ki}}$ , i.e. that estimated probabilities sum to unity. Thus, only the value of  $k$  and

$N_{max}$  need be transmitted to the decoder.

$$H_{total} = H_{ctx\_tr} + H_{1st\_tr} \quad (5.28)$$

$H_{1st\_tr}$  is the entropy contribution of the actual first order code which we have proposed to the overall rate  $H_{total}$  and  $H_{ctx\_tr}$  is the contribution of the context-based code to the overall bit-rate.

**Tables 5.21-5.23.** Table contains values for  $H_{total}$ . Here image used is LENA  $256 \times 256$ ,  $\Delta=27$  (lattice is  $D_{21}$ ).

The number of contexts are 2, 3 and 4 respectively.  $T$  is the number of distinct values each context can take and  $R$  of distinct values the output symbol can take. Ideal first order entropy = 2.72 bits/vector.

<b>R   T</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>2</b>	2.35	2.32	2.32	2.32	2.32	2.32	2.32	2.32	2.32
<b>3</b>	2.31	2.26	2.25	2.25	2.25	2.25	2.25	2.25	2.25
<b>4</b>	2.31	2.24	2.225	2.23	2.23	2.23	2.23	2.23	2.23
<b>5</b>	2.31	2.23	2.21	2.21	2.21	2.21	2.21	2.21	2.21
<b>6</b>	2.31	2.235	2.21	2.21	2.21	2.21	2.21	2.22	2.22
<b>7</b>	2.31	2.235	2.21	2.21	2.21	2.21	2.22	2.22	2.22
<b>8</b>	2.31	2.235	2.21	2.21	2.21	2.21	2.22	2.22	2.22

**Table 5.21:**

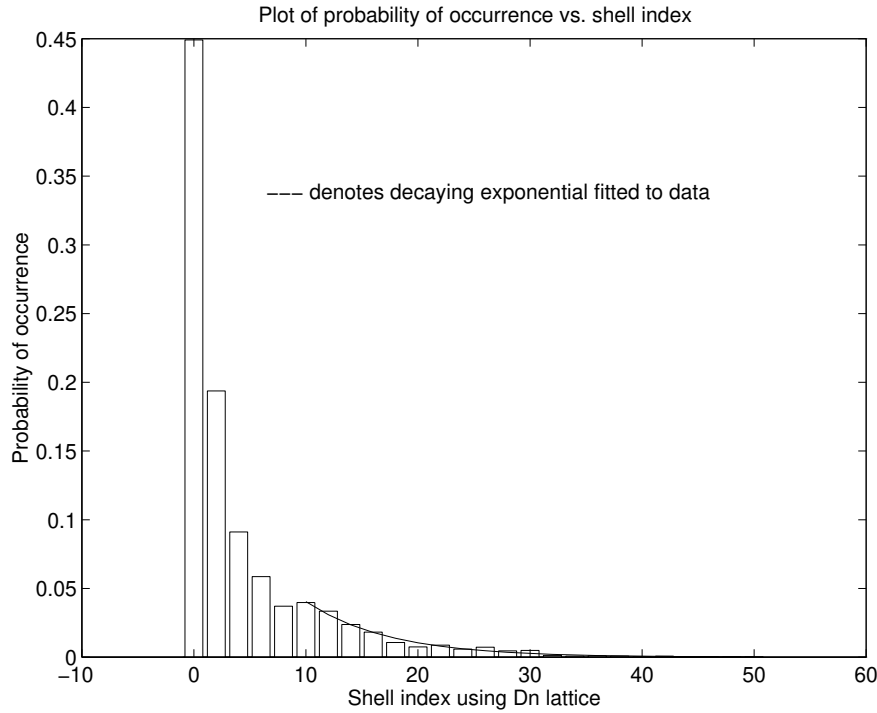
<b>R   T</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>2</b>	2.35	2.325	2.32	2.32	2.326	2.33	2.33	2.34	2.35
<b>3</b>	2.31	2.26	2.25	2.25	2.26	2.26	2.26	2.27	2.28
<b>4</b>	2.305	2.24	2.23	2.236	2.24	2.25	2.25	2.26	2.27
<b>5</b>	2.30	2.24	2.22	2.22	2.22	2.23	2.23	2.24	2.25
<b>6</b>	2.31	2.24	2.227	2.22	2.23	2.24	2.245	2.25	2.26
<b>7</b>	2.31	2.24	2.23	2.23	2.23	2.24	2.25	2.25	2.26
<b>8</b>	2.31	2.24	2.23	2.23	2.23	2.24	2.25	2.25	2.25
<b>9</b>	2.31	2.24	2.23	2.23	2.24	2.25	2.25	2.25	2.25

**Table 5.22**

<b><math>R   T</math></b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
<b>2</b>	2.31	2.29	2.29	2.30	2.28
<b>3</b>	2.26	2.22	2.22	2.24	2.26
<b>4</b>	2.25	2.2	2.19	2.22	2.23
<b>5</b>	2.25	2.19	2.18	2.21	2.21
<b>6</b>	2.26	2.2	2.18	2.21	2.215
<b>7</b>	2.26	2.2	2.17	2.215	2.21

**Table 5.23**

The method of choosing the contexts and the scan used in coding the vectors is given in appendix A.3.2. Tables 5.21-5.23 show that the potential advantages in increasing the number of contexts used are small (the conditional probability statistics are obtained from a training set of images). It should be noted that the potential advantages of increasing the values of  $T$  and  $R$  are quite small (however, as  $T$  increases, and the number of context states increases exponentially with  $T$ , a significantly larger training set (to reduce the number of empty bins) would be required to make valid comparisons). So, over a large range of bitrates and test images we have decided to use 4 contexts with  $T=4$  and  $R=8$  for future schemes.



**Fig. 5.24 Here  $\Delta=27$  (lattice is  $D_{21}$ ) for LENA.**

Figure 5.24 shows the probability of occurrence for 21-D vectors for various shell indices.

The maximum shell index is 50. It has been observed that fitting a decaying exponential to the remaining probabilities (after a context-based scheme codes the first  $R$  symbols) increases the first order entropy contribution (first-order entropy multiplied by  $(1-P_n)$ ), by only about 1.5% when the modelled probabilities are used in an entropy code, However, the increase to the total entropy is less than 0.5% because most of the contribution to the total entropy,  $H_{total}$ , is from the context-based code. The modelled probabilities are quantised to a precision of 12 bits.

### 5.6.3.1 Context state merging

Because the number of context states can grow to be quite large (in our case, setting  $T=4$ , the number of states is  $T^4=256$ ), it can be difficult to obtain accurate probabilities,  $P(X = x_i | C = c_j)$ , where the context state  $c_j$  is highly improbable in the training set and  $i$  is in the range  $[0..R-1]$  where  $R$  is the number of values that the output symbol  $X$  can take and  $j$  is in the range  $[0..T^4-1]$ . Thus there are a lot of probabilities which are zero in the training set and which have to be assigned some arbitrary minimum value in case they are needed by the image being coded. By merging states so that the entropy increase within the training set is minimised, one can perhaps obtain even better results when coding data which is outside the training set because the probabilities stored are more accurate and there are fewer empty bins in the ‘training histogram’ (thus avoiding a problem known as context dilution [52]). [41] proposed a scheme where a pair of context states is merged so as to minimise the increase in entropy and the procedure is carried on until the desired number of states is reached. A more optimal procedure, using an LBG type algorithm, is described in [43]. It should be stated, however, that this is **only** optimal for the **training set** probabilities.

We describe here a simple *novel* heuristic method which has been found to work well for the data considered in this chapter.

The heuristic merging formula is defined as follows:

$$\text{Let } jm = t_1 + t_2 + t_3 + t_4 \text{ where } 0 \leq t_1, t_2, t_3, t_4 \leq 3 \quad (5.29)$$

Here  $t_1$  is the value of the first context  $C_1$  and  $t_2$  the value of the second and so on, and  $jm$  is the index of the merged context state and has a range from  $[0..4(3)=12]$ . Thus we have reduced the total number of context states from 256 to 13. So in general, the heuristic formula merges  $T^n$  context states to generate  $n(T-1)+1$  states (in our case 13, with  $n=4$  and  $T=4$ ).

The actual merging is performed as follows:

```

For each  $jm$  (/* range from 0...12 */) do
    /* The context states defined by  $\{(C_1=t_1)\&(C_2=t_2)\&(C_3=t_3)\&(C_4=t_4)$  */
    /* such that  $t_1+t_2+t_3+t_4=jm$  are merged to form a new context state */
    /* called a merged state */
End;

```

Thus, the number of probabilities needing to be stored at the encoder/decoder is given by  $R(n(T-1)+1)$  where  $R$  is the number of values the output symbol  $X$  can take.

It should be noted that a similar formula has been used recently by Ortega and Chrysafis [52] in their method which uses context-based conditional entropy coding of scalar quantised wavelet coefficients. They, however, use a different approach for context classification which is as follows:

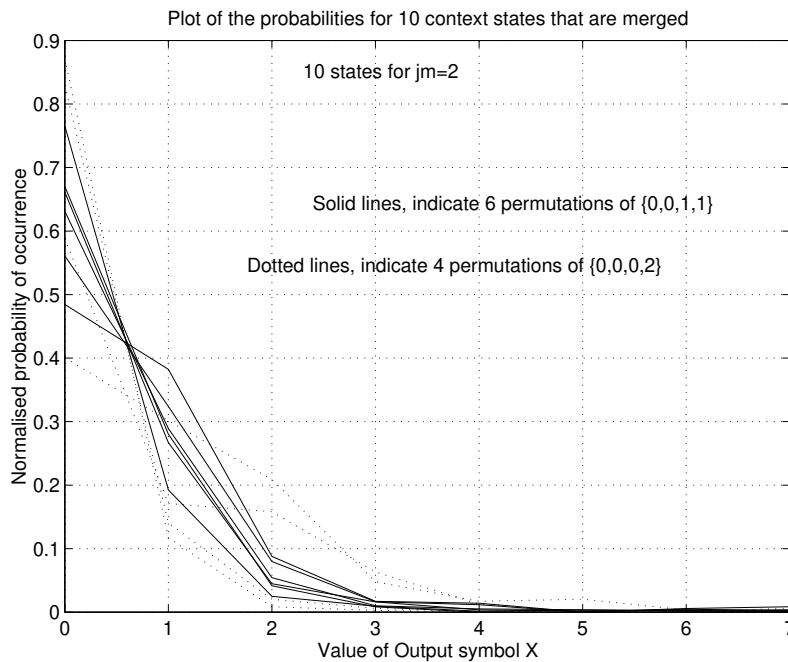
$$\hat{y} = \sum_{i=0}^{N-1} |y_i|$$

Here  $y_i$  are the bin indexes of scalar quantised coefficients (i.e. 0, +1,-1,+2,-2,...), used as contexts (total of  $N$ ) for coding a particular quantised coefficient  $x$ , and  $\hat{y}$  would be the index of a particular context state (i.e.0,1,2,...). Because the number of possible context states (i.e. different values of  $\hat{y}$ ) can be quite large, to avoid the problem of context dilution, Ortega suggests quantisation of  $\hat{y}$  by a Lloyd-Max type approach. With this approach the total number of context states (or classes) can be reduced to the desired level (Ortega and Chrysafis use  $N=13$  and 12 different context classes (one for  $\hat{y}=0$  and 11 others for  $\hat{y}>0$  using Lloyd-Max quantisation)). In a paper by Ortega which he has just submitted [64], he suggests a simple logarithmic classification (as opposed to the Lloyd-Max type approach) rule.

We can offer a partial justification for this merging formula and do so by means of an example: Consider the case for  $jm=1$ :

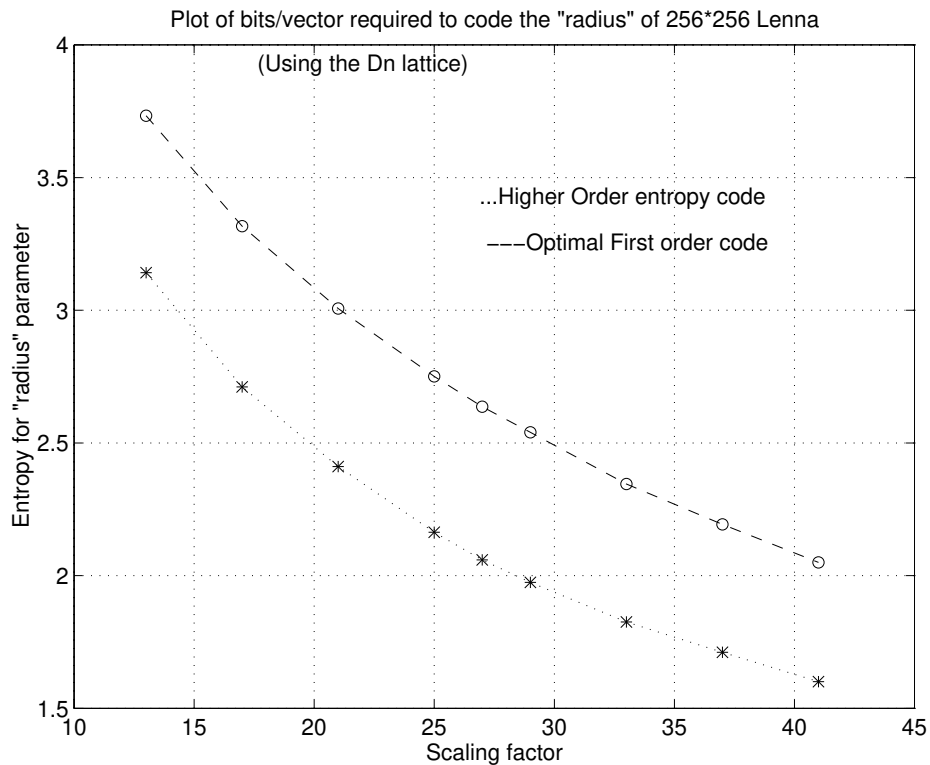
Here the 4 original context states that are merged have the following values:

$\{(0,0,0,1),(0,0,1,0),(0,1,0,0),(1,0,0,0)\}$  where for example  $(0,0,0,1)$  means  $t_1=0,t_2=0,t_3=0$  and  $t_4=1$ . Let the probability distribution function of the merged context state ( $f_1$ ) be the product of the individual 4 pdf's (assuming statistical independence). Then if the pdf's are of the same exponential form, the exponent of  $f_1$  is the sum of the four individual exponents. We shall now consider the case for  $jm=2$ .



**Figure 5.25 : Plot of 10 states to be merged for  $jm=2$  (index of a merged state)**

Fig 5.25 shows the probabilities  $P(X = x_i|C = c_j)$  for 10 value of  $c_j$  corresponding to 6 permutations of  $(1,1,0,0)$  and 4 permutations of  $(2,0,0,0)$  where  $(1,1,0,0)$  means  $t_1=1, t_2=1, t_3=0$  and  $t_4=0$ . Here the probabilities are obtained from a ‘training phase’ (with the  $D_{21}$  lattice used for quantisation) and the range for  $i = [0..7]$  (i.e. output symbol can take 8 distinct values or we have an 8 symbol alphabet). We observe that the probabilities associated with each of the original context states decay approximately exponentially (i.e. similar statistics).



**Fig 5.26: Plot of entropy reductions using higher-order code for image *Lena* using various scaling factors and the  $D_{21}$  lattice**

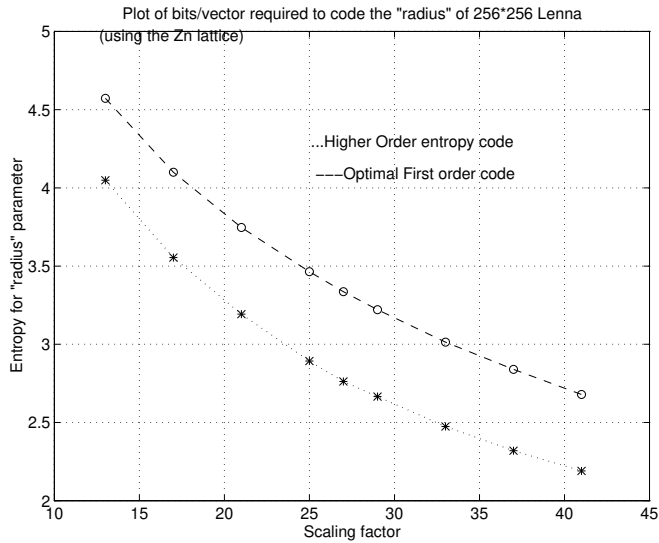
Fig 5.26 shows the reduction in entropy from using the conditional entropy code described in this section over a standard first-order entropy code. As an example, when the first-order entropy is 2.72 bits/vector, the total conditional entropy ( $H_{\text{total}}$ ) is 2.17 bits/vector, giving a gain of 0.55 bits/vector or 0.026 bits/pixel ( $0.55 \times 3072/65536$ ). The overall bit-rate turns out to be approximately 0.6 bits/pixel. The bit-rates given are theoretical but the entropy codes are based on probabilities obtained from coding a training set of 8  $256 \times 256$  images using 5 different scaling factors for each image (thus, different bit-rates, see A.5). For coding  $512 \times 512$  images, a training set of 4 images coded at 5 different bit-rates has been used (A.5). It has been found that the training data probability matrix of size  $8 \times 256$  (as  $R=8$  and no. of context states is equal to  $4^4$  (256)) contains approximately 10% empty bins. However, having applied the heuristic merging

formula, the final probability matrix of size  $8 \times 13$  contains no empty bins and in fact gives results which are slightly superior to those obtained with the initial matrix (pre-merging). The superiority is about 0.02-0.03 bits/vector (where in the case of  $256 \times 256$  images, there are 3 blocks of 1024 21-D vectors and for  $512 \times 512$  images, there are 3 blocks of 4096 21-D vectors to be coded).

A separate training data probability matrix is used for each of the three different lattices used in the quantisation phase.

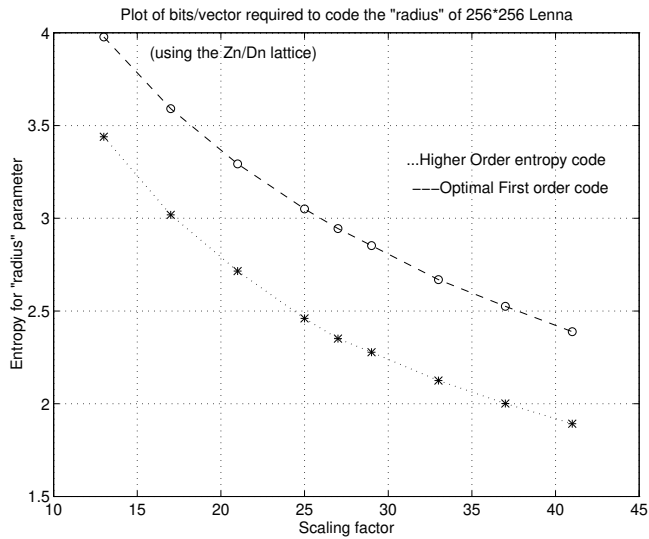
We have used the same statistics (i.e. one  $8 \times 256$  conditional probability matrix) to code each block of vectors. As seen in 5.4, one block of vectors contains horizontal edge information, another vertical edge information and a third diagonal edge information. However if one transposes the horizontal and vertical axes, one can argue that the horizontal and vertical edge statistics should be similar (when obtained from a 'training' phase). We have also found no advantage (or disadvantage) in using separate statistics for the block of vectors whose coefficients are from diagonally oriented sub-bands.

The way the 4 contexts are chosen is described in Appendix A.3.



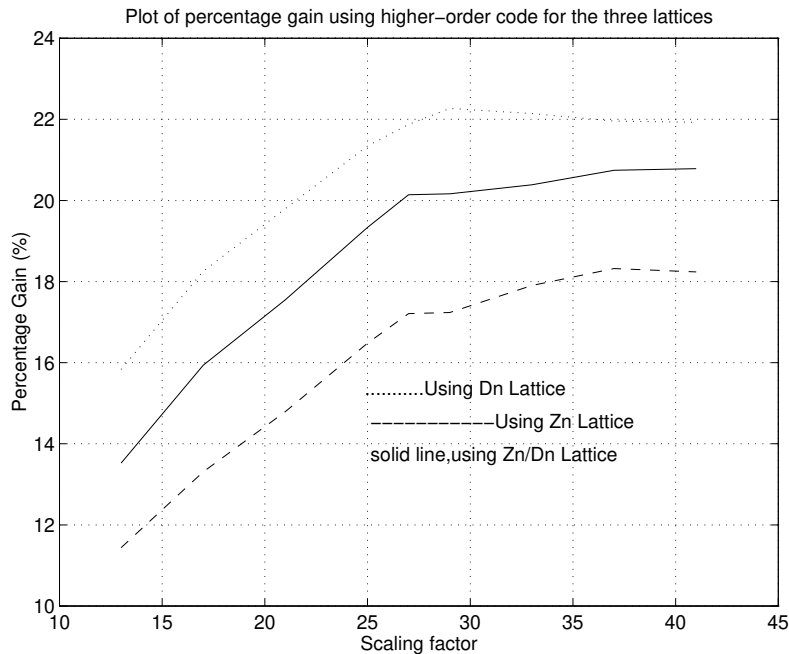
**Fig 5.27: Plot of entropy reductions using higher-order code for image *Lena* using various scaling factors and the  $Z_{21}$  lattice**

Fig 5.27 shows the reduction in entropy(bits/vector) from using the conditional entropy code (combination of context based code and separate first order code) and quantisation by the  $Z_{21}$  lattice.



**Fig 5.28: Plot of entropy reductions using higher-order code for image *Lena* using various scaling factors and the  $Z_{21}/D_{21}$  lattice**

Fig 5.28 shows the reduction in entropy (bits/vector) from using the conditional entropy code and quantisation by the  $Z_{21}/D_{21}$  lattice.



**Fig 5.29 shows percentage entropy reductions for the three lattices.**

Fig 5.29 shows the percentage reduction in entropy for the image Lena when each of the three different lattices are used for quantisation purposes. It shows that the gain is greatest for the  $D_{21}$  lattice and lowest for the  $Z_{21}$  lattice. It also shows that, for all 3 lattices at lower bit-rates (i.e. higher scaling factor), the advantage in using the context-based entropy code increases. The percentage gain in using the  $D_{21}$  lattice is between 16% and 23%, in using the  $Z_{21}/D_{21}$  lattice, it is between 14.5% and 22% and in using the  $Z_{21}$  lattice, it is between 12.5% and 19.5%.

In a practical coder, to improve the robustness of the coder for different images, the training set probabilities which are contained in the merged probability matrix (stored at both encoder/decoder) are adapted during the coding process at the encoder (this is a

trivial matter when using arithmetic coding). This adaptation can then be predicted and mirrored at the decoder. An adaptation method employed by Young [41] is used with the  $\lambda$  parameter set equal to 0.01 (i.e. a slow adaptation).

## 5.7. Coding results for $256 \times 256$ and $512 \times 512$ monochrome images

We shall be introducing coding results firstly for  $256 \times 256$  monochrome 8 bits/pixel images. Then we will show the modifications necessary to obtain results for  $512 \times 512$  images. We shall be comparing our results with standard JPEG [2], our benchmark coder (A.1), Fisher's [65] weighted pyramid scheme and Said/Pearlman's state-of-the-art entropy coded method [50].

### 5.7.1 Coding of $256 \times 256$ images

We shall begin by coding the image **Lena**. The ECPVQ coding scheme can be described as follows:

- ◆ A 3-level wavelet transform using 7/9 tap filters (A.1) is performed.
- ◆ A  $32 \times 32$  dc sub-band is uniformly quantised using the scaling factor,  $\Delta$ . The quantised coefficients are entropy-coded by the method of section 5.6.1.
- ◆ The remaining 3 blocks of 1024 21-D vectors are quantised using one of the three lattices specified in section 5.3 and using the scaling factor  $\Delta$ .
- ◆ The radius ( $K$ ),  $l_1$  norm, of each of the quantised vectors is entropy coded by the method described in section 5.6.3.
- ◆ For each quantised vector whose "radius" is not zero, its position information must be coded. For each such vector, the index of the sub-class (if  $K \leq 12$ ), super-class (for  $12 < K \leq 22$ ) or super-super-class (for  $K > 22$ ) in which it lies is entropy coded. The reason for this is described in detail later. Then the index within the sub/super/super-super class is encoded by either the Huffman code described in A.4.2. (if sub-class or super-class) or a simple binary code (for the super-super-

class). The simple binary code is as follows: Let  $c$  be the sub/super/super-super class in which the current quantised vector lies. If there are  $p$  possible code-vectors in the sub/super/super-super class, then each can be represented by  $\lceil \log_2(p) \rceil$  bits where  $\lceil x \rceil$  is the smallest integer greater than  $x$ . The Huffman code of A.4.2. can represent each possible code-vector by close to the theoretical optimum of  $\log_2(p)$  bits. If the quantised vector has “radius”,  $K > 22$ , there is very little cost in using the simplification because such vectors are highly improbable.

- ◆ The reproduction vectors at the decoder are assembled from the quantised vectors by using the side-information transmitted from the ‘multiple-mu’ method described in 5.6.2 and  $\Delta$ .

It should be noted that to avoid precision problems (as number of possible code-vectors in a **super-super-class** can be  $> 2^{50}$ ), in our implementation we have used binary digits (with binary addition, multiplication etc.) if the integer representing number of possible code-vectors in a specific **super-super-class** is greater than  $2^{50}$ . This is never the case for the **sub-classes** and **super-classes** which we consider.

### 5.7.1.1. Justification for using variations of the sub-class, super-class and super-super-class structures

We consider first the question of comparing performance of sub-classes vs. super-classes to see whether the simpler super-classes, derived by assuming the each distinct group of elements in the 21-D vectors (of size 1,4 and 16 elements respectively) come from a Laplacian pdf, are independent and have the same variance, in practice, perform almost as well as the more general sub-classes (see definitions in 5.5.)

Let the test image be (256×256) **Lena** and the scaling factor,  $\Delta = 30$  and the quantisation be done by the  $D_{21}$  lattice. Then let us look at shells indexed by 2,4,6,8,10 and 12.

The two entropies,  $e_{sub}$  and  $e_{sup}$  are defined as follows:

$$e_{sub} = \left[ \sum_i \left( -\log_2(P(S_i)) + \log_2(c_i) \right) \right] / 3072 \text{ bits/vector} \quad (5.30)$$

$$e_{sup} = \left[ \sum_i \left( -\log_2(P(SP_i)) + \log_2(cp_i) \right) \right] / 3072 \text{ bits/vector} \quad (5.31)$$

where both summations are over all quantised vectors whose “radius” ( $K$ )>0 and  $\leq 12$ . The total number of quantised vectors is 3072.  $P(S_i)$  is probability of occurrence of the sub-class in which the  $i^{\text{th}}$  vector lies and  $c_i$  is the number of possible code-vectors (including with different signs) in that sub-class. The notation  $P(SP_i)$  and  $cp_i$  corresponds to the previous notation except super-classes are being used.

The following results have been obtained:

<b><math>K</math></b>	<b>2</b>	<b>4</b>	<b>6</b>	<b>8</b>	<b>10</b>	<b>12</b>	<b>total</b>
$e_{sub}$	1.16	1.069	0.92	0.96	0.98	0.718	5.805 bits/vector
$e_{sup}$	1.163	1.088	0.927	0.967	0.987	0.713	5.835 bits/vector

We can see that using super-classes is only 0.03 bits/vector ( $0.03 \times 3072$  bits) inferior to the scheme where sub-classes are used. In fact for  $K=12$ , the super-class method is slightly superior. This is because on the shell indexed by  $K=12$ , there are 2606 sub-classes (many of whom contain empty bins from a training phase and to whom arbitrary values must be assigned in case they are needed in an entropy code for data not within the training set) and only 91 super-classes (whose probabilities can be more accurately estimated from a training set). This is the main reason we use super-classes (or modifications thereof (i.e. super-super classes)) for all  $K > 12$ .

Reducing the bit-rate by increasing value of  $\Delta$ , scaling factor, to 40 (overall bit-rate about 0.45 bits/pixel), we find that the difference between the sub-class and super-class method increases to 0.05 bits/vector (4.89 vs. 4.94 bits/vector). Increasing  $\Delta$  further to 55, we find the difference to be 0.06 bits/vector.

For the  $512 \times 512$  **Lena**, using the  $Z_{21}/D_{21}$  lattice for quantisation, the advantage in using the sub-class method at bit-rates less than 0.25 bits/pixel is equivalent to an improvement in PSNR performance (for a comparable bit-rate) of between 0.05-0.06dB.

Therefore, we can conclude that the assumptions made in deriving the super-class structure are generally valid when coding at medium to high bit-rates and even at low bit-rates the assumptions are reasonable. However, as we are mainly coding at quite low bit-rates we retain the sub-class structure for  $0 < K \leq 12$  to optimise our ECPVQ method.

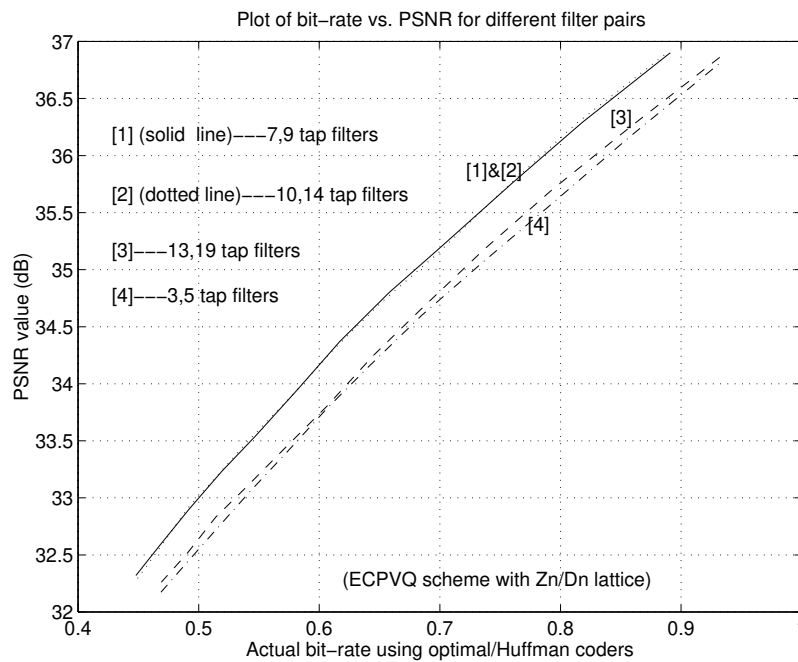
The same reasoning applies when  $K > 22$ . We find that accurate probability estimates cannot be obtained for the super-classes (If  $K=30$ , number of super-classes is 496 and there are only about 300 training data vectors). By using the merging algorithm of section 5.5.4.3., we can obtain an advantage of about 0.02-0.04 bits/vector. The advantage is greatest when quantising with the  $Z_{21}$  lattice (as there are fewer numbers of training data vectors on the outer shells because there are twice as many shells as for the  $D_{21}$  lattice). As an example, when Lena is coded at about 0.4, 0.6 and 1.0 bit(s)/pixel, the maximum value of  $K$  is respectively 36, 50 and 76. Therefore at quite low bit rates a simpler version of the super-super-class method (i.e. perhaps using only a single super-super-class for  $K > K_{min}$  where  $K_{min}$  is some small value) may be used with little degradation in performance. However, at higher bit rates, the super-super-class structure as outlined is necessary for optimal performance.

### **5.7.1.2. Performance comparison of ECPVQ using 4 different filter pairs in the wavelet transform**

Here we shall consider the performance of our ECPVQ method using 4 different filter pairs in the wavelet transform. Although non-linear phase filters have been used in coding applications [13], we shall restrict ourselves to linear phase filters. In chapter 2 we mention the potential subjective advantages of linear phase filters over non-linear phase ones.

We have chosen an example of odd symmetric filters (7/9 tap --used by Said/Pearlman [50] amongst others [38].), even symmetric filters (10/14 tap --said by Ghanbari [75] to give good subjective performance), filters which display almost perfect symmetry in wavelet and scaling function between the analysis and synthesis sides (13/19 tap --developed by Tay and Kingsbury [18]) and the simplest useful filter pair (the Le-Gall 3/5 tap filters [7]).

For each filter pair we have developed separate training data statistics for the various entropy codes.



**Fig 5.30 Shows Lena coded by 4 different filter pairs by the ECPVQ method with an optimal arithmetic and Huffman coders being used for the entropy coding (see next section) The lattice used is the  $Z_{21}/D_{21}$ .**

Fig 5.30 shows that the performance of the 7/9 tap and 10/14 tap filters is almost identical over all bit-rates. The performance of the 3/5 tap filters is surprisingly good compared with the 13/19 tap filters despite the non-smoothness of its analysis scaling and wavelet

functions (figures A.7 and A.9 of appendix A.1) (at about 0.6bpp, the performance is almost the same). The difference in performance between the best two filter pairs and the 13/19 tap filters is about 0.45dB

In appendix A.1, we do a similar performance comparison between the different filters for our less complex benchmark coder (simple first order entropy coding with scalar quantisation). We found that the 7/9 tap filters were slightly superior to the 10/14 tap filters which in turn were superior to the 13/19 tap filters by about 0.32dB (the 3/5 tap filters performed about 0.1dB worse, at 0.42dB).

We can reasonably conclude that judging the performance of filters based on a simple coding scheme is fairly accurate as far as choosing the “best” filters for more complex coding methods.

Sections of 256\*256 coded Lena



[1]

[2]

**Fig 5.31. Shows  $128 \times 128$  sections of coded Lena, both at 0.613bpp using ECPVQ method and  $Z_{21}/D_{21}$  for 2 different filter pairs (the PSNR for both is 34.31dB). [1]-7/9 tap filters and [2]-10/14 tap filters**

Although the PSNR performance of the two coded images in 5.31 is similar, we have observed that one type of artefact, in particular, is more visible when the 10/14 tap filters are used. It is noticeable that there is clearly more ringing visible in image [2] (around the hat) than in image [1]. This is probably due to the basis functions being longer for the 10/14 tap filters (see figures A.6 and A.14 in appendix A.1, of respective synthesis wavelet functions). A graphical evaluation of this hypothesis is suggested in A.1.4.

We have therefore decided to use the 7/9 tap filter pair (in the separable wavelet transform) based on a subjective and PSNR evaluation, for our ECPVQ method. This agrees with the heuristic findings in a recent paper by McCanny [71]. However, a detailed subjective evaluation of 22 bi-orthogonal filter pairs by Ghanbari and Silva [75] (including the 7/9 and 10/14 tap filters) found that the 10/14 tap filters were best (the coding scheme used in the subjective evaluation was a gain/shape lattice vector quantiser [79], with Lena coded at 0.5 bits/pixel).

A recent paper [49] suggests a new bi-orthogonal filter pair, of 10/18 taps, which the researchers find gives superior rate vs. distortion performance over the 7/9 tap filter pair. We also later give some results with that filter and compare subjective performance with the 7/9 tap filter pair. At higher bit-rates, a 28/28 [54] tap linear phase filter pair has been found [55] to give superior performance over the 7/9 and 10/18 tap filter pairs.

### **5.7.1.3 Coding results using ECPVQ method**

We have coded the image **Lena** using 6 different methods. They are 1) ECPVQ method using  $Z_{21}/D_{21}$  lattice 2) ECPVQ method using  $D_{21}$  lattice 3) ECPVQ method using  $Z_{21}$  lattice 4) Method of Said/Pearlman (entropy coded version) [50] 5) Benchmark Coder (A.1) and 6) Baseline JPEG [2].

The entropy coding, of the error coefficient values in the DPCM method, of the “radius” of each quantised vector and of the index of the sub/super/super-super class in which quantised vectors with  $l_1$  norm greater than zero lie, is done using either an

approximate arithmetic coder by Mohuddin [45] (who introduces an approximation to the multiplication operation) or an optimal arithmetic coder based on Mohuddin but using a proper multiplication function [40]. This approach tackles the precision problem by restricting the range of values of the interval register to  $0.75 \leq A(s) < 1.5$  where  $s$  is the current symbol string and  $A(s)$  is the interval register value. We set the width of the code-word and interval registers to be 16 bits long and use the efficient code-word termination method of Young [41]. We have found that when the input symbol stream to the optimal coder contains a large number of symbols, incorrect decoding occurs due to rounding errors introduced by the multiplication operator in our implementation (after encoding a symbol, the values in the interval and code-word registers need to be truncated to integers for an integer arithmetic implementation). To resolve this, we use separate arithmetic (optimal) coders for the 3 entropy codes which need to be transmitted to the decoder, namely the error coefficients from the DC band, context-based/first-order code for “radial” parameter coding, and coding of sub/super/super-super class index (note that the size of the symbol alphabet here is constantly changing, which can be easily incorporated into arithmetic coders). We also check whether proper decoding of the individual bit-streams is possible before transmitting bit-streams to the decoder. If this is not the case, we divide the  $s$  (assuming  $s$  is a power of 2) symbols by a factor of  $2^r$  ( $r = 1, 2, \dots$ ) into separate blocks, of size  $k = s/2^r$  symbols. Each  $k$  block of symbols is then separately coded and the concatenated bit-streams are transmitted (with the value of  $r$  transmitted as header information). At the decoder, the size of the blocks of symbols is obtained from the header and synchronisation of the bit-stream is obtained by the decoder being able to tell how many bits are needed to decode each  $k$  block of symbols. The efficient code-word termination method of Young [41] means that very limited redundancy is introduced by dividing the initial number of symbols,  $s$ .

This truncation problem (known as *underflow*) has been resolved in the practical arithmetic coding implementations of Rubin [85] and Cleary et al. [80].

Original Lena

Scheme [1]



Scheme [2]

Scheme [3]

**Fig. 5.32 Shows Original Lena and Lena coded by 3 different ECPVQ methods**

The following results have been obtained for a bit-rate of 0.613 bits/pixel. The entropy codes have been realised by an optimal arithmetic coder as described previously and the PSNR values of the reconstructed images are given:

*Scheme 1: ECPVQ, using the  $Z_{21}/D_{21}$  Lattice coded at 0.613bpp with the PSNR=34.31dB*

*Scheme 2: ECPVQ using the  $D_{21}$  Lattice coded at 0.613bpp with the PSNR=34.19dB*

*Scheme 3: ECPVQ using the  $Z_{21}$  Lattice coded at 0.613bpp with the PSNR=34.21dB*



**Fig 5.33 Shows original Lena and Lena coded by 3 different methods for comparison purposes.**

Here **Lena** has been coded by 3 different coders for comparison purposes and PSNR results are given.

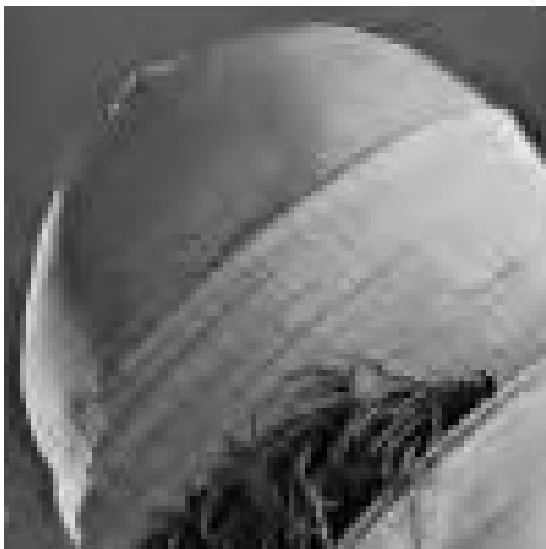
*Scheme 4: Method of Said/Pearlman coded at 0.613bpp with the PSNR=34.08dB*

*Scheme 5: Benchmark coder coded at 0.613bpp with the PSNR=32.94dB*

*Scheme 6: Baseline JPEG coded at 0.617bpp with the PSNR=30.87dB*

Original Lena

[1]



[4]

[6]

**Fig 5.34. 100×100 sections of Lena by 3 different methods**

128×128 sections of **Lena** are shown to illustrate more clearly the subjective differences.

*Scheme 1: ECPVQ using the  $Z_{21}/D_{21}$  Lattice coded at 0.613bpp with the PSNR=34.31dB*

*Scheme 4: Method of Said/Pearlman coded at 0.613bpp with the PSNR=34.08dB*

*Scheme 6: Baseline JPEG coded at 0.617bpp with the PSNR=30.87dB*



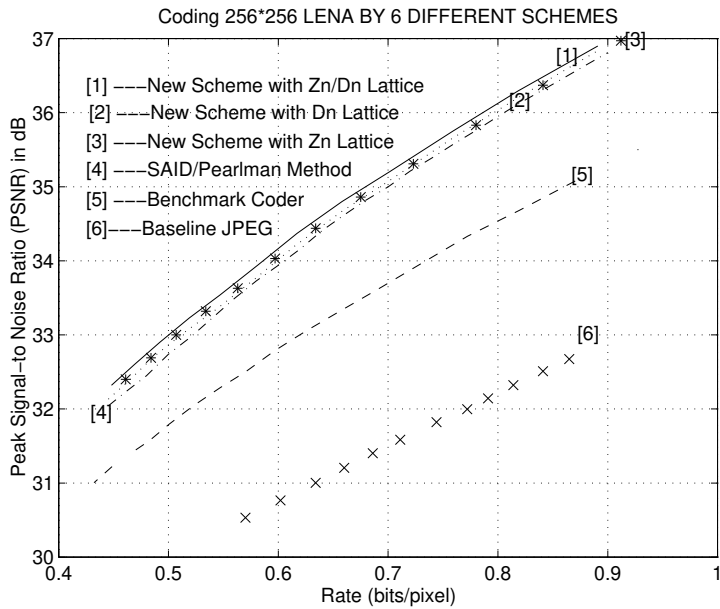
**Fig 5.35.  $80 \times 80$  Sections of Lena by 3 different methods**

Here we wish to compare the subjective performance of the three lattices we have used.

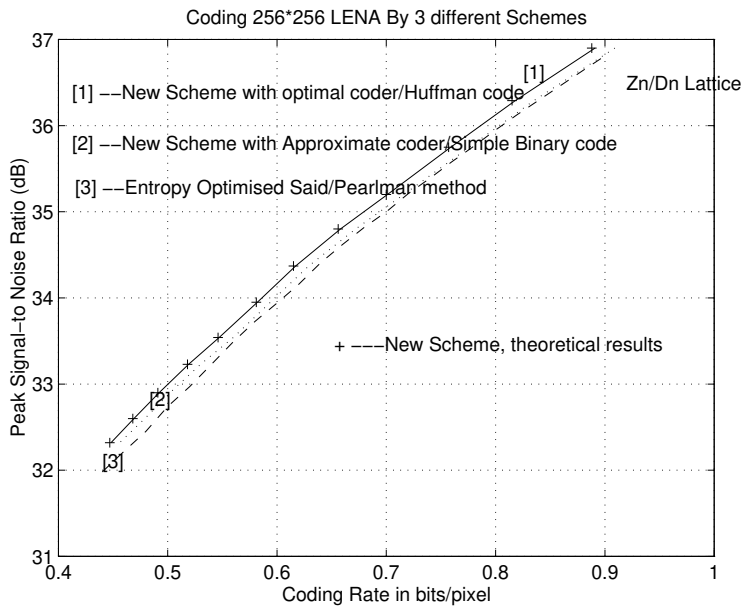
*Scheme 1: ECPVQ using the  $Z_{21}/D_{21}$  Lattice coded at 0.613bpp with the PSNR=34.31dB*

*Scheme 2: ECPVQ using the  $D_{21}$  Lattice coded at 0.613bpp with the PSNR=34.19dB*

*Scheme 3: ECPVQ using the  $Z_{21}$  Lattice coded at 0.613bpp with the PSNR=34.21dB*



**Fig 5.36: Coding Lena across a range of bit-rates for 6 different coders**  
 ([1] -solid line, [2]-dotted line, [3]-'\*', [4]-dashed-dotted line)



**Fig 5.37: Coding Lena using optimal arithmetic coder vs. approximate arithmetic coder** ([1]-solid line, [2]-dotted line, [3]-dashed line)

**Table 5.38: Efficiency (%) of various codes for Lena coded at 0.613bpp by the  $Z_{21}/D_{21}$  lattice**

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
<i>Approximate codes*</i>	99.9	97.5	95.6	98.2
<i>Optimal codes*</i>	99.95	99.97	99.98	99.62

\* The approximate codes for A, B and C are with the approximate arithmetic coder. For D, we use the simple binary code.

\* The optimal codes for A, B and C are with the optimal arithmetic coder. For D, we use the huffman coder.

*A: Efficiency of DPCM Coding*

*B: Efficiency of coding of radial parameter*

*C: Efficiency of coding of sub-class/super-class/super-super-class index*

*D: Efficiency of coding of vector within a particular sub-class/super-class/super-super-class*

Fig 5.32 shows the three decoded images at a bit-rate of 0.613 bits/pixel. The scaling factors for the respective lattices,  $Z_{21}/D_{21}$ ,  $D_{21}$  and  $Z_{21}$  were 28.25, 28.1 and 29.1. The optimal arithmetic was used for entropy coding as described previously. The maximum pyramidal radius (maximum value of  $K$ ) is 50 (for the  $D_{21}$  lattice). A comparison between the performance of the lattices reveals that the  $Z_{21}/D_{21}$  lattice does best (by about 0.1dB from the other two) and the  $Z_{21}$  and  $D_{21}$  lattices perform similarly to each other in PSNR terms (see the next sub-section on a more detailed rate vs. distortion analysis and comments).

Fig 5.35 enables us to do a subjective comparison between the lattices. The decoded image by the  $D_{21}$  lattice shows some undesirable low frequency artifacts, white specks, (around the shoulder region and around the lips) which are not so visible when quantising with the  $Z_{21}$  lattice. This is because, with the  $D_{21}$  lattice, as the sum of the elements of the 21-D vectors must be even, the lowest frequency coefficient may sometimes be forced to zero (instead of having a value of unity with the equivalent  $Z_{21}$  lattice). This subjective problem can be solved by introducing an extra shell near the origin, hence the  $Z_{21}/D_{21}$  lattice. Figure 5.36 shows that  $Z_{21}/D_{21}$  lattice is superior to the other two over all bit-rates in PSNR terms. Therefore both subjectively and in PSNR terms we conclude that the  $Z_{21}/D_{21}$  is the best for coding purposes.

For a comparison of ECPVQ using the  $Z_{21}/D_{21}$  lattice, we find from Figure 5.36, that this method is superior by between 0.19-0.27dB compared with the best method of Said/Pearlman [50]. It should be noted that we have used Said/Pearlman's original encoding/decoding programs (obtained by *ftp* from *ipl.rpi.edu*). This method is generally regarded as at least being amongst the current state of the art compression techniques. A subjective comparison at 0.613bpp can be seen in Figure 5.34, between the ECPVQ coder, Said/Pearlman's coder and JPEG, reveals 2 things. First that there is more detail in the hat region of Lena in the decoded image obtained from the ECPVQ coder than with the decoded image obtained from Said/Pearlman's method. Secondly, there are a lot of blocking artifacts visible in the JPEG decoded image. Its PSNR performance is also about 2.5dB lower than for our method.

In conclusion we can say that our method in terms of subjective and PSNR performance is superior to that of Said/Pearlman for the **Lena** image.

We now consider Fig 5.37. Here we compare the performance using an optimal arithmetic coder, for the 3 entropy codes described previously and a Huffman code as described in A.4.2 to code the index within a particular sub/super-class (an the simple binary coding method for any index within super-super-class) vs. using an approximate

arithmetic coder for the 3 entropy codes and a simple binary code for the index within a particular sub/super/super-super class. We can see that even the less efficient latter method is superior, in terms of PSNR, to the method of Said/Pearlman [50].

The efficiency of the codes is compared in Table 5.38. The dotted line gives the efficiencies using the optimal/Huffman coder method and the solid line gives the efficiencies using the approximate/simple binary coding method. The efficiency is defined as the theoretical rate divided by the actual rate multiplied by 100 to give it as a percentage. We see that the efficiencies for the optimal coder/Huffman coder method range from 99.62-99.95%. The least efficient of the 4 codes is the Huffman code (99.62%). The efficiencies for the approximate/simple binary coding method range from 95.6-99.9%. We now give the number of bits for each code for Lena coded at 0.613 bits/pixel (this is using optimal arithmetic coder/Huffman coder and with the  $Z_{21}/D_{21}$  lattice):

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>
<b>Approximate/Simple binary coder</b>	4835	7259	7312	21353	257
<b>Optimal coder/Huffman coder</b>	4832	7081	6993	21037	257
<b>Theoretical rates</b>	4830	7079	6992	20957	257

Here **A** is the entropy code for the DPCM of dc band, **B** is entropy code for “radial” parameter of quantised vectors, **C** is the entropy code for the index of a specific sub/super/super-super class, **D** is the huffman/simple binary code for the index within a specific sub/super/super-super class and **E** is the side information from the ‘multiple-mu’ method (246 bits) +11 bits for transmitting value of scaling factor (range is [0,100] and we allow 2001 equally spaced values separated by 0.05).

Table 5.39: Comparison of PSNR performance for 4 256×256 Images coded at approximately 0.6 bits/pixel by 2 different methods

IMAGE	Coding Rate	PSNR [1]	PSNR [2]
Peppers	0.56	35.41	35.34
Airplane	0.63	33.97	33.67
Baboon	0.55	26.8	26.59
Sailboat	0.59	30.1	29.68

[1]: ECPVQ using  $Z_{21}/D_{21}$  Lattice

[2]: Entropy Optimised Said/Pearlman Method [50].

We see that the advantage in PSNR terms between the two methods, from table 5.39 is between 0.07dB-0.42dB for the 4 images tested. The PSNR values are in dB.

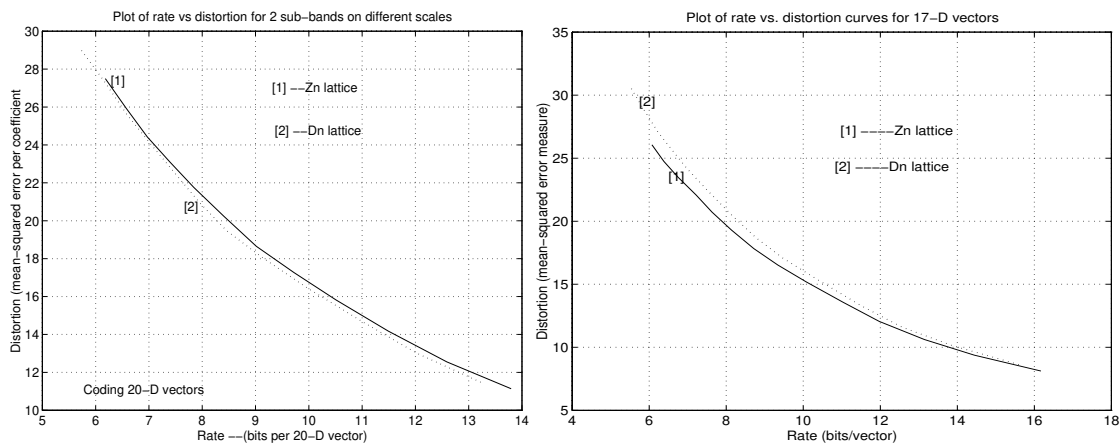
### 5.7.1.3.1. Some comparisons between different lattices used for quantisation in ECPVQ method

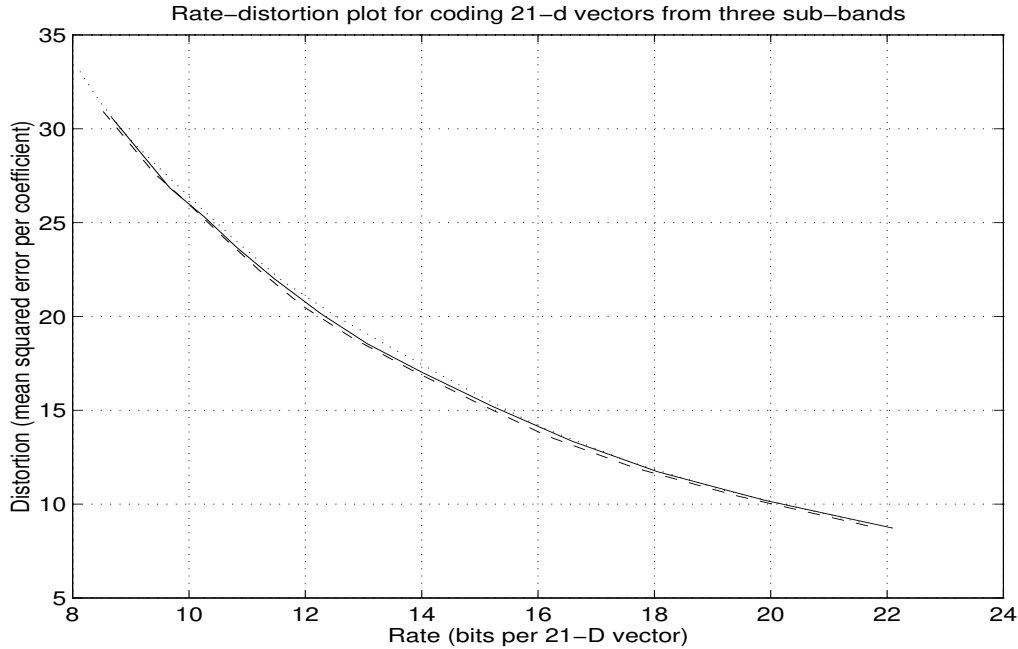
Previously, we have found that the  $Z_{21}$  lattice performs similarly (or slightly better) than the  $D_{21}$  lattice in rate vs. PSNR terms with our ECPVQ scheme. We also found that adding an extra shell (at  $l_1$  norm=1) to the  $D_{21}$  lattice actually improves PSNR performance.

When using scalar quantisers for generalised Gaussian distributed data, one finds [53,56] that increasing the nominal step size around the origin improves coding performance (as the distribution is peaked around origin). As an extension to quantisation in multiple dimensions as the  $D_n$  lattice is less dense than the  $Z_n$  lattice, one would expect the former to perform better. We have found that this is in fact the case when intra-band vectors are used in a lattice quantisation coding scheme, such as Antonini's [33,60] pyramid and spherical lattice VQ schemes (i.e. when the coefficients that each vector is made up of are from the same sub-band). We did a rate vs. distortion comparison on a

128×128 sub-band for 4, 8, 16 and 32-D vectors and found in each case that the  $D_n$  lattice is superior to the  $Z_n$  lattice for quantisation.

However different results are obtained when choosing our 21-D inter-band vectors. An intermediate result is determined for 20-D vectors (removing lowest frequency coefficient from 21-D vectors) and for 17-D vectors (removing the four medium frequency coefficients). We have obtained rate vs. distortion curves based on coding 3 similarly oriented sub-bands of sizes 32×32, 64×64 and 128×128 respectively (and 2 of those subbands of sizes 64×64 and 128×128 for the 20-D case). We use a simpler version of the ECPVQ scheme by assuming all lattice points on a particular pyramidal ( $l_1$  norm constant) shell are equally probable (equivalent to there being only 1 super-super-class on each pyramidal shell). Also a simple first order entropy code is used for coding the index of each shell (rather than a conditional entropy code). The distortion is based on a mean-squared error calculation with the reproduction vectors being determined based on experimental calculation of centroids (not the more complicated ‘multiple mu’ method we use in our ECPVQ scheme).





**Fig 5.40. Performance comparison between the  $Z_{21}$ ,  $D_{21}$  and  $Z_{21}/D_{21}$ ,  $Z_{20}$  and  $D_{20}$ , and  $Z_{17}$  and  $D_{17}$  lattices based on rate vs distortion curves for 3 horizontally oriented sub-bands. (In the graph where 21-D vectors are coded the following notation is used: solid line- $Z_{21}$  lattice, dotted line- $D_{21}$  lattice, dashed line- $Z_{21}/D_{21}$  lattice)**

Fig 5.40 shows that the  $D_{20}$  lattice performs better than the  $Z_{20}$  lattice (although performance is closer than for coding intra-band vectors) but the reverse is true for the  $D_{21}$  and  $Z_{21}$  and  $D_{17}$  and  $Z_{17}$  lattices (in our ECPVQ scheme the performance of the  $Z_{21}$  and  $D_{21}$  lattices is similar because we have found (see 5.6.2) that the ‘multiple-mu’ method for determining reproduction vectors does better (compared with a centroid approach) for the  $D_{21}$  lattice). We have also found that the  $Z_{21}/D_{21}$  lattice performs better than the  $Z_{21}$  lattice (i.e. for outer shells we use a less dense lattice, the  $D_{21}$  lattice)

The 21-D vectors contain 3 distinct groups of coefficients (1,4 and 16 elements with different variances) and the 20-D and 17-D vectors 2 distinct groups. So we feel that the main factor influencing this behaviour is the difference in variance between the

different distinct groups of coefficients. For example, the different in variance between the two distinct groups of coefficients in the 17-D vectors (i.e. 1 and 16 elements) is considerably greater than between the two distinct groups of coefficients in the 20-D vectors (i.e. 4 and 16 elements). This may explain why the  $D_{20}$  lattice performs better than the  $Z_{20}$  lattice but the  $D_{17}$  lattice seems inferior to the  $Z_{17}$  lattice.

To obtain an improvement on the  $Z_{21}$  lattice, we have found we need to quantise the vectors on the outer shells with the  $D_{21}$  lattice, i.e. our  $Z_{21}/D_{21}$  lattice.

### 5.7.2 Coding of $512 \times 512$ images

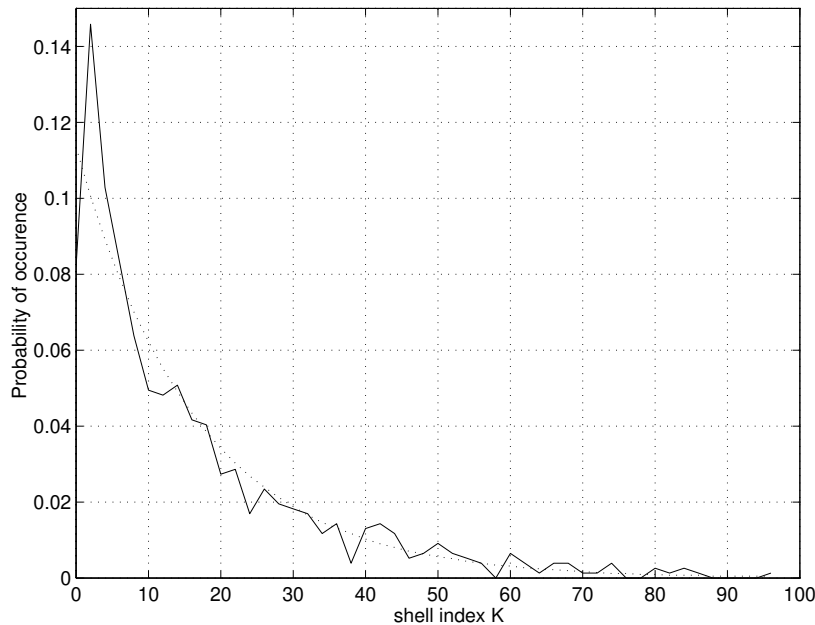
Here we shall consider coding the well known  $512 \times 512$  images **Lena** and **Goldhill**. We first describe the modifications to the coding scheme that we have adopted. We adopt the following modifications:

- ◆ Use a 5-level transform instead of 3-levels.
- ◆ The 6 a.c. sub-bands from the coarsest two levels form 3 blocks of 5-D vectors (total of 768) which are coded using a simple first order entropy code for the “radial” parameter, an entropy code for the index of the super-class and a huffman code (A.4.2) for the index within a super-class.
- ◆ The dc band of size  $16 \times 16$  is coded using the DPCM method of 5.6.1.
- ◆ The remaining sub-bands on levels 3-5 contain 3 blocks of 21-D vectors which are coded as previously except each block of vectors is of size  $64 \times 64$  instead of size  $32 \times 32$  for  $256 \times 256$  images. The total number of vectors is therefore 12288.

### 5.7.2.1. Coding of 5-D vectors

These vectors are formed by the method described in section 5.4. The reasoning behind choosing 5-D vectors is shown later. To code these vectors we need to code firstly the “radial” parameter.

Here,  $K$  is the  $l_1$  norm of the 5-element quantised vectors. We now show the probability distribution of  $K$  for the case where the lattice used to quantise these 5-D vectors is the  $D_5$  lattice and  $\Delta=30$ .



**Fig 5.41: Plot of distribution of radial parameter for 5-D vectors**

Here we group to-gether the 3 blocks of 256 5-D vectors (i.e. those vectors containing horizontal orientation, vertical orientation and diagonal orientation information). We have found no advantage in designing entropy codes separately for the 3 blocks. The modelled probabilities are again based on a decaying exponential fitted to the data (as we did for part of the range of values of  $K$  in the case of coding 21-D vectors). Here the maximum value of  $K$  is 96 as seen in Fig 5.41. The exponent of the exponential is transmitted to the

decoder along with maximum value of  $K$ . We have not considered designing a more complicated higher order code (as in the 21-D vector case) because the total number of bits taken up by coding the values of  $K$  is not significant. The modelling causes the entropy to increase by 1.5% in this case.

Now, we need to code the position information of quantised vectors that are non-zero ( $K>0$ ). We do this by means of super-class structures (we don't consider sub-class or super-super-class structures as for 21-D vectors for simplicity and because total number of bits needed to code this index is relatively small). The super-classes on these 5-D vectors are denoted by  $(r_1, r_4)$  where  $r_1 + r_4 = K$  and  $r_1$  is magnitude of first element of 5-D quantised vector and  $r_4$  is  $l_1$  norm of the next four elements (as 5-D vectors consist of 2 distinct groups of coefficients, one element and 4 elements). Thus, the total number of super-classes on a shell indexed by  $K$  is given by  $K+1$  (e.g. let  $K=2$ , then super-classes are  $\{(2,0),(1,1),(0,2)\}$ ). Enumeration encoding and decoding algorithms for indexing specific super-classes (index in range  $[0, K+1-1]$ ) are straightforward to design.

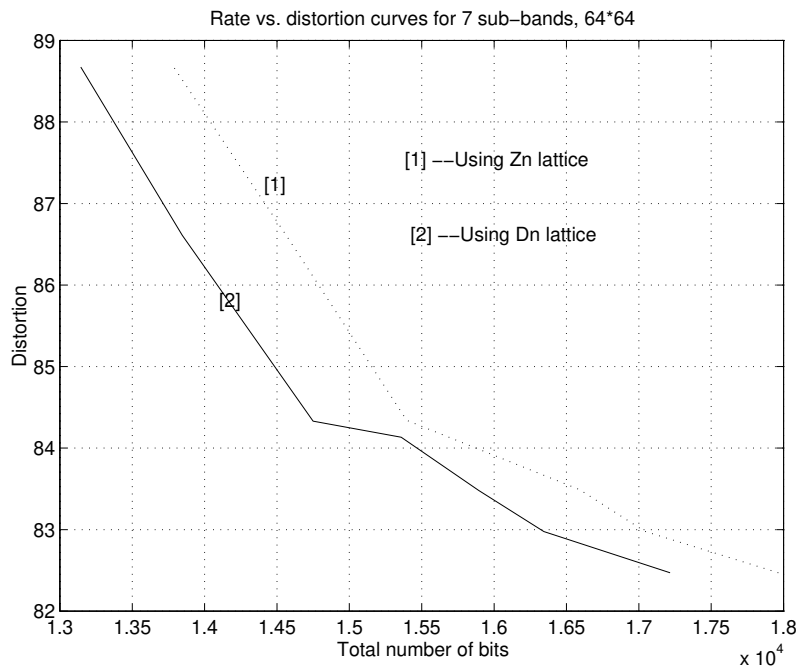
Finally for quantised vectors whose  $l_1$  norm is greater than zero, we want to code the index within the specific super-class. The number of possible code-vectors in a super-class denoted by  $(r_1, r_4)$  is given by the following: If  $r_1=0$ , it is  $N(4, r_4)$ . If  $r_4=0$ , it is 2 and if  $a_1$  is not zero, it is  $2N(4, r_4)$  where  $N(L, K)$  is defined in equation 5.9 (number of integer code-vectors of dimension  $L$  on pyramid with  $l_1$  norm of  $K$ ). The enumeration encoding/decoding algorithms are based on encoding/decoding algorithms for finding index of code-vector of dimension  $L$  on a pyramidal shell with radius  $K$ . One method is Fisher's [66] and another is given in appendix A.8. This index is coded with a huffman code (A.4.2) for  $K \leq 30$  and a simple binary code for any  $K > 30$ . This is because at low bit-rates there are few 5-D vectors lying on those outer shells and thus any inefficiency does not affect the coding rate.

We have found that grouping neighbouring 5-D vectors to form 10-D vectors does not improve performance at all. The reason is that the decay towards zero of the

probability distribution for  $K$  is much slower than for the 5-D vectors (maximum  $K=96$ ). Therefore the proportion of quantised vectors lying on shells near the origin is much less than for the 5-D vectors (so more position information bits are required).

The decay of the pdf for the radial parameter of the 5-D vectors is slower than for the 21-D vectors because the elements of these 5-D vectors have much higher variance than the elements from the 21-D vectors because they represent lower frequency information.

We now briefly consider the question of choice of lattice to quantise these 5-D vectors.



**Fig 5.42: rate vs. distortion curves for 2 lattices (quantising 5-D vectors)**

Fig 5.42 shows the rate (in bits) vs. distortion curves for the 7 sub-bands on levels 1-2 (i.e. DC band and 6 A.C. subbands from which the 768 5-D vectors are formed). The distortion is based on the mean-squared error of this  $64 \times 64$  region (where centroids are used in the 6 A.C. sub-bands to minimise the m.s.e. in each sub-band respectively). The

rate is based on the theoretical coding rate for the D.C. band and the 768 5-D vectors from the 6 A.C. sub-bands. Fig 5.39 shows that the  $D_5$  lattice is superior to the  $Z_5$  lattice. We subsequently found that the  $Z_5/D_5$  lattice is comparable in performance to the  $D_5$  lattice and have used this to match with the  $Z_{21}/D_{21}$  lattice used for coding the 21-D vectors.

### 5.7.2.2. Coding results for images *Lena* and *Goldhill*

The following modifications/improvements have been made to the version of the ECPVQ algorithm which we used to generate results in the previous section (which generated results for the  $256 \times 256$  images).

- 1) The ‘mu’ values (section 5.6.2) are differentially encoded.
- 2) The scaling factor used in the quantisation of the 5-D vectors and the DC band is 25% lower than that used in the quantisation of the 21-D vectors (see section 5.7.2.3).
- 3) The training set probabilities for indexing of sub-classes on shells with  $l_1$  norm of 1,2,4,6 and 8 are dynamically adapted during the coding process using methods similar to those described by Cleary (for shell 1) [80] and Young (for shells 2,4,6 and 8) [41].

We give the coding results for the well-known images **Lena** and **Goldhill** (the versions used are those which Said/Pearlman [50] and Shapiro [81] have used) and compare performance with the published results of Said/Pearlman [50], Fisher [65] and most recently Xiong, Ramchandran and Orchard [51] for these images. All results are obtained by use of the optimal arithmetic coder for entropy codes or the Huffman coder (for encoding index within a sub-class/super-class for 21-D vectors and a super-class for the 5-D vectors).

### Coding Lena

bit-rate (bits/pixel)	0.15	0.2	0.25	0.258	0.327	0.5
PSNR [1A] (dB)	32.11	33.36	34.31	34.46	35.49	37.32
		(33.37)	(34.33)			(37.35)
PSNR [1B] (dB)	-----	33.50	34.44	-----	-----	37.41
		(33.51)	(34.46)			(37.44)
PSNR [2] (dB)	31.90	33.16	34.12	34.26	35.33	37.22
PSNR [3] (dB)	-----	-----	-----	33.7*	34.8*	-----
PSNR [4] (dB)	-----	33.32	34.33	-----	-----	37.36
PSNR [5] (dB)	-----	-----	34.53	-----	-----	37.66
PSNR [6] (dB)	-----	<b>33.57</b>	<b>34.57</b>	-----	-----	<b>37.69</b>

\* measured from Fig. 7 in [65]

It should be noted that the PSNR values in brackets are based on coding a **Lena** used by Ortega and Chrysafis [52] (obtained from the USC database and also used by Joshi et al [48], Xiong. [51]) and is slightly different from that used by Said and Pearlman and Shapiro [50,81].

Here, [1A] is the ECPVQ method using 5-D and 21-D vectors,  $Z_n/D_n$  lattice with coding by optimal arithmetic/Huffman coder and the 7/9 tap filter pair being used in a 5-level dyadic wavelet transform, [1B] is the ECPVQ method with the 10/18 tap filter pair [49] being used in the wavelet transform, [2] is best method of Said/Pearlman [50], [3] is the method of Fisher [65] described in 5.1.1. based on the “weighted” pyramid for coding 63-D vectors and “ordinary” (non-weighted) pyramid for coding 32 or 64-D vectors (these PSNR values are measured from his paper [65]) and [4] is a recent algorithm known as space-frequency quantisation (SFQ) of Orchard et al. [51]. We see that the ECPVQ method ([1A]) outperforms the method of Said/Pearlman by between 0.1-0.21dB with the biggest gains at the low bit-rates of 0.15 and 0.25 bits/pixel. ECPVQ ([1A])

outperforms Fisher's [65] method by about 0.76dB at 0.258 bits/pixel. Orchard's method outperforms ECPVQ ([1A]) by about 0-0.01dB at rates of 0.25b/p and 0.5b/p but ECPVQ ([1A]) is superior by 0.05dB at 0.2 bits/pixel.

The method described as [5] is by Chysafis and Ortega [52] and is based on the context-based coding of scalar quantised wavelet coefficients (using a dyadic wavelet transform with 10/18 tap filters described by Villasenor [49]). The method described as [6] are the best current results recorded at the UCLA web site [55] for wavelet based coders (the PSNR number for 0.2 bits/pixel was obtained from the Estimation Quantisation algorithm [53]). We see that ECPVQ ([1B]) is between 0.07-0.22dB inferior to the method of Ortega et al. [52] with the best results at the lower bit-rate of 0.25 b/p. We can also observe that ECPVQ ([1B]) is at most **0.06-0.25 dB** inferior to the best current (web site results obtained in Mar. 1999) quoted results.

The maximum value of the radial parameter (i.e.  $l_1$  norm of quantised 21-D vector) for Lena coded respectively at 0.25, 0.5, 0.75 and 1.0 bpp is 26, 54, 76 and 94. Therefore as seen in 5.7.1.1., a simpler version of the **super-super-class** structure can be used at low bit-rates (as the super-super-class method is only used for  $K$  greater than 22). However at 0.5, 0.75 and 1.0 bit/pixel, the percentage of quantised 21-D vectors lying on "outer" shells (those with  $K > 22$ ), is respectively 2.5%, 5.5% and 8.5%. So, particularly at coding rates of 0.75bpp and 1.0bpp, the **super-super-class structure** is important for optimal performance.

### Coding Goldhill

<b>bit-rate (bits/pixel)</b>	<b>0.2</b>	<b>0.25</b>	<b>0.327</b>	<b>0.5</b>
<b>PSNR [1A] (dB)</b>	29.95	30.67	31.81	33.32
<b>PSNR [1B] (dB)</b>	<b>30.03</b>	<b>30.75</b>	-----	<b>33.39</b>
<b>PSNR [2] (dB)</b>	29.85	30.56	31.65	33.13
<b>PSNR [3] (dB)</b>	-----	30.27	-----	33.05
<b>PSNR [4] (dB)</b>	29.94	30.71	-----	33.37
<b>PSNR [5] (dB)</b>	30.07	30.80	-----	33.51
<b>PSNR [6] (dB)</b>	-----	<b>30.86</b>	-----	<b>33.53</b>

Here we see that ECPVQ ([1A]) outperforms Said/Pearlman's method by between 0.1-0.19dB. ECPVQ also outperforms Fisher's method by 0.4dB at 0.25 bits/pixel. It is inferior by 0.04-0.05 dB to Orchard's SFQ algorithm at 0.25b/p and 0.5b/p but superior by 0.01dB at 0.2 bits/pixel. ECPVQ ([1B]) is inferior by between 0.05-0.12dB to the context-based method of Ortega et al. [52].

Comparing performance against the best results at the UCLA web site [55], shows ECPVQ (with the 10/18 tap filters) inferior by about **0.11-0.14dB**.

We now do subjective comparisons on the reconstructed images for both **Lena** and **Goldhill**. The filters used, when not stated otherwise, are the 7/9 tap ones (see A.1. for coefficients).

Original 512\*512 Lena

[1]



Original 512\*512 Goldhill

[2]

**Fig 5.43 [1] --512x512 Lena coded at 0.25bpp by new ECPVQ method with PSNR=34.31dB.**

**[2] --512x512 Goldhill coded at 0.5bpp by new ECPVQ method with PSNR=33.32dB.**



**Fig 5.44: Comparison between 256×256 sections of Lena coded by 3 different methods**

*Scheme 1: New ECPVQ method, coded at 0.25bpp with PSNR =34.31dB*

*Scheme 2: Method of Said/Pearlman coded at 0.25bpp with the PSNR =34.12dB*

*Scheme 3: Baseline JPEG coded at 0.255bpp with the PSNR =31.67dB*

Original 512\*512 Goldhill



[1]



[2]



[3]

**Fig 5.45: Comparison between 256x256 sections of Goldhill coded by 3 different methods**

*Scheme 1: New ECPVQ method, coded at 0.5bpp with the PSNR =33.32dB*

*Scheme 2: Method of Said/Pearlman coded at 0.5bpp with the PSNR=33.13dB*

*Scheme 3: Baseline JPEG coded at 0.5bpp with the PSNR =31.68dB*

[1]



[2]



[3]



[4]

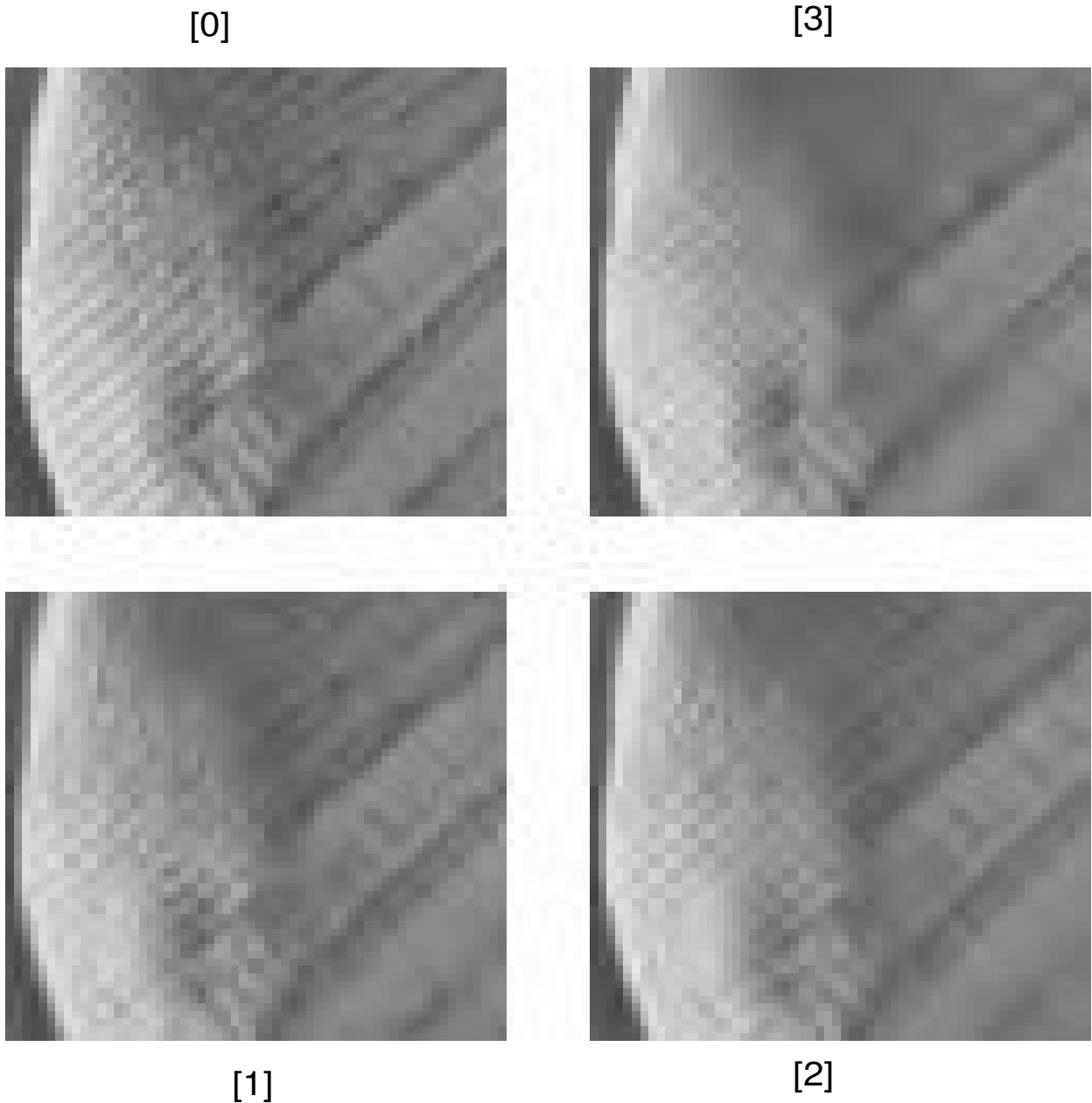
**Fig 5.46a: Comparison between 128×128 sections of Goldhill and Lena coded by two different methods using the 10/18 tap filter pair.**

[1] --Coded Lena by ECPVQ method at 0.25bpp with PSNR=34.46dB

[2] --Coded Lena by context-based method [52] at 0.25bpp and PSNR=34.53dB

[3] --Coded Goldhill by ECPVQ method at 0.5bpp with PSNR=33.39dB

[4] --Coded Goldhill by context-based method [52] at 0.5bpp with PSNR=33.51dB



**Fig 5.46b: Comparison between 60×60 sections of Lena coded by two different methods ([1] and [2]) using the 10/18 tap filter pair and one method ([3]) using the 7/9 tap filters.**

[0] -Original Lena (from USC database)

[1] --Coded Lena by ECPVQ method at 0.25bpp with PSNR=34.46dB

[2] --Coded Lena by context-based method [52] at 0.25bpp and PSNR=34.53dB

[3]--Coded Lena by method of Said/Pearlman [50] at 0.25bpp and PSNR=34.14dB

Fig 5.43 shows a comparison between the original Lena image and Lena coded at 0.25bpp with ECPVQ. It also shows a comparison between the original Goldhill image and Goldhill coded at 0.5bpp. Fig 5.44 allows us to do a subjective comparison between the Lena image coded by ECPVQ, Said/Pearlman and JPEG. We see in the JPEG image a lot of blocking artifacts as expected. Its PSNR value is also 2.64dB less than the equivalent value when coded by ECPVQ. The image coded by ECPVQ contains more detail in the hat region of Lena compared with the image coded with Said/Pearlman's method. There is also slightly less ringing around the lip region to which the arrow points.

In Fig 5.45, the JPEG image looks considerably less degraded than the corresponding JPEG coded Lena image because the rectangular objects in Goldhill hide the blocking artifacts. Its PSNR value is 1.64dB lower than for the ECPVQ method. The major subjective difference between the ECPVQ coded image and Said/Pearlman's is that there is a visually annoying artefact on some of the roof tiles, pointed by the arrow, in Said/Pearlman's decoded image. Using ECPVQ this artefact is much less severe and the texture is well preserved. There are also other regions in the ECPVQ coded Goldhill image where texture is better preserved (e.g. some of the shrubbery is more defined and the tiles on the roofs of other houses also contain more detail).

In Figs. 5.46a and 5.46b, we can see a comparison between our method and the recent context-based scalar quantisation method of Ortega and Chrysafis [52] which is currently amongst the very best coders [55] (images [2] and [4] from 5.46a were obtained directly from the authors). Comparing between the two coded Lena images, we find that ECPVQ is inferior by 0.07dB in terms of PSNR but there are certain areas which contain more texture detail. For example, below both eyelids two distinct eyelashes can be seen in [1] but they seem to have merged in [2] (Fig. 5.46a). Also in certain parts of the hat area, there is more texture detail in [1] compared with [2] (there is more fine detail in some of the lines in the hat region in [1] -Fig 5.46b) . Comparing between the two coded Goldhill images, [3] and [4], we again observe that there is a PSNR difference of 0.12dB.

However, again we can see that some of the roof tiles (top left-sloping roof) in [3] contain more texture information than those in [4]. Certain background trees in [3] also contain slightly more detail. We have also observed that an object in the top right hand corner of [4] is more sharply defined than the same object in [3] which indicates that though the subjective quality of [3] would appear to be better in most areas than [4], there is at least one region of [4] which appears to be better coded than in [3].

We performed a subjective evaluation of Lena coded with both the 7/9 and 10/18 tap filters at 0.25 bits/pixel. We found evidence of more ringing distortion in the 10/18 tap filter coded image. We also found that the texture detail in the hat region was better coded by the 10/18 tap filters but the fine hair on Lena's shoulder was better coded with the 7/9 tap filters. So, it is unclear where the superior rate/distortion performance of the 10/18 tap filters results in superior perceptual quality.

The reasoning behind the apparent better preservation of texture detail when coding using our ECPVQ method as opposed to the method of Said/Pearlman [50] is as follows: In the method of Said/Pearlman, the quantisation employed is equivalent to a uniform scalar quantiser with a quantisation region or "dead-zone" width at the origin of  $2\Delta$  where  $\Delta$  is the nominal step size. Therefore low magnitude transform coefficients are more coarsely quantised with respect to larger magnitude coefficients.

In Orchard's SFQ method, coefficients with low energy are also likely to be more coarsely quantised than higher magnitude ones (the basis premise of his algorithm is this: The frequency transform used is a dyadic wavelet using the 7/9 tap filters. A subset of mostly low-valued wavelet coefficients are selected to be set to zero by what is called zerotree spatial quantisation which sets to zero tree-structured groups of coefficients. The remainder are uniformly scalar quantised using step size  $q$  with the width of the quantisation region at the origin being  $2q$ . The two operations are performed using a jointly optimised rate vs. distortion approach).

In Chrysafis and Ortega's [52] method, the quantisation used is as follows: a uniform mid-tread scalar quantiser with step size  $q$ . The optimal quantisation reproduction level is chosen not to minimise distortion but to minimise a Lagrangian  $J = D + \mu R$  where  $D$  is the distortion associated with a particular reproduction level and  $R$  is the rate needed to transmit that information to the decoder ( $\mu$  is a parameter greater than zero). Therefore, for example, it is likely that some coefficients falling between  $\pm q/2$  and  $\pm 3q/2$  are set to zero for rate-distortion reasons.

In our ECPVQ method where the  $Z_{21}/D_{21}$  lattice is used for quantisation of the 21-D vectors, the low energy coefficients are likely to be quantised (slightly) more finely than the larger magnitude ones. Since the fine texture detail is likely to be contained within the highest frequency sub-bands (where the coefficients are generally of quite low energy), our method is less likely to set these coefficients to zero. This results not only in the preservation of more detail (i.e. improved subjective quality) but also competitive rate vs. PSNR performance, as we have shown in the reconstructed **Lena** and **Goldhill** images, particularly at low bit rates.

### 5.7.2.3 Bit allocation between sub-bands

It should be noted, that in our ECPVQ scheme, quantisation of the 5-D vectors (i.e. the 3 blocks of vectors consisting of horizontally, vertically and diagonally oriented coefficients), the 21-D vectors (consisting of 3  $64 \times 64$  sized blocks) and the dc band is done using the same scaling factor  $\Delta$ . This means an implicit bit allocation strategy between the 16 sub-bands is used. Some authors. [57,58,59] have tried to allocate bits using the "equal-slope" method which has been found [58] to be optimal for allocating bits to individual sub-bands obtained from an orthogonal transform in the mean-squared error sense. This method attempts to find the best allocation of bits for different subbands such as to minimise the overall distortion whilst constraining the total bit-rate to some fixed value.

We have attempted to assess the performance of using a common scaling factor by the following simulations: At two different bit-rates, (for different images, Lena and Goldhill), we have measured the slopes of 7  $D(R)$  or distortion vs. rate curves to assess the optimality of using a common scaling factor. The 7 curves relate to respectively the DC sub-band, coding of the 3 arrays of 5-D vectors (each involving 2 sub-bands) and coding of the 3 arrays of 21-D vectors (each involving 3 sub-bands). The quantisation is by the  $Z_{21}/D_{21}$  or  $Z_5/D_5$  lattices as appropriate in all cases. The “equal-slope” method stipulates that the overall system should operate by finding a point of equal slope along each separate  $D(R)$  curve.

We previously found that  $\Delta=36.8$  when Lena is coded at 0.25bpp. We measured the slope by taking two distortion and two rate measurements for  $\Delta=36.8$  and  $\Delta=40.0$  and assuming a linear relationship between the two points on the distortion vs. rate curve. Table 5.47. shows the results.

**Table 5.47: Tabulation of distortions and rates for each distortion vs. rate curve for image Lena.**

$D(R)$ curve	$\Delta=36.8$	$\Delta=40.0$	slope (bits <sup>-1</sup> )
Curve [1]	104.64 (6.644)	134.74 (6.570)	-406
Curve [2]	132.65 (3.953)	152.62 (3.841)	-177
Curve [3]	118.84 (2.740)	137.84 (2.627)	-167
Curve [4]	108.09 (2.697)	121.89 (2.597)	-139
Curve [5]	29.30 (0.325)	32.42 (0.295)	-105
Curve [6]	20.68 (0.158)	22.09 (0.142)	-88
Curve [7]	15.93 (0.116)	16.91 (0.107)	-98

The values in brackets in Table 5.47 are actual rates (in bits/coefficient), obtained from the arithmetic/huffman encoders and the other values are distortions (per coefficient).

Curves [1]..[7] represent the distortion vs. rate curves for respectively the DC band, the vertically, horizontally and diagonally oriented 5-D vectors (2 sub-bands each), and the vertically, horizontally and diagonally oriented 21-D vectors (3 sub-bands each).

We now tabulate similar results for the Goldhill image in table 5.48 where  $\Delta=29.5$  corresponds to a bit-rate of 0.5bpp.

**Table 5.48: Tabulation of distortions and rates for each distortion vs. rate curve for image Goldhill.**

<i>D(R)</i> curve	$\Delta=29.5$	$\Delta=32.5$	slope (bits <sup>-1</sup> )
Curve [1]	67.88 (6.667)	85.64 (6.560)	-167
Curve [2]	80.30 (3.703)	97.56 (3.572)	-130
Curve [3]	86.77 (4.048)	100.64 (3.921)	-109
Curve [4]	76.91 (2.985)	92.00 (2.859)	-119
Curve [5]	32.69 (0.558)	36.61 (0.491)	-58
Curve [6]	33.38 (0.603)	37.85 (0.535)	-65
Curve [7]	19.94 (0.183)	21.39 (0.155)	-53

The nomenclature used in table 5.48 is the same as that used in table 5.47. Based on the results contained in 5.47-5.48, it would appear that the one definitive conclusion that can be made is that for optimal mean-squared error performance (for a given overall bit-rate), the lower frequency sub-bands (i.e the DC band and the 6 A.C. sub-bands on the coarsest two decomposition levels) should be quantised more finely (i.e with a smaller scaling factor) than the 9 medium-high frequency sub-bands (represented by the 21-D vectors). However, in the case of the Lena image, quantised with  $\Delta=36.8$ , the DC band and the sub-bands represented by the 5-D vectors contribute only about 21% of the total bits to the overall bit-rate of 0.25 bits/pixel. For the Goldhill image, quantised with  $\Delta=29.5$ , the contribution of these sub-bands drops to about 12%.

Therefore, it is reasonable to conclude that using a reduced value of  $\Delta$  for the 7 lowest frequency sub-bands would result in only minimal improvement in bit-rate vs. PSNR results. To test this assumption, we coded Lena using  $\Delta=36.8$  for the 9 highest frequency sub-bands and  $\Delta=27.5$  for the 7 lowest frequency sub-bands (it was found that reducing the value of  $\Delta$  by about 25% for quantising the lowest frequency sub-bands resulted in the slopes of the distortion/rate curves of the DC band and the 5-D vectors being closer to the slopes of the distortion/rate curves of the 21-D vectors). This resulted in a rate of 0.256 bits/pixel and the PSNR of the decoded image being 34.44dB. However, using a common scaling factor of  $\Delta=35.9$  for all sub-bands results in the same rate but a PSNR value of 34.41dB. Therefore, quantising the higher frequency sub-bands more coarsely than the lower frequency sub-bands resulted in an improvement of only about 0.03dB.

We can conclude that using a common scaling factor for lattice quantisation of all the sub-bands is a reasonable low complexity solution to the bit allocation problem when coding using our ECPVQ method, particularly when coding at high bit-rates.

However, we have incorporated the modification of using a scaling factor for the 5-D vectors which is 25% less than the scaling factor used to quantise the 21-D vectors.

### **5.7.3. Computational complexity of ECPVQ**

We have implemented our ECPVQ scheme using MATLAB. Whilst this is a good development environment, typically code runs several orders of magnitude slower than if written using a language such as C/C++. We have compared the encoding/decoding times of our method with the method of Said/Pearlman also written in MATLAB. We also gives the encoding/decoding times for the C++ version of Said/Pearlman to illustrate the difference between MATLAB and C++ code and would expect a similar speed-up of ECPVQ if that were implemented in C++.

**Table 5.49: Encoding/Decoding times and flop counts on a 225 MHZ pentium processor for  $512 \times 512$  Lena coded at 0.25 bits/pixel where S/P [1] and S/P [2] are respectively the method of Said/Pearlman implemented using Matlab code or using C++ code.**

	<i>Encoding/Decoding times</i>		<i>Encoding/Decoding Mega flop counts</i>	
	<b>Encoder</b>	<b>Decoder</b>	<b>Encoder</b>	<b>Decoder</b>
<b>ECPVQ</b>	27 mins	19 mins	64	49.5
<b>S/P [1]</b>	30 mins	17 mins.	55.5	30.0
<b>S/P [2]</b>	1.5 sec	0.6 sec	---	---

Table 5.49 shows that the complexity of ECPVQ is comparable with the Said/Pearlman method. The flop counts are in units of millions and are defined as the cumulative sum of the floating point operations (additions and subtractions count one flop if result is real or two if result is not real, and multiplications and divisions count as one flop each if result is real and 6 flops if it is not). The reason for the increased complexity of the encoder is mainly due to the calculation of the ‘mu’ values in the method described in 5.6.2. Also at the encoder, we test the bit-stream generated by the arithmetic encoder for decodability (to ensure no rounding error problem as previously mentioned). To reduce the complexity of the encoder (to make it comparable with the complexity of the decoder), a simpler method other than the ‘multiple-mu’ approach (as described in 5.6.2) can be used, for the generation of reproduction vectors at the decoder, for low bit-rate coding with little degradation in PSNR performance.

The computational complexity of the encoder of Orchard, Ramachandran and Xioul’s [51] method is stated to be 8 times greater than that of the Said/Pearlman encoder. The computational complexity of the context-based coding method of Ortega and Chrysafis [52] is stated to be comparable to that of the Said and Pearlman [50] method.

### Conclusions and suggestions for future research

---

In this chapter we shall summarise the conclusions which can be drawn from the work of the previous chapter. Some suggestions for future research are also given

#### 6.1 Conclusions

We have found that ECPVQ (the coding algorithm developed in the previous chapter) is competitive with current state-of-the-art wavelet-based coders [48,50,51,52,53,55], particularly at low bit rates, in terms of PSNR and is likely to be at least comparable in terms of subjective performance (and superior to Said/Pearlman [50] and Chrysafis and Ortega's [52] methods). The PSNR advantages (compared to method of Said/Pearlman [50]) are greatest for some of the  $256 \times 256$  images tested.

We have also found that for the 21-D vectors we use, the  $Z_{21}/D_{21}$  lattice outperforms in terms of PSNR both the  $D_{21}$  and  $Z_{21}$  lattices. From a subjective point of view, the  $Z_{21}/D_{21}$  lattice is also better than the  $D_{21}$  lattice. Also it should be noted that the  $Z_{21}$  lattice only performs about 0.1dB worse in PSNR terms than the best  $Z_{21}/D_{21}$  lattice. As the  $Z_n$  lattice is equivalent to quantisation by a uniform mid-tread quantiser, it is clear that the good performance of our ECPVQ algorithm is derived from the efficient nature of the entropy coding (both the context-based conditional entropy code for coding the 'radial' parameter (or index of each shell) where a wide range of correlation information was taken into account and the sub/super/super-super class based entropy coding of "position" information).

High performance current state-of-the-art rate vs. PSNR coders such as those by Said/Pearlman [50], Orchard et al. [51] and Chrysafis and Ortega [52] apply more coarse

quantisation to low energy coefficients near the origin which can result in less detail being preserved in the reconstructed images. In our ECPVQ method we in fact reverse this, by using the  $Z_{21}/D_{21}$  lattice for quantisation of the 21-D vectors which are formed from the 9 low, medium and high frequency sub-bands, by quantising those low energy coefficients more finely whilst retaining a good rate vs. distortion performance.

We have shown that the approximations introduced in the derivation of the *super-class* structure are reasonably valid, as they result in only a small degradation in performance, when compared with the more general *sub-class* structure.

We have also shown how to extend the ECPVQ scheme from coding 256×256 images to the more widely coded 512×512 size images by introducing additional inter-band vectors of dimension five to be coded.

We have discovered that the computational complexity of ECPVQ is comparable to that of Said/Pearlman's [50] method which is not the case for Xioul, Ramachandran and Orchard's [51] space-frequency quantisation (SFQ) technique (they say that their encoder is eight times more complex).

Thus, we can say that ECPVQ is likely to be among the current state-of-the art compression algorithms in terms of rate vs. distortion, visual and computational performance.

## 6.2 Suggestions for future Research

- ◆ Our New ECPVQ method can be extended to treat video sequences. In an image sequence there are typically high correlations between consecutive frames of the sequence as well as spatial correlations which exist within each frame. Using Motion compensated predictive coding, the prediction error frames can be coded using ECPVQ. However, because the statistics of these prediction error images are considerably different from those of single frames, some adaptations of the models underlying our method may need to be changed and in addition, extra context

information (from previously coded frames) can be introduced which is not available when coding just a single frame. A comparison with current standards such as H.263 and other effective current research methods could then be undertaken.

- ◆ We have based our method on a 2-D separable wavelet transform. However, an alternative non-separable Wavelet Transform also exists called a quincunx wavelet transform which does not favour the horizontal and vertical orientations. Relatively few authors [60] have attempted lattice quantisation based on such a method. One difference with the dyadic transform is that the hierarchical relationship between a parent node and its children differs. With the quincunx method, one parent is related to 2 children (as opposed to 4 for the dyadic transform) except with the lowest frequency sub-band (there one parent is related to 1 child). A method proposed is as follows for 512×512 images: Decompose with a 6-level quincunx transform. Here coefficients from levels 3-5 can be grouped into vectors of the form (1,2,4) elements. Grouping four such vectors allows 28-D inter-band vectors to be coded. Then the coefficients from levels 1-3 can be coded as 7-D inter-band vectors. Because the lowest frequency sub-band (of size 64×64) is 32 times larger than the corresponding DC band of a dyadic transform, one needs to be able to code that efficiently. One method would be to use a 2-level dyadic (separable) wavelet transform to further decompose this sub-band leading to coding by DPCM of a 16×16 sub-band with the coding of the A.C coefficients using 5-D inter-band vectors as we used in our separable dyadic wavelet based scheme. It would be interesting to compare results both on a PSNR and subjective basis. It has been found that the reconstructed image from a quincunx transform contains an annoying type of artefact on horizontal and vertical edges known as a ‘checker-board’ pattern. We used the ‘multiple-mu’ method to minimise the mean-squared error in each sub-band. One could modify that method for the sub-bands on the finer scales (high frequency bands) by taking correlation information from the coarser scales into consideration (i.e. when deciding weighting factors for the

quantised vectors, i.e. the ‘mu’ values). Thus one may be able to eliminate these subjectively annoying artefacts as well as potentially produce good PSNR results.

- ◆ We found for our ECPVQ method that a combination of two lattices (the  $Z_n$  for low energy vectors and the  $D_n$  for higher energy vectors) produced good PSNR vs. rate results. When coding intra-band vectors, it is known that the  $D_n$  lattice outperforms the  $Z_n$  lattice but a better lattice in terms of rate vs. distortion performance may be the  $\Lambda_{21}$  lattice (densest packing lattice in 21 dimensions [34]). It would be interesting to modify our ECPVQ method such that the higher energy vectors (further from the origin) can be quantised with the  $\Lambda_{21}$  lattice and then assess where this results in any significant improvements in rate vs. PSNR performance.
- ◆ The distribution of the radial parameter (showing rapid exponential decay as the value of the radial parameter increases), when coding  $512 \times 512$  images (as opposed to ones of size  $256 \times 256$ ) showed that it might be possible to get some improvement in performance by combining neighbouring 21-D vectors to form 42-D vectors. This would require new training set statistics to be obtained. However, while the efficiency of coding the positional information may improve, the efficiency of coding the radial parameter may be reduced because the dependencies between neighbouring 42-D vectors are likely to be less than for the 21-D vectors which we currently code.
- ◆ An examination of the efficiency of our context-based scheme for coding the radial parameter (index of shell or  $l_1$  norm of vector) is required in the light of recent developments in that area vis-à-vis scalar quantised wavelet coefficients [52,64]. New approaches to avoid context dilution enabling more contexts to be used have been developed and this area in general is an interesting one for future research.

Also in our method, the context-based coding accounted for almost 25% of the overall bitrate (for the  $512 \times 512$  images). This percentage could be increased if the contribution of the first order entropy code which transmits the information about the sub/super/super-super class index is reduced. The joint optimisation of these two entropy

codes (context-based and first order entropy) is an interesting problem for future consideration.

Finally, for the  $512 \times 512$  sized images which we coded, the magnitude of the radial parameter of the 5-D vectors ( $l_1$  norm) which are coded first, could be included in the context “information” for coding the 21-D vectors of similar orientation.

## **APPENDICES**

---

## A.1 Implementation of simple scalar quantisation based benchmark coder

---

In this appendix, we shall be implementing a benchmark coder using simple scalar quantisation and first order entropy coding. We then test the efficiency of 4 different pairs of filters for the separable wavelet transform, at least 3 of which have been used previously in image coding applications [7,38,76]. All are bi-orthogonal, linear phase filters. We compared the performance of this coder with the more complex algorithms previously developed in this dissertation (i.e. the ECPVQ algorithm) in section 5.7.1.3. Finally the performance of the different filter pairs is evaluated using a novel graphical technique.

### A.1.1 Scalar quantisation based coder

In the following section, it is assumed that the quantiser of choice is the dead-zone uniform scalar quantiser with the width of the dead-zone at the origin being  $\alpha\Delta$  where  $\alpha$  and  $\Delta$  being the nominal step size. If  $\alpha = 1.0$ , then the quantiser becomes a uniform mid-tread one.

- ◆ The grayscale image **Lena** of resolution 256×256 pixels (8 bits/pixel) has been coded.
- ◆ A 3-level wavelet decomposition using 4 pairs of perfect reconstruction filters has been carried out. The filtering is performed in a separable manner (i.e. filtering the columns and then the rows leading to 10 sub-bands (one D.C. and 9 A.C.)).
- ◆ To reduce boundary effects, symmetric extension as described in section 2.6.2 is used.
- ◆ Two different scalar quantiser are used. One has  $\alpha = 2.0$ . It has been observed that the transform coefficients of real images usually contain a large number of low valued coefficients. Therefore, a slightly higher degree of compression can usually be

achieved by increasing the zero step width (or dead-zone width). We have also tried a quantiser with  $\alpha = 1.5$ . (these are known as *dead-zone* quantisers)

- ◆ For each pair of filters, the subbands are normalised such that a unit impulse in any band results in unit energy at the output (i.e. after synthesis). This is necessary as each of the filter pairs is not *orthogonal* but *bi-orthogonal*. This means that using a quantiser which has a uniform step size will result in equal contribution to the MSE by each of the subbands. This is also equivalent to noise shaping according to a flat HVS response. Fig A.4 shows a plot of the weighting factors vs. subband index for the different filter pairs. Each of the sub-bands is multiplied by the appropriate weighting factor before quantisation and this process is reversed just before the inverse wavelet transform. Then a quantiser with the *same step size* is applied to all bands. It has been observed by Karunasekera [14] among others that by using different step sizes for the different bands (used in JPEG [2]), aspects of the human visual system can be exploited. The visibility of artefacts depends mainly on frequency sensitivity. Since a wavelet split decomposes an image into spatial frequency bands, frequency sensitivity can be incorporated into a coding scheme by adjusting the quantiser step size based on the frequency sensitivity of the visual system to that particular band. Researchers [14,2] have found that by quantising the higher-frequency bands coarser than the lower frequency ones, the subjective quality (or the subjective error) of the decoded images improves for a given bitrate although the MSE increases.

Further subjective improvement can be attained by incorporating masking into a coding scheme. The idea is that the quantiser step size can be varied based on local image activity. Where there is a lot of local activity, a coarser quantiser may be used relative to where there might be little local activity. This involves varying the quantiser step size for groups of coefficients within any particular spatial frequency band. As this is an image-dependent scheme, the overheads involved may be considerable.

- ◆ Coding the DC band using a DPCM technique described in section 5.6.1.

- ◆ At the decoder, instead of using the mid-point of a quantisation interval as the reproduction level, the centroid is experimentally determined for each sub-band (except the DC band), by an exhaustive optimisation, at the encoder and transmitted to the decoder. Because the coefficients in any A.C. sub-band can be approximately modelled as being from a Laplacian pdf [37], the centroid of each quantisation interval is likely to be closer to the origin from the mid-point of that interval by an amount  $\varepsilon(\Delta)$  (which varies depending on the sub-band). For each sub-band, 101 evenly spaced values in the range  $[0, \Delta/2]$  are tried for  $\varepsilon(\Delta)$  where  $\Delta$  is the step-size and the value which minimises the mean-squared error is used.
- ◆ The probabilities used for the different entropy codes at the encoder are determined by assuming a laplacian distribution with zero-mean, and the standard-deviation for each sub-band is transmitted to the decoder so that the probabilities can be re-produced. It is assumed that the two extreme values of the probability distribution are transmitted to the decoder. This is necessary when implementation is by arithmetic coding. To reduce the amount of header information, the maximum of the extreme values (absolute values) need only be transmitted but this may unnecessarily increase the number of probabilities used for the entropy code. The probabilities are stored to a precision of 14 bits.

The  $\lambda$  parameter of the laplacian pdf is estimated from the variance of the sub-band. It can also be estimated from the mean absolute value. Experiments have shown for the two scalar quantisers considered here that the variance estimator is best for modelling the higher frequency sub-bands (“finer” bands) whereas the mean absolute value estimator better models the low-frequency sub-bands (and is quite poor for the higher frequency ones). For simplicity, we use the former method for all sub-bands.

◆ . The total entropy,  $R$ , is given as

$$R = \frac{R_D}{64} + \frac{(R_2 + R_3 + R_4)}{64} + \frac{(R_5 + R_6 + R_7)}{16} + \frac{(R_8 + R_9 + R_{10})}{4} \quad (\text{A.1.1})$$

where  $R_D$  is the entropy of the DC band and  $R_N$ ,  $N=2..10$ , is the entropy of sub-band  $N$  where the probabilities used in the entropy-code are those that have been estimated by the method described previously.

◆ . The MSE of each decoded image was measured. Also used was the peak signal to noise ratio (PSNR) given by:

$$PSNR = \frac{(255)^2}{MSE} \text{ dB} \quad (\text{A.1.2})$$

The filter pairs used are as follows:

◆ Pair A -----

$$H_0(z) = (1/8) * (1/0.832) * (-1 + 2z^{-1} + 6z^{-2} + 2z^{-3} - z^{-4}) \quad (\text{A.1.3})$$

$$H_1(z) = (1/2) * (0.832) * (1 - 2z^{-1} + z^{-2}) \quad (\text{A.1.4})$$

$$F_0(z) = (1/2) * (0.832) * (1 + 2z^{-1} + z^{-2}) \quad (\text{A.1.5})$$

$$F_1(z) = (1/8) * (1/0.832) * (1 + 2z^{-1} - 6z^{-2} + 2z^{-3} + z^{-4}) \quad (\text{A.1.6})$$

where  $H_0(z)$  and  $H_1(z)$  are the analysis low and high pass filters and  $F_0(z)$  and  $F_1(z)$  are the synthesis low and high pass filters respectively. The factor 0.832 is used to ensure unit energy at the output for a unit impulse transmitted in any of the 4 sub-bands obtained from a one level decomposition. These are the Le-Gall 3,5 tap filters [7].

◆ Pair B -----Equations A.1.7-A.1.10

$$H_0(z) = 0.0201 - 0.0116z^{-1} - 0.1426z^{-2} + 0.0448z^{-3} + 0.5889z^{-4} + 0.5889z^{-5} + 0.0448z^{-6} - 0.1426z^{-7} - 0.0116z^{-8} + 0.0201z^{-9}$$

$$H_1(z) = -0.0060 + 0.0033z^{-1} + 0.0361z^{-2} - 0.0098z^{-3} - 0.0724z^{-4} - 0.2223z^{-5} + 0.8135z^{-6} - 0.8135z^{-7} + 0.2223z^{-8} + 0.0724z^{-9} + 0.0098z^{-10} - 0.0361z^{-11} - 0.0033z^{-12} + 0.0060z^{-13}$$

$$F_0(z) = -0.0060 - 0.0033z^{-1} + 0.0361z^{-2} + 0.0098z^{-3} - 0.0724z^{-4} + 0.2223z^{-5} + 0.8135z^{-6} + 0.8135z^{-7} + 0.2223z^{-8} - 0.0724z^{-9} + 0.0098z^{-10} + 0.0361z^{-11} - 0.0033z^{-12} - 0.0060z^{-13}$$

$$F_1(z) = -0.0201 - 0.0116z^{-1} + 0.1426z^{-2} + 0.0448z^{-3} - 0.5889z^{-4} + 0.5889z^{-5} - 0.0448z^{-6} - 0.1426z^{-7} + 0.0116z^{-8} + 0.0201z^{-9}$$

These are the 10 and 14 tap filters used by Ghanbari [75]. He finds they give a good subjective performance.

◆ Pair C -----

These are the 13 and 19-tap filters designed using the method described in section 2.4 (the filters with  $c=-2/7$  and a 4-tap transformation with  $p=0$  and  $q=-3/32$  were used).

$$H_0(z) = [0.00010z^9 + 0z^8 - 0.0019z^7 - 0.0026z^6 + 0.0101z^5 + 0.0326z^4 - 0.0787z^3 - 0.0706z^2 + 0.4239z^1 + 0.7883 + 0.4239z^{-1} - 0.0706z^{-2} - 0.0787z^{-3} + 0.0326z^{-4} + 0.0101z^{-5} - 0.0026z^{-6} - 0.0019z^{-7} + 0z^{-8} + 0.0001z^{-9}]; \quad (\text{A.1.11})$$

$$H_1(z) = [-0.0026z^5 + 0z^4 - 0.0326z^3 + 0.0663z^2 - 0.0706z^1 - 0.4198 + 0.7883z^{-1} - 0.4198z^{-2} - 0.0706z^{-3} + 0.0663z^{-4} + 0.0326z^{-5} + 0z^{-6} - 0.0026z^{-7}]; \quad (\text{A.1.12})$$

$$F_0(z) = [-0.0026z^6 + 0z^5 + 0.0326z^4 - 0.0663z^3 - 0.0706z^2 + 0.4198z^1 + 0.7883 - 0.4198z^{-1} - 0.0706z^{-2} - 0.0663z^{-3} + 0.0326z^{-4} + 0z^{-5} - 0.0026z^{-6}]; \quad (\text{A.1.13})$$

$$F_1(z) = [-0.0001z^{10} + 0z^9 + 0.0019z^8 - 0.0026z^7 - 0.0101z^6 + 0.0326z^5 + 0.0787z^4 - 0.0706z^3 - 0.4239z^2 + 0.7883z^1 - 0.4239 - 0.0706z^{-1} + 0.0787z^{-2} + 0.0326z^{-3} - 0.0101z^{-4} - 0.0026z^{-5} + 0.0019z^{-6} + 0z^{-7} - 0.0001z^{-8}]; \quad (\text{A.1.14})$$

◆ Pair D----Equations A.1.15-A.1.18

These are the 9/7 tap filters that are widely used in the literature [38,50].

$$H_0(z) = 0.0378 - 0.0238z^{-1} - 0.1106z^{-2} + 0.3774z^{-3} + 0.8527z^{-4} + 0.3374z^{-5} - 0.1106z^{-6} - 0.0238z^{-7} + 0.0378z^{-8}$$

$$H_1(z) = 0.0645 - 0.0407z^{-1} - 0.4181z^{-2} + 0.7885z^{-3} - 0.4181z^{-4} - 0.0407z^{-5} + 0.0645z^{-6}$$

$$F_0(z) = -0.0645 - 0.0407z^{-1} + 0.4181z^{-2} + 0.7885z^{-3} + 0.4181z^{-4} - 0.0407z^{-5} - 0.0645z^{-6}$$

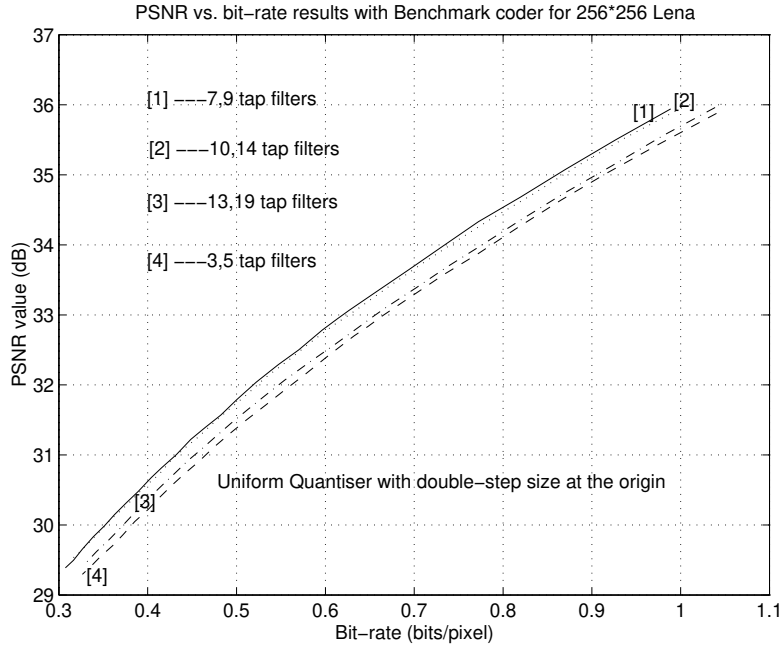
$$F_1(z) = 0.0378 + 0.0238z^{-1} - 0.1106z^{-2} - 0.3774z^{-3} + 0.8527z^{-4} - 0.3374z^{-5} - 0.1106z^{-6} + 0.0238z^{-7} + 0.0378z^{-8}$$

### A.1.2 Some results for benchmark coder

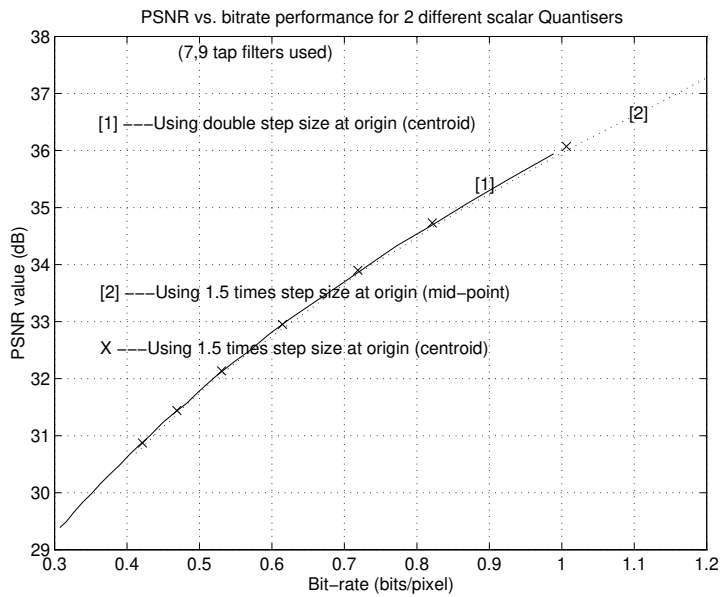
The following results in table A.1 have been obtained when using the **benchmark coder** to code the 256×256 **Lena** using the scalar quantiser with  $\alpha = 2.0$ .

<b>Filter Pair</b>	<b>Bitrate(bpp)</b>	<b>PSNR (dB)</b>
Pair A	0.613	32.51
Pair B	0.613	32.89
Pair C	0.613	32.60
Pair D	0.613	32.94
Pair B	1.0	35.96
Pair D	1.0	36.02

**Table A.1 : Scalar quantisation results for Lena**



**Fig A.2. Plot of PSNR vs. entropy for image LENA 256×256 using benchmark scalar quantiser (solid line, [1];dotted line, [2];dashed-dotted line, [3];dashed line, [4])**



**Fig A.3. Plot of PSNR vs. entropy for image LENA using 7,9 tap filters and 2 different uniform quantisers (solid line, [1];dotted line, [2])**

The benchmark coder results for Lena over a wide range of bit-rates for the 4 different linear phase filter pairs are given in figure A.2. Figure A.3 gives the PSNR vs. bit-rate results for the two different scalar quantisers (i.e.  $\alpha = 2.0$  and  $\alpha = 1.5$ ). We have found that the results from either quantiser are almost identical (when the centroid is used as the reproduction level for both quantisers). It should be noted that we found that when  $\alpha = 1.0$  (i.e. when a uniform mid-tread quantiser is used), the disadvantage in terms of PSNR at comparable bit-rates over the previous two quantisers is about 0.26dB. When the quantiser used has  $\alpha = 2.0$ , the advantage in using the centroid method (as opposed to the mid-point of a quantisation interval) is only about 0.01-0.025dB for the 7,9 tap filters. However, when the dead-zone at the origin is 1.5 times the nominal step (i.e.  $\alpha = 1.5$ ), an advantage of 0.06-0.09 dB is observed.

It has been observed that by using estimated probabilities based on the laplacian distribution, the entropy increases by approx. 4.8% over the ideal entropy (using probabilities based on the actual image) at low bit-rates (<0.7 bits/pixel).

It has also been found that transmitting the actual probabilities (quantised to 9 or 10 bits) as measured from the image being coded, for the higher frequency sub-bands (the low frequency sub-bands can continue to be modelled), where the variance is quite low (i.e. few probabilities to transmit) is better than modelling them as coming from a Laplacian pdf (so that they can be identically estimated at the decoder) in terms of leading to a reduction in terms of overall bit-rate. We have found the advantage to be about 0.12dB at 0.613bpp for coding Lena (using 7,9 tap filters).

In conclusion, we have found that for our benchmark coder, the 7,9 tap filter pair gives the best performance from a quantitative point of view.

### A.1.3 Reasons for rate vs. PSNR performance of different filters

It appears, from figure A.2, that 3,5 tap and the 13,19 tap filters have similar performance (the 3,5 tap filters being inferior by about 0.1dB). However the 7,9 tap and 10,14 tap filters out-perform the 13,19 tap filters by about 0.3dB, the 7,9 tap filters being about 0.05dB superior to the 10,14 tap ones. Due to the additional computational complexity involved with using longer length filters, the 3,5 tap files are probably adequate for applications where this is a serious consideration.

To understand the variations in performance, the different pairs of filters can be considered from a wavelet and scaling function point of view. Daubechies [10] has shown that the tree-structured system that describes a wavelet decomposition can be used to construct the scaling function  $\phi(x)$  and the wavelet function  $\psi(x)$  for a multiresolution signal decomposition. By letting  $L \rightarrow \infty$  (indefinite iteration) and :-

1. terminating the last stage with a low-pass filter; we obtain the scaling function
2. terminating the last stage with a high-pass filter; we obtain the wavelet function

The following equations were used to determine the wavelet and scaling functions:

$$h_w(n) = h_0(n) * h_{0,1}(n) * \dots * h_{0,L-2}(n) * h_{1,L-1}(n) \quad (\text{A.1.19})$$

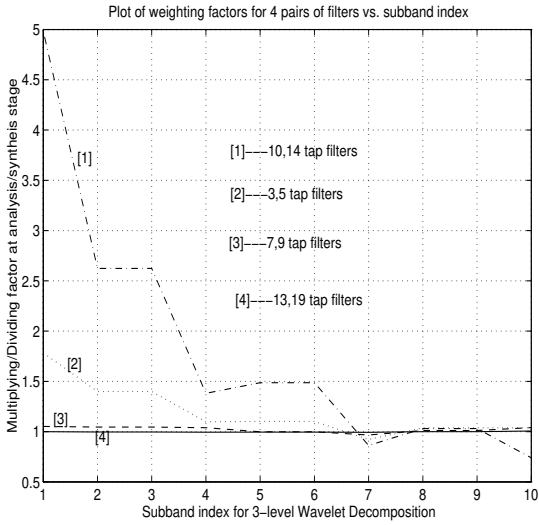
$$h_s(n) = h_0(n) * h_{0,1}(n) * \dots * h_{0,L-2}(n) * h_{0,L-1}(n) \quad (\text{A.1.20})$$

where \* denotes convolution,  $h_w(n)$  is the wavelet function,  $h_s(n)$  is the scaling function,  $h_0(n)$  is the impulse response of a low pass filter,  $h_1(n)$  is the impulse response of a high pass filter and :

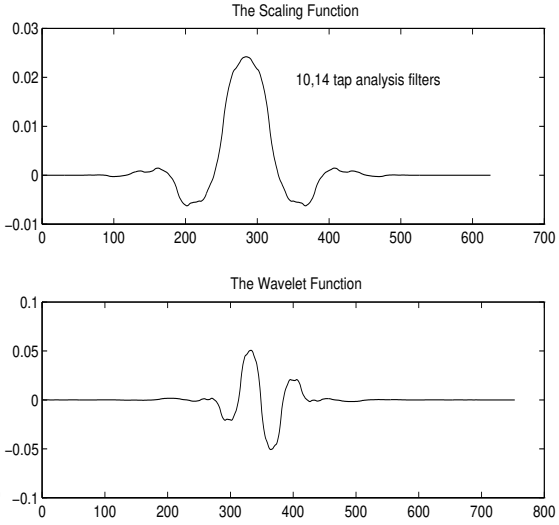
$$h_{0,i}(n) = \begin{cases} h_0(n / 2^i) & \text{if } n/2^i \text{ an integer,} \\ 0 & \text{elsewhere} \end{cases} \quad (\text{A.1.21})$$

i.e.  $h_{0,i}$  is obtained from  $h_0(n)$  by upsampling with factor  $2^i$

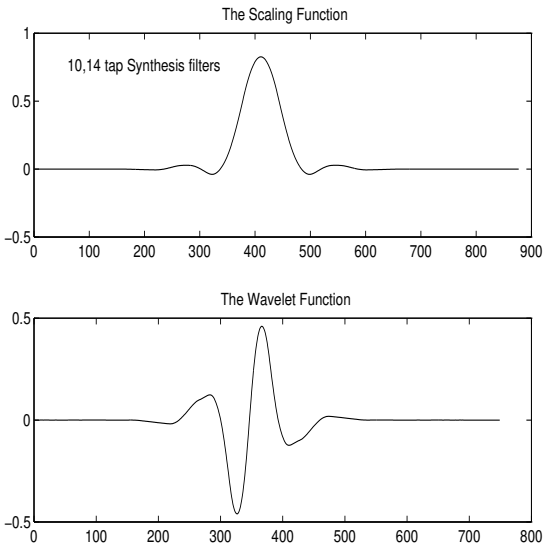
Figures A.5-A.14 show the analysis and synthesis wavelet and scaling functions for different filter pairs. The 10/18 tap filters shown are those used recently in the coding literature, by Villasenor [49] amongst others [53,55].



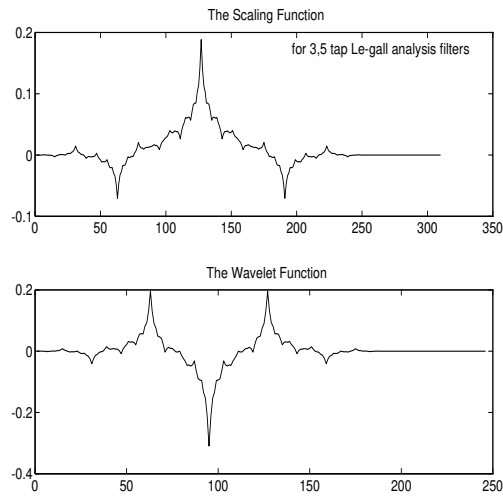
**Figure A.4 : Plot of weighting factors for 4 filters pairs vs. subband index**



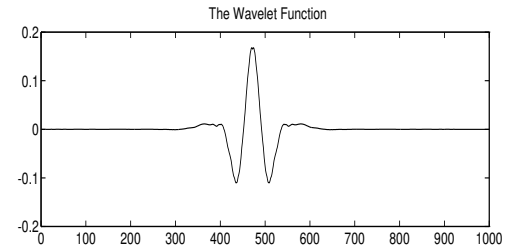
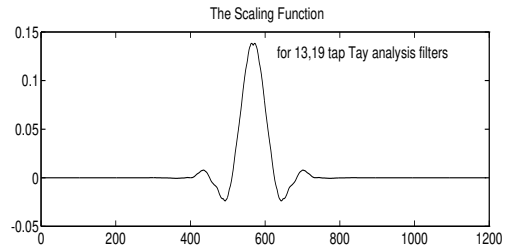
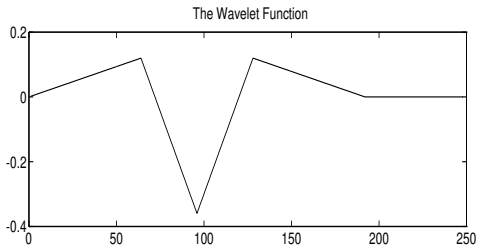
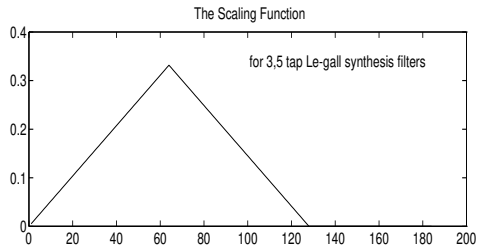
**Figure A.5 Wavelet and Scaling analysis functions using 10,14 tap filters**



**Figure A.6 :Wavelet and Scaling synthesis functions for 10,14 tap Le-gall filters**

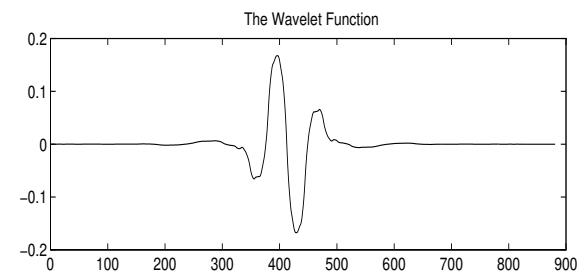
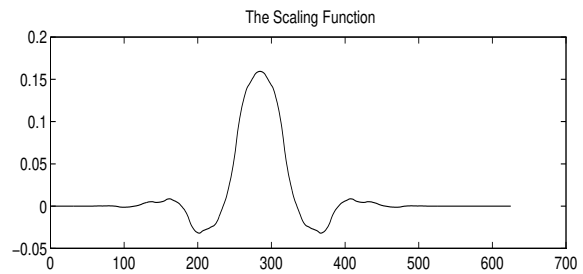
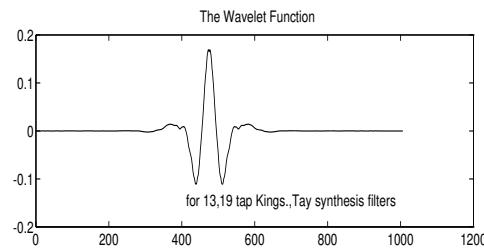
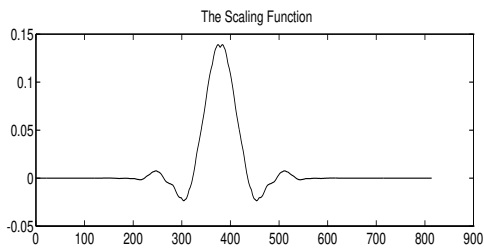


**Figure A.7: Wavelet and Scaling analysis functions for 3,5 tap Le-Gall filters**



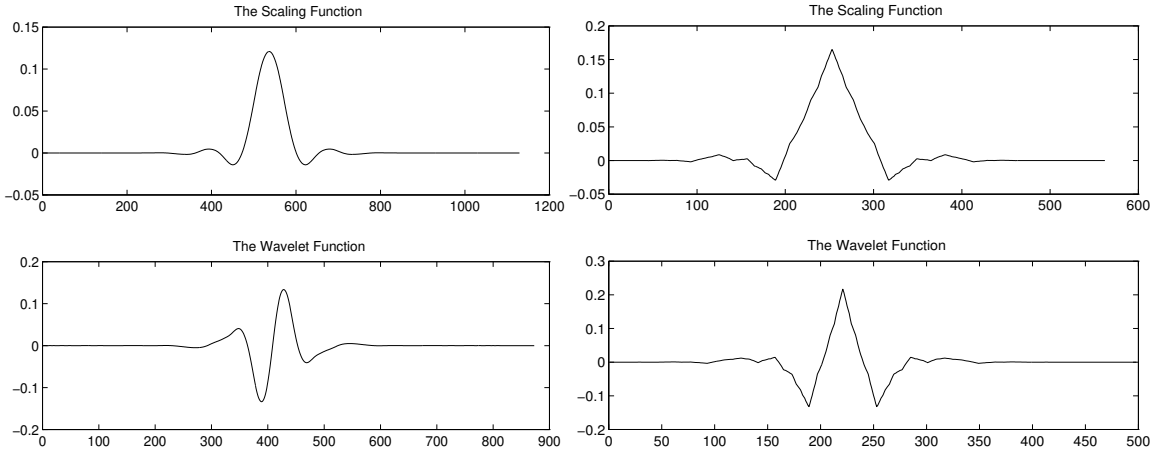
**Figure A.8:Wavelet and Scaling synthesis functions for 3,5 tap Le-gall filters**

**Figure A.9 :Wavelet and Scaling analysis functions for 13,19 tap filters**

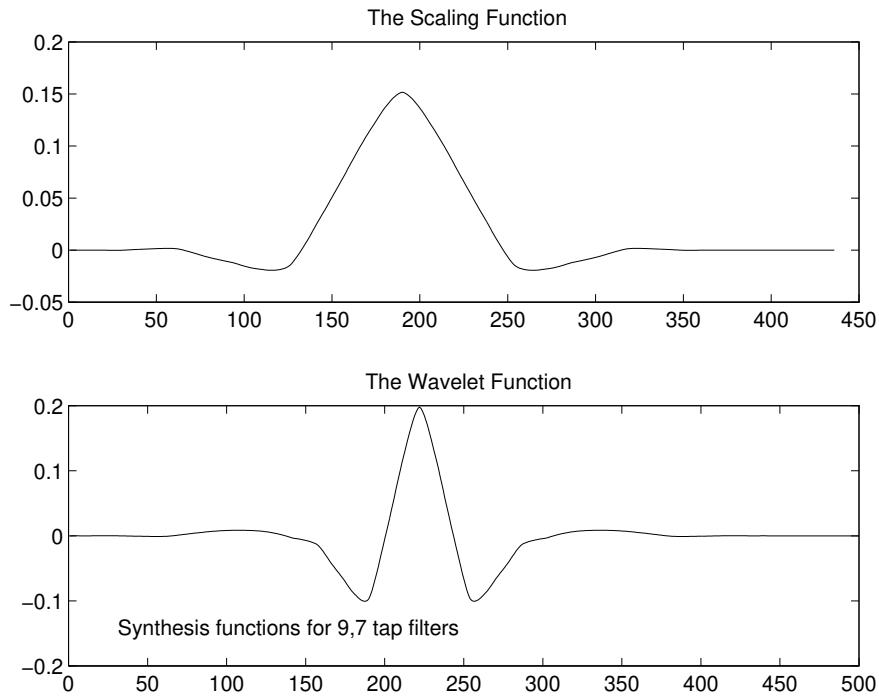


**Figure A.10:Wavelet and Scaling synthesis functions for 13, 19 tap filters**

**Figure A.11 :Wavelet and Scaling analysis functions for 10/18 tap filters**



**Figure A.12 :Wavelet and Scaling synthesis functions for 10/18 tap filters**      **Figure A.13 :Wavelet and Scaling analysis functions for 9,7 tap filters.**



**Figure A.14 :Wavelet and Scaling synthesis functions for 9,7 tap filters**

It can be seen from figures A.7-A.8 that there is a distinct lack of symmetry between the analysis and synthesis wavelet and scaling functions for the 3,5 tap filters. From figures A.9-A.10, the 13,19 tap filters appear to have both smooth wavelet and scaling functions and near symmetry between the analysis and synthesis functions (which seems to lead to some oscillations in the wavelet (synthesis most importantly) functions). The 7/9 (figures A.13-A.14) and 10/14 tap filters (figures A.5-A.6) have smooth synthesis wavelet and scaling functions but with slight irregularity on the analysis side. Both the 10/14 and 10/18 tap filters (figures A.11-A.12) have antisymmetric wavelet functions whereas the 3/5 and 13/19 tap filters have even symmetric wavelet functions.

One factor which has been suggested [75] is that for good coding performance, smooth synthesis wavelet and scaling functions are highly desirable, which result in a high degree of attenuation in the stop-band of the frequency response for both of these functions.

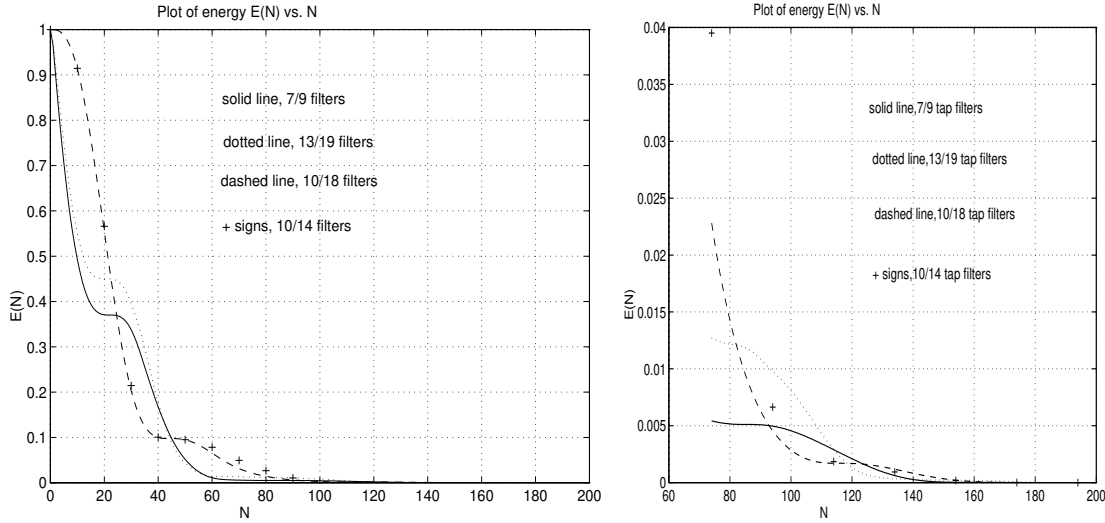
In the case of non-linear phase filters of which Daubechies [10] filters are an example, it is found that the analysis and synthesis wavelet and scaling functions of Daubechies filters are absolutely identical (apart from a time reversal). This coupled with the orthonormality of the wavelet functions has resulted in them being widely used for coding purposes. It has been suggested [75] that non-linear phase filters (coupled with a periodic extension method for filtering at the boundaries) have an *a priori* subjective disadvantage when compared with linear phase bi-orthogonal filters (or the nearly perfect reconstruction linear phase QMF filters of Johnston).

Based on our results which show an inferior rate vs. PSNR performance of the 13/19 tap filters (compared with for example the 7/9 and 10/14 tap filters), we can suggest that linear phase perfect reconstruction filters which are designed to approach orthonormality (i.e. analysis and synthesis sides are nearly identical) are inferior for wavelet based coding applications if PSNR is the performance measure being used.

#### A.1.4. Reasons for subjective performance of different filters

Another question which is typically posed is whether filters which have longer synthesis wavelet impulse responses cause more ringing distortion in the decoded image rather filters with shorter responses. De Silva and Ghanbari [75] suggested a measure called *PPR* (peak to peak value) which attempts to take into consideration the behaviour associated with the shape of the synthesis wavelet. They define *PPR* as the ratio of the maximum peak-to-peak value of the wavelet to the arithmetic mean of the second and third maximum peak-to-peak values. Therefore the larger the *PPR* is, the more “concentrated” the wavelet becomes, hence the spread of the error becomes smaller and less visible.

We suggest a graphical comparison using the following approach; let us consider the synthesis wavelet function. Let the sample about which there is symmetry (whether symmetric or antisymmetric) be labelled as sample zero (in the case of even length filters, the symmetry would be halfway between two samples). We wish to find the percentage of the energy of the wavelet function contained in the tails. Let this energy be defined as  $E(N) = \sum_{n=N}^{\infty} Es(n) + \sum_{n=-N}^{-\infty} Es(n) = 2 \sum_{n=N}^{\infty} Es(n)$  where  $Es(n)$  is the energy of sample index  $n$  of the impulse response,  $h(n)$ , (which is equivalent to  $h(n) \times h(n)$ ) and  $N=1,2,3,\dots$ . Therefore the energy contained in the tails, for odd length filters, with  $N=1$  is equal to  $E(1)=1-Es(0)$  where it is assumed that  $\sum_{n=-\infty}^{n=+\infty} Es(n) = 1$ . For even length filters, where symmetry lies between samples labelled zero and one,  $E(1) = 1-Es(0)-Es(1)$ .



**Figures A.15-A.16: Plot of the energy  $E(N)$  vs.  $N$**

Figures A.15-A.16 plot the function  $E(N)$  vs.  $N$  for 4 different filter pairs. It should be noted that figure A.16 is a close-up of the region where  $E(N) < 0.05$ . If it were found that the curve for one filter pair (pair  $X$ ) was below that of another filter pair for all  $N$  (pair  $Y$ ), then one could probably conclude that pair  $X$  would be likely to produce less visible subjective artefacts in the decoded image than pair  $Y$ .

Figures A.15-A.16 show that the 7/9 filter curve is below the 10/14 (and 10/18) tap filter curve for values of  $E(N)$  between 0.1 and 0.005 (i.e. 10% to 0.5% of the total energy is concentrated in the tails of the wavelet function). This would suggest a good subjective performance for the 7/9 tap filters which we found to be the case in section 5.7.1.2 (compared to the 10/14 tap filters). In contrast, De Silva and Ghanbari [75] found *PPR* values for the 7/9, **10/14** and 10/18 tap filters respectively of 1.45, **1.59** and 1.53.

---

## A.2 Derivation of a formula for the merging of super-classes

---

We will consider in this appendix a formula which can be used for merging super-classes (as defined in 5.5.3). The method employed (which it should be noted is not optimal when coding the training set) is to find a pair of super-classes to merge such that the increase in entropy is minimised and to continue merging pairs until the desired number of merged super-classes is reached.

The motivation for merging is simple: 1) As the shell index increases, the number of super-classes is increasing but the contribution to the overall bit-rate is diminishing as the probability of occurrence is reducing. Thus, a scheme for merging is desirable such that the number of merged super-classes is diminishing for increased shell index ( $l_1$  norm). 2) At high values of shell index, a lot of super-classes have probability zero from the training set and rather than assigning an arbitrary low value to these bins, to potentially improve performance when coding images outside the training set, merging of these super-classes is desirable to reduce the number of these empty bins.

Let  $c_1$  be a super-class with probability  $P_{c_1}$  from the training set and the number of equally probable codewords within this super-class as  $perms(c_1)$

Let  $c_2$  be a super-class with probability  $P_{c_2}$  from the training set and the number of equally probable codewords within this super-class as  $perms(c_2)$ . Then the entropy contribution of these two super-classes, the entropy contribution of a single merged super-class and the increase in entropy are derived as follows:

$$H_{\text{partial}}(c1, c2) = -P_{c1} \log_2 P_{c1} + P_{c1} \log_2(\text{perms}(c1)) - P_{c2} \log_2 P_{c2} + P_{c2} \log_2(\text{perms}(c2))$$

$$H_{\text{partial\_merged}}(c1, c2) = (P_{c1} + P_{c2}) \log_2 \left( \frac{\text{perms}(c1) + \text{perms}(c2)}{P_{c1} + P_{c2}} \right)$$

$$\text{Increase}(c1, c2) = P_{c1} \log_2 \left( \left( \frac{P_{c1}}{P_{c1} + P_{c2}} \right) \left( \frac{\text{perms}(c1) + \text{perms}(c2)}{\text{perms}(c1)} \right) \right) + P_{c2} \log_2 \left( \left( \frac{P_{c2}}{P_{c1} + P_{c2}} \right) \left( \frac{\text{perms}(c1) + \text{perms}(c2)}{\text{perms}(c2)} \right) \right)$$

Now, it is fairly easy to show that the increase must be greater than or equal to zero.

---

## A.3 Definition of contexts used in conditional entropy coding scheme

---

Here, we define the contexts used for the 2,3 and 4 context schemes mentioned in section 5.6.3. For a  $256 \times 256$  image, three  $32 \times 32$  blocks of 21-D vectors are coded. The contexts are defined for each  $32 \times 32$  block of vectors. Thus, an error which results in incorrect decoding of context information can lead to errors propagating throughout a  $32 \times 32$  block of 21-D vectors.

### A.3.1 Method for selecting contexts within each block of vectors

Here, we assume that the 2,3 or 4 contexts are in the same block of vectors as  $X$  (the current vector being coded)

#### 2 contexts

1)  $\begin{matrix} b \\ a \ X \end{matrix}$ , where  $X$  is the current vector to be coded and  $a$  and  $b$  are the vectors

which are used as the two contexts.

2) If those contexts cannot be chosen, because one of both of them are non-causal (given a scanning order, row by row, from top left to bottom right), then one of the following is used:

$$* \begin{matrix} b & a & X \end{matrix}$$

$$* \begin{matrix} a & b \\ X \end{matrix}$$

3) The default value for the two contexts of the first two vectors of each block is zero.

### 3 contexts

1)  $\begin{matrix} c & b \\ a & X \end{matrix}$ , where  $X$  is as above and  $a$ ,  $b$  and  $c$  are contexts in order of importance.

2) If one, two or all three of those contexts are non-causal, one of the following schemes is applied (whichever is appropriate)

\*  $c b a X$

\*  $\begin{matrix} a & b & c \\ X \end{matrix}$

3) The default value for the first three contexts is zero.

### 4 contexts

1)  $\begin{matrix} c & b & d \\ a & X \end{matrix}$ , If  $X$  is the current vector to be coded, then  $a$ ,  $b$ ,  $c$  and  $d$  are the four contexts.

2) If any of those contexts are non-causal, one of the following schemes is adopted:

\*  $d c b a X$

\*  $\begin{matrix} a & b & c & d \\ X \end{matrix}$

\*  $\begin{matrix} d & c & b \\ a & X \end{matrix}$

3) The default value for the first four contexts in the scan for each block is zero.

### A.3.2. Context selection using modified method

In our final coding scheme, to code the shell index, we use 4 contexts. However, the method of choosing the contexts used is slightly different depending on which block of  $32 \times 32$  vectors the current vector being coded is from. (see section 5.6.3.1)

Let a block of vectors be denoted  $blk_n$  where  $n=1 \dots 3$ . Thus  $blk_1$  denotes a block of  $32 \times 32$  vectors where each vector contains vertically orientated wavelet coefficients from sub-bands 2,5 and 8,  $blk_2$  contains horizontally orientated wavelet coefficients from sub-bands 3,6 and 9 and  $blk_3$  contains diagonally orientated wavelet coefficients from sub-bands 4,7 and 10.

The modified context choosing scheme is as follows:

- When the current vector being coded is given by  $blk_1(x,y)$  ,(i.e. containing wavelet coefficients which are vertically oriented) where  $1 \leq x \leq 32, 1 \leq y \leq 32$

the 4 contexts, ( $a, b, c$  and  $d$ ) are chosen according to the scheme in A.3.1

- When the current vector being coded is  $blk_2(x,y)$ , (i.e. containing wavelet coefficients which are horizontally oriented), where  $1 \leq x \leq 32, 1 \leq y \leq 32$

The first 3 contexts ( $a, b$  and  $c$ ) are chosen according to the “3 -context” method described in A.3.1. but the fourth context ( $d$ ) is chosen as  $blk_1(x,y)$

- When the current vector being coded is given by  $blk_3(x,y)$  where  $1 \leq x \leq 32, 1 \leq y \leq 32$ ,

The first 2 contexts ( $a$  and  $b$ ) are chosen according to the “2-context” method in A.3.1 but the third and fourth contexts ( $c$  and  $d$ ) are chosen as  $blk_2(x,y)$  and  $blk_1(x,y)$

The reasoning behind the new method is to ensure that the contexts chosen as *causal* (i.e. that the same ones can be chosen at the decoder) giving our scanning method which codes  $blk_1$ , followed by  $blk_2$  and finally  $blk_3$ .

---

## A.4. Arithmetic/huffman coding of $N$ equally probable symbols where $N$ is not a power of 2

---

We consider the question of coding an alphabet of  $N$  equally probable symbols and propose two efficient methods for doing so.

### A.4.1. Approach by arithmetic coding

We consider the question of an arithmetic coder being used to code a symbol alphabet of  $N$  (where  $N$  is not a power of 2) symbols where each symbol is equally probable (i.e. to obtain an average code-word length of  $\log_2(N)$  bits for each symbol coded), when  $N$  is very large ( $>10^{16}$ ).

Because, the codeword/Address registers are of a finite maximum length of  $k$  bits, where  $k$  is an integer, we need to quantise the  $N$  symbols of probability  $1/N$  to a precision of  $k-1$  [45] or fewer bits such that the sum of the probabilities remains unity. If  $N$  is very large, only two probabilities differing from each other by  $2^{-N}$  need be stored (rather than  $N$  probabilities which would require an extensive usage of memory) -However one also needs to know how many symbols of each probability there are. The calculation can be determined as follows:

Assume that the codeword/address registers are  $k$  bits long. We have decided to quantise using  $n$  bits (where  $n \leq k-1$ ). Therefore, the minimum probability is  $2^{-n}$  (assuming that  $2^n > N$ ),

Then, only two probabilities need be stored at the decoder/encoder.

Let  $n_1$  be the number of symbols with probability  $p_1$ . Let  $n_2$  be the number of symbols with probability  $p_2$ , where  $n_1 p_1 + n_2 p_2 = 1.0$  and  $n_1 + n_2 = N$ . Also,  $p_1, p_2 = c 2^{-n}$  where  $c$  is an integer  $\geq 1$

Now, it is easily seen that,

$$p_1 = \frac{\text{round}(2^n/N)}{2^n}, \text{ where } \text{round}(\cdot) \text{ means to round to nearest integer.}$$

$$p_2 = p_1 - \frac{d}{2^n},$$

$$\text{where } d = \frac{2^n(Np_1 - 1)}{|2^n(Np_1 - 1)|}, \text{ (i.e. } +1 \text{ or } -1)$$

$$n_1 = N - n_2$$

$$n_2 = 2^n \left| (Np_1 - 1) \right|$$

Thus, the first  $n_1$  symbols have probability  $p_1$  and the subsequent  $n_2$  symbols have probability  $p_2$ .

A simpler method whereby  $p_1 = \frac{\lfloor (2^n/N) \rfloor}{2^n}$ ,  $p_2 = 1 - p_1$  and  $n_1 = N - 1$ ,  $n_2 = 1$  can

also be used.

#### A.4.2. Approach by huffman coding

The simplest method for coding  $N$  equally probable symbols is to use  $\lceil \log_2(N) \rceil$  bits for each symbol (where  $\lceil x \rceil$  is the smallest integer greater than  $x$ ) which we term the *simple binary code* method. However, in certain applications one may desire a more optimal code whose entropy approaches  $\log_2(N)$  bits per symbol (the theoretical information content of each symbol). Using a special case of the Huffman Code one can derive the following:

Let  $n_1 = \lceil \log_2(N) \rceil$  and  $n_2 = \lfloor \log_2(N) \rfloor$  (where  $\lfloor x \rfloor$  is the greatest integer less than  $x$ )

Then  $r_1 = 2^{n_1} - N$  and  $r_2 = N - r_1$  where the first  $r_1$  symbols have  $n_2$  bit codewords and the next  $r_2$  symbols have  $n_1$  bit codewords.

**Example:**

Let  $N=10$ . Then the first 6 symbols have 3 bit codewords and the next 4 symbols have 4 bit-codewords. The actual code-words used are the 3-bit binary representations of integers 0...5 (first 6 3-bit codewords) and the 4-bit binary representations of integers 12...15 (the last 4 4-bit codewords). The codewords are as follows:

$$\{(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0,0),(1,1,0,1),(1,1,1,0),(1,1,1,1)\}.$$

Because of their prefix nature each code-word is uniquely decodable and it is very straightforward to develop fast encoding/decoding algorithm irrespective of the size of the symbol alphabet ( $N$ ).

The Huffman entropy (assuming that the actual source's symbol alphabet can accurately be modelled as coming from a uniform distribution) is as follows:

$$ent_h = r_1 n_2 + r_2 n_1 \text{ bits/symbol}$$

where  $r_1, r_2, n_1$  and  $n_2$  are as previously defined. For a comparison of the efficiency of this code compared with the theoretical rate ( $ent_t = \log_2(N)$ ), where efficiency is defined as ( $Eff = ent_t / ent_h$ ) is given in Table A.4.1

$N$	$Eff$ (%)
10	97.7
10000	99.45
100000	99.52
$1 \times 10^{10}$	99.8
$1 \times 10^{20}$	99.87

**Table A.4.1. Efficiency of Huffman Code vs. size of Symbol alphabet.**

---

## A.5. Training set images

---

We have used a collection of 8 256×256 (monochrome, 8 bits per pixel) training images for coding such sized images and a set of 4 512×512 images for coding the two 512×512 images **Lena** and **Goldhill**.

When coding 256×256 images, we have developed training data statistics based on coding each of the 8 training images at 5 different bit-rates. The different scaling factors ( $\Delta$ ) used (corresponding to different bit-rates) in the lattice quantisation, are giving in table A.5.1. for each of the 8 images. We have tried to use a wide variation of bit-rates.

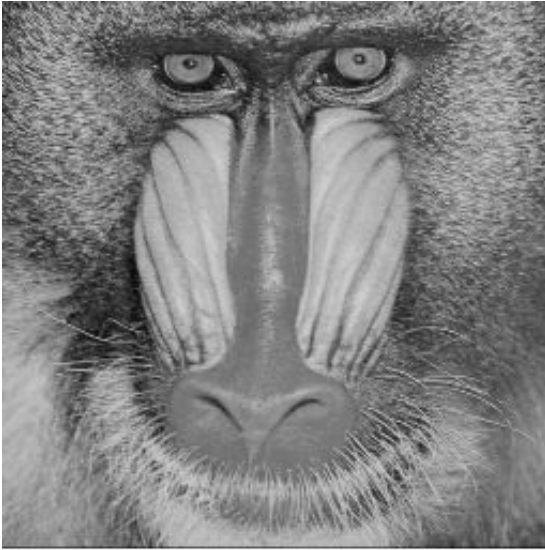
<i>Image</i>	<i>Sc-1</i>	<i>Sc-2</i>	<i>Sc-3</i>	<i>Sc-4</i>	<i>Sc-5</i>
<b>Couple</b>	14	17	20	23	29
<b>Baboon</b>	39	46	53	67	74
<b>Peppers</b>	18	22	26	30	34
<b>House</b>	11	17	20	26	29
<b>Airplane</b>	18	22	26	30	42
<b>Sailboat</b>	30	35	40	45	50
<b>Trevor</b>	11	13	17	19	27
<b>Miss America</b>	8	11	14	17	20

*Table A.5.1. 5 Different scaling factors used for each 256×256 training set image.*

*“Sc-1” is the first value of  $\Delta$  used, “Sc-2” the second and so on .....*

The 512×512 training set consists of higher resolution versions of **Airplane**, **Baboon**, **Peppers** and **Sailboat** (obtained by ftp to ipl.rpi.edu). To obtain training data statistics, each of these has also been coded at 5 different bit-rates with the same scaling factors used as for their lower resolution counter-parts. Figures A.5.1 and A.5.2 show the 8 256×256 monochrome, 8 bits per pixel images referred to in table A.5.1

Original Baboon



Original Couple



Original Miss America



Original Trevor

*Fig A.5.1. First four images of the 256×256 image training set*

Original Airplane



Original Sailboat



Original House

Original Peppers

*Fig A.5.2. Second four images of the 256×256 image training set*

---

## A.6. Relationships between symbol alphabets used in design of a conditional entropy code in 5.6.3 and shell index for 3 different lattices

---

In section 5.6.3 we described a total symbol alphabet for the output symbol  $X$  which was  $\{x_0, x_1, \dots, x_{N_{max}}\}$ . We want to describe the relationship between the symbols in the alphabet and the shell index (which is the  $l_1$  norm of the 21-D vectors which we are considering). We mentioned in 5.6.3. that for complexity reasons we constrain ourselves to selecting just the first  $R$  symbols of the alphabet (for complexity reasons) for  $X$ , where typically  $R \ll N_{max}$ .

The  $P(X=x_i)$  is the probability of occurrence of shell index  $i$  if the  $Z_n$  lattice is used, if the  $D_n$  lattice is used, then shell index  $2i$  or if the  $Z_n/D_n$  lattice is used, then shell index  $i$  if  $i \leq 2$  or else shell index  $2(i-1)$  otherwise, where  $i \leq R-2$ . Where  $i=R-1$ , it is respectively the probability of occurrence, of shell indexes greater than or equal to  $R-1$ , of shell indexes greater than or equal to  $2(R-1)$  or of shell indexes greater than or equal to  $2(R-2)$ .

Thus if the  $D_n$  lattice is used,  $N_{max} = (I/2)$  where  $I$  is the index of the pyramidal shell with the maximum  $l_1$  norm. If the  $Z_n$  lattice is used,  $N_{max} = I$ . If the  $Z_n/D_n$  lattice is used  $N_{max} = (I/2) + 1$ .

Finally, we define how the values each context  $C_n$  (see 5.6.3) can take,  $0..T-1$ , are related to the shell index where  $1 \leq n \leq nc$  indicates the  $n^{\text{th}}$  context. This is similar to the definitions for output symbol  $X$  described earlier.

**\*  $Z_n$  lattice**

$P(C_n=t_n)$  is probability of occurrence of shell index  $t_n$ ,  $0 \leq t_n \leq T-2$

$P(C_n=T-1)$  is probability of occurrence of shell indexes equal to or greater than  $T-1$ .

- ***D<sub>n</sub> lattice***

$P(C_n=t_n)$  is probability of occurrence of shell index  $2t_n$ ,  $0 \leq t_n \leq T-2$

$P(C_n=T-1)$  is probability of occurrence of shell indexes equal to or greater than  $2(T-1)$ .

- ***Z<sub>n</sub>/D<sub>n</sub> lattice***

$P(C_n=t_n)$  is probability of occurrence of shell index  $t_n$ ,  $0 \leq t_n \leq 2$

$P(C_n=t_n)$  is probability of occurrence of shell index  $2(t_n-1)$ ,  $2 < t_n \leq T-2$

$P(C_n=T-1)$  is probability of occurrence of shell indexes equal to or greater than  $2(T-2)$

---

## A.7. Enumeration algorithms for encoding/decoding *classes* and *super-classes*

---

We shall first give the encoding and decoding algorithms for the partition of an integer  $K$  into  $j$  parts. These algorithms as we have shown in section 5.5.1 are important for determining enumeration methods for the **class** structure.

We then give the encoding/decoding algorithms for indexing a specific **super-class** and finally the encoding/decoding algorithms for indexing a specific code-vector given its super-class index.

### A.7.1. Enumeration methods for partitioning an integer

The encoding and decoding algorithms are now given

#### Encoding algorithm for partitioning integer $K$ into $j$ parts

This algorithm is based on the recursive definition in equation 5.11. Here  $K$  is the shell index and  $j$  is the number of significant/non-zero elements in a quantised vector (lattice point) denoted by  $\{\dots\}$ .

```
index = Encode({.....})  
  
/* The vector denoted in {.....} is sorted in descending order of  
magnitude with sign bits and zero elements removed */  
  
if j=1,  
    index =0; return; /* Terminating Condition */  
end;  
  
if last element of vector to be encoded is a 1,  
then  
    index =0 +encode(Rest of vector with last element removed )  
else  
    subtract one from each component of the vector and then  
    index = pa(K-1,j-1) +encode(new vector)  
  
    /* We are partitioning the range of indices and then sub-partitioning  
recursively */  
  
end  
  
end; /*Encoding Function*/
```

The encoding index ranges from 0 ... $Pa(K,j)$ -1

### Decoding algorithm for partitioning integer $K$ into $j$ parts

We now give the corresponding decoding algorithm.

```
{.....} =Decode(index,K,j)
    if j>K,
        b =[]; return;
    end; /* Terminating condition 1 */
    if j=1,
        b=[K]; return;
    end; /* Terminating condition 2 */
    if index < Pa(K-1,j-1)
        b =[Decode(index,K-1,j-1) 1]
        /* Appending one to each of the parts */
    else
        index =index - Pa(K-1,j-1)
        b =[Decode(index,K-j,j)+1]
        /* Adding one to each component of vector */
    end
end; /* Of Function */
```

#### A.7.2. Enumeration algorithms for the *super-class* structure

We firstly give the encoding/decoding algorithms for indexing a *specific super-class*. Then we outline the encoding/decoding algorithms for indexing a code-vector within a specific super-class.

### A.7.2.1. Algorithms for indexing specific *super-class*

The encoding and decoding algorithms are now given, The encoding index runs from  $[0...(K+1)(K+2)/2-1]$  where  $K$  is the shell index (or  $l_1$  norm) and  $r_1$ ,  $r_4$  and  $r_{16}$  are respectively the  $l_1$  norm of the first element, next four elements and the final 16 elements of a specific vector given by  $\{\dots\dots\}$ .

#### Encoding Algorithm

*index = encode\_sup*( $\{\dots\dots\}$ )

*Determine*  $r_1$ ,  $r_4$  and  $r_{16}$

*Determine*  $K = r_1+r_4+r_{16}$

Let  $t = [1, 2, \dots, K+1]$

Let  $tx = [K, K-1, K-2, \dots, 0]$

$x$  is the index of  $r_1$  in vector  $tx$

$$index = \sum_{o=1}^{x-1} t(o) + r_4$$

return *index*

*/\* End of Function \*/*

## Decoding Algorithm

```
[r1,r4,r16] =decode_sup(index,K)
Let t =[1,2,...,K,K+1]
Let tx =[K,K-1,K-2,...,1,0]
if index is not zero,
    for i =1:no. of elements in vector t
        index =index -t(i)
        if (index ≥ 0) and (index < t(i+1))
            exit;
        end
    end
    r1 =tx(i+1); r4 =index; r16 =K-r1-r4;
else
    r1 =tx(1); r4 =0; r16 =0;
end
return r1, r4 and r16
/* End of Function */
```

### A.7.2.2. Enumeration Algorithms for indexing a specific code-vector within a given *super-class*

We now given both the encoding and decoding algorithms.

#### Encoding method:

We assume here that we have an enumeration algorithm for determining the index of a particular code-vector on a pyramidal shell  $K$ , where this index ranges from  $0 \dots N(L,K)-1$ .

Let  $\{\dots\}$  be the 21-D quantised input vector.

```
index_p = encode_posinfo({...}, r1, r4, r16)
```

```
* Let a be the index for the 4-element vector which represents a codeword on  
  a pyramid shell with L=4 and K=r4
```

```
* Let b be the index for the 16-element vector which represents a codeword on  
  a pyramid shell with L=16 and K=r16.
```

```
* index = aN(16,r16)+b (where N(L,K) is as given in equation (5.9))
```

```
  if r1 is zero,
```

```
    index_p = index
```

```
  else
```

```
    if sign of first element is negative,
```

```
      index_p = index
```

```
    else
```

```
      index_p = N(4,r4)N(16,r16) + index
```

```
    end
```

```
  end
```

```
/* End of Function */
```

The index,  $index_p$ , is an example of a **product** enumeration code.

## Decoding method

Given knowledge of the three radii  $r_1, r_4, r_{16}$ , we can tell that  $index\_p$  is in the range range  $0 \dots N(4, r_4)N(16, r_{16})-1$  if  $r_1=0$  or  $0 \dots 2N(4, r_4)N(16, r_{16})-1$ , otherwise.

$\{ \dots \} = decode\_info(index\_p, r_1, r_4, r_{16})$

if  $r_1$  is non-zero,

if  $index\_p < N(4, r_4)N(16, r_{16})$ ,

sign of first element is negative

else

sign of first element is positive

$index\_p = index\_p - N(4, r_4)N(16, r_{16})$

end

end

$a = index\_p \text{ div } N(16, r_{16})$  (div is integer division)

$b = index\_p \text{ rem } N(16, r_{16})$  (rem is remainder after integer division)

\* From  $a$ , one can decode the ordering information (including sign info) for the 4-element grouping within the 21-D vector

\*And, from  $b$ , one can similarly decode the ordering information for the 16-element grouping within the 21-D vector.

*/\* End of Function \*/*

---

## A.8. New Enumeration encoding/decoding algorithms for determining the index of a specific code-vector of dimension $L$ on a pyramidal shell of index $K$

---

We propose an alternative to Fisher's [66] original method for determining the index of a specific code-vector of dimension  $L$  on a pyramidal shell whose "radius" ( $l_1$  norm) is  $K$ . A similar technique was recently proposed [78] in conjunction with testing a fixed-rate coding scheme using pyramidal shells in a noisy channel environment.

It has been shown that (e.g [77,78] )

$$N(L, K) = \sum_{s=1}^m \binom{L}{s} \binom{K-1}{s-1} 2^s$$

where  $N(L, K)$  is the number of possible integer co-ordinate code-words on a pyramid of radius  $K$  and dimension  $L$ . The range for the enumeration *index* is  $[0, N(L, K)-1]$ . The encoding algorithm involves the following steps:

- ◆ Let  $s$  be the number of significant elements in the vector being tested,  $x: (x_1, x_2, \dots, x_L)$  and let  $index=0$ ; Then  $index = index + p_1$  where  $p_1$  is the number of code-vectors on the pyramid shell containing  $s-1$  or fewer significant elements.
- ◆ Let the vector  $x_t$  be obtained from  $x$  by removing the sign bits (i.e, all elements are positive/zero), sorting the subsequent elements into descending order and then removing the zero elements of  $x$ . Find the index of  $x_t$  among the list of all partitions of integer  $K$  into  $s$  parts (equations 5.11-5.12) Let this index be  $c_1$  where  $c_1$  is in the range  $[0, c_{\max}-1]$  ( $c_{\max}$  is the number of **partitions** of an integer  $K$  into  $s$  parts). Then for each partition indexed by  $0 \dots c_1-1$ , let this be  $pt_n$ , calculate the number of permutations of the vector,  $y_n$  (which is a **class** (section

5.5.1)), formed by appending zeros to the end of  $pt_n$ , such that  $y_n$  is an L-dimensional vector. Then  $index = index + perms\{y_n\}2^s$  for  $n = 0 \dots c_1 - 1$  and  $perms\{.\}$  being the number of permutations of the list/vector  $y_n$ .

- ♦ Find the index of the specific permutation,  $sp$ , for the vector  $x$  with sign elements removed (i.e. if  $x = (1, 0, -2, 3, -1)$ , then we wish permutation of  $(1, 0, 2, 3, 1)$  where the index ranges from 0 (for  $(3, 2, 1, 1, 0)$ ) to 59 ( $5!/2!1!1!1! - 1$ ) (for  $(0, 1, 1, 2, 3)$ ). Then  $index = index + sp2^s$
- ♦ Finally, find the specific index,  $ins$ , for the sign information, which ranges from  $0 \dots 2^s - 1$  (i.e. for previous example, index of  $(+1, -1, +1, -1)$  --index ranges from  $[0, 15]$ ). Then  $index = index + ins$ ;

The pseudo-code for the encoding/decoding algorithms is now given:

### Encoding Algorithm

$index\_pyrm = encode\_pyrm(\{ \dots \}, L, K)$

let  $w$  be the number of significant elements in  $\{ \dots \}$

let  $index\_pyrm = 0$

if  $w > 1$ ,

$$index\_pyrm = index\_pyrm + \sum_{s=1}^{w-1} \binom{L}{s} \binom{K-1}{s-1} 2^s$$

/\* Contains no. of possible codewords with 1, ...,  $w-1$  significant elements \*/.

End

let  $\mathbf{x}$  be the vector sorted in descending order of magnitude with the sign information and zero-elements moved

let  $v = encode\_part(\mathbf{x}, K, w)$  /\* Index of particular **partition** of  $K$  into  $w$  parts \*/

```

for h=0:v-1,
    y=[decode_part(h,K,w) zeroes(1,L-w)] /* Particular class
                                           indexed by h */
/* zeroes(1,z) means a 1 × z dimensional vector of 0's */
    index_pyrn =index_pyrn +perm{y}2w
    /* where perm{.} denotes the number of permutations of a
       specific vector taking repetitions into account */
end
let p be the index of specific permutation of {...} with sign
information removed. (see previous section)
let z be vector of length w containing sign bits of significant
elements with -1's replaced by 0's.
let ins =0
for s=1:w,
    ins =ins +zs2w-1
end
index_pyrn =index_pyrn + p2w + ins
/* End of algorithm */

```

## Decoding Algorithm

```
{.....} =decode_pym(index_pym,L,K)
/* To find no. of significant elements s */
let not_found =0
let s=0;
while not_found=0,
    s =s +1
    
$$d = \binom{L}{s} \binom{K-1}{s-1} 2^s$$

    if index_pym<d,
        not_found=1
    else
        index_pym =index_pym - d;
    end
end

let not_found=0; h=0; d=0;
while not_found=0,
    y =[decode_part(h,K,s) zeroes(1,L-s)] /* Particular class
                                           indexed by h */
    d =perm{y}2s
    if index_pym<d
        not_found =1;
    else
        index_pym =index_pym -d;
        h =h+1;
    end
end
```

*end*

*/\* y gives the class of vector \*/*

*p = index\_pyrn div 2<sup>s</sup> ;*

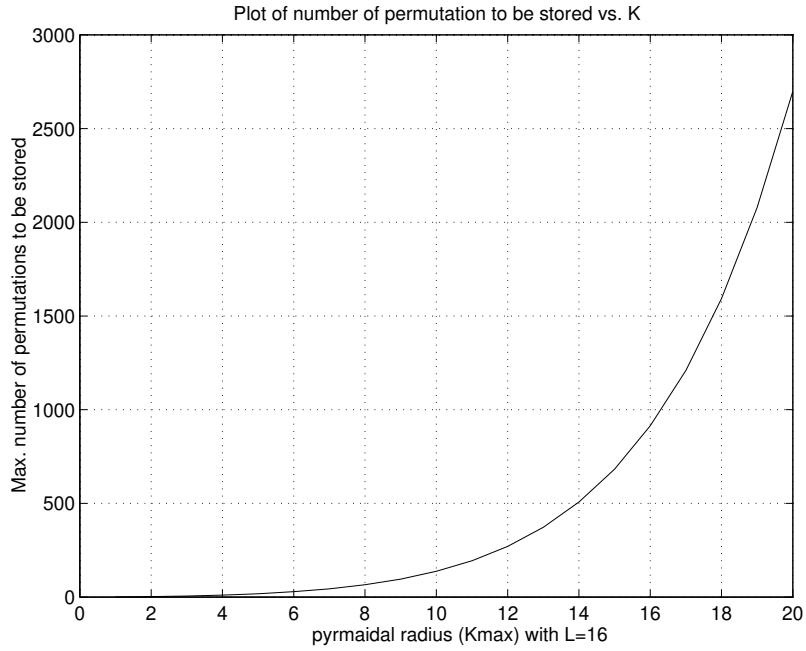
*ins =index\_pyrn rem 2<sup>s</sup>*

Given class **y** and ordering information from index *p*, one can decode the vector without sign information. (see algorithms on encoding permutations with/without repetition)

The sign information is contained in index *ins* whose range is  $[0 \dots 2^s - 1]$  where the 0's are replaced with -1's when index *ins* is denoted by a binary number.

*/\* End of Algorithm \*/*

The *encode\_part* and *decode\_part* algorithms are given in appendix A.7.1 It should be noted that these functions need not be used at the encoder/decoder if the permutation values for the **classes**, where the number of such **classes** is given by  $N^*(L,i)$  (equation 5.10), for  $i=1:K_{max}$  and with  $L=16$  and  $L=4$  and maximum pyramidal shell index  $K_{max}$  (the dimensions of two of the groups of coefficients in the 21-D vector are 4 and 16 respectively), are stored in look-up tables at both the encoder/decoder. The following graph illustrates the memory requirements for  $K_{max}$  (this value should be quite large as it is equivalent to the pyramidal shell with maximum  $l_1$  norm) with  $L=16$ .



**Table A.8.1. Plot of maximum number of permutations needed to be stored at encoder/decoder in a look-up table for  $L=16$  and max  $l_1$  norm of  $K_{max}$**

Here the number of permutations needed to be stored for  $L=16$  and  $K=K_{max}$  is given by:

$$nop(L, K_{max}) \approx \sum_{i=1}^{K_{max}} N^*(L, i)$$

where  $N^*(L, K)$  is the number of **classes** on a pyramidal shell with dimension  $L$  and  $l_1$  norm  $K$  (equation 5.10 of chapter 5)

---

## A.9. Enumeration algorithms for permutations

---

We give a brief overview of encoding/decoding algorithms for determining the index of a specific permutation within the list of all possible permutations.

### A.9.1. Permutations with no repetitions

We want to index a specific permutation from the set of all permutations. Two methods we have developed can be used if the elements are unique (i.e no repetitions)

#### Method 1

```
int encode_p(list P)
    /* P is the list to be encoded */
    if P is empty,
        Return 0,
    else
        Let S =unique elements of P, sorted into any fixed order
        Let n =number of elements of S
        Let P =first element of P
        Let Q =remainder of P after first element is removed
        Let a =index of P's location in S
        Return encode(Q)*n +a
    end
```

The decoding method involves dividing the index by the number of elements in the list -returning quotient and remainder.

## Method 2

This involves using the following approach:

If the list of elements is (2,3,4,1) (index runs from 0...4!-1)

- 1.) Find index of largest integer
- 2.) Multiply this index by the number of ways of permuting the list with the largest integer removed.
- 3.) Repeat process with this latter list,

So, index of (2,3,4,1) is  $2*3! + 1*2! + 0 + 0 = 12 + 2 + 0 = 14$

### A.9.2. Permutations with repetitions

The same process described in method 2 is used except a sub-routine is written to index all possible  ${}^n C_k$  selections (for simplification, let this notation be  $n(C)k$ ),

index running from 0 ...  $n!/(n-k)!k! - 1$

**Example:** Find index of (1,2,2,3,3,4)

Indexes run from 0 ...  $(6!/1!2!2!1!)-1 = 179$

1.) Index =  $5*5!/2!2! + (1,2,2,3,3)$

$(1,2,2,3,3) = (\text{index of } (4,5) \text{ in } (1,2,3,4,5)) = 5(C)2*3!/2! + (1,2,2)$

$(1,2,2) = (\text{index of } (2,3) \text{ in } (1,2,3)) = 3(C)2*1 + 0$

2.) Total index =  $150+27+2=179$

One method that can be used to find an indexing scheme for  $n(C)k$  is as follows:. It can be shown that

$$n(C)k = \sum_{i=k-1}^{n-1} i(C)(k-1) = \binom{n-1}{k-1} + \binom{n-2}{k-1} + \binom{n-3}{k-1} + \dots + \binom{k-1}{k-1}$$

This formula can be used recursively to partition the range of indices.

## Bibliography

Abbreviations used:

ICIP ----- International Conference on Image Processing

ICASSP ----International Conference on Acoustics, Speech and Signal  
Processing

SPIE -----The International Society for Optical Engineering

- [1] Natarayan, T., Ahmed, N., and Rao, K.: 'Discrete cosine transform', *IEEE Transactions on Computers*, 1974, 23, pp. 90-93
- [2] JPEG, 'Digital Compression and coding of Continuous tone still images'. Draft ISO 109/8., *International Organisation for Standardisation (ISO)*, 1991
- [3] Malvar, H., and Staelin, D.: 'The LOT: Transform coding without blocking effects', *IEEE Transactions on ASSP*, 1989, 37, (4), pp. 553-559
- [4] Smith, M., and Barnwell, T.: 'Exact reconstruction techniques for tree-structured subband coders', *IEEE Transactions on ASSP*, June 1986, 34, (3), pp. 434-441
- [5] Esteban, D., and Galand, C.: 'Applications of quadrature mirror filter banks', *Proc. IEEE Int. Conf. ASSP*, May 1977, pp. 191-195
- [6] Johnson, J.: 'A filter family designed for use in quadrature mirror filter banks'. Proceedings of IEEE international conference on *Acoust., speech, signal processing*, ICASSP '80, April 1980, pp. 291-294
- [7] Legall, D.: 'Sub-band coding of images for low computational complexity', Picture coding symposium, Stockholm, Sweden, June 1987
- [8] Gabor, D.: 'Theory of Communications', *J. Inst. of Elec. Eng.*, 1946, 93, pp.429-433
- [9] Mallat, S.G.: 'A theory of multiresolution signal decomposition: the wavelet representation', *IEEE Trans. Pattern Analysis and Machine Intelligence*, 1989, 11, (7), pp. 674-681
- [10] Daubechies, I.: 'Orthonormal bases of compactly supported wavelets', *Comm. of Pure and Applied Maths*, 1988, 41, pp. 901-918
- [11] Akansu, A., and Haddad, R.: 'Multiresolution signal decomposition'. (Academic Press, 1992)

- [12] Burt, P., and Adelson, E.: 'The Laplacian pyramid as a compact image code', *IEEE Comm.*, April 1983, 31, (4), pp. 532-540
- [13] Rioul, O.: 'On the choice of "wavelet" filters for still image compression', 1993, *ICASSP*, pp. 550-553
- [14] Karunasekera, S.: (PhD Thesis, University of Cambridge, June 1994)
- [15] Smith, M., and Eddins, S.: 'Analysis/synthesis techniques for sub-band coding', *IEEE Acoust. Speech Signal Process.*, 1990, 38, (8), pp.1446-1455
- [16] Barnard, D., Weber, J., and Biemond, J.: 'Efficient signal extension for subband/wavelet decomposition of arbitrary length signals', *SPIE Visual Communications and Image Processing Conference*, November 1993, pp. 966-975
- [17] Getz, N.: 'A fast discrete periodic wavelet transform'. (Memorandum no. M92/138, University of California, Berkeley, Dec. 1992)
- [18] Tay, D., and Kingsbury, N.: 'Flexible design of multi-dimensional perfect reconstruction FIR 2-band filters using a transformation of variables', *IEEE Transactions on Image Processing*, 1993, 2, (3)
- [19] Antonini, M., Barlaud, M., and Mathieu, P.: 'Image coding using vector quantisation in the wavelet transform domain', *ICASSP*, 1990, pp. 2297-2300
- [20] Linde, Y., Buzo, A., and Gray, R.: 'An algorithm for vector quantiser design', *IEEE Trans. Communications*, 1980, 28, pp. 84-95
- [21] Chou, P., Lookabaugh, T., and Gray, R.: 'Entropy-constrained vector quantisation', *IEEE Trans. Acoust. Speech Signal Process.*, 1989, 37, pp.31-42
- [22] Equitz, W.: 'A new vector quantisation clustering algorithm', *IEEE Transactions on ASSP*, October 1989, (37), pp. 1568-1573
- [23] Buzo, A., Gray, A., and Markell, J.: 'Speech coding based upon vector quantisation', *IEEE Trans. on ASSP*, October 1980, 28, (5), pp. 562-574
- [24] Lo, K-T, and Feng, J.: 'Predictive mean search algorithm for vector quantisation of images', *ICASSP*, April 1994, pp. 609-612
- [25] Ngwa-Ndifor, J., and Ellis, T.: 'Predictive partial search algorithm for vector quantisation', *Electronics Letters*, Sept. 1992, 27, pp. 1722-1723
- [26] Sabin, M., and Gray, R.: *IEEE Global Communications Conf.*, E.6.5.1-E.6.5.5, Dec. 1982, Miami
- [27] Juang, D. and Gray, R.: 'Multi stage vector quantisation for speech coding', *Proc. Intl. Conf. on ASSP*, Paris 1982, pp. 597-600

- [28] Gamal, A., Shpeling, J., and Mei, V.: 'Using Simulated Annealing to Design Good Codes', *IEEE Trans. on Information Theory*, Jan. 1987, (33), pp. 116-123
- [29] Nasrabadi, N., and Riszi, S.: 'Next-state functions for finite-state vector quantisation', ICASSP 1994, pp. 617-619
- [30] Nasrabadi, N., and Feng, Y.: 'A dynamic finite-state vector quantiser scheme', ICASSP 1990, pp. 2261-2269
- [31] Fioravanti, R.: 'An efficient neural prediction for vector quantisation', ICASSP 1994, pp. 613-616
- [32] Woolf, A., and Rogers, G.: 'Lattice vector quantisation of image wavelet coefficient vectors using a simplified form of entropy coding', Proceedings of IEEE international conference on *Acoust., speech, signal processing*, ICASSP '94, 1994, pp. 269-272
- [33] Antonini, M., Barlaud, M., and Mathieu, P.: 'Image coding using lattice vector quantisation of wavelet coefficients', ICASSP 1991, Toronto, pp. 2273-2276
- [34] Conway, J., and Sloane, N.: 'Fast quantising and decoding algorithm for lattice quantisers and codes', *IEEE Trans. Information Theory*, 1982, 28, (2), pp. 227-232
- [35] Conway, J., and Sloane, N.: 'Sphere packings, lattices and groups' (Springer NY, 1993)
- [36] Gersho, A.: 'Asymptotically optimal block quantisation', *IEEE Trans. on Information Theory*, June 1979, 25, (4), pp. 373-380
- [37] Tsern, E., and Meng, T.: 'Image coding using pyramid VQ of subband coefficients' ICASSP 1994, pp. 601-604
- [38] Barlaud, M., Sole, P., Antonini, M., and Mathieu, D.: 'A pyramidal scheme for lattice vector quantisation of wavelet transform coefficients applied to image coding', ICASSP 1992, pp. 401-404
- [39] Gallager, R.: 'Information theory and reliable communications'. (1968)
- [40] Rissanen, J., and Langdon, J.: 'Arithmetic coding', *IBM J. Res. Develop*, March 1979, 23, (2), pp. 149-162
- [41] Young, R.: 'Video coding using lapped transforms'. (PhD Thesis, University of Cambridge, U.K, Jan.1994)
- [42] Elnahas, S., and Dunham, J.: 'Entropy coding for low bit-rate visual telecommunications', *IEEE Journal in Selected areas in Communications*, August 1987, 5, (7), pp. 1175-1183

- [43] Lei, S., and Tzou, K.: ‘Design of high-order conditional entropy coding for images’, *ICASSP 1992*, 3, pp. 473-476
- [44] Michell, and Pennebake,: ‘Software Implementation of the Q-Coder’, *IBM J. Res. Develop.*, Nov. 1988, 32, (6), pp. 753-774, Nov. 1988
- [45] Rissanen, J., and Mohuddin: ‘A multiplication-free multialphabet arithmetic code’. *IEEE Trans. on Communications*, Feb. 1989, 37, (2), pp. 93-98,
- [46] Langdon, G., and Rissanen, J.: ‘Compression of black-white images with arithmetic coding’, *IEEE Transactions on Communications*, June 1981, 29(6), pp. 858-867
- [47] Chen, T., and Lei, S.: ‘Video coding using switched motion compensation and high-order entropy coding’, *SPIE 1818 Visual Communications and Image Processing Conference*, 1992, pp. 1104-1108
- [48] Joshi, R., Jararkrani, J., Kasner, J., Fisher, T., Marcelin, M. and Bamberger, R.: ‘Comparison of different methods of classification in subband coding of images’, *IEEE Trans. Image Processing*, 1997, 6, (11), pp. 1473-1484
- [49] Tsai, M., Villasenor, J., and Chen, F.: ‘Stack-run image coding’, *IEEE Trans. Circuits and Systems for Video technology*, 1996, 6, pp. 519-521
- [50] Said, A., and Pearlman, W.: ‘A new fast and efficient image codec based on set partitioning in hierarchical trees’, *IEEE Trans. Circuits and Systems for Video technology*, 1996, 6, (3), pp. 243-250
- [51] Xiong, Z., Ramchandran, K., and Orchard, M.: ‘Space-frequency quantisation for wavelet image coding’, *IEEE Trans. Image Processing*, 1997, 6, pp.677-693
- [52] Chrysafis, C., and Ortega, A.: ‘Efficient context-based entropy coding for lossy wavelet image compression’. *Proceedings IEEE Data Compression Conference*, Snowbird, U.S.A., March 25-27, 1997, pp. 241-250
- [53] Lopresto, S., Ramchandran, K., and Orchard, M.: ‘Image coding based on mixture modelling of wavelet coefficients and a fast estimation quantisation framework’. *Proceedings IEEE Data Compression Conference*, Snowbird, U.S.A., March 25-27, 1997, pp. 221-230
- [54] University of Wisconsin, “Linear phase perfect reconstruction FIR filter banks design software”, (<ftp://eceserv0.ece.wisc.edu/pub/nguyen/SOFTWARE/UIFBD>)
- [55] ‘Wavelet image coding: PSNR results’.( [UCLA web site at: http://www.icsl.ucla.edu/~ipl/psnr\\_results.html](http://www.icsl.ucla.edu/~ipl/psnr_results.html))

- [56] Cheng, N., and Kingsbury, N.: 'The ERPC: An efficient error resilient technique for encoding positional information or sparse data', *IEEE Transactions Commun.*, Jan. 1992, 40, (1), pp.140-148
- [57] Westerink, P., Biemond, J., and Boekee, D.: 'An optimal bit allocation algorithm for subband coding', *Proc. ICASSP*, 1988, pp.757-760
- [58] Shoham, Y., and Gersho, A., 'Efficient bit allocation for an arbitrary set of quantisers', *IEEE Trans. Acoust. Speech Signal Processing*, Sept. 1988, 36, (9), pp.1445-1453,
- [59] Riskin, E., and Gray, R.: 'A greedy tree growing algorithm for the design of variable rate vector quantisers', *IEEE Transactions Signal Processing*, Nov. 1991, 39, pp. 2500-2507
- [60] Barlaud, M., Sole, P., Gaidon, T., Antonini, M., and Mathieu, P.: 'Pyramidal lattice vector quantisation for multiscale image coding', *IEEE Trans. Image Processing*, 1994, 3, pp. 367-381
- [61] Marcelin, M., and Fisher, T.: 'Trellis coded quantisation of memoryless and Gaussian markov sources', *IEEE Trans. on Communications*, 1990, 38, (1), pp.82-93
- [62] Ungerboeck, G.: 'Channel coding with multilevel/phase signals', *IEEE Trans. on Information Theory*, Jan. 1982, 28, pp.5-67
- [63] Forney Jr, G.: 'The Viterbi algorithm', *Proc. IEEE*, Feb. 1984, 61, pp.169-176
- [64] Ortega, A., and Chrysafis, C.: 'Line based, reduced memory, wavelet image compression', submitted to *IEEE Trans. on Image Processing*, Jan.1999
- [65] Yusof, M., and Fisher, T.: 'Entropy coded lattice vector quantiser for transform and subband image coding', *IEEE Trans. Image Processing*, 1996, 5, (2), pp. 289-298
- [66] Fisher, T.: 'A pyramid vector quantiser', *IEEE Trans. Inform. Theory*, 1986, 32, (4), pp. 568-583
- [67] Jeong, D., and Gibson, J.: 'Image coding with uniform and piecewise-uniform vector quantisers', *IEEE Trans. Image Processing*, 1995, 4, (2), pp.140-146
- [68] Alessandro, P., and Lancini, R.: 'Video coding scheme using DCT-pyramid vector quantisation', *IEEE Trans. Image Processing*, 1995, 4, (3), pp.309-319
- [69] Fisher, T.: 'Geometric source coding and vector quantisation', *IEEE Trans. Information Theory*, 1989, 35, (1), pp.137-145

- [70] Kasei, S., and Deriche, M.: 'Fingerprint compression using a piecewise-uniform pyramid lattice vector quantisation'. Proceedings of IEEE international conference on *Acoust., speech, signal processing*, ICASSP '97, 1997, pp.3117-3120
- [71] S. Masud and J. McCanny, "Finding a suitable wavelet for image compression applications", vol 5, pp. 2581-2584, ICASSP 1998
- [72] Villasenor, J., Belzer, B., and Liao, J.: 'Filter evaluation and selection in wavelet image compression', *IEEE Trans. on Image Processing*, 1995, (8), pp.1053-1060
- [73] Andrew, J.: 'A simple and efficient hierarchical image coder', vol. 3, October 1997, pp. 658-661, International Conference on Image Processing (ICIP)
- [74] Nister, D., and Christopoulos, C.: 'An embedded DCT-based still image coding algorithm', vol 5, pp.2617-2620, ICASSP 1998
- [75] Silva, E., and Ghanbari, M.: 'On the Performance of linear phase wavelet transforms in low bit rate image coding', *IEEE Trans. on Image Processing*, May 1996, (5), pp. 689-704
- [76] Silva, E., Sampson, D., and Ghanbari, M.: 'A successive approximation vector quantiser for wavelet transform image coding', *IEEE Trans. on Image Processing*, 1996, 5, (2), pp. 299-310
- [77] Yeo, B., Yeung, M., and Sarkar, S.: 'A Fixed-rate vector quantiser based on pyramid-bounded integer lattices for image compression', pp. 578-581, ICIP Nov. 1994.
- [78] Hung, A., and Meng, T.: 'Error resilient pyramid vector quantisation for image compression', pp. 583-587, ICIP, Nov. 1994.
- [79] Sampson, D., and Ghanbari, M.: 'Fast lattice-based gain-shape vector quantisation for image-sequence coding', *Proc. IEE pt. I, Communications, Speech and Vision*, Feb. 1993, 140, (1)
- [80] Witten, I., Neal, R., and Cleary, J.: 'Arithmetic coding for data compression', *Communications of the ACM*, 1987, 30, (6), pp.520-540
- [81] Shapiro, J.: 'Embedded image coding using zerotrees of wavelet coefficients', *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1993, 42, (12), pp. 1345-1362
- [82] Berge, C., 'Principles of combinatorics' (Prentice-Hall, 1991)
- [83] Katto, J., and Yasuda, Y.: 'Performance evaluation of subband coding', Proc. 1991 Picture Coding Symp., PCS '91, Sept. 1991, pp. 399-403.

- [84] Zador, P.: 'Asymptotic quantisation error of continuous signals and their quantisation dimension', *IEEE Trans. on Information Theory*, 1982, 28, (1), pp. 139-149.
- [85] Rubin, F.: 'Arithmetic stream coding using fixed precision registers', *IEEE Trans. on Information Theory*, Nov. 1979, 25, no.6, pp. 672-674
- [86] Adelson, E., Simoncelli, E., and Hingorani, R.: 'Orthogonal pyramid transforms for image coding', *Proc. SPIE*, Oct. 1987, 845, pp. 50-58
- [87] Rabbani, M., and Jones, P., "Digital image compression techniques", SPIE Opt. Eng. Press, Washington 1991
- [88] Darragh, J.: 'Subband and transform coding of images', PhD Thesis, UCLA 1989
- [89] A. K. Jain, 'Fundamentals of Digital Image Processing'. (Prentice-Hall, 1989)