MORGAN & CLAYPOOL PUBLISHERS

Wavelet Image Compression

William A. Pearlman

SYNTHESIS LECTURES ON IMAGE, VIDEO & MULTIMEDIA PROCESSING

Alan C. Bovik, *Series Editor*

Wavelet Image Compression

Synthesis Lectures on Image, Video, and Multimedia Processing

Editor
Alan C. Bovik, ***University of Texas, Austin***

The Lectures on Image, Video and Multimedia Processing are intended to provide a unique and groundbreaking forum for the world's experts in the field to express their knowledge in unique and effective ways. It is our intention that the Series will contain Lectures of basic, intermediate, and advanced material depending on the topical matter and the authors' level of discourse. It is also intended that these Lectures depart from the usual dry textbook format and instead give the author the opportunity to speak more directly to the reader, and to unfold the subject matter from a more personal point of view. The success of this candid approach to technical writing will rest on our selection of exceptionally distinguished authors, who have been chosen for their noteworthy leadership in developing new ideas in image, video, and multimedia processing research, development, and education.

In terms of the subject matter for the series, there are few limitations that we will impose other than the Lectures be related to aspects of the imaging sciences that are relevant to furthering our understanding of the processes by which images, videos, and multimedia signals are formed, processed for various tasks, and perceived by human viewers. These categories are naturally quite broad, for two reasons: First, measuring, processing, and understanding perceptual signals involves broad categories of scientific inquiry, including optics, surface physics, visual psychophysics and neurophysiology, information theory, computer graphics, display and printing technology, artificial intelligence, neural networks, harmonic analysis, and so on. Secondly, the domain of application of these methods is limited only by the number of branches of science, engineering, and industry that utilize audio, visual, and other perceptual signals to convey information. We anticipate that the Lectures in this series will dramatically influence future thought on these subjects as the Twenty-First Century unfolds.

Wavelet Image Compression
William A. Pearlman
2013

Remote Sensing Image Processing
Gustavo Camps-Valls, Devis Tuia, Luis Gómez-Chova, Sandra Jiménez, and Jesús Malo
2011

The Structure and Properties of Color Spaces and the Representation of Color Images
Eric Dubois
2009

Biomedical Image Analysis: Segmentation
Scott T. Acton and Nilanjan Ray
2009

Joint Source-Channel Video Transmission
Fan Zhai and Aggelos Katsaggelos
2007

Super Resolution of Images and Video
Aggelos K. Katsaggelos, Rafael Molina, and Javier Mateos
2007

Tensor Voting: A Perceptual Organization Approach to Computer Vision and Machine Learning
Philippos Mordohai and Gérard Medioni
2006

Light Field Sampling
Cha Zhang and Tsuhan Chen
2006

Real-Time Image and Video Processing: From Research to Reality
Nasser Kehtarnavaz and Mark Gamadia
2006

MPEG-4 Beyond Conventional Video Coding: Object Coding, Resilience, and Scalability
Mihaela van der Schaar, Deepak S Turaga, and Thomas Stockhammer
2006

Modern Image Quality Assessment
Zhou Wang and Alan C. Bovik
2006

Biomedical Image Analysis: Tracking
Scott T. Acton and Nilanjan Ray
2006

Recognition of Humans and Their Activities Using Video
Rama Chellappa, Amit K. Roy-Chowdhury, and S. Kevin Zhou
2005

Copyright © 2013 by Morgan & Claypool

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopy, recording, or any other except for brief quotations in printed reviews, without the prior permission of the publisher.

Wavelet Image Compression

William A. Pearlman

www.morganclaypool.com

ISBN: 9781627051316 paperback
ISBN: 9781627051323 ebook

DOI 10.2200/S00464ED1V01Y201212IVM013

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON IMAGE, VIDEO, AND MULTIMEDIA PROCESSING

Lecture #13
Series Editor: Alan C. Bovik, *University of Texas, Austin*
Series ISSN
Synthesis Lectures on Image, Video, and Multimedia Processing
Print 1559-8136 Electronic 1559-8144

Wavelet Image Compression

William A. Pearlman
Rensselaer Polytechnic Institute

SYNTHESIS LECTURES ON IMAGE, VIDEO, AND MULTIMEDIA PROCESSING #13

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

This book explains the stages necessary to create a wavelet compression system for images and describes state-of-the-art systems used in image compression standards and current research. It starts with a high level discussion of the properties of the wavelet transform, especially the decomposition into multi-resolution subbands. It continues with an exposition of the null-zone, uniform quantization used in most subband coding systems and the optimal allocation of bitrate to the different subbands. Then the image compression systems of the FBI Fingerprint Compression Standard and the JPEG2000 Standard are described in detail. Following that, the set partitioning coders SPECK and SPIHT, and EZW are explained in detail and compared via a fictitious wavelet transform in actions and number of bits coded in a single pass in the top bit plane. The presentation teaches that, besides producing efficient compression, these coding systems, except for the FBI Standard, are capable of writing bit streams that have attributes of rate scalability, resolution scalability, and random access decoding. Many diagrams and tables accompany the text to aid understanding. The book is generous in pointing out references and resources to help the reader who wishes to expand his knowledge, know the origins of the methods, or find resources for running the various algorithms or building his own coding system.

KEYWORDS

image coding, wavelet transform, null-zone quantization, entropy coding, FBI Fingerprint Standard, JPEG2000, SPIHT, SPECK, EZW, rate control, scalability

Contents

Glossary

DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DWT	discrete wavelet transform
EZBC	Embedded Zero Block Coder
EZW	Embedded Zerotree Wavelet
FFT	Fast Fourier Transform
FIR	finite impulse response
JPEG	Joint Photographic Experts Group
JPEG2000	international standard for image compression set by JPEG Working Group and ITU in 2000
LIP	list of insignificant pixels or points
LIS	list of insignificant sets
LSP	list of significant pixels or points
PSNR	peak signal-to-noise ratio
QMF	quadrature mirror filters
SBHP	Subblock Hierarchical Partitioning coder
SOT	spatial orientation tree
SPECK	Set Partitioning Embedded bloCK coder
SPIHT	Set Partitioning In Hierarchical Trees

CHAPTER 1

Introduction

The origin of wavelet image compression hearkens back to the rate-distortion theory introduced by Shannon [44] and its extension to stationary Gaussian processes by Pinsker and reported by Kolmogorov [30]. In these writings, the formulation of the rate-distortion function for mean squared error distortion measure and its proof of optimality use a mathematical transform (Karhunen-Loeve) of the source sequence to produce a sequence of uncorrelated, and independent if Gaussian, non-identically distributed random variables that are encoded together as a block. Huang and Schultheiss [21] proved that even when the transform coefficients are quantized and coded independently, performance gains would be realized in comparison to independent quantization and coding of the source sequence itself. That eventually started a flurry of attempts to encode realistic sources using a transform followed by independent quantization and coding of the transform coefficients. In the case of image sources, the research groups led by Wintz at Purdue University (e.g., see [19, 53]) and Pratt and Andrews at the University of Southern California (e.g., see [39, 40]) were the most prolific and noteworthy. The optimal transform was and remains hard to compute, so the Discrete Fourier Transform (DFT), which computes via the Fast Fourier Transform (FFT), had been the transform of choice until it was supplanted by the Discrete Cosine Transform (DCT) in 1978 [2], when it was proved to converge to the optimal one much faster with length than the DFT and also had a fast computational algorithm. The Joint Photographic Experts Group (JPEG) standard encodes 16×16 DCT blocks and is still the compression format prevalent in consumer cameras and cell phones. Today's video standards use smaller blocks down to size 4×4 transformed by integerized versions of the DCT.

Another kind of transform, the subband transform, appeared around 1976 in the field of speech coding. It was first promulgated by researchers at Bell Telephone Laboratories (see [12, 16]) and is now used mainly in high quality audio coding including MP3. In 1986, O'Neil and Woods published the first journal paper [54] that used a subband transform for image coding. The performance surpassed that of block DCT coding that utilized the same principles of bit allocation and coding. That set off a flurry of research in coding subband transforms of images.

A subband transform may be produced by a bank of anti-aliasing, perfect reconstruction filters. Early subband transforms used quadrature mirror filters (QMF) in a tree-structured filter bank that produced subbands of uniform size. The QMF filters reconstruct the original input almost perfectly, are orthogonal and are relatively short in tap length. Then perfect reconstruction, biorthogonal, short finite impulse response (FIR) filters were developed and used in image subband coding. The wavelet transform is a type of subband transform that was introduced for image coding in the early 1990s. The basis vectors for the transform are dilated and shifted replicas of a waveform having

short, compact support. Filtering with shifted replicas give a series of coefficients that capture local activity content at the different coordinates; and the dilations, which are by factors of two, provide the scale or resolution of the local content. Therefore, the wavelet transform can be reconstructed as a sequence of subimages having octave steps of resolution. In comparison, the DCT coefficients are formed from the entire image or image block. Therefore, local amplitude variations cannot be captured by its coefficients. The solution has been to transform images in small blocks that cover the image, as has been done in the standards. Furthermore, the DCT does not decompose the image into subimages of different resolutions, as does the wavelet transform. These properties of multiresolution decomposition and local content capture make the wavelet transform attractive for the coding of images. The paper on wavelet transform coding by Antonini, Barlaud et al. [4] published in 1992 sparked the research activity that led to the present-day wavelet coding systems that will be taught in this book.

A complete coding or compression system comprises a conjunction of compression algorithms, entropy coding methods, source transformations, statistical estimation, and ingenuity. The intention of this book is to explain clearly and logically the stages of a compression system at a level comprehensible both to an undergraduate student and a practicing professional in the field. The usual objective of a coding system is compression efficiency, stated as the smallest rate (or number of bits) for a given distortion for lossy coding or smallest rate in lossless coding. However, other attributes may be even more important for a particular scenario. For example, in medical diagnosis, decoding time may be the primary concern. For mobile devices, low complexity in the form of small memory and low power consumption is essential. For broadcasting over packet networks, scalability in bit rate and/or and resolution may take precedence, so that from a single bitstream users can decode and display on devices having different reception and display capabilities. A compromise may have to be made between competing objectives, such as maximum compression efficiency and minimum complexity. Of course, one tries to obtain as much efficiency as possible for the given set of attributes wanted for the system. We shall elucidate the principles and methods for obtaining efficient compression and achieving the attributes of scalability in rate and resolution. We shall feature the modern wavelet compression methods of the FBI Fingerprint Compression Standard, the JPEG2000 Standard, Embedded Zerotree Wavelet (EZW), SPIHT (Set Partitioning in Hierarchical Trees) coding, and SPECK (Set Partitioning Embeded bloCK) coding.

CHAPTER 2

Characteristics of the Wavelet Transform

The basis functions of the wavelet transform are contained within a number of orthogonal or biorthogonal sets of time- or space-shifted waveforms. Within a set, the resolution or scale is the same, but between sets the scale differs by an octave (factor of 2). In fact, the set of waveforms at a given scale is a basis for the basic waveform in the set in the next lower scale. Therefore, the wavelet transform decomposes signals into a hierarchy of increasing resolutions. The coefficients are grouped into subbands belonging to different resolutions or scales with octave frequency separation. A depiction of this hierarchy is shown in Fig. 2.1 for three levels of decomposition of the lowest frequency subband of the 512 × 512 Lena image. The lowest frequency subband at a given scale consists of a smoothed and scale-reduced version of the original image. The wavelet transform to this scale comprises this scale's lowpass image with all the higher frequency subbands produced by decompositions up to this scale. The number of transform coefficients is equal to that of the original image. Clearly, this transform is a natural platform for producing streams of code bits (hereinafter called *codestreams*) that can be decoded at multiple resolutions. Furthermore, since the basis functions are space-shifted replicas of short waveforms, the coefficients retain local characteristics of the image.

Most natural images exhibit wavelet transforms with magnitudes of their coefficients generally decreasing as the subband scale or resolution increases. In other words, most of the energy of these images is packed into the lower frequency subbands. Furthermore, there are intra-subband and inter-scale statistical dependencies that can be exploited for compression. The values of the coefficients may be negative, except in the lowest frequency subband in the top left corner, and require precision exceeding the eight bits of most displays. Therefore, for display purposes, the coefficients in all subbands were scaled to the range of 256 grey levels. In all but the lowest frequency subband, the zero value corresponds to the middle grey level of 128. One can clearly see the edge detail propagating across scales to the same relative locations in the higher frequency subbands as one moves from the top left to the top right, bottom left, or bottom right of the transform. However, this detail becomes quite faint in the highest frequency subbands, where the coefficients are predominantly middle grey with actual values of zero. Notice also that within subbands, the close neighbors of a coefficient do not change much in value, except across edges. All these characteristics make the wavelet transform an effective platform for efficient compression with the attributes of resolution scalability and random access decoding. The arrangement of the coefficients into subbands belonging to different scales or resolutions makes possible encoding or decoding different scales. Random access to selected regions

of interest in the image is possible, because of the local nature of the transform, since one can select regions of subbands at different scales to decode a particular region of an image.

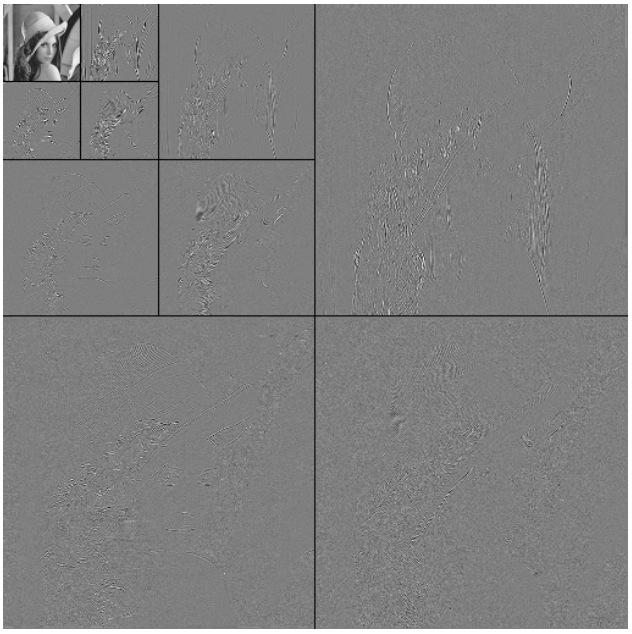

Figure 2.1: Hierarchy of three levels of decomposition of the 512 × 512 Lena image. The sizes of the successive levels are 256 × 256, 128 × 128, and 64 × 64. Middle grey level of 128 corresponds to 0 value of a coefficient in all subbands, excluding the lowest frequency one in the top left corner.

The subband labelling diagram for the wavelet transform in Fig. 2.1 is depicted in Fig. 2.2. This subband arrangement was produced by three stages of alternate low and high pass horizontal and vertical filterings of the resulting low horizontal and low vertical subbands followed by 2:1 downsampling. (The first stage input is just the source image itself.) The filters are discrete-time (or discrete-space) filters, so the transform is called a discrete wavelet transform (DWT). We show the analysis and synthesis for two stages explicitly in Figures 2.3 and 2.4. The LL2 (low horizontal, low vertical, second stage) subband is just a coarse, factor of 2^2 reduction (in each dimension) of the original image. Upsampling and filtering of LL2, LH2, HL2, and HH2 subbands yields the LL1 subband, which is two times the scale of LL2, but still scaled down by a factor of 2^1 from the original. And so it repeats on the LL1 subband for one more stage to obtain the full scale reconstruction of the original input. Therefore, at each synthesis stage, we can obtain a reduced scale version of the original image.

The best filters to use from the standpoint of achieving the best compression efficiency or highest coding gain have real number tap values, represented as floating point numbers that are precise only within the limits of the computer. These filters produce floating point wavelet coeffi-

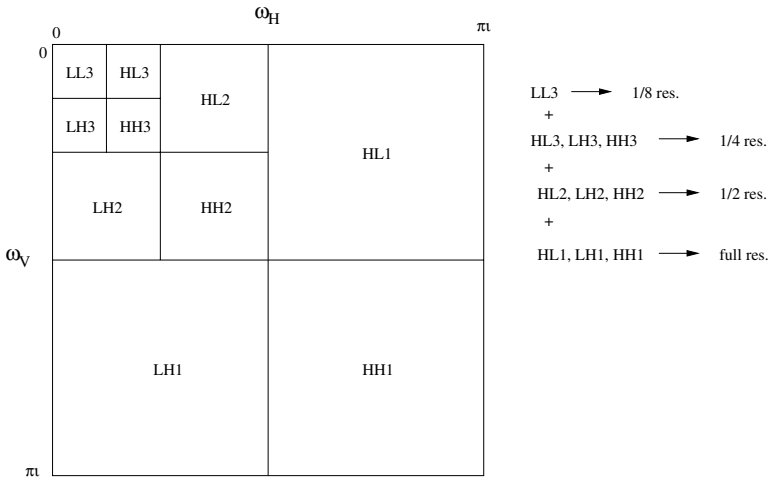

Figure 2.2: Subbands of a three-level, dyadic wavelet transform.

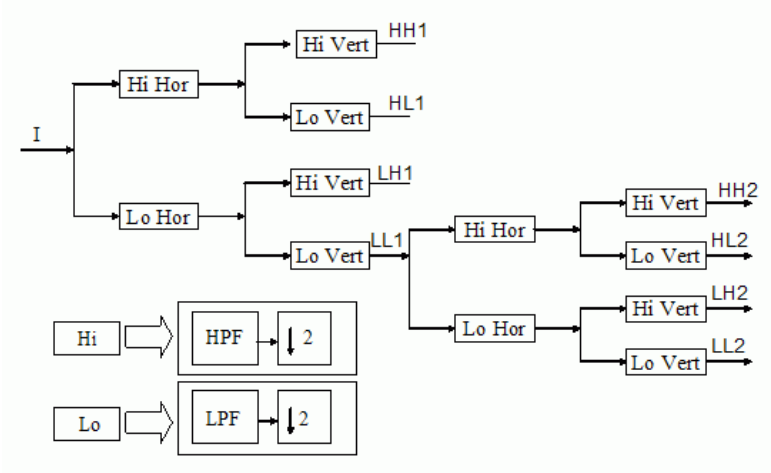

Figure 2.3: Two-level recursive lowpass filter analysis of image I.

cients. Hence, the inverse transform, done again with floating point filters, may not produce an exact replica of the source. For example, if a filter tap value contains a factor of $1 + \sqrt{3}$, then the floating point representation can not be exact and all subsequent mathematical operations will propagate this inexactness. Coding the wavelet coefficients means converting them to a compact integer representation. Therefore, even if the coding is perfectly lossless, the inverse wavelet transform may reconstruct the image with error from the original. Therefore, one can not guarantee perfectly lossless image compression in a wavelet transform system using floating point filters. One achieves what is often

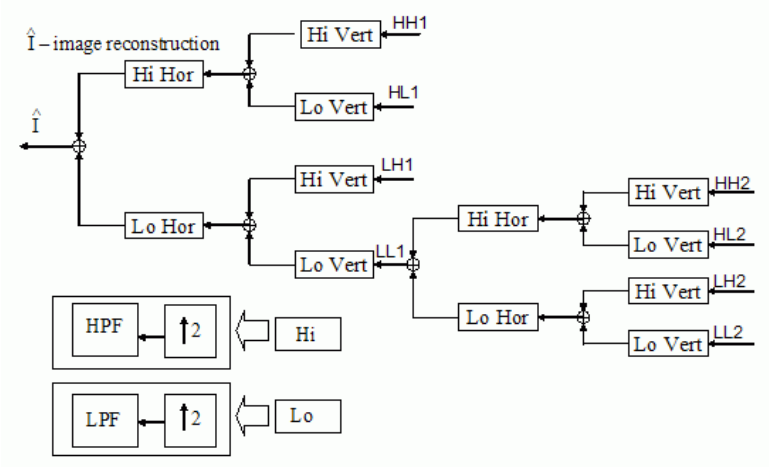

Figure 2.4: Two-level recursive lowpass filter synthesis of image $\hat{I}$.

called *virtually lossless* reconstruction. In most practical applications, that is not a hindrance, but sometimes, often for legal reasons that arise in certain fields such as diagnostic medicine, it is crucial to achieve perfectly lossless compression. Therefore, for perfectly lossless image compression in a wavelet-based coding system, one must use filters that operate with integer arithmetic to produce integer transform coefficients. This also assures that there are no errors due to limited precision in the synthesis stage. There will be a small, usually tolerable, reduction in potential coding gain from the use of these integer-to-integer filters for wavelet transformation.

2.1 WAVELET PACKET TRANSFORM

The subbands of a true wavelet transform reduce in size by factors of two in each dimension at every decomposition level. This phenomenon occurs because only the lowpass subband is decomposed to produce the next lower resolution. When one decomposes in addition one or more higher frequency subbands at a given level, one obtains what is called a *wavelet packet transform*. Suppose, for example, when we decompose the HL2 and HL1 subbands in Fig. 2.2, we obtain the subband arrangement of Fig. 2.5. In this way, one is able to capture finer detail in the high horizontal and low vertical spatial frequencies.

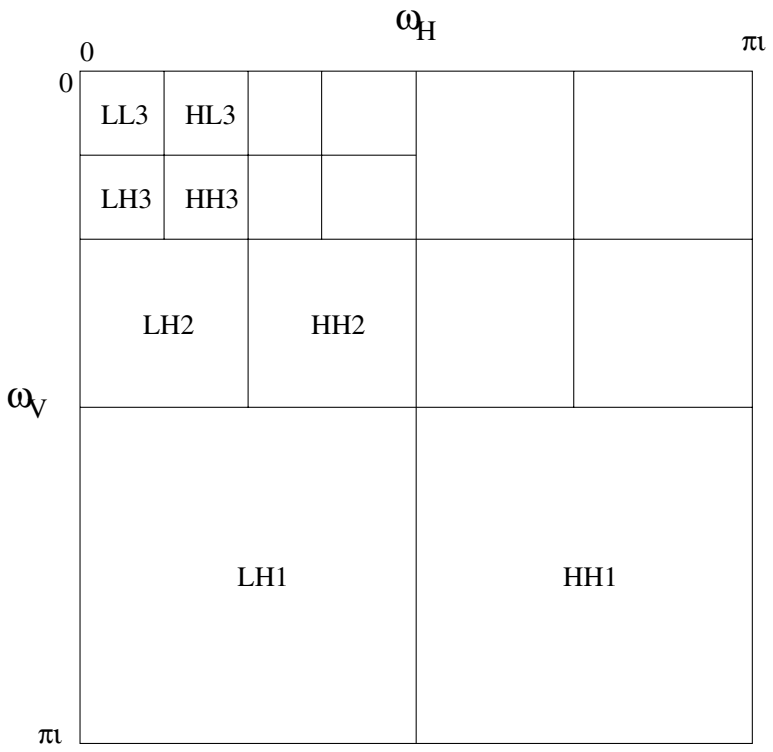

Figure 2.5: Subband arrangement of a wavelet packet transform, where the HL1 and HL2 subbands of the wavelet transform were further decomposed.

CHAPTER 3

Generic Wavelet-based Coding Systems

The generic wavelet transform coding system, regardless of the source, normally starts with subband/wavelet transformation of the source input. There is often then an optional pre-processing step that consists of statistical estimation, segmentation, weighting, and/or classification of the transform coefficients. Then the coefficients are subjected to quantization and/or set partitioning, usually followed by entropy coding, such as Huffman or arithmetic coding. Along with overhead information generated by the pre-processor, the encoded transform coefficients are written to the output codestream. The block diagram of this generic system is shown in Fig. 3.1.

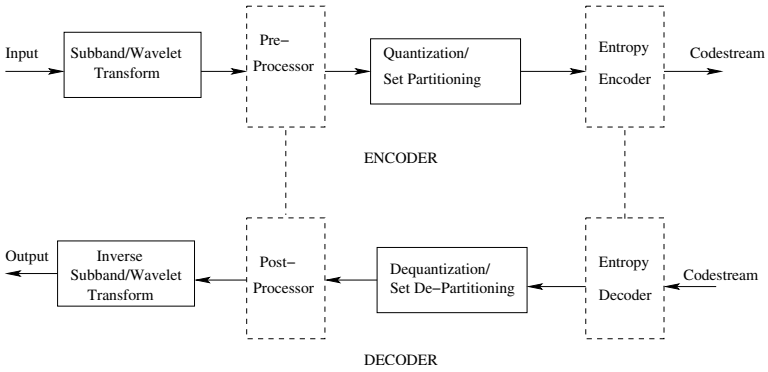

Figure 3.1: Encoder and decoder of subband/wavelet transform coding system. The boxes with dashed lines denote optional actions.

The decoding system, also shown in Fig. 3.1, reverses the process by decoding the codestream to reconstruct the wavelet transform, post-processing this transform, according to the pre-processor's actions, and then inverting the wavelet transform to reconstruct the source. One simple example of pre-processing is weighting transform coefficients to affect the distribution of code bits, in order to enhance visual quality of an image or aural quality of audio. The post-processing step must do inverse weighting, or the reconstruction will be distorted. The pairs of the pre- and post-processors and the entropy encoder and decoder are optional for some coding systems, so are depicted in dashed line boxes in Fig. 3.1.

3.1 QUANTIZATION

The goal of quantization is to output an integer that determines the range of values of the input signal. Within the Quantization/Set Partitioning system block in Fig. 3.1, quantization is necessary for floating point transform coefficients and optional for integer ones. Any quantization will result in reconstruction error, which is unavoidable for floating point coefficients. The coefficients in wavelet coding systems are quantized with a particular type of uniform quantizer, called a uniform, null-zone quantizer. For this quantizer, thresholds are uniformly spaced by step size Δ, except for the interval containing zero, called the null- (or dead-) zone, which extends from $-t\Delta/2$ to $+t\Delta/2$, where $1 \leq t \leq 2$. In this interval, the input is quantized to 0. Most often, $t = 2$, giving a double width null zone. Figure 3.2 illustrates the input-output characteristic of a uniform, double-width null-zone quantizer with seven quantizer levels and mid-point reconstruction values ($\xi = 0.5$ in Eqn. (3.1) below).

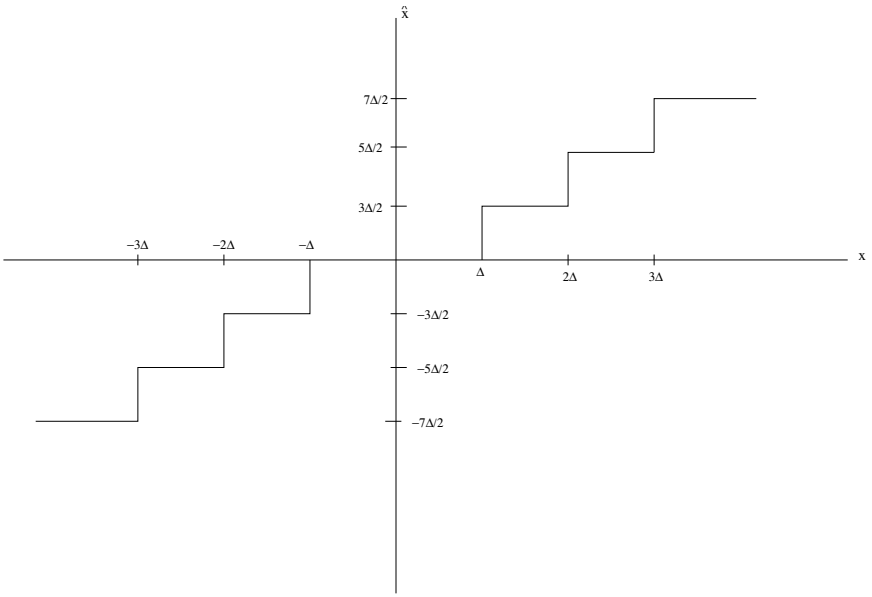

Figure 3.2: Input-output characteristic of a seven level, uniform null-zone quantizer.

This quantization is enacted by scaling by the step size and truncating to integers to produce the indices of the decision intervals (often called quantization bins). The mathematical operation upon the input x to produce a bin index q, given a quantizer step size Δ is

$$q = \begin{cases} \lceil x/\Delta - t/2 \rceil, & x \geq 0 \\ \lfloor x/\Delta + t/2 \rfloor, & x \leq 0 \end{cases} \tag{3.1}$$

The reconstruction (de-quantization) y is given by

$$y = \begin{cases} (q - 1 + \xi + t/2)\Delta, & q > 0 \\ (q + 1 - \xi - t/2)\Delta, & q < 0 \\ 0, & q = 0 \end{cases}, \tag{3.2}$$

where $0 \leq \xi < 1$. Heretofore, the bin index q will be called the *quantizer level*. The parameter ξ is often set to place the reconstruction value at the centroid of the decision interval. It has been derived through a model and confirmed in practice that $\xi \approx 0.38$ usually works well. In many cases, $\xi = 0.5$ is used, which places the reconstruction at the interval's midpoint. It is important to realize that when $-t\Delta/2 < x < t\Delta/2$, the quantizer level and reconstruction value are both 0. For a subband or linear transform, there may be many coefficients, belonging especially to higher frequencies, that are set to 0. The array of quantizer levels (bin indices) q are further encoded losslessly. As will be seen, clusters of zeros can be represented with very few bits.

For optimal coding that minimizes mean squared error for a given number of bits, statistically independent subbands encoded with non-zero rate should exhibit the same mean squared error. Although not statistically independent, the subbands are often modeled as independent and are encoded independently. In such circumstances, use of the same step size for all subbands minimizes the mean-squared reconstruction error for high rates and orthonormal synthesis filters. In accordance with this usual model, notwithstanding less than optimal coding, all subbands encoded with non-zero rates are quantized with the same step size Δ.

Often, the pre-processor function of weighting subbands to enhance perceptual quality or to compensate for scaling of non-orthonormal synthesis filters is combined with quantization by using different step sizes among the subbands. In this case, there is posited a set of step sizes, $\{\Delta_m\}$, $m = 1, 2, \ldots, M$, with Δ_m being the step size for subband m. The quantization follows the same formulas as (3.1) and (3.2).

Entropy coding of the indices normally follows the quantization. Entropy coding is a lossless operation that attempts to produce a minimum number of output bits, which is theoretically equal to the entropy of the input probability distribution. If the input signal is an independent, identically distributed (i.i.d.) Gaussian process, the optimal mean squared error $d(R)$ per sample of the quantizer/entropy coder for a bit rate R is given approximately by

$$d(R) = g\sigma^2 2^{-2R},$$

where g is a constant and σ^2 is the variance of the input signal to the quantizer. Actually g is not a constant, but varies with rate R much more slowly than exponentially. We can state that $g > 1$ for $R > 0$ and $g = 1$ for $R = 0$.

In almost all wavelet transform coding methods, the quantizer levels are represented by their sign and magnitude. In the sections that follow, we shall describe specifically some of the many methods used to code the quantizer levels of wavelet transforms of images. For images that are represented by 8 bits per pixel per color, the quality criterion is peak signal-to-noise ratio (PSNR),

defined by

$$\text{PSNR} = \frac{255^2}{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(x[i,j]-\hat{x}[i,j])^2} \tag{3.3}$$

where $x[i, j]$ and $\hat{x}[i, j]$ denote the original and reconstructed image values, respectively, at coordinates (i, j), and M and N the total row and column elements, respectively. The denominator is just the mean squared error per pixel. Usually PSNR is expressed in dB, so that it expresses the dB difference between the peak value and RMS (root mean squared) error. We remind the reader that mean-squared error is exactly the same whether calculated in the source or transform domain, if the transform or filters are orthonormal.

3.2 RATE ALLOCATION

Now that our input signal has been converted to a stream of integer indices, the question arises as to how to encode them and with how many bits. The early subband coding methods relied on statistical models to answer these questions. We adopt the usual naive model of the subband processes that is strictly true only when the source is Gaussian and the subband processes are white. That is, we assume that the coefficients within each subband are Gussian and independent with the same probability distribution. Therefore, each coefficient in a given subband will be quantized and encoded independently and have the same distortion (mean squared error) versus rate function. Among different subbands, only the variances will differ, so that their distortion-rate functions will be the same unit variance characteristic,

$$\rho(R) = g2^{-2R},$$

scaled by the subband variances.

For the sake of simplicity, we shall ignore the dyadically increasing sizes of subbands and consider a general subband decomposition of the source with N samples into M subbands, denoted by vectors $\mathbf{y}_m$, each with n_m coefficients, $m = 0, 1, 2, \ldots, M-1$. We can envision that each subband was produced by an equivalent analysis filter for its frequency range and downsampling by a factor of $V_m = N/n_m$. To reconstruct the source, each subband is upsampled by a factor of V_m and interpolated by a (conjugate) filter in its corresponding frequency range. Then the sum of these upsampled and interpolated subband processes reconstructs the source. A block diagram of this subband coding system is depicted in Fig. 3.3. We shall assume that the filters are orthonormal, so that the total squared error and variance sum to the same value in the signal and transform domains.

The first objective in coding is allocating a given number of bits among the subbands to achieve the minimum distortion. Distortion is defined to be mean squared error. Let there be N samples from the source and a given code rate of R bits/sample. If b_m bits are given to the m^{th} subband, then

$$R = \frac{1}{N}\sum_{m=0}^{M-1} b_m \tag{3.4}$$

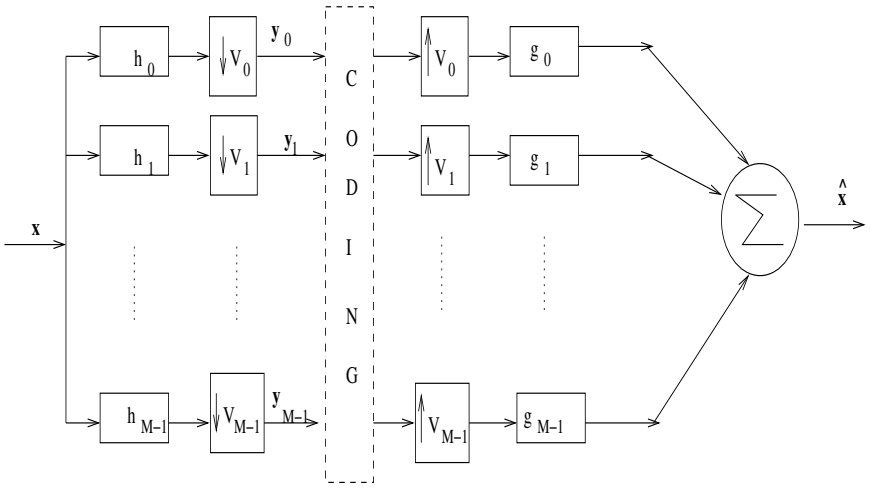

Figure 3.3: A subband analysis, coding, and synthesis system.

because the subbands are statistically independent and coded independently.

We assign the same rate r_m to code the coefficients of $\mathbf{y}_m$, so that $b_m = n_m r_m$. Therefore, the average code rate in bits per sample can be expressed as

$$R = \sum_{m=0}^{M-1} \eta_m r_m. \tag{3.5}$$

where $\eta_m = 1/V_m = n_m/N$, the fraction of the number of samples (coefficients) in the subband $\mathbf{y}_m$. Each subband $\mathbf{y}_m$ has samples with the same variance σ_m^2, $m = 1, 2, \ldots, M$. The subband samples (or coefficients) are then quantized to obtain minimum mean squared error for the given rate r_m. We shall denote this distortion versus rate function by $d_{Q_m}(r_m)$.

Since all n_m samples in the subbands are independently encoded with the same rate r_m, the distortion in the subband is

$$D_m(r_m) = n_m d_{Q_m}(r_m),$$

and the overall distortion per sample is

$$\begin{aligned} D &= \frac{1}{N} \sum_{m=0}^{M-1} D_m(r_m) && (3.6) \\ &= \frac{1}{N} \sum_{m=0}^{M-1} n_m d_{Q_m}(r_m) \\ &= \sum_{m=0}^{M-1} \eta_m d_{Q_m}(r_m) && (3.7) \end{aligned}$$

The assumed orthonormality of the filters means that the sum squared error remains the same after upsampling by V_m and filtering in each subband. Therefore, D is also the per-sample distortion of the reconstructed signal.

We seek the set of rates $\{r_m\}$ that minimizes the Lagrangian objective function

$$J(\lambda) = \sum_{m=0}^{M-1} (\eta_m d_{Q_m}(r_m) + \lambda \eta_m r_m) \tag{3.8}$$

for Lagarange parameter $\lambda > 0$ and rates $r_m \geq 0$ for all $m = 0, 1, 2, \ldots, M-1$. Therefore, for a given average target rate R, we seek the set of rates $\{r_m\}_{m=1}^{M}$ that minimize the distortion D in (3.7) subject to the average rate constraint

$$\sum_{m=0}^{M-1} \eta_m r_m = R. \tag{3.9}$$

Differentiating $J(\lambda)$ with respect to each r_m and setting to zero, we obtain the following solution:

$$g\sigma_m^2 2^{-2r_m} = d_{Q_m}(r_m) = g\theta, \quad m : r_m \geq 0, \tag{3.10}$$

where $\theta = \lambda/(2g \ln 2)$ is a re-scaled Lagrange parameter. Notice that the per-sample distortion is equal across all subbands receiving non-zero rate. Solving the above equation for r_m, we obtain

$$r_m = \begin{cases} \frac{1}{2} \log \frac{\sigma_m^2}{\theta}, & \sigma_m^2 \geq \theta \\ 0, & \sigma_m^2 < \theta \end{cases}$$

For $\sigma_m^2 < \theta$, the rate formula in the first line above yields negative values, so the best we can do is set $r_m = 0$ for $\sigma_m^2 < \theta$. In this case, the minimum distortion is σ_m^2. Let the index set of positive rates be defined as $\mathcal{I}_p = \{m : \sigma_m^2 > \theta\}$. Therefore, from (3.5) and (3.7), the rate and distortion are given by the following expressions.

$$R = \sum_{m \in \mathcal{I}_p} \eta_m \frac{1}{2} \log \frac{\sigma_m^2}{\theta} \tag{3.11}$$

$$D = \sum_{m \in \mathcal{I}_p} \eta_m g\theta + \sum_{m \notin \mathcal{I}_p} \eta_m \sigma_m^2 \tag{3.12}$$

Sometimes, it is advisable to employ a perceptual model whereby the mean squared error in each subband is multiplied by a weighting factor w_m. Then the solution in (3.10) is modified so that

$$w_m d_{Q_m}(r_m) = g w_m \sigma_m^2 2^{-2r_m} = g\theta.$$

So in all formulas that follow, we can replace the subband variances σ_m^2 by the weighted variances

$$\sigma_m'^2 = w_m \sigma_m^2 .$$

Also, when the filters are not orthonormal, we have to employ weighting factors to the filter responses in certain subbands to make them orthonormal. The squares of these weighting factors are likewise absorbed into the variances in the formulas for rate allocation.

Iterative Procedure Another way to calculate the optimal rate allocation is facilitated by ordering the variances σ_n^2 from largest to smallest, and re-indexing them so that

$$\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_N^2$$

If the variances are weighted, substitute weighted variances for variances in the above ordering and in all the expressions that follow.

As the specified target rate R_T increases, more and more variances exceed the threshold θ. Let us assume that for some rate R, there are N_θ coefficients whose weighted variances exceed θ. Therefore, the rate R can be expressed as

$$R = \frac{1}{N} \sum_{n=0}^{N_\theta - 1} \frac{1}{2} \log \frac{\sigma_n^2}{\theta} \tag{3.13}$$

$$= \frac{1}{2N} \log \frac{\prod_{n=0}^{N_\theta - 1} \sigma_n^2}{\theta^{N_\theta}} \tag{3.14}$$

Solving for θ and substituting it into the rate formulas

$$r_n = \frac{1}{2} \log \frac{\sigma_n^2}{\theta} \quad n = 0, 1, 2, \ldots, N_\theta - 1,$$

we obtain the rate assignments

$$r_n = \begin{cases} \frac{NR}{N_\theta} + \frac{1}{2} \log \frac{\sigma_n^2}{(\prod_{n=1}^{N_\theta} \sigma_n^2)^{1/N_\theta}}, & 0 \leq n \leq N_\theta - 1 \\ = 0, & N_\theta \leq n \leq N - 1 \end{cases} \tag{3.15}$$

A simple iterative procedure for determining N_θ is to set initially $N_\theta = N$ and calculate the r_n's according to Eqn. (3.15). If all the rates r_n are non-negative, then they are the optimal rates. Otherwise, iterate steps of decrementing N_θ by 1 and calculating the rates until all the rates are non-negative. The explicit steps of this procedure follow.

1. Set $N_\theta = N$.
2. Calculate rates r_n using Eqn. (3.15).
3. If $r_n \geq 0$, for all $n = 0, 1, 2, \ldots, N - 1$, stop. Otherwise, set $N_\theta = N_\theta - 1$ and go to Step 2.

These rates are numbers of bits per transform coefficient, so the question arises whether they should be integers. When you consider the source as a sequence of N-dimensional vectors,

the transform coefficients are considered to be independent scalar sources. These scalar sources are quantized and encoded with a variable length entropy code. Each realization of an entropy code has to have an integer length in bits, but the average length should be close to the entropy, which need not be an integer. These rates are the average lengths of the entropy codes, so do not have to be integers.

3.3 SUBBAND CODING GAIN

The case of all subbands receiving non-zero bit rates is of special interest. The rate and distortion equations in (3.11) and (3.12) consequently simplify to

$$R = \sum_{m=0}^{M-1} \eta_m \frac{1}{2} \log \frac{\sigma_m^2}{\theta} \tag{3.16}$$

$$D = g\theta, \tag{3.17}$$

since $\sum_{m \in \mathcal{I}_p} \eta_m = 1$. Now we can solve for θ as

$$\theta = \left(\prod_{m=0}^{M-1} (\sigma_m^2)^{\eta_m} \right) 2^{-2R}, \tag{3.18}$$

and explicitly express the m-th subband rate and overall distortion per sample by

$$r_m = \frac{1}{2} \log \frac{\sigma_m^2}{\prod_{m=0}^{M-1} (\sigma_m^2)^{\eta_m}} 2^{-2R} \tag{3.19}$$

$$D = g \left(\prod_{m=0}^{M-1} (\sigma_m^2)^{\eta_m} \right) 2^{-2R}. \tag{3.20}$$

Now we are in a position to compare the distortions of subband coding and PCM coding at the same rate R. Independent PCM coding of samples of a Gaussian signal yields the distortion versus rate function

$$D_{PCM} = g\sigma^2 2^{-2R}, \tag{3.21}$$

where σ^2 is the signal variance. The ratio of these two distortions, D_{PCM} and D from (3.21) and (3.20), respectively, is defined to be the *subband coding gain*,

$$G_{SB} \equiv \frac{D_{PCM}}{D} = \frac{\sigma^2}{\prod_{m=0}^{M-1} (\sigma_m^2)^{\eta_m}}, \tag{3.22}$$

To prove that G_{SB} is truly a gain, we use $\sigma^2 = \sum_{m=0}^{M-1} \eta_m \sigma_m^2$ and the convexity of the logarithm function, as follows.

$$
\begin{aligned}
G_{SB} &= \frac{\sum_{m=0}^{M-1} \eta_m \sigma_m^2}{\prod_{m=0}^{M-1} (\sigma_m^2)^{\eta_m}} \\
\log G_{SB} &= \log \sum_{m=0}^{M-1} \eta_m \sigma_m^2 - \log \prod_{m=0}^{M-1} (\sigma_m^2)^{\eta_m} \\
&= \log \sum_{m=0}^{M-1} \eta_m \sigma_m^2 - \sum_{m=0}^{M-1} \log(\sigma_m^2)^{\eta_m} \\
&= \log \sum_{m=0}^{M-1} \eta_m \sigma_m^2 - \sum_{m=0}^{M-1} \eta_m \log(\sigma_m^2) \\
&\geq \log \sum_{m=0}^{M-1} \eta_m \sigma_m^2 - \log \sum_{m=0}^{M-1} \eta_m \sigma_m^2 = 0.
\end{aligned}
$$

Therefore,

$$G_{SB} \geq 1,$$

with equality if and only if σ_m^2 is the same for all m, which holds true when the signal process is white. The coding gain is often used as a figure of merit for different transforms. It essentially measures the degree of compaction of the signal energy among the transform coefficients.

CHAPTER 4

The FBI Fingerprint Image Compression Standard

A good example of applying the methods studied thus far resides in the FBI Fingerprint Image Compression Standard. This standard uses adaptive entropy-coded scalar quantization of wavelet packet subbands. We shall describe briefly the salient parts of this method.

The 64-subband arrangement of the chosen wavelet packet transform is shown in Fig. 4.1. The order of visiting the subbands for coding is indicated by the number affixed to the suband.

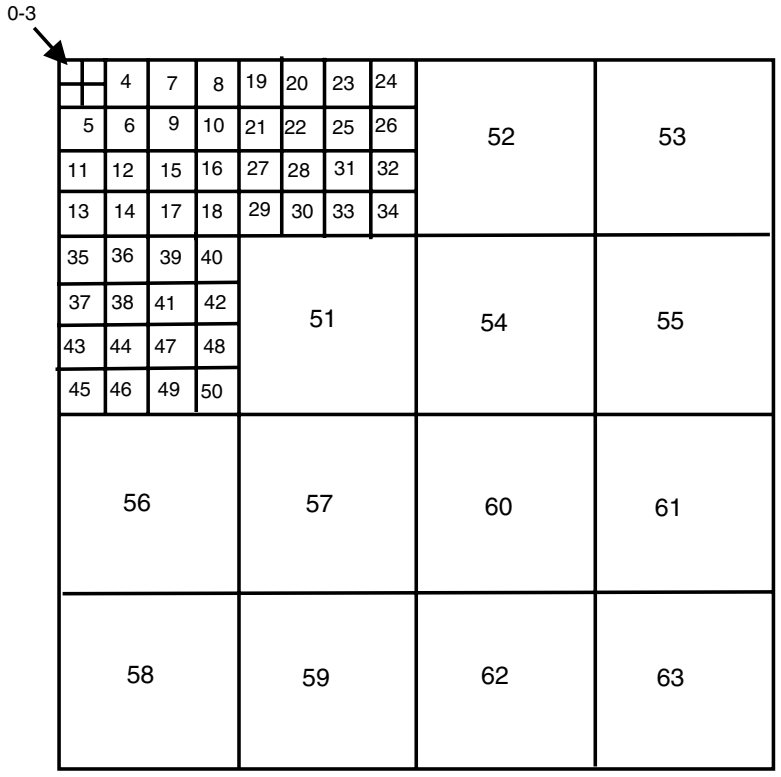

Figure 4.1: Subband arrangement of the FBI standard packet transform. The numbers indicate the order of coding the subbands.

The building block filter pair are the biorthogonal CDF (Cohen, Daubechies, Feauveau) 9/7 filters [9]. The sizes and locations of the subbands were chosen carefully in order to capture the fine detail of the ridges and valleys of fingerprints.

The image I is normalized prior to transformation according to the following formula

$$I' = \frac{I - \overline{I}}{F}, \quad F = \frac{1}{128}\max\{I_{\max} - \overline{I}, \overline{I} - I_{\min}\}, \tag{4.1}$$

where $\overline{I}$ is the image mean, $I_{\max}$ is the image maximum, and $I_{\min}$ is the image minimum. The purpose is to make the mean value of the lowest frequency subband close to zero. The other subbands have means approximately zero, whether normalized or not.[1]

Subbands 60, 61, 62, and 63 are not coded and are simply set to all zeros. The quantizers for the other subbands are uniform, null-zone quantizers, having different quantization intervals of width Δ_m outside the null zone. For every subband, $t = 1.2$ to give a null-zone width of $1.2\Delta_m$. The determination of the quantizer intervals Δ_m is driven by the perceptual and spectral characteristics of fingerprints and a scaling parameter dependent on the target rate R. Please refer to the references for the details [5, 15]. An algorithm iterating on values of θ that uses formulas analogous to (3.16) and (3.17) determines the subband rates r_m. The variances σ_m^2 are estimated from the image transform and are sent as overhead to the decoder. The parameter ξ is germane only to reconstruction in the decoder and is set to $\xi = 0.56$ through offline estimation of probability distributions in fingerprint wavelet transforms.

The quantizer outputs, which are indices of the quantizer bins, are entropy coded using an adaptive Huffman code of runlength/value symbols. For entropy coding only, the subbands are grouped into three to eight blocks. The entropy coding crosses subband boundaries, but remains within its block. Up to eight code tables are accepted by the standard. A grouping into three blocks using two code tables is shown in the table below.

Block	Subbands	Huffman Table
1	0–18	1
2	19–51	1
3	52–59	2

The breaking of blocks between subbands 18 and 19 and between 51 and 52 must be maintained in every grouping arrangement.

The Huffman code model uses distinct symbols for run lengths up to 100, distinct symbols for signed integers -73 to +74, and escape symbols for outliers. The definitions of the symbols for the model are presented in the table shown below.

[1] A non-zero mean random variable is almost always quantized after removal of its mean, which is then restored in the reconstruction.

Symbol	Value
1	zero run of length 1
2	zero run of length 2
3	zero run of length 3
⋮	⋮
100	zero run of length 100
101	escape for positive 8 bit quantizer bin index
102	escape for negative 8 bit quantizer bin index
103	escape for positive 16 bit quantizer bin index
104	escape for negative 16 bit quantizer bin index
105	escape for 8 bit zero run length
106	escape for 16 bit zero run length
107	quantizer bin index value -73
108	quantizer bin index value -72
109	quantizer bin index value -71
⋮	⋮
180	*Not used. Use symbol 1.*
⋮	⋮
253	quantizer bin index value 73
254	quantizer bin index value 74

For example, a run of 4 zeros followed by an index value of -9 corresponds to the composite symbol (4, 171) that is encoded with a Huffman codeword. The outlier of 127 zeros corresponds to the escape symbol 105 followed by the run length in hexadecimal 0x7F. When this run is followed by the non-outlier value +24, the composite symbol is (1050x7F, 204).

The probabilities of these composite symbols are estimated for each block and Huffman code tables are constructed following the same procedure detailed in the JPEG image compression standard. Please consult the references for further details [23, 37]. The decoder re-creates the composite symbols that correspond to their runlength and bin index values according to the table above. The bin indices are then mapped to their corresponding reconstruction values (see Eqn. (3.2)), which then undergo an inverse transform to produce the reconstructed image.

CHAPTER 5

Set Partition Embedded Block (SPECK) Coding

In this and following sections, we shall describe in detail methods for coding of subbands in which the bit allocation is not statistically based and which attempt to represent individual coefficient amplitudes with their minimal numbers of natural bits. We begin with a set partitioning method that has a quadtree representation and is part of the coding algorithm called SPECK (Set Partition Embedded bloCK).

Assume the subbands are square arrays of coefficients of size $2^m \times 2^m$ for some $m > 1$. The coefficients have been quantized to integer quantizer levels that are represented in sign and integer magnitude format. Let $c_{i,j}$ denote a quantizer level at coordinates (i, j) within the set $\mathcal{S}$. Heretofore, we shall call the quantizer levels *pixels*, as is customary. We specify a test of every $|c_{i,j}|$ in $\mathcal{S}$ against a threshold 2^n for some integer $n \geq 0$ by defining the following function:

$$\Gamma_n(\mathcal{S}) = \begin{cases} 1, & \max\limits_{(i,j)\in\mathcal{S}} \{|c_{i,j}|\} \geq 2^n, \\ 0, & \text{otherwise}, \end{cases} \tag{5.1}$$

Such a test is called a *significance test*. We say that the set is significant (for threshold 2^n) when there exists at least one pixel contained in the set that is greater than equal to the threshold. Otherwise, the set is deemed *insignificant*. To simplify the notation of single pixel sets, we write $\Gamma_n(\{(i, j)\})$ as $\Gamma_n(i, j)$.

When the set $\mathcal{S}$ tests as significant ($\Gamma_n(\mathcal{S}) = 1$), the square array is split into four subsets of equal size. A subband of size $2^m \times 2^m$ is then split into four quadrants, each of size $2^{m-1} \times 2^{m-1}$, as depicted for $m = 4$ in Fig. 5.1 where the 16×16 block is split into four 8×8 blocks. When a significant 2×2 set is split into four single pixels, we know now that we can encode insignificant pixels with n bits, because they are less than 2^n. If pixels in that set tested insignificant at threshold 2^{n+1} and test significant at 2^n, we can represent them with n bits, since their values are greater than 2^n and less than 2^{n+1}.

Recursive use of this basic procedure serves to code the pixels of the subband. We begin with the highest power of 2 threshold that does not exceed the maximum pixel magnitude in the array, defined formally as

$$T = 2^{n_{\max}}, \tag{5.2}$$

$$n_{\max} \equiv \left\lfloor \max_{(i,j)\in\mathcal{S}} \{\log_2 |c_(i, j)|\} \right\rfloor. \tag{5.3}$$

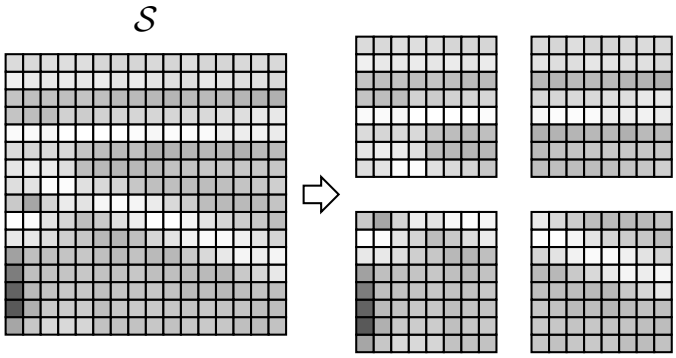

Figure 5.1: Partitioning of an array of pixels $\mathcal{S}$ into four equal-sized subsets. Gray values are used for representing pixel values.

We split the full size set into four quadrants. At least one of those quadrants contains a significant pixel (by definition of $n_{\max}$), so is significant. Label the significant quadrants with '1' and the insignificant quadrants with '0'. The 1-labelled quadrants are split again into four quarter-size sets, which are then labelled as '1' if significant and '0' if insignificant. All 0-labelled (insignificant) sets are left alone and their top left coordinates put on a list called the LIS (List of Insignificant Sets). Significant (1-labelled) sets continue to be quadrisected (quad-split) until significant pixels are isolated. An illustration of this procedure of recursive quadrisection and labelling quadrants is shown in Fig. 5.2 in an 8×8 pixel array.

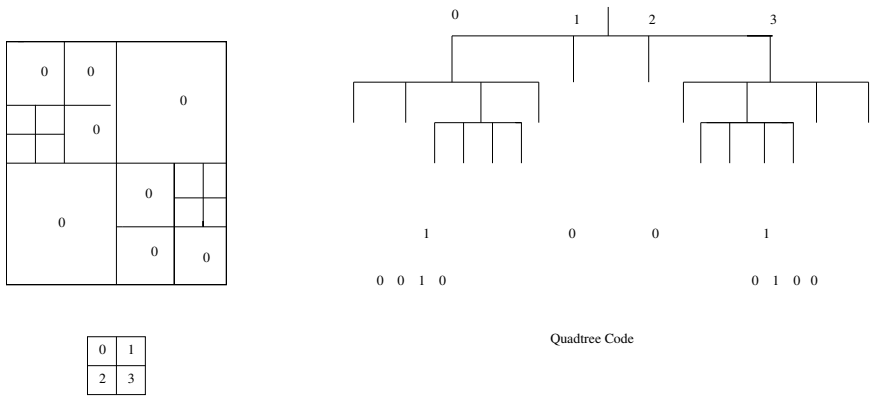

Figure 5.2: Three levels of quadrisection of 8x8 block and associated quadtree and code.

Every time a set is split into four quadrants, their labels create a four bit binary mask that can be mapped onto a quadtree. A split generates four branches and lack of a split creates a terminal

node. The quadtree and corresponding binary code are shown alongside the labelled squares of the 8×8 block in Fig. 5.2. This type of code is often called a *quadtree code* for this reason.

The insignificant (0-labelled) multi-pixel sets that have been recorded on the LIS have all their elements less than $2^{n_{\max}}$. Therefore, these elements can be represented with less than $n_{\max}$ bits, but we would like to further narrow down this bit range. To do so, we now lower the threshold exponent by 1 to $n_{\max} - 1$. We now repeat the testing and quadrisecting process at the threshold $2^n = 2^{n_{\max}-1}$ for the previously insignificant multi-pixel sets on the LIS. In this way, we locate significant and insignificant single pixels, each one of which can be represented with $n_{\max} - 1$ bits. Also, those multi-pixel sets that are found to be significant are removed from the LIS and newly insignificant (smaller) sets are added to the LIS in this process. We continue by iteratively lowering the current threshold exponent n by 1 to locate single pixels that can be encoded with one less bit. The process may be stopped at any $n > 0$. Pixels contained in sets remaining on the LIS are set to their most probable value of 0.

The quadtree code, which is the aggregate of 4-bit binary masks, conveys the execution path of the algorithm and therefore the sizes and locations of the sets and pixels. It is advantageous to entropy code these 4-bit masks instead of sending the raw quadtree code to the coded bitstream. Even a simple fixed Huffman code of 15 symbols (0000 cannot occur) shows improvement. A more sophisticated fixed Huffman code can be conditioned on the size of the block and/or the particular subband. Another level of sophistication is to adapt the code by estimating and updating the significance state probabilities as coding proceeds. Certainly, arithmetic coding may be substituted for Huffman coding in the above scenarios for entropy coding.

5.1 OCTAVE BAND PARTITIONING

We now consider the usual wavelet transform, where only the lowest frequency subband is split at each level. A good example is the subband arrangement of the three-level wavelet transform of Fig. 2.2. We wish to encode these subbands by the method introduced in the previous section. The order in which these subbands are coded matter greatly, because their statistical properties are different. A common order in which the subbands are coded follows the indicated zigzag path from lowest to highest frequency subband, as illustrated in Fig. 5.3. Coding efficiency is maximized when the higher magnitude coefficients are encoded before lower magnitude ones. Because energy in subbands tends to decrease with increasing spatial frequency in most natural images, the zigzag scan is often reasonably efficient for coding.

An inefficiency of the zigzag scanning mode is that each subband has to be tested individually for significance, so that areas larger than a single subband can never be insignificant and signified with a single 0 bit. To remedy this situation, another layer of partitioning, called *octave band partitioning* precedes the partitioning of the coding procedure. The idea is to consider the DWT as initially consisting of two sets, one being the LL band marked as an $\mathcal{S}$ set and the remainder of the DWT marked as an $\mathcal{I}$ set, as depicted in Fig. 5.4. The $\mathcal{S}$ set is coded by set partitioning, after which the $\mathcal{I}$ set is tested. If not significant, a '0' is output and the threshold is lowered or a new maximum is

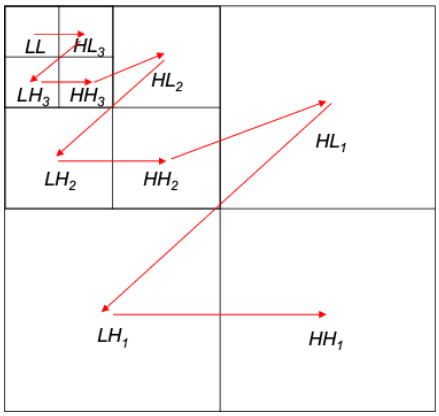

Figure 5.3: Scanning order of subbands in a 3-level wavelet decomposition.

calculated, depending on whether fixed or data-dependent thresholds are used. If $\mathcal{I}$ is significant, it is partitioned into three subbands marked as $\mathcal{S}$ at the same level adjacent to the previous $\mathcal{S}$ set and the remainder set marked as $\mathcal{I}$. This partitioning of $\mathcal{I}$ is illustrated in Fig. 5.5. Testing and partitioning of $\mathcal{I}$ continue in the same manner until $\mathcal{I}$ becomes empty, as indicated in Figure 5.5. Octave band partitioning was first used in wavelet transform coding by Andrew [3] in the so-called SWEET method and then utilized in SPECK [22], which will be described in the next section.

A binary digit is sent to the codestream after every significance test. An insignificant subband then is indicated by a single '0' at the given threshold. So the maximal threshold of a subband is conveyed by a succession of '0's and a single '1' when it becomes significant. Sometimes, a '0' indicating insignificance is shared among insignificant subbands, as in an $\mathcal{I}$ set. The zigzag subband scan sent the maximal threshold of every subband to the codestream. Although it did not require another bit to skip insignificant subbands, more bits are needed for conveying the maximal threshold of every subband than in the method of octave band partitioning.

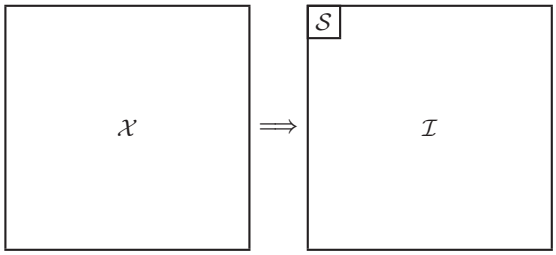

Figure 5.4: Partitioning of image $\mathcal{X}$ into sets $\mathcal{S}$ and $\mathcal{I}$.

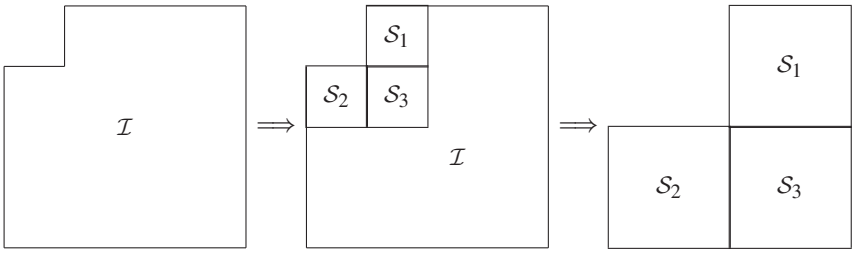

Figure 5.5: Partitioning of set $\mathcal{I}$.

5.2 PROGRESSIVE PROPERTIES

Encoding every subband completely in the zigzag order above produces a resolution-progressive codestream. Furthermore, each subband's codestream is approximately progressive in quality or value, because coefficients significant at larger thresholds precede those with smaller thresholds. Figure 5.6 illustrates the magnitude ordering of coefficients in a value-progressive scan. The coefficients are represented by their magnitude bits stacked vertically from highest to lowest significant bit. Although the subbands individually are progressive in quality, the composite codestream is not. The bits from larger coefficients in a subband scanned later may follow those of smaller coefficients from a subband scanned earlier. One can re-organize this codestream to be progressive in value, if we insert indicators in the form of markers or counts of bits into the codestream header, to separate the bits from coding passes at different thresholds. The code bits of coefficients significant for the same threshold are kept together in the codestream. With the aid of these indicators, we can gather together from different subbands the code bits of coefficients significant for the same threshold. Therefore, in the decoder, the coefficients with larger significance are decoded before those of lesser significance. This re-organization scheme does not order the values within the same threshold, so is only partially progressive in value. Furthermore, this re-organization may vitiate the progressiveness in resolution. One or more lower resolution subbands may not have significant coefficients at the current threshold, while higher resolution ones do. If we continue to collect the bits from these higher resolution significant coefficients at the current threshold, then we will not be keeping together code bits from the same resolution.

When coefficients are grouped to be partially progressive in value, they can be encoded so that bits belonging to a higher bitplane precede those belonging to lower ones in the coded bitstream. Such a coded bitstream is said to be *bit-embedded*. When a coefficient is found to be significant at a certain threshold, its coordinates are placed on a list called the LSP (List of Significant Pixels). The thresholds are decreased in octave steps, so Fig. 5.6 illustrates coefficients on the LSP where their highest '1' magnitude (most significant) bits decrease monotonically from top of the list on the left to bottom of the list on the right. As indicated by the arrows in Fig. 5.6, the value-progressive scan reads code bits of each coefficient from the most significant bit downward to the least significant

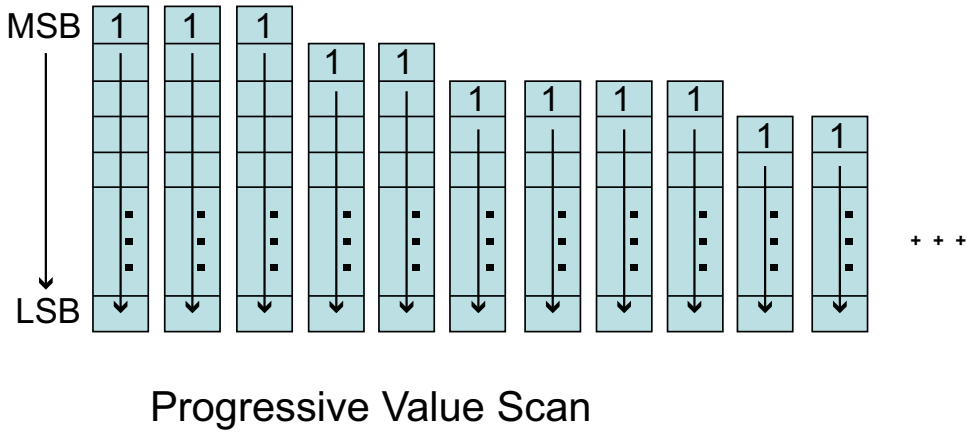

Figure 5.6: Coding order for a progressive value codestream.

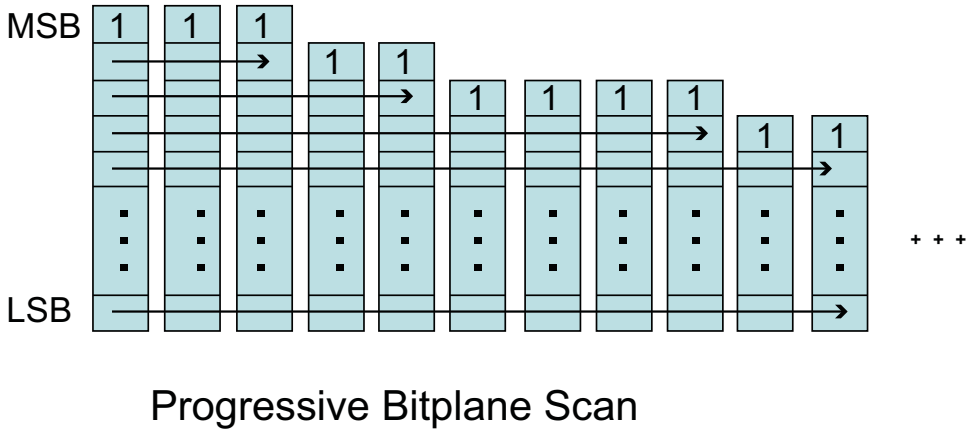

Figure 5.7: Coding order for a bit embedded codestream.

before proceeding to the next coefficient with the same most significant bit. We can produce a bit-embedded codestream, if starting with the largest most significant bitplane, we always move to the same bitplane of the next coefficient in the scan, whether in the same or a different subband. Referring to Fig. 5.7, this path corresponds to scanning a bitplane from extreme left to extreme right at the same horizontal (bitplane) level, dropping down one level, and returning to the extreme left to repeat the rightward scan holding the same level. This procedure is called *progressive bitplane coding*. A progressive bitplane coded bitstream can be truncated to any given length to yield the optimal code (producing the least mean squared error) for that length. Both progressive bitplane coding and

progressive value coding are said to be *quality or rate scalable*, since their codestreams contain all their lower rate or lower quality codestreams.

5.3 PROGRESSIVE QUALITY CODING WITH SPECK

The SPECK algorithm comprises the three procedures just described: quadri-section set partitioning, octave band partitioning, and progressive bitplane coding. The algorithm maintains two control lists:

1. LIS—list of insignificant sets, initially lowest frequency subband.
2. LSP—list of significant pixels, initially empty.

First, $n_{\max}$, the power of two of the initial threshold is calculated. The testing and partitioning of $\mathcal{S}$ sets on the LIS produce a series of 4-bit binary masks that can optionally be entropy coded, either by a Huffman or arithmetic code. Entropy coding of the masks usually results in bitrate reductions of a few percent. Once an $\mathcal{S}$ set has been iteratively quadri-sected into single elements, significant ones are added to the LSP and insignificant ones are added to the LIS. The sets in the LIS are ordered by size with the smallest at the top and the largest at the bottom. Once an $\mathcal{S}$ set has been quadri-sected down to single elements at a given threshold, the bits in the current bitplane of LSP elements previously found to be significant at higher thresholds are sent to the bitstream. These bits are called *refinement bits*, since they refine the values of the coefficients that are significant at higher thresholds by sending their lower order bits in the current bitplane. The threshold is then halved (or bitplane lowered by one), whereupon the whole procedure is repeated for multi-element LIS sets. This process continues until the bit budget is satisfied. For a description of the SPECK algorithm in pseudo-code, the interested reader is referred to the original SPECK paper [22] or more comprehensive journal paper [35].

	0	1	2	3	4	5	6	7
0	63	–34	49	10	7	13	–12	7
1	–31	23	14	–13	3	4	6	–1
2	15	14	3	–12	5	–7	3	9
3	–9	–7	–14	8	4	–2	3	2
4	–5	9	–1	47	4	6	–2	2
5	3	0	–3	2	3	–2	0	4
6	2	–3	6	–4	3	6	3	6
7	5	11	5	6	0	3	–4	4

Figure 5.8: Example of coefficients in an 8 × 8 transform used in coding examples. The numbers in the first row and first column outside the box are horizontal and vertical coordinates.

5.3.1 AN EXAMPLE OF SPECK CODING

In order to clarify the SPECK coding procedure, we present an example of encoding data of the type resulting from an 8×8 two-level wavelet transform. Fig. 5.8 depicts this data in the usual pyramidal subband structure.

The output bits, actions, and population of the lists are displayed in Table 5.1 and explained below for the top bitplane pass of the algorithm at $n = 5$. The following explains the notational conventions.

- i is row index and j is column index in coordinates (i, j).
- $S^k(i, j)$ under Point or Set denotes $2^k \times 2^k$ set with (i, j) upper left corner coordinate.
- $(i, j)\mathbf{k}$ under Control Lists denotes $2^k \times 2^k$ set with (i, j) upper left corner coordinate.
- (i, j) in LSP is always a single point.

We explain now the steps of the coding procedure for the top bitplane pass and summarize them in Table 5.1. Only the initial and final lists of this pass are shown in the table. The maximum magnitude of the full transform is 63, so $n = 5$ is the initial bitplane significance level. The initial $\mathcal{S}$ set is the top left 2×2 subband, so $\mathcal{S} = S^1(0, 0)$ and (0,0)**1** initializes the LIS, and LSP is initially empty. The set $S^1(0, 0)$ is tested and is significant, so it is quadrisected into four singleton sets added to the LIS and a '1' is output to the bitstream. These singleton sets (pixels) are tested in turn for significance. The point (0,0) with magnitude 63 is significant, so a '1' designating 'significant' and a '+' indicating its sign is sent to the bitstream and it is moved to the LSP. Likewise, (0,1) is significant and negative, so a '1—' is output and its coordinate is moved to the LSP. Both (1,0) and (1,1) are insignificant with magnitudes below 32, so a '0' is output for each and they stay in the LIS. Next, the remainder set $\mathcal{I}$ is tested for significance, so a '1' is sent and it is partitioned into three new $\mathcal{S}$ sets and a new $\mathcal{I}$. Each of these new $\mathcal{S}$ sets, $S^1(0, 2)$, $S^1(2, 0)$, and $S^1(2, 2)$ are processed in turn as was $S^1(0, 0)$. Of the three, only the first tests significant and is further quadrisected to four points, one of which, (0,2), moves to the LSP. The other three points, (0,3), (1,2), and (1,3) stay in the LIS and three '0's are output to indicate their insignificance. The other two insignificant $\mathcal{S}$ sets, $S^1(2, 0)$ and $S^1(2, 2)$, are added to the LIS after the single point sets with bold suffixes **1**, since they are size 2×2, larger than the single point sets. Three '0's are also output to indicate the insignificance of each.

The algorithm continues in this way until the $n = 5$ sorting pass is complete. The lists at the end of the $n = 5$ pass are the initial lists for the next sorting pass at $n = 4$. No octave band partitioning is performed for the $n = 4$ or lower bitplanes. Note that SPECK puts out 29 raw (uncoded) bits in this first ($n = 5$) pass. Subsequent entropy coding can reduce this number of bits.

The decoder will duplicate the action of the encoder when receiving the code bitstream. In fact, if you replace the words in the column "Output Bits" by "Input Bits" in Table 5.1, the same exact table will be built from the codestream.

Table 5.1: Example of SPECK Coding of Wavelet Transform, from Bit-plane $n = 5$

Comment	Point or Set	Output Bits	Action
$n = 5$ Sorting $\mathcal{S}$=$S^1(0,0)$, $\mathcal{I}$ = rest			LIS = {(0,0)**1**} LSP = ϕ
	$S^1(0,0)$	1	quad split, add to LIS(0)
	(0,0)	1+	(0,0) to LSP LSP = {(0,0)}
	(0,1)	1–	(0,1) to LSP
	(1,0)	0	none
	(1,1)	0	none
Test $\mathcal{I}$	$\mathcal{S}(\mathcal{I})$	1	split to 3 $\mathcal{S}$'s, new $\mathcal{I}$
	$S^1(0,2)$	1	quad split, add to LIS(0)
	(0,2)	1+	(0,2) to LSP
	(0,3)	0	none
	(1,2)	0	none
	(1,3)	0	none
Test $\mathcal{I}$	$S^1(2,0)$	0	add to LIS(1)
	$S^1(2,2)$	0	add to LIS(1)
	$\mathcal{S}(\mathcal{I})$	1	split to 3 $\mathcal{S}$'s
	$S^2(0,4)$	0	add to LIS(2)
	$S^2(4,0)$	1	quad split, add to LIS(1)
	$S^1(4,0)$	0	none
	$S^1(4,2)$	1	quad split, add to LIS(0)
	(4,2)	0	none
	(4,3)	1+	move (4,3) to LSP
	(5,2)	0	none
	(5,3)	0	none
	$S^1(6,0)$	0	none
	$S^1(6,2)$	0	none
	$S^2(4,4)$	0	add to LIS(2)

End $n = 5$	End of Pass Control Lists
	LIS = {(1,0)**0**,(1,1)**0**,(0,3)**0**, (1,2)**0**,(1,3)**0**,(4,2)**0**,(5,2)**0**,(5,3)**0**, (2,0)**1**,(2,2)**1** ,(4,0)**1**, (4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**, (4,4)**2**} LSP = {(0,0),(0,1),(0,2),(4,3)}

We follow with an example of reconstructions from SPECK coded images.

Example 5.1 Reconstructions of different quality from a SPECK codestream. The wavelet transform of the Goldhill image is encoded, but this time using progressive bit plane coding to produce a bit-embedded codestream of size 65,536 bytes (rate of 2.00 bits/pixel). (The transform coefficients are not scaled and are truncated to integers, if necessary.) The full codestream and its truncation to sizes of 32,768 bytes (1.00 bpp), 16,384 bytes (0.50 bpp), and 8,192 bytes (0.25 bpp) are then decoded, de-quantized, and inverse wavelet transformed. The reconstructions and their PSNR's are shown in Fig. 5.9.

5.4 PROGRESSIVE RESOLUTION CODING

Processing the subbands in the order of the zigzag scan or the order governed by octave band partitioning realizes the framework for progressive resolution coding. In either method, the (quantized) subbands are visited in order of low to high spatial frequency. When we encode a subband or resolution completely (passing through all thresholds) before moving to the next one, the subbands of the same resolution are kept together in the codestream. Therefore, one can truncate the codestream according to resolution. For example, in the ten subband wavelet decomposition shown in Fig. 2.2, one can strip all the bits belong to the highest resolution consisting of subbands HL_1, LH_1, and HH_1. Alternatively, one could stop coding just before reaching this resolution. In either case, the decoder can reconstruct a half resolution image by inverting the transform using one less stage of synthesis filtering or a full size image with reduced resolution by setting all coefficients in the highest resolution to zero in the final stage of synthesis filtering. The following example illustrates use and displays results from resolution scalable algorithm.

Example 5.2 Resolution scalable coding.
The source image is the 512×512, 8 bits per pixel, grey Goldhill image, shown in Fig. 5.10. This image is then wavelet transformed and its wavelet coefficients are then quantized through scaling by a factor of 0.31 (step size of 1/0.31) and truncating to integers. The resolution scalable coding algorithm described above encodes the wavelet transform's quantization bin indices losslessly to a rate of 1.329 bpp (codestream size of 43,538 bytes). Portions of this codestream corresponding to the desired resolution are then decoded, de-quantized, and inverse wavelet transformed, to produce the reconstructed images. Fig. 5.11 shows the reconstructed images for full, one-half, and one-quarter resolutions.

Figure 5.9: Reconstructions of Goldhill from same codestream by a quality scalable coding method.

(a) 2.00 bpp, 42.02 dB	(b) 1.00 bpp, 36.55 dB
(c) 0.50 bpp, 33.13 dB	(d) 0.25 bpp, 30.56 dB

5.5 THE EMBEDDED ZERO BLOCK CODER (EZBC)

The Embedded Zero-Block Coder (EZBC) [20] is a variant of SPECK that utilizes SPECK's quadri-section coding of full wavelet subbands and visits them in the zigzag order as shown in Fig. 5.3. The transition from one subband to the next can be either at the same threshold or after passing through all thresholds from top to bottom. The former produces a bit-embedded composite

Figure 5.10: Original 512 × 512, 8 bits/pixel Goldhill image.

codestream and the latter a resolution-progressive one with separate bit-embedded subband codestreams. What distinguishes EZBC from normal SPECK is the entropy coding of the significance map, sign bits, and refinement bits. The original SPECK used arithmetic coding of the 4-bit masks, but did no entropy coding of sign and refinement bits. EZBC entropy encodes the sign and refinement bits in the manner of JPEG2000, which will be described later. It also performs entropy coding of the significance decision bits in the masks adaptively using a context template consisting of significance bits in neighboring masks and parent subbands. An innovation is the use of conditional de-quantization, by which a crude estimate of the probabilistic centroid of the quantizer decision interval is calculated from next lower order refinement bits. For most images, the efficiency of EZBC surpasses that of JPEG2000, most likely due to the fact that JPEG2000 disallows out-of-subband context for arithmetic coding and conditional de-quantization in order to retain region-of-interest and random access capabilities. EZBC lacks these capabilities.

5.6 CODING OF SUBBAND SUB-BLOCKS

Coding blocks that are contained within subbands, unlike direct coding of blocks of source samples, will not produce blocking artifacts at low code rates, because the filtering in the inverse transform performs a weighted average of the reconstructed coefficients that crosses block boundaries.[1] This realization led to systems that divide the subbands into square blocks, typically 32 × 32 or 64 × 64

[1] This phenomenon holds also for full-size subband blocks, because of the successive upsampling, filtering, and combining needed to synthesize an image from its subbands.

Figure 5.11: Reconstructions from codestream of Goldhill coded to rate 1.329 bpp, quantizer step size = 1/0.31 at full, 1/2, and 1/4 resolutions.

in their dimensions, and that code these blocks. The subbands of a 3-level wavelet transform divided into these so-called *subblocks* are depicted in Fig. 5.12. The smallest subbands are not divided, if their size is comparable to that of the subblock. The figure therefore shows no division of the subbands at the coarsest level.

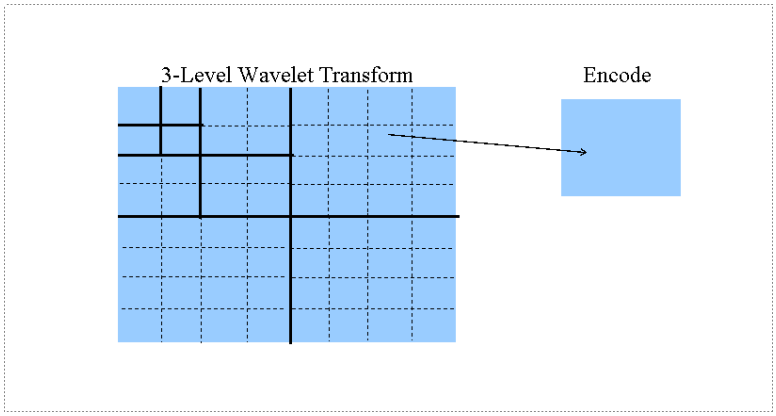

Figure 5.12: Division of wavelet subbands into subblocks. Note subbands of coarsest level are too small to be divided.

These subblocks may be encoded by any of the means previously presented for coding of blocks that are full subbands. All the subblocks within a given subband will be encoded before transition to the next subband. The order of encoding subblocks within a subband is pre-determined. Usually it is raster scan order, that is, horizontal from left to right and then vertical from top to bottom, but it could be the zigzag or some other space-filling scan. The advantages of coding subblocks are small memory usage and random access or region-of-interest (ROI) capability. The independent coding of subblocks allows the latter capability, because subblocks forming a group of spatial orientation trees can be selected from the image or the codestream for encoding or decoding a certain spatial region of the image.

The coding of subblocks of subbands entails extra codestream overhead in the form of the initial (largest) threshold for every subblock and markers to delineate boundaries between subblocks.[2] The thresholds are either group indices or bitplane maxima and can be compressed, if necessary. For example, consider again the 5-level wavelet decomposition of a 512×512 image, for a subblock size of 32×32. There are 64 subblocks in each of the three level-1 subbands, 16 subblocks in each of the three level-2 subbands, 4 subblocks in each of the level-3 subbands, 1 subblock in each of the three level-4 subbands, and 1 subblock making up remaining subbands for a total of 256 blocks. Assuming that the thresholds are maximal bitplane levels, no threshold can exceed 13 for an original 8-bit image. Therefore, four bits for each threshold gives an overhead of 1024 bits or 0.004 bits per pixel. At very low bit rates, such overhead may be problematical. These thresholds, especially those associated with subblocks within a subband, are statistically dependent and can be encoded together to save overhead rate. Both JPEG2000 and SBHP, to be described soon, use a quadri-section code of bitplane number differences from the maximum bitplane number of the subband.

5.7 THE SBHP METHOD

We now describe two methods to code the subblocks: SBHP and JPEG2000. First, we consider the simpler of the two methods, called SBHP for Subband Block Hierarchical Partitioning [8]. The coding method of SBHP is SPECK, initialized by an $\mathcal{S}$ set consisting of the 2×2 array of coefficients in the top left corner of the subblock and with the $\mathcal{I}$ as the remainder of the subblock, as illustrated in Fig. 5.4. When the threshold is lowered such that the $\mathcal{I}$ set becomes significant, the $\mathcal{I}$ set is split into three $\mathcal{S}$ sets adjoining the existing $\mathcal{S}$ and another $\mathcal{I}$ set for the remainder, as shown in Fig. 5.5. Recall that the $\mathcal{S}$ sets are encoded by recursive quadrature splitting according to significance tests with fixed decreasing power of 2 thresholds. For each subblock, the method uses an LIS list and an LSP list to store coordinates of insignificant sets (including singleton sets) and significant coefficients, respectively. However, the subblock is fully encoded from the top threshold down to the threshold associated either with some large bit rate for lossy coding or through the 2^0 bitplane for lossless coding. The lists are then cleared and re-initialized for the next subblock to be coded. All the subblocks in every subband are independently coded to the same lossy bit rate or to its lossless bit rate in the same way. We pass through the subbands in the progressive

[2] Often, instead of markers, the counts of code bytes representing the subblocks are written to the codestream header.

resolution order and through subblocks within a subband in raster order. We obtain a sequence of codestreams, one for each subblock, where each codestream is bit-embedded, but when assembled together into one composite codestream is not embedded in any sense. We may reorganize this composite codestream by interleaving the sub-codestreams at the same bit plane level to obtain an embedded codestream. The effect is the same as holding the threshold in the current pass across the subblocks and subbands, before lowering the threshold for the next pass. We can exercise rate control by cutting the reorganized codestream at the desired file size.

Another way to control the rate is the method to be described later in more detail in Section 7. By this method, a series of increasing slope parameters $\lambda_1, \lambda_2, \ldots, \lambda_J$ are specified and every sub-codestream is cut, so that the magnitude of its distortion-rate slope matches each parameter in turn until the desired total number of bits among the codestreams is reached. Because SPECK codes subblocks in progressive bitplane order, approximate, but fairly accurate calculations of distortion changes with rate can be made very fast and in-line by counting the numbers of newly significant bits at each threshold. Then the codestream can be reorganized for quality and/or resolution scalability.

Entropy Coding in SBHP The sign and refinement bits in SBHP are not entropy coded. However, the 4-bit masks are encoded with a simple, fixed Huffman code for each of three contexts. These contexts are defined in the following table.

Table 5.2: Contexts for SBHP entropy coding

Context	Meaning
0	Sets with more than 4 pixels
1	Sets with 4 pixels revealed during a previous pass
2	Sets with 4 pixels revealed in the current pass

For each context, a simple, fixed Huffman code is used for coding the 15 possible mask patterns. Recall that all-zero cannot occur. The label of a pattern is the decimal number corresponding to the four binary symbols read in raster order, left-to-right and top-to-bottom. Table 5.3 contains Huffman codes for each context.

This entropy code having just three simple contexts, requiring no calculation, and composed of non-adaptive codes of only 15 symbols for each context, adds very little complexity and is quite effective for natural, non-synthetic gray level images and leads to very fast encoding and decoding.

5.8 JPEG2000 CODING

The framework for the entropy coding engine of the JPEG2000 coding system is the same as that of SBHP. JPEG2000 encodes subblocks of subbands independently and visits the subbands and subblocks in the same order as previously described. The entropy coding engine is called EBCOT for Embedded Block Coding with Optimized Truncation and is the brainchild of David S. Taubman [49]. In the parlance of JPEG2000, the subblocks are called *code-blocks*. The method of coding

Table 5.3: Huffman Codes for Four-Bit Masks in SBHP

Symbol	Context					
	0		1		2	
	Length	Codeword	Length	Codeword	Length	Codeword
1	3	000	3	000	2	00
2	3	001	3	001	3	010
3	4	1000	4	1000	4	1010
4	3	010	3	010	3	011
5	4	1001	4	1001	4	1011
6	5	11010	4	1010	5	11100
7	5	11011	5	11010	6	111010
8	3	011	3	011	3	100
9	5	11100	5	11011	6	111011
10	4	1010	4	1011	4	1100
11	5	11101	5	11100	6	111100
12	4	1011	4	1100	4	1101
13	5	11110	5	11101	6	111101
14	5	11111	5	11110	6	111110
15	4	1100	5	11111	6	111111

is context-based, binary arithmetic coding (CABAC) of bitplanes from top (most significant) to bottom (least significant), as will be described.

First of all, there is no sorting procedure, as in the set partitioning coders, to partially order the coefficients' quantizer values (indices) by their most significant (highest non-zero) bitplanes. Henceforth, we shall call the location of a coefficient a *pixel* and its quantizer level (index) a *pixel value*. We start with the highest non-zero bitplane in the code-block, i.e., having index $n_{\max}$. When encoding any given bitplane, we execute three passes:

1. the Significance Propagation Pass (SPP)
2. the Magnitude Refinement Pass (MRP)
3. the Clean Up Pass (CUP)

Each of these passes encodes a different set of bits and together these passes encode the bits of all pixels in the bitplane. That is why these passes are called *fractional bitplane coding*. The specific order above is chosen to code first those bits that are likely to give the greatest distortion reductions. The first pass, the SPP, visits only (previously) insignificant pixels with a "preferred" neighborhood, where at least one of its eight nearest neighbors is (previously) significant. An 8-pixel neighborhood of the current pixel y is depicted in Fig. 5.13. If a '1' is encountered, changing the state from insignificant to

significant, then its associated sign is encoded. The SPP pass is skipped for the first ($n_{\max}$) bitplane, because nothing is yet significant, meaning that there is no preferred neighborhood.

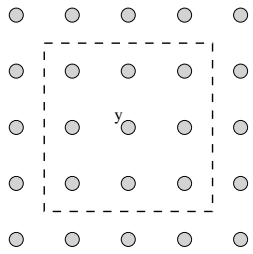

Figure 5.13: Eight pixel neighborhood of pixel y for context formation.

The second pass, the MRP, is the same as the refinement pass for the set partitioning coders. It visits only (previously) significant pixels (those with their first '1' in a higher bitplane) and codes the associated bits in the current bitplane. This pass is also skipped for the $n_{\max}$ bitplane for the same reason.

The third pass (CUP) visits the locations not yet visited in the SPP and MRP. Clearly, these locations store insignificant pixels without preferred neighborhoods. Therefore, this pass is the only one for the $n_{\max}$ bitplane.

We explain next how we encode the bits encountered in these three passes. The short explanation is that each pass has associated with it sets of contexts that are functions of significance patterns for the 8-pixel neighborhood of the bit to be encoded. Probability estimates of the bit values (0 or 1) given each context are accumulated as encoding proceeds and the bit is arithmetically encoded using these probability estimates. Of course, the devil is in the details, on which we shall elaborate.

Significance Coding - Normal Mode We now describe the normal mode of significance coding that is used exclusively in the SPP and partially in the CUP passes. As mentioned, the coding method is context-based, adaptive arithmetic coding of the binary symbols within a bitplane. The context template is the 8-pixel neighborhood of Fig. 5.13, where the number of possible (binary) significance patterns is $2^8 = 256$. So-called context quantization or reduction is necessary for reasons of lowering computational complexity and collection of sufficient statistical data (to avoid so-called *context dilution*) for each pattern. A reduction to nine contexts is achieved by counting numbers of significant pixels in the horizontal, vertical, and diagonal neighbors. Table 5.4 lists the nine contexts for code-blocks in LL and LH subbands. The context label descends in value from strongest to weakest horizontal neighbor significance. The HL subbands show strong dependence in the vertical direction, so their contexts, shown in Table 5.5, are numbered with the reversal of the Sum H and Sum V columns in Table 5.4.

The HH subbands show diagonal dependencies, so the contexts for code-blocks in these subbands are numbered by descending diagonal significance, accordingly in Table 5.6.

Table 5.4: Contexts for Code-Blocks in LL and LH subbands

Context Label	Sum H	Sum V	Sum D
8	2		
7	1	≥ 1	
6	1	0	≥ 1
5	1	0	0
4	0	2	
3	0	1	
2	0	0	≥ 2
1	0	0	1
0	0	0	0

Sum H—sum of significance states (0 or 1) of two horizontal neighbors
Sum V—sum of significance states (0 or 1) of two vertical neighbors
Sum D—sum of significance states (0 or 1) of four diagonal neighbors

Sign Coding In the SPP or CUP of a bitplane, whenever a '1' is encountered, changing the state from insignificant (0) to significant (1), the pixel value's sign is encoded also. The context template for sign coding consists just of the two horizontal and two vertical neighbors in Fig. 5.13. An intermediate value characterizes the significance and sign pattern either of the horizontal or vertical neighbors. This intermediate value, denoted as $\bar{\chi}$, takes values as follows:

$\bar{\chi} = 1$ when one neighbor is insignificant and the other significant and positive, or both neighbors significant and positive;

$\bar{\chi} = -1$ when one neighbor is insignificant and the other significant and negative, or both neighbors significant and negative;

$\bar{\chi} = 0$ when both neighbors are significant and opposite in sign, or both neighbors insignificant.

Since the horizontal and vertical neighbor pairs have three intermediate values each, there are $3^2 = 9$ possible patterns or context values. We expect that the conditional distribution of the sign to be coded given a particular pattern to be identical to the negative of that sign given the sign complementary pattern. Therefore, we group a pattern and its sign complementary pattern into a single context, thereby reducing to 5 distinct sign contexts. A flipping factor χ^{flip} =1 or -1 is used to change the sign of the bit to be coded according to the pattern or its complement, respectively. The contexts, their labels, and flipping factors are exhibited in Table 5.7. Denoting the sign to be coded as χ, the coded binary symbol κ_{sign} for the given context is

$$\begin{aligned} \kappa_{sign} &= 0 \quad \text{if} \quad \chi \cdot \chi^{flip} = 1 \\ \kappa_{sign} &= 1 \quad \text{if} \quad \chi \cdot \chi^{flip} = -1 \end{aligned} \tag{5.4}$$

Magnitude Refinement Coding Only previously significant pixels are visited in the MRP (Magnitude Refinement Pass) through a given bitplane. The associated bits refine these pixel values. The coded bit equals the actual bit of the pixel in the bitplane. We define three contexts in the context template for magnitude refinement coding. The value of each context is determined by whether or not the bit to be coded is the first refinement bit and the sum of the significance states of the eight pixels in the template. We distinguish three contexts as follows:

— Not the first refinement bit

— First refinement bit and sum of 8-pixel significance states is 0

— First refinement bit and sum of 8-pixel significance states is non-zero

We summarize the context definitions and labels in Table 5.8.

When a refinement bit is 0, the value of the associated coefficient is in the lower half of the quantization interval. When it is 1, the value is in the upper half. The typical probability density function of a coefficient is peaked and steep near the origin for small magnitudes and much flatter and shallower for high magnitudes. Therefore, when the coefficient has small magnitude, the probability is higher for the lower part of the interval. When the coefficient has large magnitude, the probability is close to 1/2 for both the lower and upper half. That is the reason for distinguishing between the first refinement bit, when the magnitude is larger, and the subsequent refinement bits of a pixel when the magnitude is smaller. Because of statistical dependence among immediate neighbors, we distinguish whether or not there is at least one significant neighbor when the first refinement bit of a pixel is being coded.

Table 5.5: Contexts for Code-Blocks in HL subbands

Context Label	Sum H	Sum V	Sum D
8		2	
7	≥ 1	1	
6	0	1	≥ 1
5	0	1	0
4	2	0	
3	1	0	
2	0	0	≥ 2
1	0	0	1
0	0	0	0

Sum H—sum of significance states (0 or 1) of two horizontal neighbors
Sum V—sum of significance states (0 or 1) of two vertical neighbors
Sum D—sum of significance states (0 or 1) of four diagonal neighbors

Table 5.6: Contexts for Code-Blocks in HH subbands

Context Label	Sum H + Sum V	Sum D
8		≥ 3
7	≥ 1	2
6	0	2
5	≥ 2	1
4	1	1
3	0	1
2	≥ 2	0
1	1	0
0	0	0

Sum H—sum of significance states (0 or 1) of two horizontal neighbors
Sum V—sum of significance states (0 or 1) of two vertical neighbors
Sum D—sum of significance states (0 or 1) of four diagonal neighbors

Run Mode Coding The set partitioning coders employ a sorting pass that locates significant pixels and insignificant sets of pixels for a given threshold or bitplane. The insignificant sets are encoded with a single 0 symbol. The EBCOT method in JPEG2000 has no such sorting pass and must visit and encode the bit of every pixel in the bitplane. The pixels visited in the Clean Up Pass (CUP) are all previously insignificant with insignificant neighborhoods. These pixels are likely to remain insignificant, especially for the higher bitplanes, so we employ a run-length coding mode in addition to the normal mode. This run mode is entered when four consecutive pixels are insignificant with insignificant neighborhoods. Therefore, the initial coding mode for the CUP is the run mode. However, when we encounter pixels with 1's in the scan of the bitplane, they become significant and make the neighborhoods of succeeding pixels 'preferred,' thereby triggering normal mode coding.

The preceding coding modes were not dependent on the scan pattern through the code-block. However, in order to describe the details of run mode coding, we need to specify this scan pattern. The code-block is divided into horizontal stripes of four rows per stripe. Within each stripe, the scan proceeds down the four-pixel columns from the leftmost to rightmost column. After the last pixel of the rightmost column is visited, the scan moves to the leftmost column of the next stripe below.[3] This scan pattern is illustrated in Fig. 5.14.

The run mode is entered when all three of the following conditions are met.

1. Four consecutive pixels in the scan shown in Fig. 5.14 must currently be insignificant.
2. All four of these pixels must have insignificant neighborhoods.

[3]We are assuming that code-block dimensions are multiples of four. For non-square images, some code-blocks may not have such dimensions, leaving stripes with fewer than four rows. Run mode is not invoked for stripes with fewer than four rows.

Table 5.7: Contexts for Code-Blocks for Sign Coding

Context Label	$\bar{\chi}_H$	$\bar{\chi}_V$	χ^{flip}
14	1	1	1
13	1	0	1
12	1	-1	1
11	0	1	1
10	0	0	1
11	0	-1	-1
12	-1	1	-1
13	-1	0	-1
14	-1	-1	-1

$\bar{\chi}_H$—intermediate value for horizontal neighbors
$\bar{\chi}_V$—intermediate value for vertical neighbors
$\bar{\chi}_D$—intermediate value for diagonal neighbors

Table 5.8: Contexts for Magnitude Refinement Coding

Context Label	Pixel's first Refinement Bit?	Sum H + Sum V + Sum D
17	No	
16	Yes	> 0
15	Yes	0

Sum H—sum of significance states (0 or 1) of two horizontal neighbors
Sum V—sum of significance states (0 or 1) of two vertical neighbors
Sum D—sum of significance states (0 or 1) of four diagonal neighbors

3. The four consecutive pixels must be aligned with a column of a stripe.

In run mode, a binary "run interruption" symbol is coded to indicate whether (0) or not (1) all four pixels remain insignificant in the current bitplane. This symbol is coded using its natural context of four consecutive pixels remaining insignificant or not. JPEG2000 labels this context of four consecutive pixels as number 9.[4] A pixel becomes significant when a '1' is read in the current bitplane. We count the number r of '0's until a '1' is reached and encode this run number followed by the sign of the pixel associated with the '1' terminating the run. The run-length r ranges from 0 to 3, because it has already been signified that there is at least one '1'. The run-length r is close to uniform in distribution, so its values are encoded with 2 bits, the most significant one sent first to

[4]Some textbooks count this context as two states, significant and insignificant four consecutive pixels.

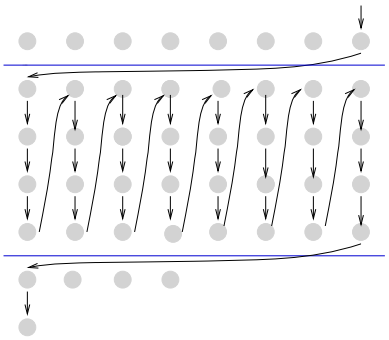

Figure 5.14: Column-wise scan pattern of stripes within a code-block.

the codestream.[5] The significances of the remaining samples of the four, if any, are encoded using the normal mode. Coding continues in normal mode until conditions to enter run mode are again encountered.

Example 5.3 Run Mode Coding Example.
Suppose we are in run mode and the run interruption symbol '1' signifies that one or more of the four pixels in a column has become significant. Let the bitplane bits of the four pixels be 0010. Since $r = 2, 1$ and then 0 are sent to the codestream. Furthermore, the sign associated with the 1 following 00 is coded in the usual way and sent to the codestream. The next bit of 0 is then coded using the normal mode.

The operation of JPEG2000's arithmetic coder, which is the MQ-Coder, has purposely not been described here. We refer the reader to JPEG2000 textbooks [50][1] for such material. Clearly, JPEG2000 coding is heavily dependent on arithmetic coding, much more so than set partitioning coders, and for that reason is considered to be computationally complex. The payoff is excellent compression efficiency. Due to its bitplane coding from most to least significant, the resulting codestream of every code-block is bit-embedded. Also, the coding of subblocks of subbands independently allows resolution scalability and random access to spatial regions directly within the codestream. Despite its excellent performance and range of scalability options, some applications, especially those involving hardware implementation constrained in size and power consumption, cannot tolerate the high complexity. This complexity is one of the reasons that restrains wider adoption of JPEG2000.

Scalable Lossy Coding in JPEG2000 The JPEG2000 method of coding of subblocks produces bit-embedded sub-codestreams. Each sub-codestream has a distortion versus rate characteristic, because it is associated with a different part of the wavelet transform. In order to achieve the best lossy reconstruction at a given rate, each sub-codestream must be cut to its optimal size, so that

[5] In the standard, the MQ-coder is set to a non-adaptive mode with a fixed uniform probability distribution to code the run lengths.

the total of the sizes equals the size corresponding to the given rate as closely as possible. Sets of sub-codestream sizes or cutting points corresponding to a number of rates can be calculated during encoding. The codestream is then considered to be built up in *layers*. A method to determine the optimal cutting points for a number of layers will be described later in Section 7. Briefly, according to this method, a series of increasing slope parameters $\lambda_1, \lambda_2, \ldots, \lambda_J$ are specified and every sub-codestream is cut, so that the magnitude of its distortion-rate slope matches each parameter in turn until the desired total number of bits among the codestreams is reached.

CHAPTER 6

Tree-based Wavelet Transform Coding Systems

We have described several block-based wavelet transform coding systems, so now we turn to two algorithms which are tree-based, Set Partitioning In Hierarchical Trees (SPIHT) [41] and EZW [45]. The SPIHT and EZW algorithms operate on trees in the transform domain, because they require the energy compaction properties of transforms to be efficient. These algorithms will be described in their breadth-first search or bit-embedded coding modes and without entropy coding of their output bits. Here we shall explain how to realize the attributes of resolution scalability and random access in the case of SPIHT coding. EZW can be configured similar to SPIHT for random access, but is not amenable to generating a resolution scalable codestream.

6.1 SPIHT CODING

The SPIHT coding algorithm operates on a different partition of the wavelet transform into structures called spatial orientation trees (SOT's). The transform space is partitioned into trees descending from three of the four pixels in every 2×2 group plus the remaining upper left corner pixels in the lowest frequency subband, denoted by $\mathcal{H}$. A partition generated from a 2×2 group is called a tree-block and is depicted in Fig. 6.1. The members of this partition or tree-block are three spatial orientation trees (SOT's), branching vertically, diagonally, and horizontally, and the corner pixel. The constituent sets of an SOT are all descendants from a root node, called a $\mathcal{D}$ set, the set of four offspring from a node, called an $\mathcal{O}$ set, and the grand-descendant set, denoted by $\mathcal{L}$, consisting of all descendents of members of the offspring set, $\mathcal{O}$. These sets are illustrated in Fig. 6.1 for the particular case where the root node is in the lowest frequency subband.

The algorithm requires two passes through the SOT's of the wavelet transform. These passes, called the *sorting* and *refinement* passes, will be described in literal detail in the next paragraphs. The original SPIHT paper [41] presents a pseudo-code description.

The Sorting Pass First, the largest integer power of 2 threshold, $\tau_{max} = 2^{n_{\max}}$, that does not exceed the largest magnitude coefficient in the entire transform is found. The algorithm maintains three control lists: the LIP (list of insignificant pixels), LIS (list of insignificant sets), and LSP (list of significant pixels). The LIP is initialized with all coordinates in $\mathcal{H}$, the LIS with all coordinates in $\mathcal{H}$ with descendants, and LSP as empty. The initial entries in the LIS are marked as Type A to indicate that the set includes all descendants of their roots. The algorithm then proceeds to test members of

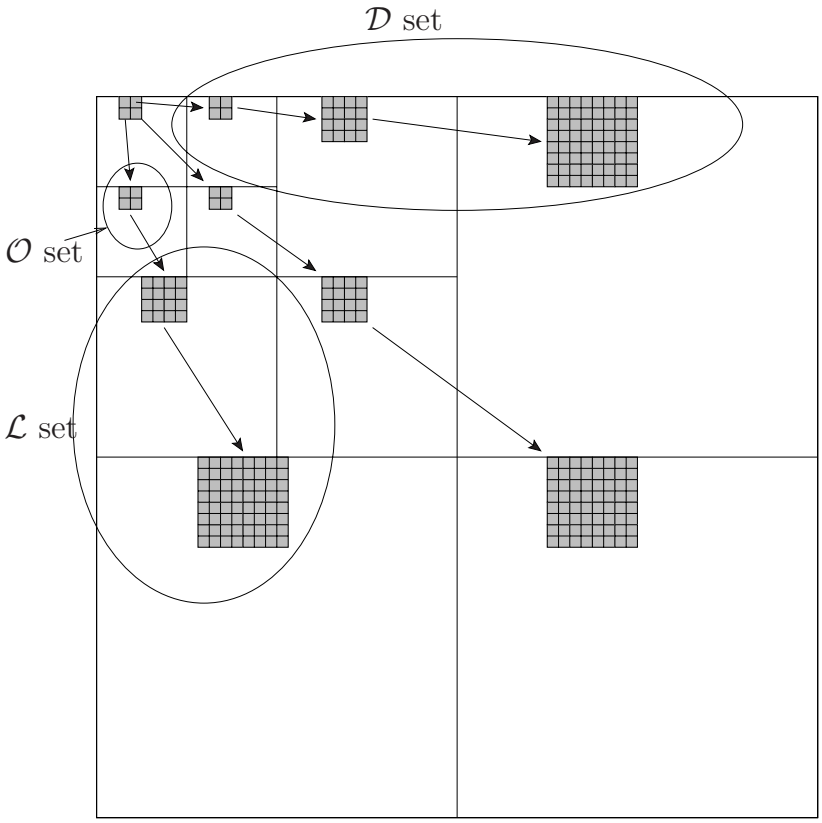

Figure 6.1: Illustration of set types in a tree-block with its three constituent SOT's descending from a 2×2 group in the lowest frequency subband in a three level, 2D wavelet transform. A full descendant $\mathcal{D}$ set, an offspring $\mathcal{O}$ set, and a grand-descendant $\mathcal{L}$ set are encircled in the diagram. All pixels (coefficients) in grey, including the upper left corner pixel in the 2×2 group, belong to the tree-block.

the LIP for significance at the current threshold. A binary symbol of 0 for "insignificant" or 1 for "significant" is sent to the codestream for each test. When a pixel is significant, its sign bit is also sent to the codestream and its coordinates are moved to the LSP. Then the entries in the LIS are visited to ascertain whether or not their associated trees are significant. A tree is said to be significant if any member pixel is significant. Again, the binary result of the significance test is sent to the codestream. If insignificant (0), then the algorithm moves to the next LIS member. If significant (1), the tree is split into its child nodes (pixels) and the set of all descendants of these children, the so-called *grand-descendant set*. The children are then individually tested for significance. As before, if a child node is significant, a 1 and its sign are written to the codestream and its coordinates are moved and appended to the end of the LSP. If insignificant, a 0 is written and its coordinates are moved and

appended to the end of the LIP. The grand-descendant set is placed on the LIS and is designated by the coordinates of its tree root and a Type B tag to indicate that it is a grand-descendant set. That distinguishes it from LIS sets of Type A. The grand-descendant set (if non-empty) is tested for significance, and as usual, the binary result is written immediately to the codestream. If significant, the set is divided into the four sub-trees emanating from the children nodes. The coordinates of the roots of these sub-trees, which are the children nodes, are appended to the LIS and tagged as Type A. They will be tested at the current threshold after the starting LIS entries of this sorting pass. The grandparent node of these sub-trees is now removed from the LIS. If the grand-descendant set is insignificant, it stays on the LIS as a Type B set.

During the course of the algorithm, new entries are being added to the end of the LIS, through the splitting of significant grand-descendant sets into four sub-trees. So these entries will be tested for significance at the same threshold under which they were added. Since at least one of these sub-tree sets is significant, that will engender a sequence of analogous splitting of successively smaller sets at later generations of the SOT until the significant single elements are located. Later generations of the SOT are equivalent to higher resolution levels. Therefore, we note that pixels and sets belonging to different resolutions are intermingled in the lists. We also note that the LIS significance search is breadth first, because LIS entries in the starting list of the pass are tested before those created at a later generation of the tree corresponding to the next higher level of resolution.

When all entries on the LIS have been processed, the sorting pass at the current threshold is complete. The threshold is then lowered by a factor of 2 to start the next sorting pass. The algorithm visits the LIP left from the last pass to test it against the new lower threshold and to code its pixels as described above. It then enters the top of the LIS remaining from the previous pass to process its entries for the new threshold.

The Refinement Pass The refinement pass is executed immediately after every sorting pass, except for the first one at threshold $2^{n_{\max}}$, before lowering the threshold. The refinement pass at a threshold 2^n consists of outputting the n-th bit of the magnitude of pixels in the LSP that became significant at a higher threshold. This corresponds to collecting and writing the bits through bit plane n of all coefficients previously found to be significant.

The Decoder The conveyance of the outcomes of the significance tests allows the decoder to recreate the actions of the encoder. The decoder initializes the same lists and populates them according to the significance test bits received from the codestream. Then the sign and refinement bits for that threshold can be read. Once the sorting and refinement passes have been completed for a given threshold (2^n), the source image can be reconstructed at the precision corresponding to the bit plane of that threshold (n)through the following operations.

- For each LSP entry,

    1. decode sign and refinement bits of LSP coordinate to a quantizer index.

2. de-quantize to a reconstruction value of the associated coefficient. The usual de-quantization parameter is $\xi = 3/8$, which reconstructs at a point that is 3/8 of the width of the decision interval.

- Set all coefficients not represented in the LSP to 0.
- Apply the inverse wavelet transform to the reconstructed coefficient array to produce the reconstruction of the source image.

One may choose to partially reconstruct the image and update the reconstruction values after every threshold pass.

6.1.1 EXAMPLE OF SPIHT CODING

We describe now in detail an example of SPIHT coding of the 8×8, three-level wavelet transform in Figure 5.8. We assume the integer values were either natural or quantized values of wavelet coefficients. We applied the SPIHT algorithm to the this set of data, for one pass. The results are shown in Table 6.1, indicating the sets or pixels tested, the emitted code bits and the actions taken to update the control lists. To save space only, we have omitted showing the coordinates on the control lists. The notation is defined in the previous description of the algorithm. For a quick reference, here are some of the important definitions.

LIS List of insignificant sets: contains sets of wavelet coefficients which are defined by tree structures, and which had been found to have magnitude smaller than a threshold (are insignificant). The sets exclude the coefficient corresponding to the tree or all subtree roots, and have at least four elements.

LIP List of insignificant pixels: contains individual coefficients that have magnitude smaller than the threshold.

LSP List of significant pixels: pixels found to have magnitude larger that the threshold (are significant).

$\mathcal{O}(i, j)$ in the tree structures, the set of offspring (direct descendants) of a tree node defined by pixel location (i, j).

$\mathcal{D}(i, j)$ set of descendants of node defined by pixel location (i, j).

$\mathcal{L}(i, j)$ set defined by $\mathcal{L}(i, j) = \mathcal{D}(i, j) - \mathcal{O}(i, j)$.

The following refer to the respective numbered entries in Table 6.1:

(1) These are the initial SPIHT settings. The initial threshold is set to 32. The notation (i,j)A or (i,j)B, indicates that an LIS entry is of type 'A' or 'B', respectively. Note the duplication of coordinates in the lists, as the sets in the LIS are trees without the roots. The coefficient (0,0) is not considered a root.

(2) SPIHT begins coding the significance of the individual pixels in the LIP. When a coefficient is found to be significant it is moved to the LSP, and its sign is also coded. We used the notation 1+ and 1– to indicate when a bit 1 is immediately followed by a sign bit.

(3) After testing pixels it begins to test sets, following the entries in the LIS (active entry indicated by bold letters). In this example $\mathcal{D}(0, 1)$ is the set of 20 coefficients {(0,2), (0,3), (1,2), (1,3), (0,4), (0,5), (0,6), (0,7), (1,4), (1,5), (1,6), (1,7), (2,4), (2,5), (2,6), (2,7), (3,4), (3,5), (3,6), (3,7)}. Because $\mathcal{D}(0, 1)$ is significant SPIHT next tests the significance of the four offspring {(0,2), (0,3), (1,2), (1,3)}.

(4) After all offspring are tested, (0,1) is moved to the end of the LIS, and its type changes from 'A' to 'B', meaning that the new LIS entry meaning changed from $\mathcal{D}(0, 1)$ to $\mathcal{L}(0, 1)$ (i.e., from set of all descendants to set of all descendants minus offspring).

(5) Same procedure as in comments (3) and (4) applies to set $\mathcal{D}(1, 0)$. Note that even though no offspring of (1,0) is significant, $\mathcal{D}(1, 0)$ is significant because $\mathcal{L}(1, 0)$ is significant.

(6) Since $\mathcal{D}(1, 1)$ is insignificant, no action need to be taken. The algorithm moves to the next element in the LIS.

(7) The next LIS element, (0,1), is of type 'B', and thus $\mathcal{L}(0, 1)$ is tested. Note that the coordinate (0,1) was moved from the beginning of the LIS in this pass. It is now tested again, but with another interpretation by the algorithm.

(8) Same as above, but $\mathcal{L}(1, 0)$ is significant, so the set is partitioned in $\mathcal{D}(2, 0)$, $\mathcal{D}(2, 1)$, $\mathcal{D}(3, 0)$, and $\mathcal{D}(3, 1)$, and the corresponding entries are added to the LIS. At the same time, the entry (1,0)B is removed from the LIS.

(9) The algorithm keeps evaluating the set entries as they are appended to the LIS.

(10) Each new entry is treated as in the previous cases. In this case, the offspring of (2,1) are tested.

(11) In this case, because $\mathcal{L}(2, 1) = \emptyset$ (no descendant other than offspring), the entry (2,1)A is removed from the LIS (instead of having its type changed to 'B').

(12) Finally, the last two entries of the LIS correspond to insignificant sets, and no action is taken. The sorting pass ends after the last entry of the LIS is tested.

(13) The final list entries in this sorting pass form the initial lists in the next sorting pass, when the threshold value is 16.

Without using any other form of entropy coding, the SPIHT algorithm used 29 bits in this first pass, exactly the same number of bits as SPECK in its first pass for the same data. The initial lists for the second pass of SPIHT at $n = 4$ are those at the end of the first pass in the last line (13) of Table 6.1.

Table 6.1: Example of image coding using the SPIHT method.

Comm.	Pixel or Set Tested	Output Bit	Action
(1)			LIS = {(0,1)A,(1,0)A,(1,1)A} LIP = {(0,0),(0,1),(1,0),(1,1)} LSP = ∅
(2)	(0, 0)	1+	(0,0) to LSP
	(0, 1)	1–	(0,1) to LSP
	(1, 0)	0	none
	(1, 1)	0	none
(3)	$\mathcal{D}(0, 1)$	1	test offspring
	(0, 2)	1+	(0,2) to LSP
	(0, 3)	0	(0,3) to LIP
	(1, 2)	0	(1,2) to LIP
	(1, 3)	0	(1,3) to LIP
(4)			LIS (0,1)A to (0,1)B
(5)	$\mathcal{D}(1, 0)$	1	test offspring
	(2, 0)	0	(2,0) to LIP
	(2, 1)	0	(2,1) to LIP
	(3, 0)	0	(3,0) to LIP
	(3, 1)	0	(3,1) to LIP
			LIS (1,0)A to (1,0)B
(6)	$\mathcal{D}(1, 1)$	0	none
(7)	$\mathcal{L}(0, 1)$	0	none
(8)	$\mathcal{L}(1, 0)$	1	add offspring sets to LIS
(9)	$\mathcal{D}(2, 0)$	0	none
(10)	$\mathcal{D}(2, 1)$	1	test offspring
	(4, 2)	0	(4,2) to LIP
	(4, 3)	1+	(4,3) to LSP
	(5, 2)	0	(5,2) to LIP
	(5, 3)	0	(5,3) to LIP
(11)			(2,1) removed from LIS
(12)	$\mathcal{D}(3, 0)$	0	none
	$\mathcal{D}(3, 1)$	0	none
			End of pass lists
(13)			LIS = {(1,1)A,(0,1)B,(2,0)A,(3,0)A,(3,1)A} LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0), (2,1),(3,0),(3,1),(4,2),(5,2),(5,3)} LSP = {(0,0),(0,1),(0,2),(4,3)}

There is no refinement pass after the first pass, because there are no higher thresholds. The significant pixels on the final LSP are known to be in the magnitude interval [32,63]. A midpoint ($\xi = 1/2$) reconstruction would put their magnitudes as 48. The 3/8 point reconstruction would put their magnitudes as 44. Adding their signs gives the following reconstructed values.

coordinate	3/8 point	1/2 point
(0,0)	44	48
(0,1)	-44	-48
(0,2)	44	48
(4,3)	44	48

All other coordinates have value 0.

6.2 RESOLUTION SCALABLE SPIHT

The original SPIHT algorithm just described above is scalable in quality at the bit level. Each additional bit written to the bitstream reduces the distortion no more than any of the preceding bits. However, it cannot be simultaneously scalable in resolution. Scalable in resolution means that all bits of coefficients from a lower resolution are written to the bitstream before those from a higher resolution. We see in the original quality scalable SPIHT that resolution boundaries are not distinguished, so that bits from higher bitplanes in high resolution subbands might be written to the codestream before lower bitplane bits belonging to lower resolutions. We therefore have to change the priority of the coding path to encode all bitplanes of lower resolution subbands before higher resolution ones.

Referring to Fig. 5.3, the correspondence of subbands to resolution level r is described in Table 6.2. The missing LL_r subbands in this table are the union of all the subbands at resolutions lower than r. To reconstruct an image at a given resolution $r > 0$, all subbands at the current and lower resolution levels are needed. For example, we need subbands HL_2, LH_2, HH_2, HL_3, LH_3, HH_3, LL_3 in order to reconstruct the image at resolution $r = 2$.

Table 6.2: Correspondences of Resolution Levels to Subbands

Resolution Level r	Subbands
0	LL (LL_3)
1	HL_3, LH_3, HH_3
2	HL_2, LH_2, HH_2
3	HL_1, LH_1, HH_1

The way to distinguish resolution levels and avoid their intermingling on the lists is to maintain separate LIP, LIS, and LSP lists for each resolution. In doing so, one must indicate LIP and LSP entries with the threshold at which they were put on the list. In this way, we can maintain these list entries in order of decreasing threshold. These threshold indicators, just like Type A or B, do not have to be sent, because the decoder can re-create them when it populates its lists. The details of maintaining separate lists for each resolution and placing indicators into the codestream are explained in a paper by Christophe and Pearlman [7].

6.3 BLOCK-ORIENTED CODING AND RANDOM ACCESS DECODING

We have introduced a partition of the wavelet transform into spatial orientation trees (SOT's). One can form an SOT or a contiguous group of SOT's into a block, called a *tree-block*, as depicted in Fig. 6.2. A tree-block represents the wavelet transform of an image region with size and location depending on its relative location and size within the lowest frequency subband. Therefore, coding tree-blocks is commensurate with coding rectangular regions of interest within the image. Any block coding method or even SPIHT can be chosen to code the tree-blocks. In order to achieve a designated rate for the entire image, one has to run a rate allocation algorithm to determine the optimal tree-block rates that meet the rate target. For example, we may use the iterative procedure in Section 3.2, in which tree-blocks replace subbands.

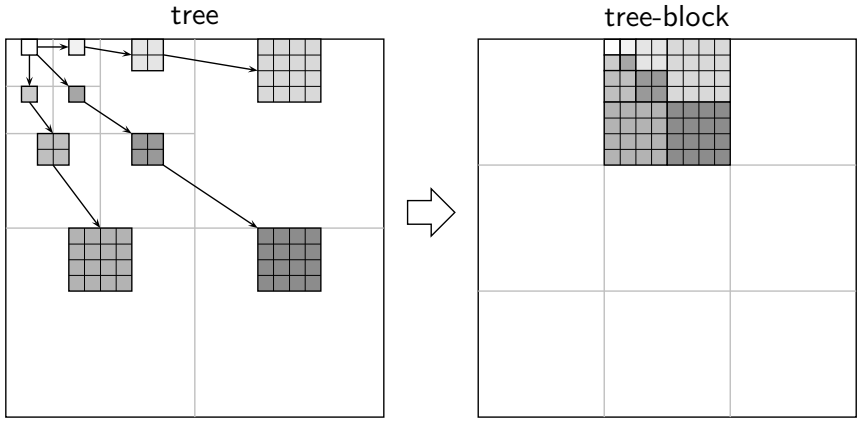

Figure 6.2: Rearranging a spatial orientation tree into a tree-block placed in the position of the image region it represents.

Coding tree-blocks, often called *block-oriented coding*, provides the capability of *random access decoding*. Random access decoding means that a designated portion of the image can be reconstructed by extracting and decoding only an associated segment of the codestream. The segment consists of the code of a group of tree-blocks that are associated with the rectangular region of the image that we wish to decode and reconstruct. Consider, as an example, the tree-block in Fig. 6.1, which corresponds to a region of the image geometrically similar in size and position of its 2×2 root group within the LL subband. The dimensions of the LL subband and image are 8×8 and 64×64, respectively. Therefore, the second 16×16 block at the top of the image is similar to the 2×2 group root in the LL subband. In reality, this 2×2 wavelet coefficient group does not reconstruct the corresponding 16×16 image block exactly, because surrounding coefficients within the filter span contribute to values of pixels near the borders on the image block. However, the correct image block is reproduced,

although not perfectly. The tree-blocks are mutually exclusive and their union forms the full wavelet transform.

Random access decoding can be realized therefore by coding all the tree-blocks separately. When we use SPIHT, we just initialize the lists in the usual way, but only with the coordinates of the 2×2 group in the LL subband that is the root of the tree-block to be coded. The steps of the SPIHT algorithm are exactly as before. When the algorithm finishes a tree-block, it moves to the next one until all the tree-blocks are coded.

Example 6.1 *Using block-oriented SPIHT to decode region of interest.* We demonstrate an example of using block-oriented SPIHT coding to reconstruct a region interest from a portion of the codestream. Again, as in Example 5.2, the Goldhill source image's wavelet transform is quantized with step size 1/0.31, and the bin indices are losslessly coded to a codestream size corresponding to 1.329 bpp. The coding algorithm is block-oriented, so a segment of the codestream containing the coded blocks corresponding to the desired region of interest is extracted and decoded. Figure 6.3 shows the reconstruction of the region of interest beside that of the fully coded image.

Figure 6.3: Reconstructions from same codestream of 512×512 Goldhill, quantizer step size = 1/0.31, coded to rate 1.329 bpp, and of 70×128 region at coordinates (343,239).

6.4 EMBEDDED ZEROTREE WAVELET (EZW) CODING

An embedded SOT coding method that is not strictly a set partioning coder is the Embedded Zerotree Wavelet (EZW) Coder [45].

This coder inspired the creation of SPIHT and shares many of its characteristics. It operates on SOT's of a wavelet transform like SPIHT, but locates only significant coefficients and trees, all of whose coefficients, including the root, are insignificant. When the thresholds are integer powers of 2, insignificant trees means that all the bits in the corresponding bit plane located at the coordinates of the tree nodes are 0's. Hence, arises the term *zerotree*. SPIHT has also been called a *zerotree* coder, but it locates zerotrees in a broader sense than the EZW zerotrees [6]. For our purposes here, it is better to view SPIHT as a set partitioning coder.

The structure of a full spatial orientation tree used in EZW is shown in the left frame of Fig. 6.2. The EZW algorithm determines certain attributes of these trees and their subtrees. Given a significance threshold, EZW visits a wavelet coefficient and makes the following determinations about its significance and the subtree rooted at that coefficient. Subtrees, whose coefficients (including the root) are all insignificant, are called *zerotrees* and its root is deemed a *zerotree root*. When the subtree root is insignificant and some of its descendants are significant, it is called an *isolated zero*. When the root of a subtree is significant, it is noted whether it is positive or negative. The descendants in its subtree are not characterized in this case. Therefore, when the algorithm comes to a certain coefficient having descendants in its wavelet transform subtree, it emits one of four symbols:

- ZTR - zerotree root
- IZ - isolated zero
- POS - significant and positive
- NEG - significant and negative

When a coordinate has no descendants (i.e., it is contained within a Level 1 subband or is a leaf node), then, if insignificant, a zero symbol (Z) is emitted. Note that raw encoding each of the four symbols for non-leaf nodes requires 2 bits, while encoding of the Z symbol for a leaf node sends a single '0' to the codestream. For coefficients associated with leaf nodes, the symbol alphabet changes to Z, POS, NEG, corresponding to the raw bits 0, 11, and 10, respectively. The encoder and decoder will know when the leaf node set has been reached, so can adjust its symbol alphabet accordingly.

Having characterized the symbols used in EZW, we can now describe this algorithm in more detail. Since the trees or subtrees in EZW include the root, we introduce the notation for a tree (or subtree) rooted at coordinates (i, j) as

$$\mathcal{T}(i, j) = (i, j) \bigcup \mathcal{D}(i, j), \tag{6.1}$$

the union of the root coordinate and the set of its descendants. Referring to the branching policy shown in Fig. 6.2, if (i, j) has descendants, we test both the root coefficient $c(i, j)$ and all the

coefficients in $\mathcal{T}(i, j)$ for significance. Using the significance test in (5.1), the outcomes are as follows.

$$\begin{aligned} \Gamma_n(\mathcal{T}(i, j)) &= 0, \text{ output ZTR} \\ \Gamma_n(\mathcal{T}(i, j)) &= 1 \text{ AND } \Gamma_n(i, j) = 0, \text{ output IZ} \\ \Gamma_n(i, j) &= 1 \text{ AND } c(i, j) > 0, \text{ output POS} \\ \Gamma_n(i, j) &= 1 \text{ AND } c(i, j) < 0, \text{ output NEG} \end{aligned} \tag{6.2}$$

If (i, j) has no descendants, the outcomes are:

$$\begin{aligned} \Gamma_n(i, j) &= 0, \text{ output Z} \\ \Gamma_n(i, j) &= 1 \text{ AND } c(i, j) > 0, \text{ output POS} \\ \Gamma_n(i, j) &= 1 \text{ AND } c(i, j) < 0, \text{ output NEG} \end{aligned} \tag{6.3}$$

We maintain two ordered control lists, the dominant list (DL) and the subordinate list (SL).[1] The dominant list keeps track of coordinates of subtree roots that are to be examined at the current and lower thresholds. The subordinate list (SL) stores coordinates of significant coefficients, so is identical to the LSP in the previous algorithms. The dominant list (DL) is initialized with the coordinates in the lowest frequency subband. The algorithm first tests entries in order from the DL. When the outcome is not ZTR and there are offspring, the coordinates of the offspring are added to the end of the list DL. When the coefficient for the entry is significant, its coordinates are added to the end of the list SL. These actions for populating the lists may be summarized as follows.

for (i, j) in DL,

- if (i, j) not ZTR AND not Z, add (k, l) in $\mathcal{O}(i, j)$ to end of DL.
- if $\Gamma_n(i, j) = 1$, add (i, j) to end of SL.

Once a coefficient tests significant, it should not be tested again. It is customary to keep it on the DL, but mark it with an internal flag to skip over it. Putting the previous procedures together leads us to the full description of the EZW algorithm. As before, we calculate $n_{\max}$, initialize DL with the coordinates of $\mathcal{H}$, and set SL to empty. The order of scanning of the subbands follows the zigzag shown in Fig. 5.3. Coordinates within a subband are visited in raster scan order from the top to the bottom row. We then visit the coordinates in the DL in order starting at $n = n_{\max}$. We test these coordinates and output the symbols according to (6.2) and re-populate the DL and SL by the rules above. We continue through the subbands in the prescribed scan order at the same threshold. When we reach DL coordinates belonging to the leaf node set in the Level 1 subbands, we test and output symbols by (6.3), but nothing more is added to the DL, since there are no offspring. The so-called *dominant pass* is complete, once all the DL coordinates have been visited at a given

[1]These lists keep track of the actions of the algorithm and point to future actions. Other means that do not require these lists can be employed to achieve the same actions. For example, see Taubman and Marcellin [50]. SPIHT can also be implemented without lists (*e.g.*, see [52]).

threshold n. Then the SL is entered to output the n-th bit of coefficients found significant at previous higher values of n. This pass through the SL, called in EZW the *subordinate pass*, which is identical to the *refinement pass* previously described for progressive bitplane coding, is omitted for coefficients found significant on the current pass, because the output symbols, POS and NEG, already signify that the n-th bit is 1. When the subordinate pass is complete, the threshold is lowered by a factor of 2 (n is decremented by 1) and the dominant pass at this new threshold is executed on the coordinates in the DL left from the previous pass. The coordinates belonging to coefficients that have become significant on previous passes are removed or marked or set to 0 to be skipped. This process continues either until completion of the $n = 0$ passes or the bit budget is exhausted.

6.4.1 AN EZW CODING EXAMPLE

We now present an example of EZW coding of the same 8×8 wavelet transform shown in Fig. 5.8. The notation of **F** following a coordinate on the dominant list means that an internal flag is set to indicate "significant" on that pass and its magnitude on the dominant list is set to 0 for subsequent passes. Recall that i signifies row index and j column index. We display only the actions and outputs for the first dominant pass at the top bitplane $n = 5$. Table 6.3 shows the results.

The DL and SL lists at the end of the $n = 5$ are the initial lists for the next pass at $n = 4$. The coordinates on the DL not flagged with **F** are tested in the order in which they appear on the list.

Assuming the EZW uses (at least initially) two bits to code symbols in the alphabet {POS, NEG, ZTR, IZ}, and one bit to code the symbol Z, the EZW algorithm used 26+7=33 bits in the first pass. Since the SPECK, SPIHT, and EZW methods are coding the same bit-plane defined by the threshold 32, both find the same set of significant coefficients, yielding images with the same mean squared error. However, SPIHT and SPECK used 29 bits, about 10% fewer bits to obtain the same results, because they coded different symbols. EZW uses arithmetic coding of its output symbols to reduce its bitrate. Even with arithmetic-coded symbols, it does not surpass the compression performance of SPIHT or SPECK without entropy coding.

The final lists of EZW and SPIHT may have some equal coordinates, but as shown in the examples, the interpretation and use of those coordinates by the two methods are quite different. Also, in the passes through the lower bitplanes they grow and change differently.

Table 6.3: EZW coding of wavelet transform in Fig. 5.8.

DL: dominant list, SL: subordinate list		
Initial Lists $n = 5$ pass		DL = {(0,0)} SL = ∅
Tree Root	Output Symbol	Action
(0,0)	POS	DL = {(0,0)**F**,(0,1),(1,0),(1,1)} SL = {(0,0)}
(0,1)	NEG	append (1,1),(0,2),(0,3),(1,2),(1,3),(0,1)**F** to DL append (0,0),(0,1) to SL
(1,0)	IZ	append (2,0),(2,1),(3,0),(3,1) to DL
(1,1)	ZTR	
(0,2)	POS	append (0,4),(0,5),(1,4),(1,5),(0,2)**F**} to DL append (0,2) to SL
(0,3)	ZTR	
(1,2)	ZTR	
(1,3)	ZTR	
(2,0)	ZTR	
(2,1)	IZ	append (4,2),(4,3),(5,2),(5,3) to DL
(3,0)	ZTR	
(3,1)	ZTR	
(0,4)	Z	
(0,5)	Z	
(1,4)	Z	
(1,5)	Z	
(4,2)	Z	
(4,3)	POS	append (4,3)**F**} to DL append (4,3) to SL
(5,2)	Z	
(5,3)	Z	
End $n = 5$ pass		DL = {(0,0)**F**,(1,0),(1,1),(0,3),(1,2),(1,3),(0,1)**F**, (2,0),(2,1),(3,0),(3,1),(0,4),(0,5),(1,4),(1,5), (0,2)**F**, (4,2),(5,2),(5,3),(4,3)**F**} SL = {(0,0),(0,1),(0,2),(4,3)}

CHAPTER 7

Rate Control for Embedded Block Coders

In order to exercise rate control when encoding the wavelet transform in separate blocks, such as we have in JPEG2000, SBHP, and block-oriented SPIHT, we need a method to determine the rate in each block that minimizes the distortion for a given average bit rate target. When the blocks are bit-embedded, their codestreams can simply be truncated to the sizes corresponding to the bit rates determined by such an optimization method. We assume that blocks have been coded to the bottom bitplane or threshold and are bit-embedded either naturally or by reorganization.

Given the target number of codestream bits B_T, the task is to assign rates (measured in bits) $b_1, b_2, \cdots, b_K$ to the blocks $\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_K$ respectively, so that the average distortion $D(B_T)$ is a minimum. Stated more formally, we seek the solution of rates $b_1, b_2, \cdots, b_K$ that minimizes

$$\begin{aligned} D(B_T) &= \sum_{k=1}^{K} d_{\mathcal{B}_k}(b_k) \quad \text{subject to} \\ B_T &= \sum_{k=1}^{K} b_k \end{aligned} \tag{7.1}$$

where $d_{\mathcal{B}_k}(b_k)$ denotes the distortion in block $\mathcal{B}_k$ for rate b_k. We assume that this function is convex and monotonically non-increasing. Since the distortion measure is squared error, the total distortion is the sum of those of the blocks. The minimization of the objective function,

$$J(\lambda) = \sum_{k=1}^{K} (d_{\mathcal{B}_k}(b_k) + \lambda b_k), \tag{7.2}$$

with Lagrange multiplier $\lambda > 0$, yields the solution that at the optimal rates, all the individual distortion-rate slopes must be equal to $-\lambda$. Since we do not have formulas for the block distortion-rate functions, we offer the following practical procedure to approximate this solution.

In the course of coding any block, say $\mathcal{B}_k$, the algorithm can calculate and record a set of distortion-rate points $(d^i_{\mathcal{B}_k}, b^i_k)$. The index i of these points increases from low to high rates. According to the convex assumption, the distortion decreases with increasing rate and index i. For a given λ, we start calculating distortion-rate slope magnitudes starting from high rates to lower rates.

These magnitudes increase with decreasing rate. We stop when we first reach

$$\frac{d^i_{\mathcal{B}_k} - d^{i+1}_{\mathcal{B}_k}}{b^{i+1}_k - b^i_k} \geq \lambda. \tag{7.3}$$

and declare the smaller of the rates b^i_k as the rate solution for block $\mathcal{B}_k$. We repeat for all $k = 1, 2, \ldots, K$. Algorithm 7 details this method.

Optimal bit allocation to independent blocks

Notation:

- i: index of points increasing with rate
- i-th rate in bits to block $\mathcal{B}_k$: b^i_k
- Distortion $d^i_{\mathcal{B}_k}$ for rate b^i_k

1. *Initialization.*
   (a) Obtain sequence of (rate, distortion) points $\{(b^i_k, d^i_{\mathcal{B}_k})\}$ for every block $\mathcal{B}_k$, $k = 1, 2, \ldots, K$.
   (b) Set a bit budget target B_T to allocate.
   (c) Choose $\lambda > 0$.
2. *Main:*
   For $k = 1, 2, \ldots, K$ (for all blocks)
   (a) set initial i large for large rate
   (b) for steadily decreasing i calculate
   $\frac{\delta d^i_k}{\delta b^i_k} = \frac{d^i_{\mathcal{B}_k} - d^{i+1}_{\mathcal{B}_k}}{b^{i+1}_k - b^i_k}$
   only until $\geq \lambda_j$ and stop.[1] Let $i = i^*_k$ be the stopping index at smaller of the two rates.
   (c) Denote rate solution for k-th block $b_k(\lambda) = b^{i^*_k}_k$
3. Total number of bits is $B(\lambda) = \sum_{k=1}^{K} b^{i_k *}_k$.

When we choose λ, the total rate, $B(\lambda) = \sum_k b_k$, is revealed only after the optimization procedure. We do know that λ is the magnitude of the common slope at the rate solution points. Therefore, we start with small λ corresponding to a high rate and steadily increase λ to test slopes

at smaller rates. In this way, we can store a table of correspondences of λ to $B(\lambda)$ and then know the value of λ needed for the target rate B_T.

One of the problems with this procedure is that a rather odd set of total rates usually results. More often, one would like to find the cutting points for a given set of rates, such as 0.25, 0.50, 0.75, 1.0, and 2.0 bpp perhaps. Therefore, one needs to determine the unique slope parameter λ that produces each one of these rates. The bisection method starts with an interval of slope magnitudes that contains the solution for the prescribed rate and determines in which half of this interval lies the solution. Then the solution point is narrowed to one of the halves of this half-interval and so forth until the interval containing the solution is sufficiently small. The bisection procedure for a single rate is detailed in Algorithm 7.

Bisection procedure for optimal bit allocation

1. Set target rate $B_T = \sum_{k=1}^{K} b_k(\lambda^*)$. To find λ^*. Set precision parameter $\epsilon > 0$.
2. Find slope parameters $\lambda_1 > 0$ and $\lambda_2 > \lambda_1$ such that $B(\lambda_2) < B_T < B(\lambda_1)$.
3. Let $\lambda_3 = (\lambda_2 + \lambda_1)/2$.
4. Set $i = 3$.
5. If $B(\lambda_i) < B_T$, $\lambda_{i+1} = (\lambda_i + \lambda_{j^*})/2$, $j^* = \arg\min_{j<i} B(\lambda_j) > B_T$.
   If $B(\lambda_i) > B_T$, $\lambda_{i+1} = (\lambda_i + \lambda_{j^*})/2$, $j^* = \arg\max_{j<i} B(\lambda_j) < B_T$.
   (Recursively determines half interval containing target rate.)
6. If $|B(\lambda_{i+1}) - B_T| < \epsilon$, stop. Let $\lambda_{i+1} = \lambda^*$.
   Else, set $i = i + 1$ and return to Step 5.

In calculating these rates for the various values of λ, we needed to calculate the distortion-rate slopes as in Step 2b in Algorithm 7. When you need to find the optimal parameters λ^* for a series of rates, it is particularly easy to set initial intervals once the first λ^* for the largest rate is found. Taking the rate targets in decreasing order, the distortion-rate slopes are increasing, so one can always use the last solution as the lower end point of the next slope interval.

CHAPTER 8

Conclusion

In this book we have attempted to present the principles and practice of image or two-dimensional wavelet compression systems. Besides compression efficiency as an objective, we have shown how to imbue such systems with features of resolution and quality scalability and random access to source regions within the codestream. The simultaneous achievement of more than one of these features is a unique aspect of the systems described here. We have tried to present the material at a level accessible to students and useful to professionals in the field.

The coding methods featured in this chapter are the basic ones. The purpose of this book was to explain the fundamental methods, so as to provide students and professionals with the tools to extend or enhance these methods according to their own purposes. Many enhancements and hybrids appear in the literature. We shall mention just a few of them to suggest further readings. For example, only the basic JPEG2000 (Part 1) coding algorithm was described along with its scalability features. There is a Part 2 standard with many extensions. Even in Part 1, built-in features of error resilience and region-of-interest encoding (max-shift method) have not been presented. Interested readers may consult the JPEG2000 standardization documents [24][26] and the book by Taubman and Marcellin [50].

There have been published many extensions and enhancements of the SPIHT and SPECK methods. SPIHT especially has inspired thousands of attempts to improve upon the original work [41]. One way was to dispense with the control lists, especially the LSP, because they grow long, requiring large memory, for higher rates and large images. Articles by Wheeler and Pearlman [52] and Shively et al. [47] report the use of fixed memory arrays to store the states of pixels and sets, thereby alleviating the memory problem. A problem of SPIHT that detracts from its efficiency, especially at low bit rates, is the identification of an insignificant pixel at a high threshold. For each such pixel one bit has to be encoded for every lower threshold. Therefore, there are attempts in the literature to cluster these insignificant pixels, so that a single '0' indicates their common insignificance. One such successful attempt is the block-tree approach of Moinuddin and Khan [33]. Another approach by Khan and Ghanbari [28] clustered insignificant offspring together by creating virtual root nodes on top of the real root nodes of the LL subband.

There have been some notable attempts to combine SPIHT or EZW and vector quantization. The idea is that the significance search can sort vectors according to their locations between shells in higher dimensional spaces. Among such efforts are those by da Silva et al. [13] and Cosman *et al.* [38][10], who developed vector EZW coders, and Mukherjee *et al.* [34], who developed a vector SPIHT coder.

Finally, the two-dimensional coding methods in this chapter extend readily to three-dimensional data, such as volume medical and hyperspectral images and video, and even to one-dimensional data, such as audio and electro-cardiogram signals. For three-dimensional data, the two-dimensional algorithms can operate on the axially transformed frames or slices of a three-dimensional transform. An example is the multi-component version of JPEG2000 in Part 2 of the Standard [25, 26]. Furthermore, certain coding methods treated in this chapter are independent of the dimensionality of the transform. They operate in the usual fashion once the tree structure has been created from the transform. For example, to extend SPIHT to three dimensions, one can take a three-dimensional wavelet transform of an image sequence, and create an SOT that branches to eight children from every parent node (e.g., see [29]). For a one-dimensional data source, the transform is one-dimensional and a parent node of an SOT branches to two children (e.g., see [31]). For the SPECK method, three-dimensional transform blocks split into eight subblocks (subcubes) to form an oct-tree and one-dimensional transform blocks split into two subblocks to form a bintree. The search for significance in SPECK proceeds among eight or two subblocks, depending upon whether the transform is three- or one-dimensional, respectively, (e.g., see [48]). Three-dimensional versions of JPEG2000 are specified in Part 10 of the Standard [43] and were developed in an earlier article by Xu et al.[56]. In these works, subcubes of the wavelet transform are coded with a context-based, adaptive, arithmetic coder, where the context is three-dimensional.

APPENDIX A

Further Reading and Resources

Much of the material in this book has been extracted and adapted from the textbook by Pearlman and Said, entitled *Digital Signal Compression: Principles and Practice*, published by Cambridge University Press [36]. Readers interested in a deeper and more tutorial treatment of the subjects in the present book should find the present book to be helpful in this regard. In order to keep the treatment of wavelet image compression basic and uncluttered, the present book only hints at extensions to sources of different dimensionality. More material on this subject may be found in Chapter 10 of the aforementioned book on "Color and multi-component image and video coding." Also helpful for understanding basic concepts is an earlier tutorial article by Xiong and Ramchandran [55] on wavelet image compression. An advanced and thorough treatment of quantization, entropy coding, wavelet filters, and JPEG2000 is contained in the book by Taubman and Marcellin [50]. Other notable textbooks that teach image compression, including wavelet image compression, are

Introduction to DataCompression by K. Sayood [42]

DataCompression: The Complete Reference by D. Salomon [14]

Image and Video Processing for Multimedia Engineering by Y. Q. Shi and H. Sun [46].

The last book above focuses mostly on the standards in image and video compression. The next two books focus on wavelets and subbands in signal processing and include some material on image compression.

A Wavelet Tour of Signal Processing by S. Mallatt [32]

Wavelets and Subband Coding by M. Vetterli and J. Kovačević [51]

Several books are recommended for exploring the foundations of source coding (compression).

Information Theory and Reliable Communication by R. G. Gallager [17]—the classic textbook on information theory, rate distortion theory, and source and channel coding

Elements of Information Theory by T. M. Cover and J. A. Thomas [11]—the modern reference on all aspects of information theory

Digital Coding of Waveforms by N. S. Jayant and P. Noll [27]—the first textbook dealing with compression of realistic sources

Vector Quantization and Signal Compression by A. Gersho and R. M. Gray [18]—a clear treatment of fundamentals of quantization and coding methods with encyclopedic coverage of vector quantization

The JPEG website, http://www.jpeg.org, contains information on description and explanation and links to books and articles about JPEG2000. JPEG2000 source code can be downloaded from the following sites:

http://www.kakadusoftware.com—the best and fastest software, and inexpensive

http://www.ece.uvic.ca/~frodo/jasper/—Part 1 only and free

The following hyperlinks will take you to image datasets, SPIHT and other coding software, animations, and tutorial material.

http://www.cipr.rpi.edu

http://www.cipr.ri.edu/~pearlman

http://www.cipr.rpi.edu/research/SPIHT

Bibliography

[1] Tinku Acharya and Ping-Sing Tsai. *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*. Wiley-Interscience, Hoboken, NJ, October 2004. DOI: 10.1002/0471653748 44

[2] Nasir Ahmed, T. Raj Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE Trans. Comput.*, 23(1):90–93, January 1974. DOI: 10.1109/T-C.1974.223784 1

[3] Jim Andrew. A simple and efficient hierarchical image coder. In *Proc. IEEE Int. Conf. Image Process.*, volume 3, pages 658–661, Santa Barbara, CA, October 1997. DOI: 10.1109/ICIP.1997.632207 26

[4] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 1(2):205–220, April 1992. DOI: 10.1109/83.136597 2

[5] C. Brislawn. The FBI fingerprint image compression standard. In P. N. Topiwala, editor, *Wavelet Image and Video Compression*, chapter 16, pages 271–288. Kluwer Academic Publishers, Boston/Dordrecht/London, 1998. 20

[6] Yushin Cho and William A. Pearlman. Quantifying the coding performance of zerotrees of wavelet coefficients: degree-k zerotree. *IEEE Trans. Signal Process.*, 55(6):2425–2431, June 2007. DOI: 10.1109/TSP.2007.893218 56

[7] Emmanuel Christophe and William A. Pearlman. Three-dimensional SPIHT coding of volume images with random access and resolution scalability. *EURASIP J. Image Video Process.*, 2008(Article ID 248905), 2008. 13 pages. DOI: 10.1155/2008/248905 53

[8] Christos Chrysafis, Amir Said, Alex Drukarev, Asad Islam, and William A. Pearlman. SBHP–a low complexity wavelet coder. In *Proc. IEEE Int. Conf. Acoustics Speech Signal Process.*, volume 4, pages 2035–2038, Istanbul, Turkey, June 2000. DOI: 10.1109/ICASSP.2000.859233 36

[9] A. Cohen, Ingrid Daubechies, and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Commun. Pure Appl. Math.*, 45(5):485–560, June 1992. DOI: 10.1002/cpa.3160450502 20

[10] Pamela C. Cosman, Sharon M. Perlmutter, and Keren O. Perlmutter. Tree-structured vector quantization with significance map for wavelet image coding. In *IEEE Data Compression Conf.*, pages 33–41, Snowbird, UT, March 1995. DOI: 10.1109/DCC.1995.515493 65

[11] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley and Sons, New York, NY, 1991, 2006. DOI: 10.1002/0471200611 67

[12] R. E. Crochiere, S. A. Weber, and J. L. Flanagan. Digital coding of speech in sub-bands. *Bell Systems Technical Journal*, pages 1069–1085, September 1976. DOI: 10.1109/ICASSP.1976.1170079 1

[13] Eduardo A. B. da Silva, Demetrios G. Sampson, and Mohammed Ghanbari. A successive approximation vector quantizer for wavelet transform image coding. *IEEE Trans. Image Process.*, 5(2):299–310, February 1996. DOI: 10.1109/83.480765 65

[14] D.Salomon. *Data Compression: The Complete Reference*. Springer, New York, 3 edition, 2004. 67

[15] Federal Bureau of Investigation, Washington, DC. *WSQ Gray-scale Fingerprint Image Compression Specification*, February 1993. 20

[16] J. Flanagan, M. Schroeder, B. Atal, R. Crochiere, N. Jayant, and J. Tribolet. Speech coding. *IEEE Transactions on Communications*, 27(4):710– 737, 1979. DOI: 10.1109/TCOM.1979.1094470 1

[17] Robert G. Gallager. *Information Theory and Reliable Communication*. John Wiley and Sons, New York, NY, 1968. 67

[18] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1992. DOI: 10.1007/978-1-4615-3626-0 68

[19] A. Habibi and P. Wintz. Image coding by linear transformation and block quantization. *IEEE Transactions on Communication Technology*, COM-19(1):50–62, February 1971. DOI: 10.1109/TCOM.1971.1090601 1

[20] Shih-Ta Hsiang and John W. Woods. Embedded image coding using zeroblocks of subband/waveletcoefficients and context modeling. In *Proc. IEEE Int. Symp. Circuits Syst.*, volume 3, pages 662–665, Geneva, Switzerland, May 2000. DOI: 10.1109/ISCAS.2000.856147 33

[21] J. J. Y. Huang and P. M. Schultheiss. Block quantization of correlated gaussian random variables. *IEEE Transactions on Communication Systems*, COM-11:289–296, September 1963. DOI: 10.1109/TCOM.1963.1088759 1

[22] Asad Islam and William A. Pearlman. Embedded and efficient low-complexity hierarchical image coder. In *Proc. SPIE Vol. 3653: Visual Commun. Image Process.*, pages 294–305, San Jose, CA, January 1999. DOI: 10.1117/12.334677 26, 29

[23] ISO/IEC 10918-1 and ITU-T Recommendation T.81. *Information technology–digital compression and coding of continuous-tone still images: Requirements and guidelines*, 1994. 21

[24] ISO/IEC Int. Standard 15444-1, Geneva, Switzerland. *Information Technology—JPEG2000 Image Coding System, Part 1: Core Coding System*, 2000. 65

[25] ISO/IEC Int. Standard 15444-10, Geneva, Switzerland. *Information Technology—JPEG2000 Extensions, Part 10: Core Coding System*, 2007. 66

[26] ISO/IEC Int. Standard 15444-2, Geneva, Switzerland. *Information Technology—JPEG2000 Extensions, Part 2: Core Coding System*, 2001. 65, 66

[27] Nikil S. Jayant and Peter Noll. *Digital Coding of Waveforms*. Prentice Hall, Englewood Cliffs, NJ, 1984. 67

[28] Ekram Khan and Mohammed Ghanbari. Very low bit rate video coding using virtual SPIHT. *IEE Electron. Lett.*, 37:40–42, January 2001. DOI: 10.1049/el:20010028 65

[29] B-J. Kim, Z. Xiong, and W. A. Pearlman. Low bit-rate scalable video coding with 3d set partitioning in hierarchical trees (3d spiht). *IEEE Trans. Circuits and Systems for Video Technology*, 10:1374–1387, December 2000. DOI: 10.1109/76.889025 66

[30] A. N. Kolmogorov. On the shannon theory of information transmission in the case of continuous signals. *IRE Transactions on Information Theory*, IT-2:102–108, December 1956. DOI: 10.1109/TIT.1956.1056823 1

[31] Z. Lu, D. Y. Kim, and W. A. Pearlman. Wavelet compression of ecg signals by the set partitioning in hierarchical trees (spiht) algorithm. *IEEE Transactions on Biomedical Engineering*, 47(7):849–856, July 2000. DOI: 10.1109/10.846678 66

[32] Stèphane Mallatt. *A Wavelet Tour of Signal Processing*. Academic Press, San Diego, CA, 1998. 67

[33] Athar A. Moinuddin and Ekram Khan. Wavelet based embedded image coding using unified zero-block-zero-tree approach. In *Proc. IEEE Int. Conf. Acoustics Speech Signal Process.*, volume 2, pages 453–456, Toulouse, France, May 2006. DOI: 10.1109/ICASSP.2006.1660377 65

[34] Debargha Mukherjee and Sanjit K. Mitra. Successive refinement lattice vector quantization. *IEEE Trans. Image Process.*, 11(12):1337–1348, December 2002. DOI: 10.1109/TIP.2002.806235 65

[35] William A. Pearlman, Asad Islam, Nithin Nagaraj, and Amir Said. Efficient, low-complexity image coding with a set-partitioning embedded block coder. *IEEE Trans. Circuits Syst. Video Technol.*, 14(11):1219–1235, November 2004. DOI: 10.1109/TCSVT.2004.835150 29

[36] William A. Pearlman and Amir Said. *Digital Signal Compression*. Cambridge University Press, Cambridge, England, 2011. DOI: 10.1017/CBO9780511984655 67

[37] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, 1993. 21

[38] Sharon M. Perlmutter, Keren O. Perlmutter, and Pamela C. Cosman. Vector quantization with zerotree significance map for wavelet image coding. In *Rec. Twenty-Ninth Asilomar Conf. Signals Syst. Comput.*, volume 2, pages 1419–1423, Pacific Grove, CA, October 1995. DOI: 10.1109/ACSSC.1995.540932 65

[39] W. K. Pratt, W. Chen, and L. R. Welch. Slant transform image coding. *IEEE Transactions on Communications*, COM-22(8):1075–1093, August 1974. DOI: 10.1109/TCOM.1974.1092335 1

[40] W. K. Pratt, J. Kane, and H. C. Andrews. Hadamard transform image coding. *Proceedings of the IEEE*, 57(1):58–68, January 1969. DOI: 10.1109/PROC.1969.6869 1

[41] Amir Said and William A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Trans. Circuits Syst. Video Technol.*, 6(3):243–250, June 1996. DOI: 10.1109/76.499834 47, 65

[42] Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann Publishers (Elsevier), San Francisco, CA, third edition, 2006. 67

[43] P. Schelkens, C. Brislawn, J. Barbarien, A. Munteanu, and J. Cornelis. Jpeg2000 - part 10:volumetric imaging. In *Proceedings of SPIE Annual Meeting, Applications of Digital Image Processing XXVI*, pages 296–305, San Diego CA, July 2003. DOI: 10.1117/12.512540 66

[44] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, IL, 1949. 1

[45] Jerome M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. Signal Process.*, 41(12):3445–3462, December 1993. DOI: 10.1109/78.258085 47, 56

[46] Y. Q. Shi and H. Sun. *Image and Video Compression for Multimedia Engineering*. CRC Press, Taylor & Francis Group, Boca Raton, 2008. DOI: 10.1201/9781420007268 67

[47] R. R. Shively, E. Ammicht, and P. D. Davis. Generalizing SPIHT: a family of efficient image compression algorithms. In *Proc. IEEE Int. Conf. Acoustics Speech Signal Process.*, volume 4, pages 2059–2062, Istanbul, Turkey, June 2000. DOI: 10.1109/ICASSP.2000.859239 65

[48] X. Tang and W. A. Pearlman. Three-dimensional wavelet-based compression of hyperspectral images. In G. Motta, F. Rizzo, and J. A. Storer, editors, *Hyperspectral Data Compression*, chapter 10, pages 273–308. Springer Science+Business Media, Inc., Boston/Dordrecht/London, 2006. DOI: 10.1007/0-387-28600-4 66

[49] David S. Taubman. High performance scalable image compression with EBCOT. *IEEE Trans. Image Process.*, 9(7):1158–1170, July 2000. DOI: 10.1109/83.847830 37

[50] David S. Taubman and Michael W. Marcellin. *JPEG2000: Image Compression Fundamentals, Standards, and Practice*. Kluwer Academic Publishers, Norwell, MA, 2002. DOI: 10.1007/978-1-4615-0799-4 44, 57, 65, 67

[51] Martin Vetterli and Jelena Kovačević. *Wavelets and Subband Coding*. Prentice Hall, Englewood Cliffs, NJ, 1995. 67

[52] Frederick W. Wheeler and William A. Pearlman. SPIHT image compression without lists. In *Proc. IEEE Int. Conf. Acoustics Speech Signal Process.*, volume 4, pages 2047–2050, Istanbul, Turkey, June 2000. DOI: 10.1109/ICASSP.2000.859236 57, 65

[53] P.A. Wintz. Transform picture coding. *Proceedings of the IEEE*, 60(7):809–820, July 1972. DOI: 10.1109/PROC.1972.8780 1

[54] John W. Woods and Sean D. O'Neil. Subband coding of images. *IEEE Trans. Acoust., Speech, Signal Process.*, 34(5):1278–1288, October 1986. DOI: 10.1109/TASSP.1986.1164962 1

[55] Zixiang Xiong and Kannan Ramchandran. Wavelet image compression. In Alan Bovik, editor, *Handbook of Image and Video Processing*, chapter 5.4. Elsevier, Amsterdam, The Netherlands, 2005. 67

[56] J. Xu, Z. Xiong, S. Li, and Y.-Q. Zhang. 3-d embedded subband coding with optimal truncation (3-d escot). *Applied and Computational Harmonic Analysis: Special Issue on Wavelet Applications in Engineering*, 10(5):290–315, May 2001. DOI: 10.1006/acha.2000.0345 66

Author's Biography

WILLIAM PEARLMAN

William Pearlman William A. Pearlman is currently Professor Emeritus in the Electrical, Computer and Systems Engineering at Rensselaer Polytechnic Institute (RPI). He obtained B.S. and M.S. degrees from MIT in 1963 and his Ph.D. degree from Stanford University in 1974. He joined RPI in 1979 and became Professor in 1988. Prior to joining RPI, he had been a faculty member at the University of Wisconsin-Madison for five years. In addition, he had held industrial positions at Lockheed Missiles and Space Company and GTE-Sylvania and has consulted for several organizations. He has authored or co-authored more than 200 publications in the fields of signal, image and video compression, information theory, communications theory, and digital signal processing. He is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) and of SPIE- The International Society for Optical Engineering. At Visual Communications and Image Processing (VCIP) 2010 in Huangshan, China, where he was keynote speaker, he was presented a Certificate of Appreciation in recognition of his leadership and contributions to this conference since its inception in 1986. He received the IEEE Circuits and Systems Society 1998 Video Technology Transactions Best Paper Award and the IEEE Signal Processing Society 1998 Best Paper Award in the Area of Multidimensional Signal and Image Processing. He is co-inventor of two celebrated image compression algorithms, Set Partitioning in Hierarchical Trees (SPIHT) and Set Partitioning Embedded Block (SPECK) coding. In 2001, he received the Robert Bosch Foundation Award in Appreciation of Outstanding Works in the Field of Picture Coding. He has also delivered several keynote or invited plenary lectures on topics in signal compression and is lead author of the recent textbook, *Digital Signal Compression: Principles and Practice,* with co-author Amir Said, published by Cambridge University Press. He is also founder, president, and chief scientific officer of PrimaComp, Inc., a company that specializes in developing innovative software for signal and image compression.

Index