TRELLIS SOURCE CODING
AND
MEMORY CONSTRAINED IMAGE CODING

By

Frederick W. Wheeler

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Electrical, Computer and Systems Engineering

Approved by the
Examining Committee:

William A. Pearlman, Thesis Adviser

John W. Woods, Member

Gary J. Saulnier, Member

David Isaacson, Member

Rensselaer Polytechnic Institute
Troy, New York

September 2000
(For Graduation December 2000)

to Sue

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

Though technically, a Ph.D. thesis is the work of an individual, this is never truly the case. Completing this work would have been impossible without a network of teachers, friends and family. I would like to acknowledge some of them here. Of course there have been many others along the way who have helped and guided me in large and small ways. As grateful as I am, it is not possible to list them here, though I thank them as well.

I have been quite fortunate to have Prof. William A. Pearlman as my thesis advisor. His selfless support and trust and his careful guidance have been both an asset and and inspiration.

I would like to thank Profs. John W. Woods, Gary J. Saulnier and David Isaacson for their help and thoughtful comments on this work and for serving on my doctoral committee.

I owe a great debt to Prof. Richard Vaz of Worcester Polytechnic Institute. Prof. Vaz was a caring advisor to me when I was an undergraduate student. He helped instilled in me a love of learning and research, and ultimately guided me to graduate school.

During my time at Rensselaer Polytechnic Institute I have been generously financially supported by the Defense Advanced Projects Research Agency, the Academic Computing Services department, the Center for Digital Video and Media Research and Sun Microsystems. I have also benefited greatly from the use of the facilities of the Center for Image Processing Research.

Finally, the love and support of my wife and family is a bedrock of stability without which my attempts to complete this work would have been fruitless.

ABSTRACT

With highly structured symmetric ECTCQ codebooks used at low rates, certain paths through the trellis are more likely followed than others. Analysis of the relationship between these dominant paths leads to a new method for ECTCQ codebook design. This new method finds an 8-state ECTCQ codebook branch assignment that outperforms the most widely used assignment.

A new class of image coding systems are developed: Spatial Block SPIHT (SB-SPIHT), Subband Block SPECK (SB-SPECK), and the general Hybrid Block Codec (HBC). Each is designed for coding large images in a constrained memory environment. Subband decomposition coefficients are partitioned into small hierarchical tree preserving spatial blocks and/or subband blocks which are independently coded using SPIHT or SPECK, a quad-tree partitioning codec. Approximate rate distortion characteristics are formed for each block and used for quality of service layered rate allocation. The bitstreams for each block are packetized through one of several priority schemes. The final bitstream can be fidelity progressive with a small rate overhead. SB-SPECK provides the ability to use integer wavelet transforms that scale energy differently in each band. SB-SPECK and HBC offer resolution scalability and their decomposition may have the dyadic or a wavelet packet structure. Each system can code color images.

A variant SPIHT image compression algorithm called No List SPIHT (NLS) is presented. NLS operates without linked lists and is suitable for a fast, simple hardware implementation. The image data is stored in a 1-D array following the Morton (recursive Z) scan for computational efficiency and algorithmic simplicity. Performance of NLS on standard test images is nearly the same as that of SPIHT.

The SB-SPECK system is enhanced to code video and 3-D medical images. A dyadic DWT is applied in the third dimension of a group of frames. Each transformed frame is partitioned into subband blocks which are coded by the core SPECK algorithm. The final bitstream is scalable in fidelity, resolution and frame rate. Results show a large gain from using a 3-D transform on medical imagery.

CHAPTER 1
Introduction

This thesis has two major parts: a trellis coding part, and an image coding part. Each part contains its own review and background material, and new research and contributions. This first chapter holds introductory material and an overview for both parts. Following the two major parts, the final chapter contains general conclusions and a summary of the contributions of both parts this work.

1.1 Trellis Source Coding

Trellis Coded Quantization (TCQ) is an effective and computationally efficient source coding technique for memoryless data. Trellis source coders can perform very close to the rate distortion bound without the complexity of vector quantization. They are practical and have proven useful in applications. This first part of this document starts with a review of the development of trellis source coding techniques from early trellis coders through to trellis coded quantization and Entropy-Constrained Trellis Coded Quantization (ECTCQ).

The earliest trellis based source coders are reviewed in Chapter 2. This chapter begins a theme of design complexity reduction. This theme is followed throughout the trellis coding part of this thesis. The development of trellis coding has followed a clear trend of reductions in the codebook size and increases in the structure of the reproduction sequences. As trellis source coders have evolved, their design complexity has decreased.

Early trellis codes, in general, had little structure to their reproduction sequences, except for that provided by the trellis itself. Often, each branch of the trellis was independently assigned a codevalue at random. An initial randomly generated trellis code could then be improved with an extension of the Linde-Buzo-Gray (LBG) algorithm [1] given by Stewart [2], or improved using a conjugate gradient optimization procedure described by Freeman [3].

Alphabet Constrained Rate Distortion Theory (ACRDT) by Finamore and

Pearlman [4] showed that trellis codes could be simplified somewhat by proving that nearly optimal coders can be formed using a limited set of reproduction values to populate the trellis. In practice, the set of reproduction values depends on the source distribution and the values are assigned to the branches of a non-stationary trellis randomly [4, 5].

Beginning with the introduction of trellis coded quantization, in Chapter 3, the techniques are described in more detail, though a general knowledge of source coding and quantization is assumed. Marcellin and Fischer introduced Trellis Coded Quantization (TCQ) [6, 7], motivated by the duality with Trellis Coded Modulation (TCM) by Ungerboeck [8, 9, 10]. Like TCM, TCQ is characterized by a codebook that is approximately twice the size that would be used for scalar coding. The values in this codebook are then partitioned into subsets and assigned to the branches of the trellis in a particular structured manner.

The trellis coding review continues with Entropy-Constrained TCQ (ECTCQ) in Chapter 4. Fischer developed an ECTCQ [11] that is a straightforward extension of scalar entropy-constrained coding [12] to TCQ. Fischer's technique, however, can not be used at rates below 1 bit per sample because the path through the trellis must still be separately transmitted. Marcellin extended ECTCQ [13] in such a way that the state transitions themselves are no longer necessarily equally used and are also subject to the entropy constraint. In this state-entropy ECTCQ, for each input sample, a single output symbol specifies both the trellis branch to follow and the reproduction value to use from that branch.

With the review of trellis coding completed, Chapter 5 presents some techniques and observations that are new contributions of this thesis. Still following the theme of design complexity reduction, a symmetry constraint which simplifies ECTCQ is described in this chapter. This symmetry reduces the implementation complexity without affecting performance. There are several advantages to this constraint, including reduced training time and codebook storage requirements.

The well known Ungerboeck trellis branch labelings, designed for Trellis Coded Modulation (TCM), lead to non-symmetric reproduction sequences, which is contrary to our intuition of what would work well for symmetric sources. We will find

that some new trellis branch labelings designed specifically for source coding with ECTCQ outperform the Ungerboeck labelings.

At low rates, the entropy constraint of ECTCQ causes half of the branches of the trellis to be used with greater probability than the other half. This is necessary for the rate to go below 1, but also occurs at rates slightly greater than 1. Instead of simply using the branch labelings found by Ungerboeck for Trellis Coded Modulation (TCM), Ungerboeck's rules for assigning codebooks to branches are reexamined. A larger set of branch labelings that meet Ungerboeck's basic TCM design guidelines are tested to determine their rate-distortion performance. The result is that, for ECTCQ, some new branch labelings are found that perform slightly better than the commonly used TCM labelings.

The final section of the trellis coding part examines some interesting behavior of ECTCQ. As the performance of ECTCQ on the Gaussian random source is improved by increasing the number of trellis states, the reproduction sequences move closer to an optimal reproduction sphere given by Sakrison [14].

1.2 Image Coding

In the second part of this thesis, trellis coding is left for the investigation of image compression using discrete wavelet transforms, scalar quantization, and entropy coding based on set partitioning. This change in focus is inspired by Shapiro's observation that when coding a wavelet transform a great deal of the rate must go to conveying which coefficients should be reproduced as non-zero [15]. His insight is that it is better to first seek the significance map gain, and then the granular gain.

Several aspects of image bitplane coding by set partitioning are investigated in this part. The dominant theme is the practical application of image coding by set partitioning in a memory constrained environment. We will focus on techniques designed to work efficiently with a cache, or two-level, memory structure. In general, cache memory is smaller, faster and more expensive, and external memory is larger, slower and less expensive. On general purpose computers, a cache controller attempts to predict what data will be needed next and moves it from the external memory to the cache. When this prediction is incorrect, time-consuming swapping

occurs. Computation is faster if the CPU can operate for longer periods accessing just the cache memory, with only occasional swapping. Several chapters will focus on adapting the well known Set Partitioning In Hierarchical Trees (SPIHT) image codec by Said and Pearlman [16] and the Set Partitioned Embedded Block Coder (SPECK) image codec by Islam and Pearlman [17, 18] to operate efficiently in a cached memory environment.

This part begins with Chapter 6 which presents background material on images and image coding. Image representation, the goals of image coding, the wavelet transform, quantization, and entropy coding are reviewed. Desirable bitstream properties such as fidelity and resolution scaling are defined and discussed in this chapter. Two specific well known entropy coding algorithms, SPIHT and SPECK, are then described in detail. In later chapters, new image coding systems are derived from these core algorithms.

Chapter 7 introduces a new image compression system designed to operate in a cached memory system. Many wavelet based image compression systems require repeated access to all of the data of the image and wavelet decomposition. SPIHT and SPECK are two such algorithms. As discussed above, this becomes a serious problem if the data for the entire image or wavelet decomposition does not fit in cache memory. Memory constraints become important when coding very large images or when designing an embedded coder where memory is a critical resource. The solution of this problem is an explicit requirement of the JPEG 2000 image compression standard request for proposals, and that requirement directly motivated this work.

The new algorithm, Spatial Block SPIHT (SB-SPIHT), organizes the wavelet decomposition into hierarchical tree preserving spatial blocks. The SPIHT algorithm is then applied to each spatial block independently. A fast approximate rate-distortion characterization has been developed for this work and is applied in each spatial block so that a nearly optimal layered rate allocation procedure can determine how much of the final bitstream to devote to each spatial block. This chapter includes detailed analysis of the memory requirements for the SB-SPIHT encoder, making use of the line-based transform engine described by Chrysafis and Ortega [19, 20].

The application of SPIHT, or the related Embedded Zerotree Wavelets (EZW) algorithm by Shapiro [15], to independent spatial blocks has been investigated by others for various purposes. Rogers and Cosman [21] have applied SPIHT and EZW to spatial blocks in a manner similar to SB-SPIHT. However, the purpose of their development is robustness against packet loss, which is reflected in their algorithm. Their system transmits the encoded data for as many spatial blocks as possible in each of a series of fixed size packets. SB-SPIHT is not concerned with packet transmission or loss, and instead produces a more flexible bitstream that is scalable in fidelity.

Creusere has also adapted EZW to work independently within spatial blocks for memory efficiency [22], in this case to enable parallel processing. In this system, the bitstreams from each spatial block can be interleaved to provide a fidelity embedded final bitstream, however this corresponds to sub-optimal uniform bit allocation. An important and useful feature of SB-SPIHT is its nearly optimal bit allocation procedure which decides how much rate is assigned to each spatial block. Other work by Creusere [23, 24] also applies EZW independently to spatial blocks, for region of interest coding and error resilience, however, rate allocation for overall fidelity maximization is not addressed.

The basic techniques used to apply SPIHT in a cached memory environment also work for the SPECK algorithm. In Chapter 8 the Subband Block SPECK (SB-SPECK) image compression system is described. This system organizes the wavelet decomposition into subband blocks instead of spatial blocks. Subband blocks are simply blocks within a subband and are naturally suited to SPECK. SPECK then is applied independently to each block. The fast rate-distortion characterization used for SB-SPIHT works just as well for SB-SPECK.

In part of his thesis [18], Islam also applied SPECK independently to subband blocks, however, in that system, rate was allocated to each block based on sub-bitplane milestones of the algorithm and not on actual rate-distortion performance within each block. We will see that rate-distortion based bit allocation offers a significant improvement.

The proposed JPEG 2000 image compression standard, which is based heavily

on the EBCOT image coder [25], has many characteristics in common with SB-SPECK. Both codecs divide discrete wavelet transform coefficients into subband blocks, encode each block independently, and use rate-distortion performance based bit allocation and a packetized final bitstream. The primary difference between the algorithms is in the core entropy codec and its computational complexity. Where JPEG 2000 will use EBCOT, SB-SPECK uses the comparatively faster SPECK algorithm to encode each block of coefficients.

SB-SPIHT and SB-SPECK have similar computational complexities and memory usage characteristics, however, there are a few differentiating features of the algorithms that affect which is the better technique in particular situations. Two important features of SB-SPECK that are not conveniently available with SB-SPIHT are resolution scalability and the ability to use integer wavelet transforms that scale energy differently in each band. SB-SPECK, however, has reduced rate-distortion performance when used on very small images or on very small tiles, as investigated in Chapter 11.

Chapter 9 describes another independent block image coding scheme, the Hybrid Block Codec (HBC), which is a generalization of SB-SPIHT and SB-SPECK. The HBC system partitions low frequency subbands into spatial blocks, as in SB-SPIHT, and high frequency subbands into subbands blocks, as in SB-SPECK. SPIHT is applied to the spatial blocks, and SPECK is applied to the subband blocks. This scheme was motivated by the fact that, in reported results, SPIHT tends to outperform SPECK, except for images with elevated high spatial frequency energy. The rate-distortion characterization used for SB-SPIHT and SB-SPECK is general enough that it can be applied to a combination of spatial and subband blocks. HBC inherits the resolution scalability feature of SB-SPECK. The HBC system is also extended to encode three component color images, making use of a YCbCr transform.

In Chapter 10 a new alternate implementation of the basic SPIHT image compression algorithm is presented. No List SPIHT (NLS) is an image codec that functions almost identically to SPIHT, but with a significant savings in memory due to a new technique for keeping track of which sets have been tested in each bitplane.

Instead of using growing linked lists to keep track of which sets have been tested, No List SPIHT uses a fixed array of memory and a novel efficient scanning technique. This feature makes NLS attractive for fast hardware based image compression. No List SPIHT operates with the same set-partitioning rules and significance tests as SPIHT, but with a slightly different set significance test order.

No List SPIHT uses a recursive zig-zag, or Morton scan [26], of the wavelet coefficients. It turns out that this scan, or indexing scheme, is very well suited to applications using hierarchical trees. Once the 2-D wavelet coefficient array is reordered in a 1-D array according to the Morton scan, certain operations performed very frequently in SPIHT, such as moving to the parent coefficient, moving to the child coefficient, or iterating over four children coefficients require fewer operations. Others, such as Seetharaman and Zavidovique [27], have used this type of scan for image segmentation problems, but not for zero-tree based compression.

Several others have also reported SPIHT implementations that do not use linked lists. Lin, Burgess, Ng and Bouzerdoum have developed listless zerotree codecs for images [28, 29] and video [30]. Their codecs, however, sacrifice some rate-distortion performance by performing a depth first search of the hierarchical trees. NLS uses a breadth first tree search that closely mimics the search order of the original algorithm. Shively, Ammicht and Davis have developed flexible listless SPIHT implementation [31], but rely on bitstream reordering after the SPIHT encoding to avoid the need for an efficient scan of the hierarchical trees.

Variable Quality SPIHT (vqSPIHT), a codec developed by Järvi, Lehtinen and Nevalainen [32] also uses a fixed array of memory in which the set partitioning state is kept. For some passes, the array is scanned, but for others a linked list, implemented within the memory of the fixed 2-D array, is effectively followed. Lehtinen has also described the Distortion Limited Wavelet Image Codec (DLWIC) [33], which is based on EZW. DLWIC uses a single depth first scan of the hierarchical trees for each bitplane, and thus can easily keep all set partitioning state information in a 2-D array, but sacrifices fidelity scalability.

It is sometimes desirable to divide an image into tiles before the transform and to encode each tile separately. Such a division offers simple region of interest

extraction at the decoder. If a spatial tree codec is applied to a small tile it has the advantage of full-size spatial trees being present. Subband block codecs such as SB-SPECK and EBCOT [25], as used in the proposed JPEG 2000 standard, suffer in this environment because low frequency subbands will be very small. Results demonstrating this behavior are presented in Chapter 11.

In Chapter 12 the SB-SPECK image codec is extended to 3-D to code video sequences and 3-D medical imagery. This extension of SB-SPECK is analogous to the extension of HBC to code color images. The 3-D SB-SPECK system begins with a Group Of Frames (GOF) and a dyadic DWT in the third dimension. As in the 2-D SB-SPECK system, each transformed frame is then divided into small 2-D blocks within each subband which are independently encoded with the core SPECK algorithm. The rate allocation procedure assigns space in the final bitstream to each subband block in the GOF, maximizing overall fidelity. As with the 2-D system, the final bitstream is scalable in resolution and fidelity. The 3-D system is also scalable in frame rate. Results show that 3-D SB-SPECK without motion compensation is effective for 3-D medical imagery with both lossy and lossless encoding.

Other applications of set partitioning based coding to 3-D data are reviewed in this chapter. The Cube-Splitting codec, by Schelkens, Barbarien and Cornelis [34] is a 3-D extension of quad-tree splitting, similar to SPECK, to 3-D blocks in a 3-D subband decomposition. This algorithm relies on a context based arithmetic coder for significance test results and for refinement bits, making it computationally complex relative to SB-SPECK.

Quad-tree splitting has also been applied to 3-D coding by Hsiang and Woods who have developed the 3-D Embedded Zero Block Coder (EZBC) [35, 36]. 3-D EZBC uses a motion compensated 3-D DWT and then encodes each transformed frame with EZBC, a 2-D algorithm. EZBC codes significance tests in a subband with an adaptive arithmetic coder in a context formed using information from other bands. Taking context information from other bands enhances the rate-distortion performance of this codec, but adds to its memory and cache requirements.

Finally, Chapter 13 offers general conclusions with a summary of the major contributions of this work and some suggestions for future extensions of this work.

PART I

Trellis Source Coding

CHAPTER 2
Trellis Source Coding Background

2.1 Source Coding

Figure 2.1 shows a block diagram of the major components of a digital communication system. The source generates a message which is to be communicated to the user. For our purposes, we assume this to be a sequence of real values, $\mathbf{x} = x_0, x_1, \ldots$. The message from the source is converted into bits by the source encoder. The channel encoder then takes these bits and converts them to a signal appropriate for the channel being used. The channel decoder receives this signal and recovers the sequence of bits that the source encoder produced. These bits are fed into the source decoder, which produces an approximation of the original message, $\hat{\mathbf{x}} = \hat{x}_0, \hat{x}_1, \ldots$ [37].

This work is concerned only with the source encoder and decoder, together called the source codec. We assume that the channel decoder reproduces the exact bit sequence generated by the source encoder. This amounts to the bits traveling along a virtual path shown in Fig. 2.1 as a dashed line.

If the source values, $x_0, x_1, \ldots$, have a continuous range, the source codec cannot perfectly reproduce the original source sequence. This would require an infinite number of bits for each source value. Instead, the encoder uses, on average, R bits per source sample and produces a reproduction sequence as close as possible to the original source sequence, according to some distance function $d(\mathbf{x}, \hat{\mathbf{x}})$. This

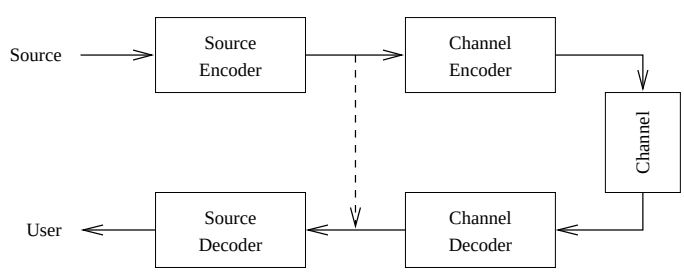

Figure 2.1: Block diagram of a digital communication system.

is the fundamental problem addressed by rate distortion theory [38]. Because of its nice analytic properties and because it corresponds to the power of the reproduction error, the sum-squared-difference distortion measure is usually used,

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \lim_{N \to \infty} \frac{1}{N} \sum_{n=0}^{N-1} (x_n - \hat{x}_n)^2. \tag{2.1}$$

The source, $\mathbf{x}$, is often modeled as a random sequence. The *rate distortion function* $R(D)$, for a given random source, defines the lowest possible rate needed to reproduce the source with expected distortion less than or equal to D, $E\{d(\mathbf{x}, \hat{\mathbf{x}})\} \leq D$.

Two broad categories of source coders are scalar quantizers and vector quantizers. A scalar quantizer can operate on a single source value at a time. Its action on each source value is independent. A vector quantizer blocks the source values into vectors and operates on one vector at a time. Its action on each source vector is independent. Some coders, including trellis coders, do not fall into these two categories.

An important aspect of source coding not addressed by rate distortion theory is computational complexity. In fact, according to the theory, optimal performance is only known to be reached in the limit of infinite computational and memory complexity. A source coder is a procedure or algorithm that converts real source values to bits (encoding) and then converts these bits to a reproduction of the source (decoding). In a communication system, this procedure takes place electronically in a dedicated circuit or computer. When designing source coders one must consider the ultimate cost of this circuit or computer.

2.2 Motivation

Scalar quantization schemes have an inherent performance limitation. At high rates, their rate distortion performance cannot be closer than 0.255 bits per sample, or 1.54 dB to the rate distortion bound for sufficiently smooth source distributions [39]. This bound is known to hold for scalar quantizers even at low rates [40] for a wide class of sources. The reason for this performance limitation is that a scalar quantizer, when applied to a sequence of source values, generates reproduc-

tion sequences on a square lattice. The decision regions for these sequences are hypercubes leading to non-minimal granular distortion. This performance limitation may or may not be a problem, depending on the application. The 0.255 bit penalty can have a large effect on coders that quantize data at low rates. At high rates, 0.255 bits per sample may be a small portion of the overall rate.

Vector quantizers overcome this limitation and achieve performance approaching the rate distortion bound as their dimension increases. Vector quantizers do not restrict their reproduction vectors to a lattice. They have decision regions that are more rounded, giving them a granularity gain compared to scalar quantizers. Scalar quantizers are simple to implement. Vector quantizers, however, have a high computational complexity and storage requirement. It is time consuming to compute the best reproduction vector from a high dimensional vector codebook, and memory consuming to store the codebook.

Structured quantization schemes have been developed to overcome both the scalar quantization rate distortion penalty and the vector quantization computation penalty. To overcome the rate distortion penalty, structured quantizers have decision regions that are not as granular as those from a scalar quantizer. To reduce their computational requirements, structured quantizers are organized so that the search for a suitable reproduction sequence is not exhaustive.

Some examples of structured quantizers are: tree structured vector quantizers, lattice vector quantizers, finite state quantizers, tree codes and trellis codes [41]. This work concentrates on trellis coded quantization, a specialized type of trellis coding.

The early trellis coders described in this chapter had little codebook structure. They could have a different codevalue on each branch and were often non-stationary, even when designed to encode an independent and identically distributed series of random source values. A series of simplifications leads to TCQ described in more detail in the next chapter.

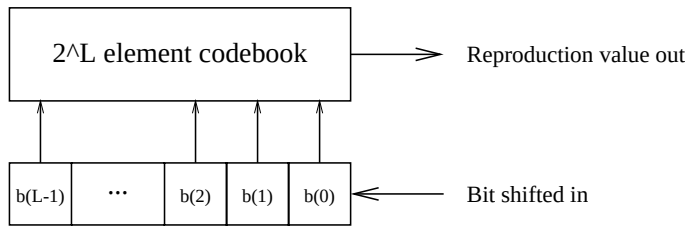

Figure 2.2: Trellis coding decoder with an L bit shift register indexing a codebook with 2^L reproduction values.

2.3 Trellis Coders

Trellis source coding developed naturally from tree coding. Tree codes have the disadvantage of an exponential growth in the number of states as the code progresses. Trellis codes overcome this problem by wrapping the tree back into itself to create a structure more like a vine.

At the heart of the decoder of a trellis source coder is a shift register. This register is an L bit vector, $\mathbf{b} = (b_0, b_1, \ldots, b_{L-1})$. Each bit moves over one position as a new bit is shifted in. One new bit is shifted in for each source value. During each interval, b_0 becomes the new bit shifted in, b_1 becomes the previous value of b_0, b_2 becomes the previous value of b_1, and so on. More general trellis configurations, not considered here, can have more than one bit shifted in at a time.

The decoder outputs reproduction values that are a function of the shift register, $\mathbf{b}$. This function is a codebook of 2^L reproduction values. After each new bit is shifted into the register, a new reproduction value is output. Figure 2.2 shows a diagram of the decoder.

The trellis decoder can also be described with a state transition diagram. The first $L-1$ bits of the shift register, $b_0, b_1, \ldots, b_{L-1}$, define the state. Given the current state and the new bits shifted in, we know the next state. Figure 2.3 shows the state transition diagram for a 3-bit shift register. The possible current states are on the left, and the possible next states are on the right. The possible state transitions, or branches, are either solid or dashed lines. Solid lines correspond to a 0 and dashed lines correspond to a 1 shifted into the register. The diagram shown is a single stage of a trellis diagram. As bits are repeatedly shifted into the register,

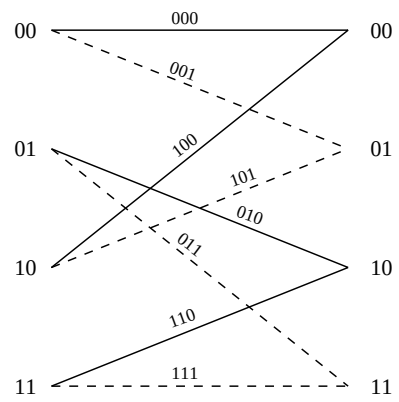

Figure 2.3: This state transition diagram for an $L = 3$, 4-state trellis has its branches labeled with the corresponding values in the 3-bit shift register.

the state transitions follow a path along an infinite repeated series of these stages.

The trellis diagram, as described, shows all possible values of the shift register. The reproduction values are a function of the shift register and can be shown by placing them on the branches of the trellis diagram. Each branch of the diagram corresponds to a unique value of the shift register.

The task of the encoder is more difficult. The encoder inputs the original source sequence and must select a sequence of bits to send to the decoder. This bitstream is generally selected so that the resulting reproduction sequence will be close to the source sequence with regard to a distance metric. Trellis source codes have low computational complexity because the search for the bitstream that results in the optimal reproduction sequence can be completed efficiently using the Viterbi algorithm [42]. A sub-optimal bitstream can be found with less complexity using the M-algorithm [43]. This algorithm is especially beneficial when L, the shift register size, is large.

To complete the design of the trellis source coder, we must define the reproduction values in the codebook. Viterbi proved the existence of trellis source codes that approach optimal rate distortion performance as the constraint length of the shift register increases [44]. He did not, however, provide a method for defining the codebook. Early trellis coders populated the trellis randomly according to the distribution of the source. That leads to a large storage requirement, especially if

the trellis is non-stationary to match a non-stationary source distribution.

Techniques have been developed to improve existing trellis codebook designs. These techniques are often used to improve an initial codebook generated randomly. Stewart [2] provides an algorithm that is is a generalization of the Linde-Buzo-Gray (LBG) algorithm [1]. Initial trellis codes were also iteratively improved by Freeman [3] who used conjugate gradient optimization.

2.4 Constrained Reproduction Alphabet

A major reduction in the codebook complexity of trellis coders came from Alphabet Constrained Rate Distortion Theory (ACRDT) [4]. In this theory, Finamore and Pearlman develop an approximate discrete rate distortion theory by finely quantizing a continuous source and restricting reproduction values to a small finite set. The source quantization adds negligible distortion. Discretization allows one to find optimal discrete test-channels with the Blahut-Arimoto algorithm [45]. Their results prove that nearly optimal coders exist which have a limited repertoire of reproduction values.

Nearly optimal constrained alphabet coders using a trellis structure were then found for the independent Gaussian and Laplacian sources [4, 5]. For rate $R = 1$, only 4 distinct reproduction values are used. These values depend on the source distribution and are assigned to the branches randomly. These coders are similar to that in Fig. 2.2. One difference is that they are non-stationary, so the codebook changes for each time increment. The important change is that, no matter what the time increment, the codebook is restricted to a small number of values.

The small set of reproduction values only contains complementary pairs. If x is in the set, so is $-x$. However, this does not imply symmetry in the reproduction sequences because of the random branch assignment.

The number of reproduction values needed to code close to the rate distortion bound for memoryless Gaussian and Laplacian sources is made clear by the work of Pearlman and Chekima [46]. For coding Gaussian sources at a rate of R bits per sample, 2^{R+1} reproduction values can put rate distortion performance very close to the bound. For Laplacian sources, it appears that 2^{R+2} values should be used.

CHAPTER 3
Trellis Coded Quantization

Trellis coding took another step forward, when, motivated by a duality with Trellis Coded Modulation (TCM) [8, 9, 10], Marcellin and Fischer introduced Trellis Coded Quantization (TCQ) [6, 7]. TCQ is characterized by a reproduction set expansion that approximately doubles its size compared to what would be used for scalar quantization at the same rate. The reproduction set is then partitioned into subsets (usually four) which are assigned to the branches of a trellis according to a particular scheme.

3.1 TCQ Decoder

The TCQ decoder, like the trellis decoder, has an L bit shift-register at its core. In contrast to the trellis coder, the reproduction values are not an arbitrary function of the shift register value. Generator vectors operate with the shift register to specify a subset of the whole codebook. Then, a reproduction value is selected from the subset. So, instead of placing a reproduction value on each branch of the trellis diagram, TCQ assigns one of a small number of subsets (usually four) to each branch with a particular structure.

For each value to be decoded, a single bit is entered into the shift-register. As in trellis coded modulation, two L-bit generator-vectors are used to define which codebook subset to use given the shift-register value. Separately, the generator-vectors undergo a bitwise AND with the shift register value and the resulting L-bit vector is modulo-2 summed (parity sum). Thus the single input bit results in two output bits which are used to select one of four subsets. Only certain sequences of codebook subset indices are possible, since the sequence is determined from a single input bit per stage. Figure 3.1 shows this process. Generator vectors that perform well for TCM are reported by Ungerboeck [8]. These generators are also universally used for TCQ.

The codebook subsets are essentially individual codebooks themselves. At the

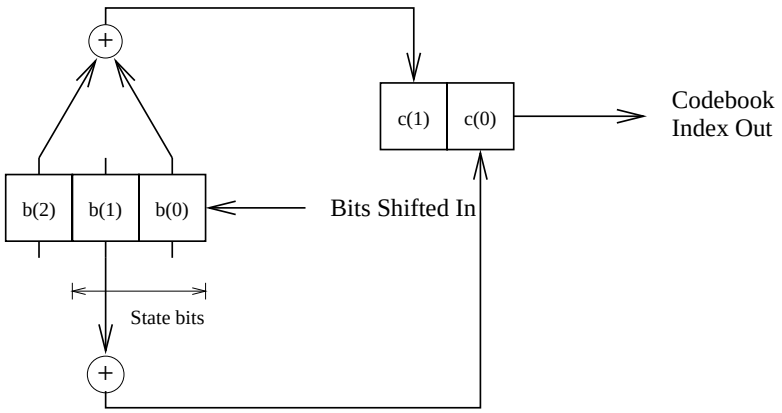

Figure 3.1: Shift register used to select subsets for TCQ. The $L = 3$ bit shift register feeds the two Ungerboeck generator vectors: $(1, 0, 1)$ and $(0, 1, 0)$.

simplest, each subset holds a single reproduction value, but they can be different sizes, or entropy-constrained, as we will see in Chapter 4. They can even hold vectors, so long as each is the same length.

A second bitstream is needed for TCQ to select individual reproduction values from the subsets. This bitstream is absent if the codebooks each contain only a single value. In practice the bitstream for the trellis path and the bitstream for the reproduction value selection are interleaved. The bits are placed on a single bitstream as soon as they are generated.

A trellis coded quantizer, like a trellis coder, can be represented graphically by a state transition diagram. Figure 3.2 shows one stage of such a diagram for an $L = 3$, 4-state trellis. From left to right this diagram shows the possible consecutive states as one bit is shifted into the shift register. Each of the $2^L = 8$ lines, or branches, uniquely corresponds to a value of the shift register. The value on the left of each branch shows the corresponding shift register value for the branch. The value on the right, in parenthesis, is the codebook index defined by the generators. The diagram shows only one stage of the trellis diagram; that stage is repeated indefinitely.

It is worthwhile to reconsider the two parts of the bitstream with the trellis diagram in mind. The first part of the bitstream sends bits for the shift register. As

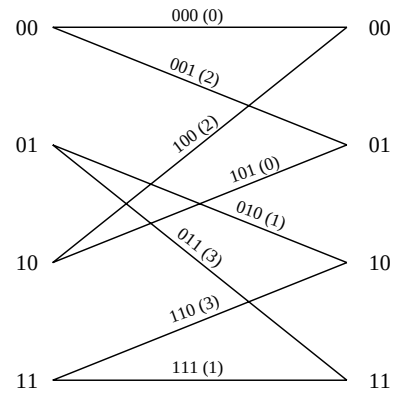

Figure 3.2: This state transition diagram for an $L = 3$, 4-state trellis has its branches labeled with the corresponding values in the 3-bit shift register. In parenthesis is the codebook index assigned by the Ungerboeck labeling.

far as the trellis diagram is concerned, these bits choose a path through the trellis with 1 bit per stage. This chosen path defines a sequence of codebooks.

The second part of the bitstream extracts reproduction values from the subsets. If each of the 4 subsets has N members, a simple indexing scheme will use $\lceil \log_2 N \rceil$ bits per stage. The total rate is $\lceil \log_2 N \rceil + 1$ bits per stage. Assuming that the subsets are scalar codebooks, the rate is $R = \lceil \log_2 N \rceil + 1$ bits per sample. Rates below 1 bit per sample will be achieved with trellis coded vector quantization and entropy-constrained TCQ, explored in the next chapter.

Trellis coders are generally assumed to have an infinite number of stages. In practice the coding must start and end. To avoid extra distortion of the initial source values, the trellis may be allowed to start in any state. This requires sending an extra $L-1$ bits to the decoder to initialize the state in the shift register. As the number of source values encoded gets large, this fixed overhead becomes insignificant. Sending the $L - 1$ bits which specify the initial state is equivalent to starting at state 0 but having $L - 1$ startup stages without codebooks on their branches, as shown in Fig. 3.3.

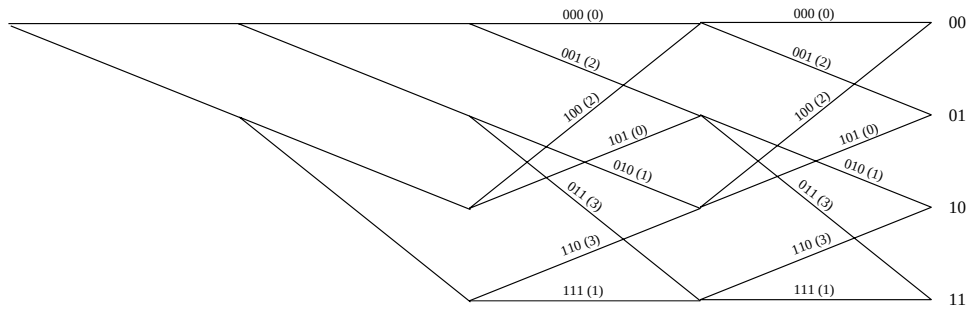

Figure 3.3: For an $L = 3$, 4-state trellis, $L - 1 = 2$ incomplete initial stages without codebooks assigned to their branches allow the decoder to be in any state when the complete stages begin.

3.2 TCQ Encoder

It is the job of the encoder to choose a path through the trellis and a sequence of codevalues from the subset on each stage of that path. The resulting sequence must satisfy some criteria with the source sequence. The most common criteria is that the reproduction sequence is as close as any possible reproduction sequence to the original source sequence, given some definition of distance. Usually, sum-squared distance is used. The bits defining the trellis path and choosing codebook elements are sent to the decoder so it can generate the reproduction sequence.

The most common reproduction sequence selection criteria is that the sequence introduces the least distortion of all possible sequences. If $\mathbf{x} = x_0, x_1, \ldots, x_{N-1}$ is the source sequence and $\hat{\mathbf{x}} = \hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{N-1}$ is the reproduction sequence, the distortion measure is written as $d(\mathbf{x}, \hat{\mathbf{x}})$. The sum-squared-difference distortion measure is,

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{n=0}^{N-1} (x_n - \hat{x}_n)^2. \tag{3.1}$$

Since the total sequence distortion is equal to the sum of the distortions of the sequence values taken individually we have,

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{n=0}^{N-1} d(x_n, \hat{x}_n), \tag{3.2}$$

which means this is a single-letter distortion measure.

For a single-letter distortion criteria, the seemingly difficult task of finding the optimal reproduction sequence is handled readily by the Viterbi algorithm [42]. This algorithm is used for all coders presented here.

3.3 Trellis Coding Gain

Trellis coded quantization has been described here in a general manner with regard to the codebooks assigned to the branches. This should not give the impression that TCQ is a technique to improve any existing type of codebook. The codebooks are not independent and must be carefully chosen to work together.

Trellis coded quantizers offer granular and shaping gain [47, 48] without high computational complexity. Scalar quantizers allow each source value to be quantized individually. Vector quantizers group them into blocks. Their complexity comes from this grouping. TCQ quantizes whole sequences of source values at a time. However, the structure of TCQ allows low complexity searching.

Trellis coded quantizers perform well with respect to rate, distortion, and computational complexity. A TCQ generates reproduction sequences which give granular gain and shaping gain over scalar quantization [47, 48]. However, the finite state structure allows the optimal reproduction sequence to be computed at the encoder with low complexity using the Viterbi algorithm or the similar nearly optimal M-algorithm.

It needs to be emphasized that TCQ is for memoryless data. There are many paths leading up to any particular branch. Thus, the reproduction values on a branch cannot be dependent on past reproduction values. Still, there are ways to code correlated data with TCQ. For example, Marcellin developed a modified TCQ in which a predictor operated on each surviving path in the encoder and the branch codebooks quantized the prediction error [7].

Some insight into why TCQ can perform as well as it does is provided by Alphabet Constrained Rate Distortion Theory (ACRDT) [4, 5]. For Gaussian distributed sources, 2^{R+1} reproduction values are sufficient to make a rate R coder operating near the rate distortion bound. ACRDT shows that such coders exist because it can find a test channel with mutual information R and distortion near

$R(D)$, but it does not construct such coders. TCQ provides a way of using a finite set of about 2^{R+1} reproduction values to create a nearly optimal encoder. TCQ uses about the same number of reproduction values as the trellis coders by Finamore and Pearlman [4, 5], but keeps them much more organized. They are all members of one of four subsets which are placed on branches of the trellis in an orderly way.

Initially, Marcellin optimized TCQ codebooks with a gradient search [6, 7]. The cost function is the quantization error after encoding a random training sequence. Like Freeman, Marcellin takes advantage of naturally occurring symmetry in the coder to reduce the search dimension.

CHAPTER 4
Entropy-Constrained Trellis Coded Quantization

Fundamentally, a trellis coded quantizer can have any type of codebook assigned to its branches. This chapter is concerned with the use of entropy-constrained codebooks [12]. We will start with a straightforward use of entropy-constrained codebooks and then introduce an important modification called state-entropy which will greatly improve the utility of ECTCQ.

4.1 Entropy-Constrained Quantization

The rate calculations used thus far have assumed the use of a fixed rate code. With a fixed rate code, each value from a codebook has a sequence of bits, or codeword, that represent it, and each of these codewords is the same length. This is not desirable if some of the codevalues are used more often than others. A variable length code that assigns short codewords to codevalues used often and long codewords to codevalues used infrequently will result in a lower average rate. A perfect variable length code will have a rate equal to the entropy of the codevalues.

It is straightforward to determine the optimal codeword lengths if the probability of using each codevalue is known [41]. If the probability of using codevalue x_i is p_i, then codevalue x_i should be represented by a codeword with length $l_i = -log_2 p_i$ bits. The amount of information that must be sent to the receiver in the event that a source value is represented by the codevalue x_i is l_i bits. If the codeword lengths, l_i, are integers, or at least well approximated by integers, then the Huffman technique can find codewords with the desired lengths. Otherwise, arithmetic coding can be used [49, 50]. In the following discussion, we will be more concerned with the codeword lengths than the codewords themselves.

An entropy-constrained quantizer uses a variable length code. So, its codebook has a codeword length l_i associated with each codevalue x_i. But, it also takes this codeword length into account when selecting codevalues. It does this by incorporating the codeword length into the distance function used to determine the

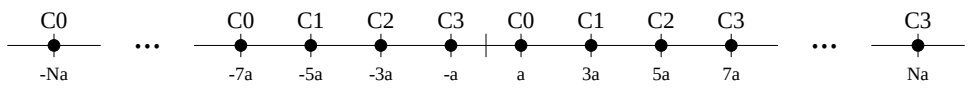

Figure 4.1: The uniform amplitude modulation codebook is depicted.

optimal reproduction value,

$$d(x, x_i) = (x - x_i)^2 + \lambda l_i. \tag{4.1}$$

The sum squared distance is modified to account for the cost of transmitting the codeword.

The Lloyd-Max codebook optimization [51] algorithm is readily extended to entropy-constrained coders, as is the LBG algorithm [1].

4.2 Subset-Entropy ECTCQ

To overcome the rate restrictions of TCQ with simple codebook indexing, Fischer developed Entropy-Constrained Trellis Coded Quantization (ECTCQ) [11]. ECTCQ has subsets of scalar reproduction vales with associated codeword lengths. The coders described by Fischer, however, still have the restriction that the rate be greater than 1 bit per sample since one bit per sample is needed for the shift register to define the path through the trellis.

Training an entropy-constrained TCQ can be accomplished with the generalized LBG algorithm [1]. The TCM partitioned amplitude modulation codebook in Fig. 4.1 is a reasonable choice as an initial codebook. This codebook has uniformly spaced reproduction values, with alternating subset assignments. The reproduction values extend from $-Na$ to Na where a is the reproduction value spacing and N is chosen large enough that the probability of using the reproduction values near $-Na$ and Na is negligible. The initial probability table for the subsets is uniform.

The codebook is trained by applying the ECTCQ to test data. Then, each codevalue is replaced by the mean of the source values it has been chosen to represent and the codeword lengths are replaced using the frequency of use for each codevalue. These training runs are repeated until the change in rate distortion performance

becomes negligible. Generally, N is selected larger than needed and the first training iteration eliminates the outlier reproduction values.

The rate of the coder depends on the Laplacian multiplier, λ. To design a series of codebooks which operate at different rates, λ starts at a value that gives the highest desired rate and is iterated through a series of values. After each increase, the codebook is retrained and saved. A logarithmic progression of λ results in a series of codebooks with rates progressing approximately uniformly.

The amount of training required depends on the initial codebook. The most training is needed to design the first ECTCQ from the uniform initial codebook. If the changes in λ are small, then each designed codebook is a good starting point from which to start training the next one. Codebooks designed for a particular number of trellis states also provide a good starting point for trellises with more states at the same λ. It is a good idea to initially train codebooks for 4-state trellises, and to use these as initial codebooks when training for trellises with more states, since they require more computation.

4.3 State-Entropy ECTCQ

The ECTCQ system described above uses a codebook with four subsets that are independent in the sense that each has its own frequency of use table. This table tells the relative frequencies with which each codevalue is selected, given that that subset is used.

An important modification can be made if the branches are labeled with subsets following Ungerboeck's rules [8]. Under Ungerboeck's branch labeling rules, the four codebooks are grouped into two pairs: C_0 with C_2, and C_1 with C_3. At any state, the two possible codebooks for the next stage are one of these pairs.

Marcellin combined these two pairs of subsets into joint subsets, $D_0 = C_0 \cup C_2$ and $D_1 = C_1 \cup C_3$. The joint subsets have a combined probability table. Selecting a codevalue from, say, D_0 has a secondary effect of selecting a branch to follow to the next stage of the trellis. The branch taken depends on whether the codevalue selected came originally from C_0 and C_2.

This change tightly integrates the bitstream to specify the path through the

trellis and the bitstream to select a codevalue from the codebooks. The codeword length tables of C_0 and C_2 are combined, which is why this enhancement is called state-entropy ECTCQ. The entropy to get from one state to another is used, as opposed to the entropy to select a codevalue from a subset. Initially in the design process, C_0 and C_2 are combined to form D_0 and 1 is added to all the codeword lengths. The probabilities of using C_0 or C_2, based on these codeword lengths, are initially both 1/2. There is no constraint to maintain this, however, and as the Lagrangian multiplier is increased in training, these probabilities diverge and the rate goes below 1 bit per source value. As this happens, some branches of the trellis become more frequently used.

Figure 4.2 show a plot of the rate distortion performance of 4-state state-entropy ECTCQ on the unit variance memoryless Gaussian source using the squared error distortion measure. Also plotted is the performance of scalar entropy coded quantization, the rate distortion bound and Gish-Pierce bound for comparison. The ECTCQ plot is essentially identical to Marcellin's [13]. Figure 4.3 shows the performance of 64-state state entropy ECTCQ, in addition to the graphs in Fig. 4.2.

The trellis coders used to compute the results shown in these figures were implemented both on a general purpose scalar computer and a massively parallel 2048 processor MasPar computer. The operation of training TCQ has straightforward parallelism. Random test data is generated for each processor and simultaneously coded. This is efficient for training ECTCQ with a low number of states. When the per-processor memory limit is exceeded by a trellis with more states or adaptive entropy coding is used, the scalar computer version must be used.

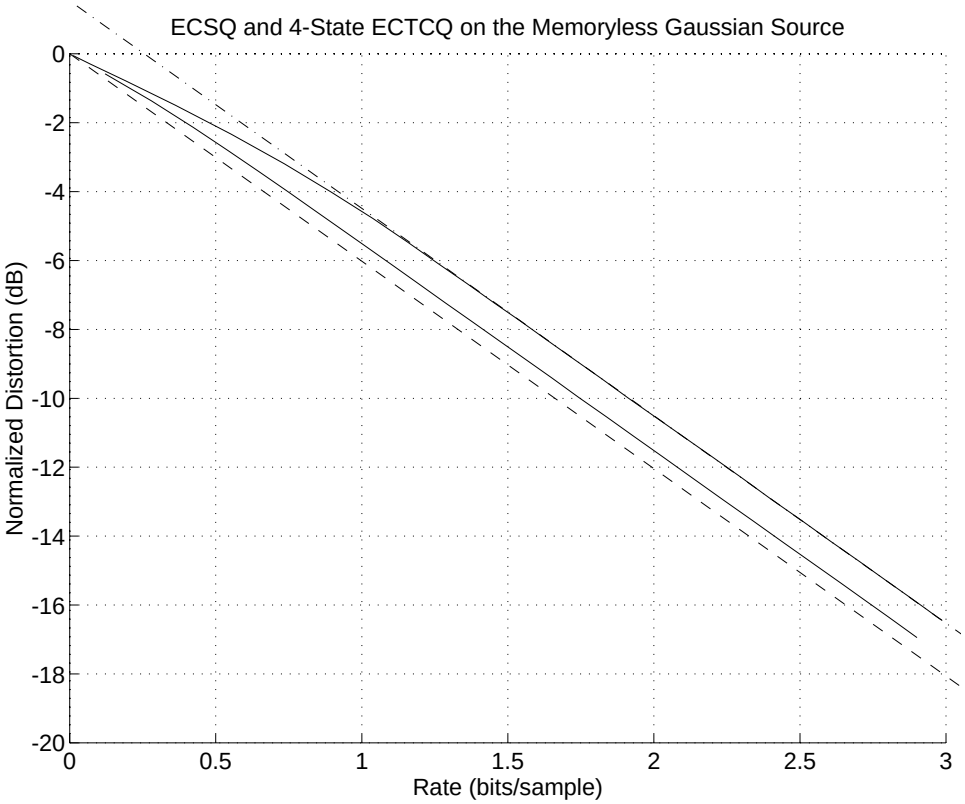

Figure 4.2: Rate distortion performance for the memoryless Gaussian source are graphed. The lowest dashed graph is the rate distortion bound. Just above the bound is the performance of 4-state ECTCQ as a solid line. The performance of scalar entropy-constrained quantization is also solid. At most rates, this graph overlaps the Gish-Pierce bound shown in dash-dot.

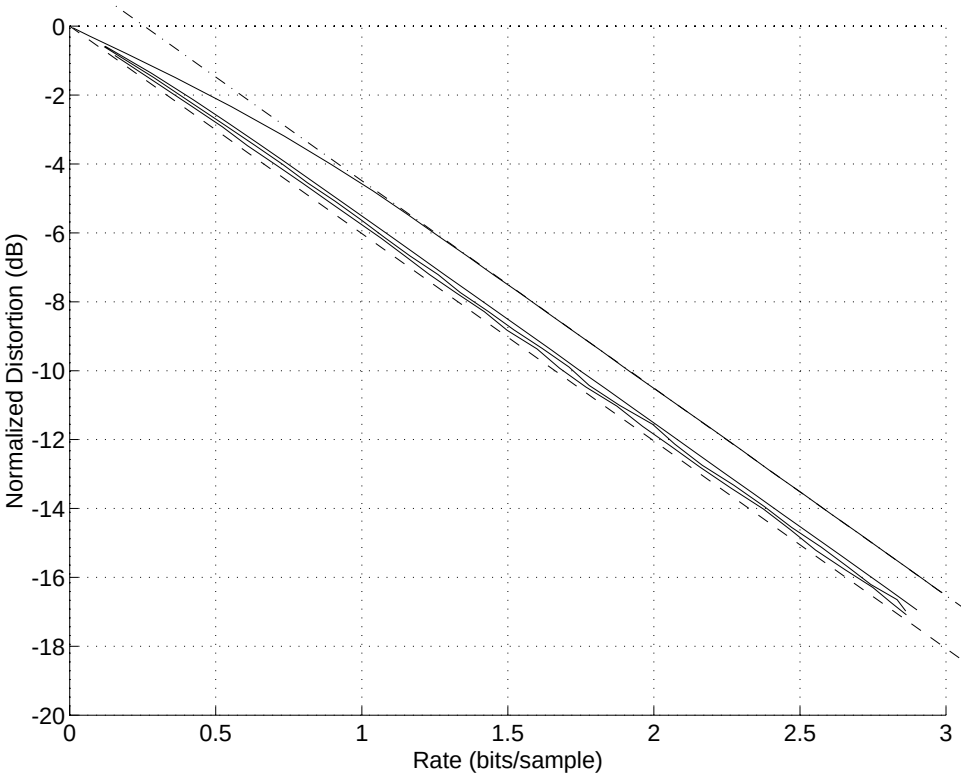

Figure 4.3: The same plots as in Fig. 4.2 and the performance of 16 and 64-state ECTCQ.

CHAPTER 5
Probable Paths and Alternate Branch Assignments

In this chapter, state-entropy ECTCQ is enhanced with a codebook constraint that imposes symmetry. The codebook for the state-entropy ECTCQ has two subsets, D_0 and D_1, with independent codevalues and codeword lengths. The codebook structure can be constrained by making the codevalues of D_0 and D_1 additive inverses with identical probability tables. Below, we will make a case for the logic of this constraint, describe similar symmetric constraints used in the past, discuss the benefits of the symmetry and present some empirical results using symmetric state-entropy ECTCQ. We will find that the constrained coder has identical rate distortion performance as the unconstrained coder and some advantages.

5.1 Symmetry

Figure 5.1 shows the uniform codebook that is used as an initial codebook when training ECTCQ. Each codevalue is labeled as a member of one of the four subsets, C_0, C_1, C_2 or C_3. There is symmetry in this codebook defined by $C_0 = \{-c : c \in C_3\}$ and $C_2 = \{-c : c \in C_1\}$. We can constrain the codebook so that this symmetry is enforced through its training in an ECTCQ. The codevalue symmetry is maintained, and the codeword lengths of additive inverse pairs of codevalues are kept equal.

This symmetry constraint is applicable to codebooks used for the subset-entropy or state-entropy versions of ECTCQ. Since state-entropy ECTCQ is so advantageous and more widely used, we restrict the experiments below to state-entropy ECTCQ.

Even when the symmetry is not enforced, it occurs naturally. This is illustrated

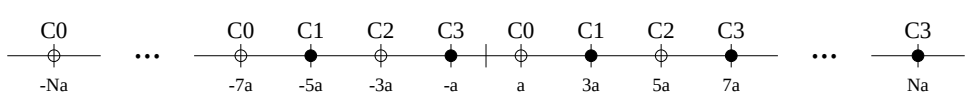

Figure 5.1: Uniform amplitude modulation codebook showing symmetric subsets.

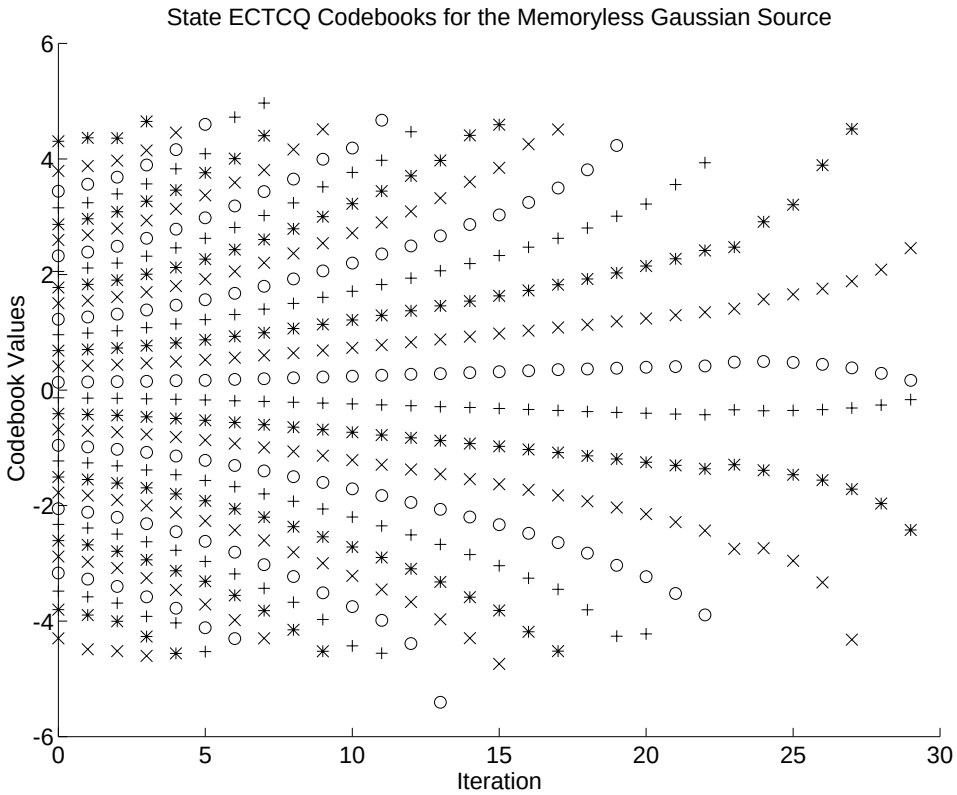

Figure 5.2: State-entropy ECTCQ codebook values for the memoryless Gaussian source. Each symbol type corresponds to a particular subset.

in Fig. 5.2 which shows state-entropy ECTCQ codevalues for the Gaussian source and in Fig. 5.3 which shows symmetric ECTCQ codevalues for the same source. Because the symmetry is evident even when not enforced, it is no surprise that the symmetry has no detrimental effect on rate distortion performance.

The codevalues shown in Figs. 5.2 and 5.3 are plotted versus iteration number instead of the more appropriate codebook rate. As the training progressed, the Laplacian multiplier, λ was increased logarithmically. This results in an approximately linear decrease in rate. Thus, the horizontal axis of the plots are approximately linear with respect to rate.

To express more clearly the subset symmetry we introduce some new notation: $+A = C_0$, $-B = C_1$, $+B = C_2$ and $-A = C_3$. We will denote the state-entropy joint subsets in a similar way, $+C = D_0$ and $-C = D_1$. The four subsets and the

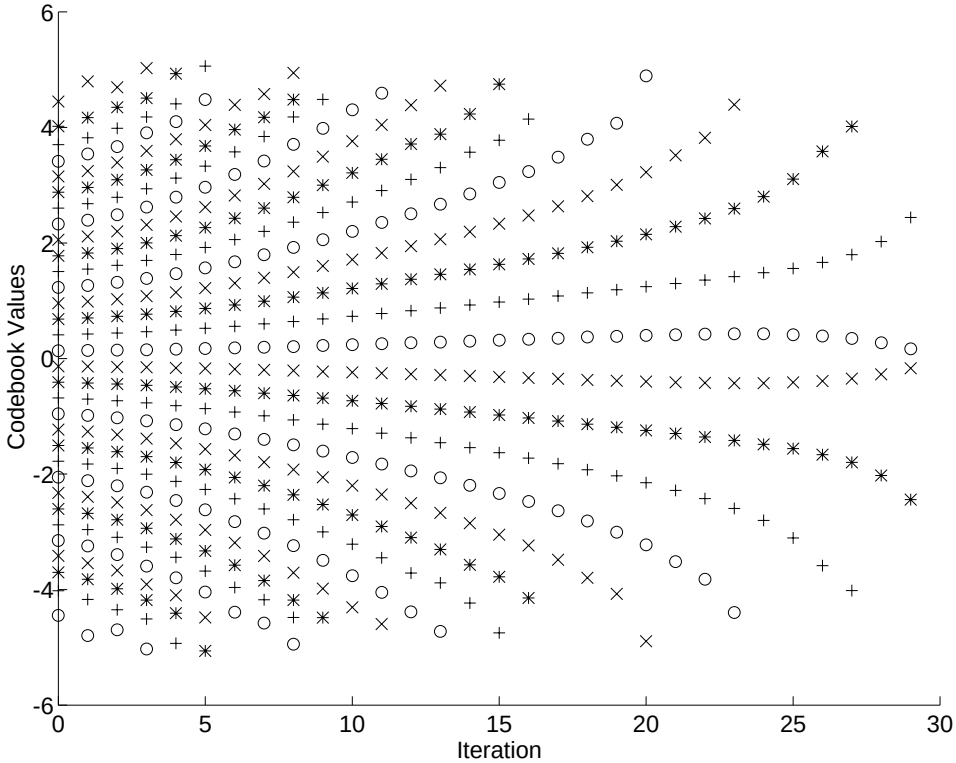

Figure 5.3: Symmetric ECTCQ codebook values for the memoryless Gaussian source. Each symbol type corresponds to a particular subset.

two joint subsets that contain them are given in both notations in this table.

Individual Subset	Joint Subset	Symmetric Subset	Symmetric Joint Subset
C_0	D_0	+A	+C
C_1	D_1	-B	-C
C_2	D_0	+B	+C
C_3	D_1	-A	-C

A generalized Linde-Buzo-Gray (LBG) algorithm [1] is often used to train entropy-constrained TCQ. The ECTCQ is used to code a sequence of test data. Then, the mean of all the test values reproduced by a particular codevalue replaces that codevalue. With the symmetry constraint, when a test value x is reproduced by a codevalue $\hat{x}$ in, say, $-A$, count that as $-x$ being reproduced by the codevalue

$-\hat{x}$ in $+A$. Codevalues from $-B$ are handled in the same manner.

5.2 Prior Research

In early work [6, 52], Marcellin enforced this same symmetry on TCQ codebooks. These codebooks were not entropy-constrained. They were trained with a numerical optimization algorithm which would converge faster because the number of free variables is cut in half.

In his thesis, van der Vlueten has designed trellis codes using the fake process approach [53]. He found this same symmetry property as a condition for a random walk of the trellis to generate a white output sequence. Further, Jafarkhani, Farvardin and Lee mention the use of codebook symmetry with ECTCQ [54, 55].

5.3 Symmetry Advantages

The symmetry constraint gives the coder a few straightforward advantages. These advantages are in both the design of the coder and in implementation. This symmetry reduces the training time for the design of an ECTCQ coder with the generalized LBG technique. Since the coder has half as many parameters, training takes place in about half the time taken by an ECTCQ coder without the symmetry constraint. Since the number of parameters which define the codebook is cut in half, the storage requirement is halved. For the entropy-constrained codebook, the number codevalues and codeword lengths are halved.

The most important advantage may occur when the ECTCQ has its codewords coded by an adaptive entropy coder, such as an adaptive arithmetic coder. An adaptive entropy coder will improve the performance if the source distribution is different from the distribution the coder was designed for. This could occur if it is not known precisely or if the source is non-stationary. With symmetric ECTCQ, the adaptive entropy coder will adapt faster because there is one symbol stream to estimate statistics for instead of two.

The symmetry condition can not improve the rate distortion performance of ECTCQ. Viewing ECTCQ design as an optimization problem, this is obvious. Without the symmetry condition, the design problem is to find the codebook that

gives the best rate distortion performance. Constraining the codebook can not improve this performance.

The crucial concern is whether the symmetry hurts rate distortion performance. Given that the source has a symmetric distribution, it is intuitive that there is no penalty for the symmetry constraint. Empirical results show this to be the case. The plot in Fig. 4.2 show the rate distortion operating points of ECTCQ coders for the Gaussian source. The performance of coders with and without the symmetry constraint are essentially identical.

5.4 Probable Paths

When the ECTCQ described here operates at rates above about 2 bits per sample, the four subsets are used with nearly equal frequency. As the rate drops, either A or B will become more probable. Because the initial codebook from Fig. 5.1 has members of subset A closer to the origin, subset A invariably becomes more frequently used. We will assume that this is given and call A the more probable subset. Figure 5.4 shows the probability of subsets A and B splitting as rate drops for an ECTCQ designed for the Gaussian source.

Figure 5.5 illustrates the more probable sequences, or paths. These three trellis stages have symmetric codebook labels corresponding to Ungerboeck's assignments for a 4-state trellis. Branches labeled with +A or -A are bold to emphasize the higher probability of these subsets. Branches labeled with the inverse subsets, -A and -B, are dashed.

The probable sequence leaving each state is the one made up of branches labeled with a +A or a -A, or following the bold path. For each state, these sequences are: +A,+A,+A; -A,-A,+A; +A,-A,-A; and -A,+A,-A.

The probable sequences of the trellis in Fig. 5.5 are not symmetric. As an example, one of the subset sequences is +A,+A,+A, but -A,-A,-A cannot be produced by the trellis.

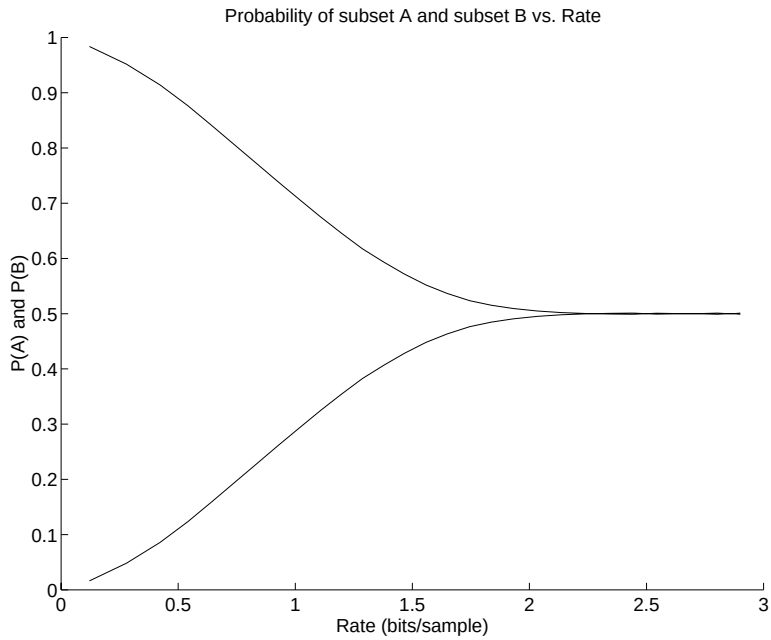

Figure 5.4: Probability of subset A and subset B versus rate for Symmetric ECTCQ on the memoryless Gaussian source.

5.5 Alternate Branch Labelings

Trellis coded dequantizers use generator vectors to indicate which codebook is used on each branch of the trellis, following the TCM techniques by Ungerboeck. Ungerboeck provided three rules for determining which branch labelings will work well for TCM. We will call branch labelings that follow the spirit of these rules, in the TCQ domain, *valid* labelings. There are valid labelings that cannot be implemented by generator vectors.

The three design rules given by Ungerboeck [9, 10] are quoted below, along with explanations as to how the rules apply to valid labelings for TCQ.

1. "Parallel transitions are associated with signals with (the largest possible) maximum distance ... between them." A parallel transition is from one sequence to another sequence that shares the same path through the trellis but differs at one element. This rule is already satisfied because of the way in which the individual reproduction values are assigned to subsets, as seen in Figs. 4.1 and 5.1.

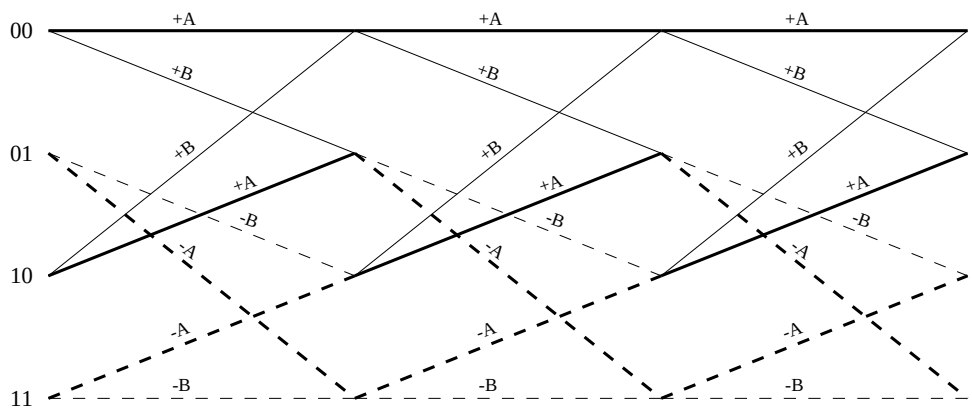

Figure 5.5: TCQ stages with symmetric codebook labels show the most frequent paths.

2. "Four transitions originating from or merging in one state are labeled with signals with (the next greatest possible distance)." For TCQ, this means that the two branches leaving a state must be labeled +A and +B or be labeled -A and -B. Likewise, the same applies to the two branches leaving any state. For example, it is not allowed for the two branches leaving a state to be labeled +A and -B because these two subsets have reproduction values that are close to each other.

3. "All ... signals are used in the trellis diagram with equal frequency." Thus, for TCQ, the subsets +A, -A, +B and -B each appear in the trellis diagram with equal frequency. In TCQ, in contrast to TCM, because of the non-uniform source distribution and entropy constraints, not all subsets, or reproduction values within the subsets can be used equally.

Figure 5.6 shows two such valid branch labelings for the 4-state trellis. If we include the labeling in Fig. 5.5 we have shown all the valid labelings for 4-state trellises disregarding labelings that are equivalent after some renumbering of the states.

These two new branch assignments have been pursued because they produce symmetric subset sequences. However, though each new labeling has rate distortion performance near the trellis with Ungerboeck's labels, neither performs as well. The minimum free distance of the new labelings is not as high as the Ungerboeck labeling.

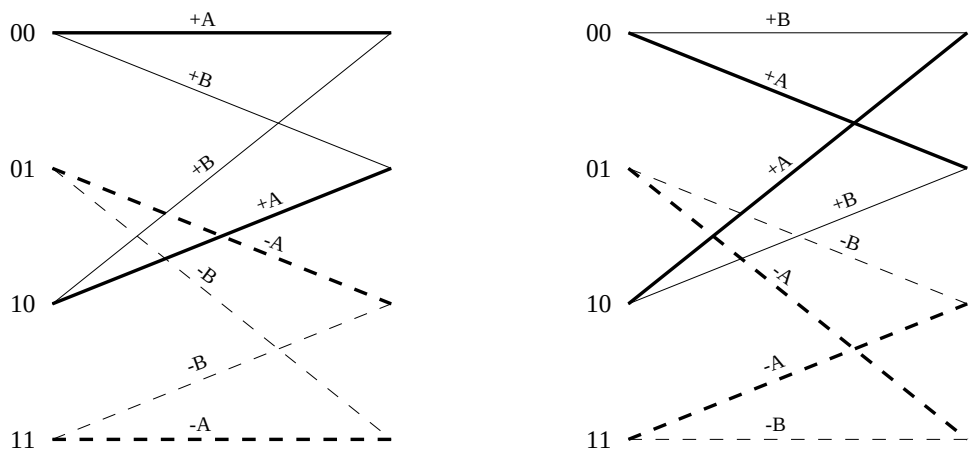

Figure 5.6: TCQ stages with alternate symmetric codebook labels show the most frequent paths.

A similar search for alternate branch labelings has been undertaken for 8-state trellises. There are 30 valid branch labelings for 8-state trellises that meet Ungerboeck's rules. Some of these can be implemented with generator vectors, but most cannot. Of alternate labelings that cannot be implemented with generator vectors, none were found to outperform the standard Ungerboeck labeling for coding the Gaussian source. The same cannot be said for the non-Ungerboeck labelings that can be implemented with generator vectors.

For 8-state trellises, Ungerboeck prescribes the generator vectors 4_8 and 13_8. The generator vectors 4_8 and 17_8 give a slight improvement for symmetric ECTCQ coding the memoryless Gaussian source. Figure 5.7 shows the trellis branch labels resulting from these generator vectors, and Figure 5.8 shows the coding results. The improvement is slight, amounting to a maximum decrease in distortion of about 0.018dB. The distortion is decreased near this amount only for rates between about 0.5 and 1.0 bits/sample.

A curious point about the 4_8, 17_8 generator vectors is that the resulting trellis does not produce symmetric subset sequences, yet the Ungerboeck generator vectors do.

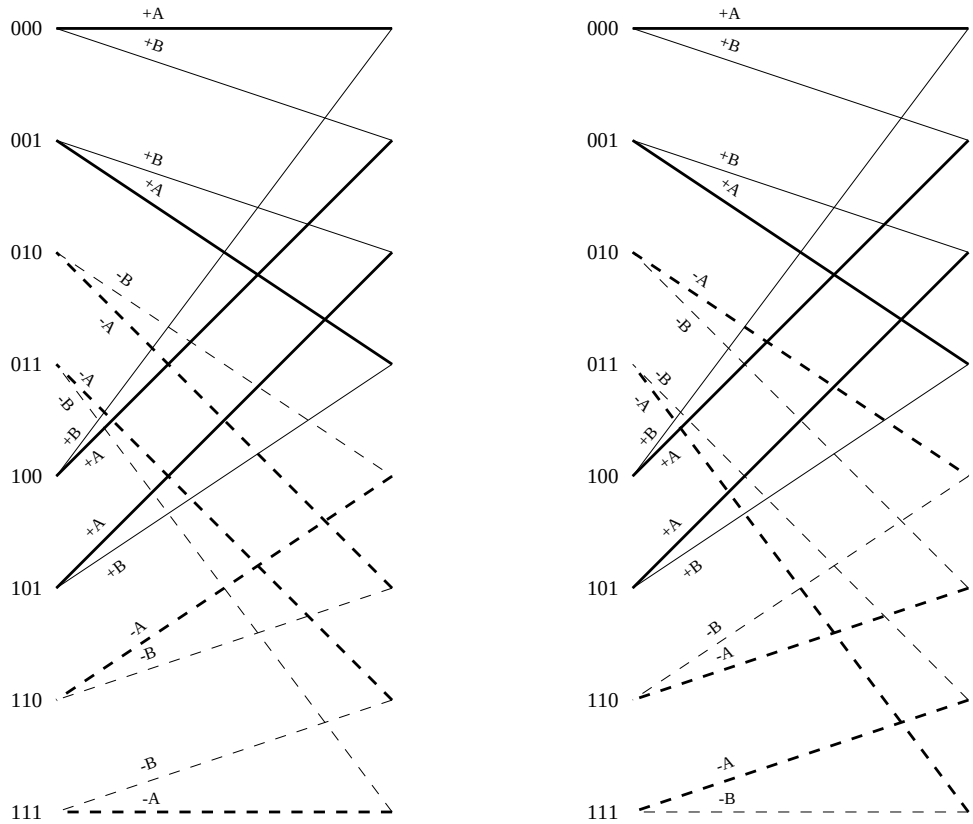

Figure 5.7: These two 8-state trellis stages show the branch labeling for the Ungerboeck ($04_8, 13_8$) generator vectors and for the new ECTCQ ($04_8, 17_8$) generator vectors.

5.6 Statistical Properties of ECTCQ

In this section, we will look at a statistical property of entropy-constrained TCQ reproduction sequences for the memoryless Gaussian source. This property demonstrates how ECTCQ operates more efficiently as the number of states increases. As the number of states increases, the reproduction radius of the reproduction sequences approaches the optimal reproduction sphere given by Sakrison [14].

Sakrison considers source coding independent Gaussian random variables with variance σ^2 in vectors of length L. He uses geometric arguments to derive the rate distortion bound for this source, $D(R) = \sigma^2 2^{-2R}$ and shows that as L grows larger,

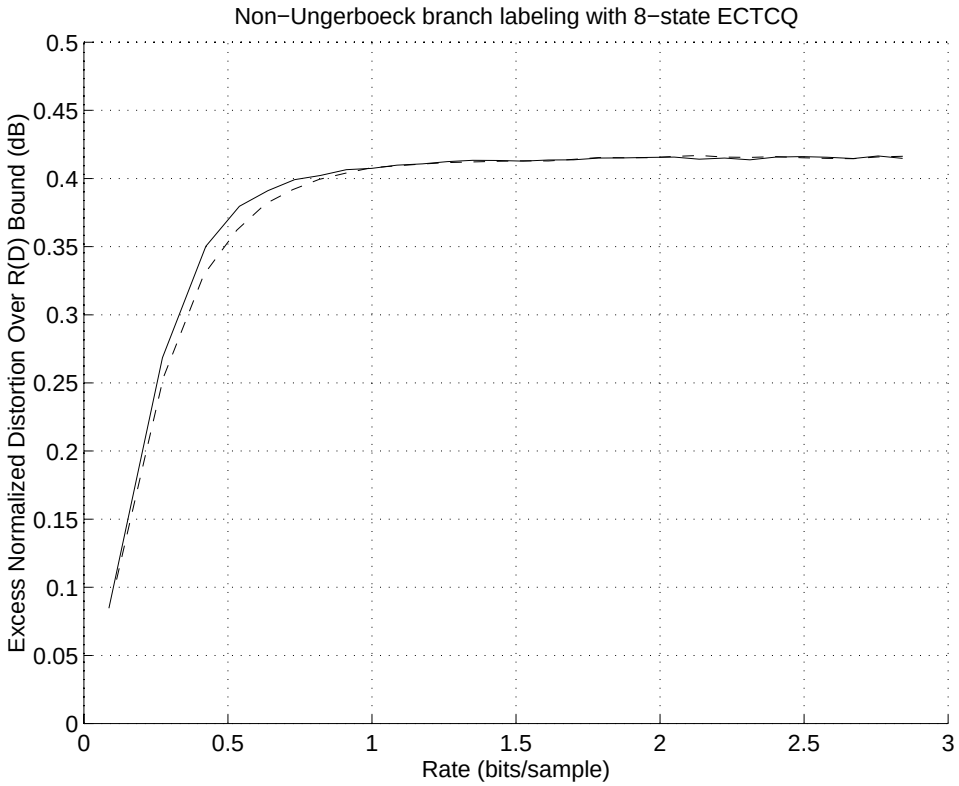

Figure 5.8: Shown is the excess distortion above the rate distortion bound for the Gaussian source of two ECTCQ coders. The upper solid plot line is for ECTCQ with Ungerboeck's generator vectors, 4_8 and 13_8. The improved dashed plot line is for the generator vectors 4_8 and 17_8.

the source vectors will be arbitrarily close to the L-sphere with radius σ. Further, he shows that, as l grows larger, the rate distortion bound can be approached if the 2^{RL} reproduction vectors are all on the L-sphere with radius $\sigma\sqrt{1-D(R)}$. Sakrison's proofs rely on the spherical symmetry of the multi-dimensional Gaussian distribution, so his results are limited to that source.

A trellis coded quantizer defines a set of infinite length reproduction sequences. These sequences are constrained by the limited codebooks and the trellis structure, so they are by no means equivalent to an infinite length vector quantizer. As the number of states increases, the reproduction sequences gain flexibility to be distributed such that the rate distortion performance of the TCQ improves. We will observe below that increasing the number of states used for ECTCQ causes the

reproduction sequences to approach the optimal reproduction sphere.

The idea of a reproduction sphere must be approached a bit differently for ECTCQ than for VQ. A VQ codebook has a finite number of vectors and their lengths are found readily. There are an infinite number of reproduction sequences that can be produced by an ECTCQ decoder.

To get around this problem, we use a probabilistic approach. Let x_n be the source sequence being quantized and $\hat{x}_n$ the quantized representation. We will only consider $\hat{x}_n$ for investigating the reproduction radius of ECTCQ. This will allow us to avoid the infinite number of reproduction sequences.

For a symmetric ECTCQ, appropriately designed by training for the Gaussian source, we can find some statistics of the reproduction sequence $\hat{x}_n$. Because the source has a symmetric distribution, and because the codebooks are symmetric, we will assume that the mean of $\hat{x}_n$ is zero. The fact that the codebooks are symmetric and entropy-constrained makes it easy to find the variance of $\hat{x}_n$. Depending on the state of the trellis, $\hat{x}_n$ may be drawn from either of the joint subsets, +C or -C. But, since we are concerned with variance, the sign does not matter so we assume that $\hat{x}_n$ is drawn from +C. Since the codebook is entropy-constrained we have a probability table that specifies the frequency with which the elements of +C are used. The variance of $\hat{x}_n$ is simply $\sigma_r^2 = E\{x^2\}$ where x is drawn from +C according to the probability table. The reproduction radius of the quantizer is then σ_r.

Figure 5.9 shows plots of these values for the ECTCQ coding of the Gaussian source with variance $\sigma^2 = 1$ for rates from 0 to 3 bits. This plot shows $\sqrt{1 - D(R)}$, the optimal reproduction radius, and the experimental reproduction radius $\sqrt{E\{\hat{x}_n^2\}}$ for Entropy Constrained Scalar Quantization (ECSQ) (1-state ECTCQ), 4-state ECTCQ and 64-state ECTCQ. 8-state, 16-state and 32-state were left out to keep the plot legible. The plots for these other trellises fall in order between the 4-state and 64-state plots. We can see that as the number of states increases, the reproduction radius of the sequences produced by the ECTCQ decoder increases, approaching the optimal radius, $\sqrt{1 - D(R)}$, for Gaussian sources, given by Sakrison.

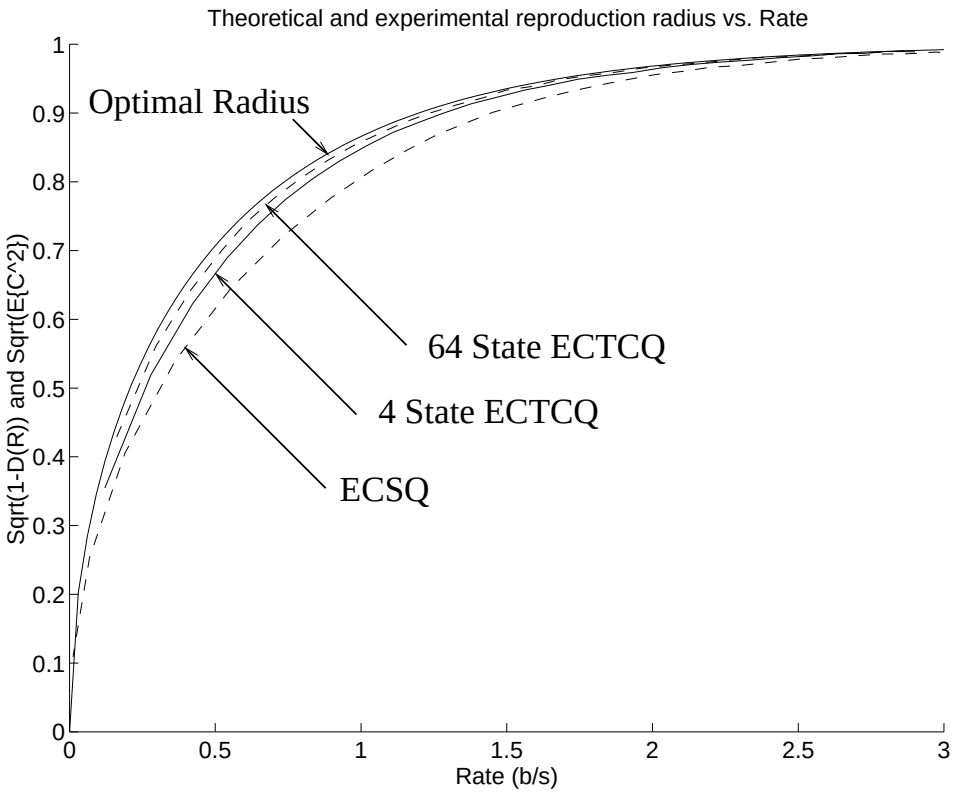

Figure 5.9: The optimal reproduction radius is compared to the experimental reproduction radius for ECSQ, 4-state ECTCQ and 64-state ECTCQ.

PART II

Image Coding

CHAPTER 6
Image Coding Background

This chapter begins with a brief discussion of imagery and image coding to establish notation and motivate this work. This is followed by an overview of the subband transform and scalar quantization. Then some important bitstream properties will be defined and discussed. At the end of this chapter the SPIHT and SPECK image coding algorithms, which form the core of later work, are described. We will work through the major steps in an image coding system shown in Fig. 6.1.

6.1 Images

Comprehensive background in image acquisition and processing is available in several texts [56, 57, 58, 59]. For the purposes of this work, an image is an N_r row by N_c column array of discrete values, or pixels, representing a projected view of some natural scene. Digital photographs are the most common example. Let $\mathcal{I}$ represent such an image, and $\mathcal{I}(n_r, n_c)$ one pixel of the image where $n_r = 0, \ldots, N_r - 1$ is the row and $n_c = 0, \ldots, N_c - 1$ is the column.

For grayscale images, each pixel is usually an integer between 0 and 255, pro-

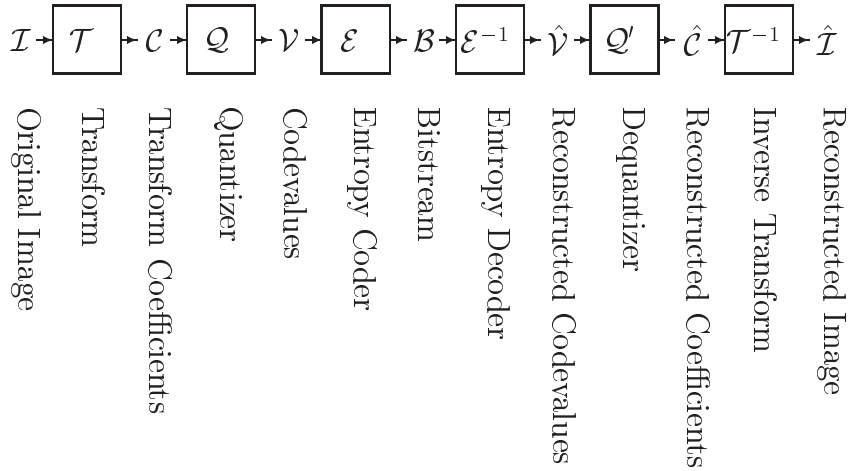

Figure 6.1: Major stages of a typical image coder.

portional to brightness. This range can be stored in 8 bits, or 1 byte. For color images, each pixel is represented by three such values. Often the three values represent the red, green and blue brightness for the pixel, but other color representations are also used.

In their raw form, images require 8 bits per pixel for storage or transmission. In most applications, this is a considerable burden. Thankfully, images of interest contain less than 8 bits per pixel of actual information.

6.2 Image Coding

A clear ongoing trend in the digital imaging industry is the steady increase in both image size and resolution. This is due to the development of higher quality and less expensive image acquisition devices. As new and better image acquisition techniques and products are developed in the future, this trend will continue, resulting in greater need for image compression technology.

The primary goal of image coding is to reduce the number of bits needed to represent an image. An image codec is an encoder and decoder pair. The image encoder maps an image $\mathcal{I}$ to a bitstream $\mathcal{B}$ that is a sequence of B bits. The image decoder maps the bitstream $\mathcal{B}$ to a reproduced image $\hat{\mathcal{I}}$.

If $\mathcal{I} = \hat{\mathcal{I}}$ the codec is said to be *lossless*. The performance of a lossless codec on a particular image should judged not only by the rate but also by its computational resource needs. The rate, $R = B/N_r N_c$ is the number of bits placed on the bitstream per pixel. Important computational resources include CPU and memory usage.

If $\mathcal{I} \neq \hat{\mathcal{I}}$ the codec is *lossy*. In many applications it is worthwhile to allow $\mathcal{I} \neq \hat{\mathcal{I}}$ because it can lead to a significant reduction in rate. The performance of a lossy codec is not only judged by the rate, R, and computational needs, but also by the distortion in the reconstructed image. A distortion measure, $d(\mathcal{I}, \hat{\mathcal{I}})$, is needed to evaluate the quality of the reconstructed image. The most common measure of distortion by far is Mean Squared Error (MSE),

$$D = d(\mathcal{I}, \hat{\mathcal{I}}) = \frac{1}{N_r N_c} \sum_{n_r} \sum_{n_c} (\mathcal{I}(n_r, n_c) - \hat{\mathcal{I}}(n_r, n_c))^2. \tag{6.1}$$

The distortion is conventionally expressed in decibels as the Peak Signal to Noise Ratio (PSNR),

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{D}, \tag{6.2}$$

when the peak pixel value of the image is 255. Usually a lossy image codec, controlled by some parameter, has many possible rate-distortion (R, D) operating points.

6.3 Subband Transform

The image codec strives to reduce rate by exploiting redundancy, or correlation, in the image. This task is simplified if we begin by partially decorrelating the image with a suitable transform, $\mathcal{T}$. Let $\mathcal{C} = \mathcal{T}(\mathcal{I})$ be an N_r by N_c array of transform coefficients. Most image coding transforms preserve the number of values and the arrangement in rows and columns. The encoder then generates a bitstream from the transform coefficients. The decoder converts the bitstream to reconstructed coefficients $\hat{\mathcal{C}}$. The reconstructed image is obtained with the inverse transform, $\hat{\mathcal{I}} = \mathcal{T}^{-1}(\hat{\mathcal{C}})$.

In the past the Discrete Cosine Transform (DCT) was a heavily favored decorrelating transform. It is used in the original JPEG image compression standard and is the primary transform in almost all MPEG video compression standard modes.

Recent research and some standards have favored the subband transform, first applied to imagery by Woods and O'Neil [60, 61], which is now usually implemented with discrete wavelet transforms [62]. The forthcoming JPEG 2000 image compression standard and the texture image compression mode of the MPEG-4 video compression standard will both use a wavelet transform.

The multi-level dyadic wavelet transform provides a signal representation convenient for exploiting the specific statistical dependencies of images. One level of a 2-D wavelet transform is performed on an image via a series of core 1-D wavelet transforms. First, a 1-D wavelet transform is applied to each row of the image, then to each column of the result. A reflection extension is used at the image boundaries to reduce edge effects and preserve the number of coefficients. An L level dyadic wavelet transform is performed by recursively applying this procedure to the lowest spatial frequency band of the $L-1$ level decomposition. Most wavelet im-

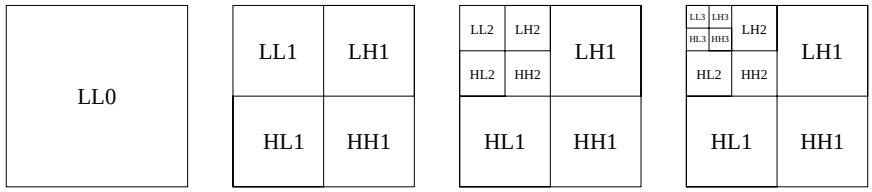

Figure 6.2: This diagram shows an image, its 1 level, 2 level and 3 level dyadic wavelet transforms. The whole original image is labeled LL0 for the sake of notational continuity.

age codecs operate on a dyadic decomposition of about $L = 5$ levels. The inverse wavelet transform is formed by simply reversing these steps, applying the inverse 1-D wavelet transform.

Figure 6.2 diagrams the structure of the dyadic wavelet decomposition. The whole original image is labeled LL0 for the sake of notational continuity. A one level transform splits the image into 4 bands, LL1, HL1, LH1 and HH1. The LL1 band is the low pass band. The others are high pass bands of different orientations. To produce the two level decomposition, only the LL1 band is changed. It is further split into 4 bands.

In Fig. 6.3 is an example of a dyadic wavelet decomposition of a real image. In the wavelet decomposition image, brightness is proportional to the logarithm of the magnitude of the coefficients. The logarithm is used here because coefficient magnitudes in the high frequency bands are generally too low to be seen in contrast with low frequency bands.

The wavelet transforms we will use are not unitary, but they are close, so we will rely on the approximation,

$$d(\mathcal{I}, \hat{\mathcal{I}}) \approx d(\mathcal{C}, \hat{\mathcal{C}}). \tag{6.3}$$

It will be convenient and computationally necessary to estimate the reconstructed image distortion from the reconstructed transform coefficients.

Depending upon the core 1-D wavelet transform used, the coefficients of $\mathcal{C}$ may be real or integers. The range of the wavelet coefficients may be greater than that of the original pixel values and the coefficients may be negative. We can assume that

Figure 6.3: This diagram shows the original goldhill image and the 3 level dyadic wavelet transform. The wavelet transform coefficients are on a log scale so that the structure in each band is visible.

the wavelet transform is perfectly reversible, so $\mathcal{I} = \mathcal{T}(\mathcal{T}^{-1}(\mathcal{I}))$, for any $\mathcal{I}$, though the next step, quantization, will change this.

A useful feature of the dyadic wavelet decomposition is that one can acquire lower resolution versions of the image by omitting some number of high frequency levels.

6.4 Quantization

If the transform coefficients are real values, they must be quantized. The quantizer, $\mathcal{Q}$, maps the coefficients to integer codevalues, $\mathcal{V} = \mathcal{Q}(\mathcal{C})$. The encoder converts these codevalues to the bitstream and the decoder recovers the codevalues, as well as possible, as $\hat{\mathcal{V}}$, from the bitstream. A dequantizer maps recovered codevalues to reconstructed coefficients, $\hat{\mathcal{C}} = \mathcal{Q}'(\hat{\mathcal{V}})$. The dequantizer is denoted by $\mathcal{Q}'$, and not $\mathcal{Q}^{-1}$, because $\mathcal{Q}$ does not have an inverse.

In most lossy source coding frameworks, the codevalues are recovered perfectly at the decoder, so $\mathcal{V} = \hat{\mathcal{V}}$ and the distortion is due solely to quantization. For the particular class of core image compression schemes we will be using (SPIHT and SPECK), the boundary between quantization and entropy coding is blurred. For these schemes, the quantization used is quite simple and introduces insignifi-

cant distortion. The quantization used will simply be to round each real transform coefficient to the nearest integer, and the dequantizer simply maintains the reconstructed integer, but as a real value. Distortion will be introduced at the entropy coding stage which may not perfectly reproduce the codevalues.

Because such simple quantization is used, the fact that real transform coefficients are converted to integral codevalues before entropy coding is often overlooked. We will usually consider the coefficients $\mathcal{C}$ and the codevalues $\mathcal{V}$ as equivalent, and call them both coefficients.

6.5 Entropy Coding

The codevalues $\mathcal{V}$ are converted to the bitstream by the entropy encoder $\mathcal{E}$. The entropy decoder $\mathcal{E}^{-1}$ converts the bitstream back to reconstructed codevalues, $\hat{\mathcal{V}}$. The entropy encoder is often referred to simply as the encoder, and the entropy decoder as the decoder.

The entropy encoders and decoders we will be concerned with are bitplane encoders. The codevalues $\mathcal{V}$ are represented in sign-magnitude binary form in a set of bitplanes. Working from the Most Significant Bit (MSB) to the Least Significant Bit (LSB) the encoder sends bits to the bitstream which communicate to the decoder the setting of each bit in the plane. An important feature of bitplane encoding is that the encoder can stop at any time, and approximate reconstructed codevalues can be generated at the decoder. The more bits the encoder can send to the decoder, the more accurately the codevalues will be reproduced because they will have more significant bits determined.

For some transforms, quantization that maps each coefficient to the nearest integer prevents lossless coding because $\mathcal{I} \neq \mathcal{T}^{-1}(\mathcal{Q}'(\mathcal{Q}(\mathcal{T}(\mathcal{I}))))$. If an entropy encoder and decoder perfectly reconstruct the quantized codevalues such that $\mathcal{V} = \mathcal{E}^{-1}(\mathcal{E}(\mathcal{V}))$, the only distortion is due to quantization. In this case, the coding system is referred to as *nearly lossless*.

6.6 Bitstream Properties

As mentioned above, most image codecs can operate at many rate-distortion points, depending on encoder parameters. In general, the rate is selected before encoding and each rate selection results in a completely unrelated bitstream.

Some codecs, especially bitplane codecs, have the *embedded* property which means that the rate selection can be reduced after encoding by simply truncating the bitstream. The B_0 bit bitstream $\mathcal{B}_0$ is the first B_0 bits of the B_1 bit bitstream $\mathcal{B}_1$ when $B_0 < B_1$.

There are different types of bitstream embedding, depending on what happens to the reconstructed image as more of the bitstream is received. If receiving more bits means the reconstructed image has lower distortion (increased fidelity), then the bitstream is *fidelity embedded*. Some image codecs are very finely fidelity embedded to the point where every few additional bits reduces the distortion.

If receiving more bits means that the decoder can produce a larger reconstructed image using additional higher frequency subbands, then the bitstream is *resolution embedded*. A resolution embedded bitstream has several points in the bitstream B_l, $l = L, \ldots, 0$, at which the coefficients up to the lth decomposition level are recoverable. If it is possible for the decoder to easily find and extract a non-initial, possibly non-contiguous, subset of the bitstream such that it can then reproduce a lower resolution version of the image then the bitstream is *resolution parsable*.

6.7 Set Partition Based Image Coding

After the wavelet transformation and quantization to integers it is the task of the entropy codec to convert the field of codevalues to a bitstream and back. The two core entropy codecs we will be concerned with, SPIHT and SPECK, are described in detail in the remainder of this chapter. These state of the art core coding algorithms are then built upon in later chapters. We begin by describing their many common features in this section and then proceed to details specific to SPIHT, and then SPECK.

The SPIHT (Set Partitioning In Hierarchical Trees) image codec was developed

by Said and Pearlman [16] and the SPECK (Set Partitioned Embedded Block Coder) image codec by Islam and Pearlman [17, 18]. SPIHT is based on the earlier EZW (Embedded Zerotree Wavelets) algorithm by Shapiro [15]. Both SPIHT and SPECK are fast and efficient wavelet-based codecs, produce fidelity embedded bitstreams and achieve state of the art compression performance. Munteanu, Cornelis, Van der Auwera and Cristea have developed and published an image compression technique very similar to the SPECK codec [63, 64].

We will use the term *entropy codec* to refer to the high level algorithm that maps codevalues to a bitstream. To differentiate low level symbol entropy codecs, such as Huffman or arithmetic codecs [49, 65] we call them *back-end entropy codecs*, because they can be used on symbols produced by a high level algorithm such as SPIHT or SPECK.

The terms SPIHT and SPECK are usually used to describe end to end image compression systems, including the wavelet transform and quantization. Here, we will use the terms SPIHT and SPECK to refer only to the specific algorithms that convert the codevalue field to a bitstream and back. This is the most interesting and novel part of the systems anyway. In the following sections that describe the common aspects of the SPIHT and SPECK algorithms, we will often refer to the two algorithms simply as the algorithms.

6.7.1 Bitplane Coding

The algorithms are both *bitplane codecs* that encode each bitplane in turn. Each codevalue $v(n_r, n_c)$ in $\mathcal{V}$ is stored in sign-magnitude binary format. The sign of the codevalue is stored in a single bit, and the magnitude is stored in a fixed number of bits. This is different than the more common twos complement storage format. Let b index a magnitude bit position of the codevalues, with bit $b = 0$ being the LSB. Let $v_s(n_r, n_c)$ be the sign bit of $v(n_r, n_c)$ and $v_m(n_r, n_c, b)$ be the bth magnitude bit of $v(n_r, n_c)$. The bth bits of all codevalues, $v_m(n_r, n_c, b)$, $n_r = 0, \ldots, N_r - 1$, $n_c = 0, \ldots, N_c - 1$ taken together, form a bitplane of the codevalues.

The algorithms begin by finding the index, b_{msp}, of the most significant non-zero bitplane. This number is placed on the bitstream. Bitplanes above b_{msp} are

then known by the decoder to be zero, and need not be further encoded.

Bitplane algorithms start with the most significant non-zero bitplane, $b = b_{msp}$, and work down to the least significant plane, $b = 0$. For each bitplane, the encoder places encoded bits on the bitstream that communicate to the decoder the values of all bits in the plane. It is important to do this using fewer bits than are in the bitplane itself ($N_r N_c$), otherwise there will be no data compression.

It is primarily due to the nature of bitplane coding that the algorithms are fidelity progressive. If the encoder does not encode all of the bitplanes, or if the bitstream is truncated, some number of the most significant bitplanes can still be decoded, and the image can be reconstructed to some lower, but possibly acceptable, fidelity level.

The nature of the SPIHT and SPECK algorithms offers an even greater degree of fidelity progression. As we will see in detail, if the bitstream is interrupted in the middle of bitplane b, some codevalues will be known to bit position b, and the rest to bit position $b - 1$. The fidelity of the overall image is smoothly improved during the processing for each plane.

6.7.2 Significance Field

Let $b_s(n_r, n_c)$ be the index of the most significant non-zero bit in codevalue $v(n_r, n_c)$, or -1 if $v(n_r, n_c) = 0$. This array is called the *significance field* and is conceptually important to both algorithms. Even after the decorrelating wavelet transform, on natural images, there is statistical dependence between values of the significance field. The algorithms exploit that dependence for data reduction.

Within each bitplane, each algorithm has two main stages, the *significance passes* and the *refinement pass*. Significance passes is plural because both algorithms happen to have more than one pass in that stage. The significance passes used by SPIHT and SPECK differ, but they have the same purpose.

While encoding bitplane b, a codevalue is deemed *significant* if $b_s(n_r, n_c) \geq b$. In the significance passes, the encoder communicates to the decoder which codevalues are *newly significant*. In bitplane b, these are the codevalues such that $b_s(n_r, n_c) = b$. These codevalues have a magnitude in the range $[s, 2s)$, with $s = 2^b$

known as the significance level. The decoder is already aware of which codevalues satisfy $b_s(n_r, n_c) > b$ because they have been found during the passes for previous bitplanes. In the refinement pass, the encoder communicates the value of the bth bit for each codevalue found newly significant in any previous bitplane. Thus, at the end of the significance and refinement passes for bitplane b, the decoder can determine the values of all the bits for bitplane b and for the more significant planes.

6.7.3 Set Partitioning

Both algorithms use set partitioning techniques to communicate to the decoder which codevalues are newly significant in each bitplane. Set partitioning is a divide and conquer search process. Codevalues are all initially grouped into large sets which are iteratively split, or partitioned, into smaller and smaller sets. The specific initial sets for SPIHT and SPECK are different, and are detailed in separate sections below.

If a set of codevalues contains no significant codevalues for bitplane b, the set is called an *insignificant set*. If a set contains one or more significant codevalues, the set is called a *significant set*. For each bitplane, each set is tested for significance by the encoder and the Boolean result of that test is placed on the bitstream. If the set is insignificant, nothing more needs to be done, and the set is not visited again until the next bitplane is encoded. If the set is significant, then it is partitioned into smaller child sets which are tested for this bitplane exactly as the larger parent set was. Sets that have been partitioned are split up permanently. They are not rejoined when the algorithm moves to the next bitplane. The search for newly significant codevalues in any bitplane after the first starts with sets split up from processing all previous bitplanes. The specific partitioning rules differ for SPIHT and SPECK. Again, the details are in separate sections below.

In the process to isolate newly significant codevalues, eventually the algorithm will find a significant set that holds only a single codevalue. When this happens, the sign of that codevalue is also placed on the bitstream. Once the decoder receives that information, it can approximate the codevalue to a value other than 0.

In both algorithms, the initial set grouping is fixed and known to both the

encoder and decoder. Also, the way in which significant sets are split is predetermined. The only information passed from the encoder to the decoder via the bitstream is the results of the significance tests, the codevalue sign bits, and the refinement bits. Both the encoder and the decoder have data structures to keep track of the current set groupings. The decoder needs only the significance test results to stay synchronized.

The set partitioning techniques are effective because newly significant codevalues tend to cluster. Many large sets are insignificant, so large numbers of codevalues are eliminated with a small cost in rate during the search for newly significant codevalues.

6.7.4 Reconstruction

At some point, possibly before all bitplanes are completely decoded, the decoder will reconstruct the codevalues. Before a particular codevalue has been found significant, the decoder only knows that some number of its most significant bits are zero and does not know its sign. Such codevalues are reconstructed as 0.

Once a codevalue is found significant, and possibly refined, the decoder knows all of its bits down to some bitplane b, and that its magnitude lies in $[hs, (h+1)s)$, for some positive integer h. The decoder then reconstructs the codevalue as $(h + 1/2)s$, choosing the midpoint of the quantization bin.

If the distribution of the codevalues is known, or assumed, then better estimates of the codevalue can be made given the quantization bin. Generally, this does not improve the reconstructed image much, so midpoint reconstruction is commonly used. Another practical technique is to reconstruct the codevalue as $(h + 3/8)s$. Using 3/8 instead of 1/2 moves the reconstructed codevalue closer to 0, which is justified by codevalue distributions that decay as the magnitude increases. The value 3/8 is well suited to the integer codevalues because its binary representation is simply 0.011_b.

To fully define the SPIHT and SPECK algorithms, all that remains is to specify the initial sets, the partitioning rules and the set test order. These details are in the following two sections.

6.8 SPIHT Set Partitioning

The SPIHT algorithm groups codevalues into sets based on hierarchical spatial trees in the wavelet decomposition. Spatial trees are formed using a parent child relationship extending across levels of the decomposition. Each codevalue, except those in the three highest frequency bands, and some of those in the DC band (see below), has four children. The children are in the 2 by 2 block of codevalues in the next higher frequency band at the same orientation and the same spatial location. Assuming that the number of image rows and columns are both even multiples of 2^{L+1}, where L is the number of decomposition levels, and the wavelet decomposition is organized in the ordinary manner, a codevalue at coordinates (n_r, n_c) has four children at coordinates $(2n_r, 2n_c)$, $(2n_r, 2n_c+1)$, $(2n_r+1, 2n_c)$ and $(2n_r+1, 2n_c+1)$. Figure 6.4 depicts several examples of this relationship. The codevalue labeled R1 is the parent of the four codevalues labeled C1; R2 is the parent codevalue of those labeled C2; and so on. To see why this structure is used, note the visible magnitude correlation in the wavelet decomposition in Fig. 6.3.

The DC band of the wavelet decomposition is an exception to some of these statements. For a codevalue in the DC band, the four children are not at the same orientation as the DC band and may not represent the same spatial location. This is of no consequence because, while locations in the DC band are used to denote roots of hierarchical trees, they are never grouped in the same set as their children.

The original SPIHT algorithm [16] used different DC codevalues to designate the roots of the full size hierarchical trees. The differences between that system and the one used here are purely notational and have no effect on the output of the algorithm. In the original algorithm, the DC band is divided into 2 by 2 blocks. Within each 2 by 2 block, all codevalues except for the upper left are designated as the root of a hierarchical tree. The upper right codevalue is the root of the hierarchical tree in the LH bands, the lower left codevalue is the root for the tree in the HL bands and the lower right codevalue is the root for the tree in the HH bands.

In the equivalent algorithm presented here, the codevalues in the upper left quadrant of the whole DC band do not represent the roots of any tree. The codeval-

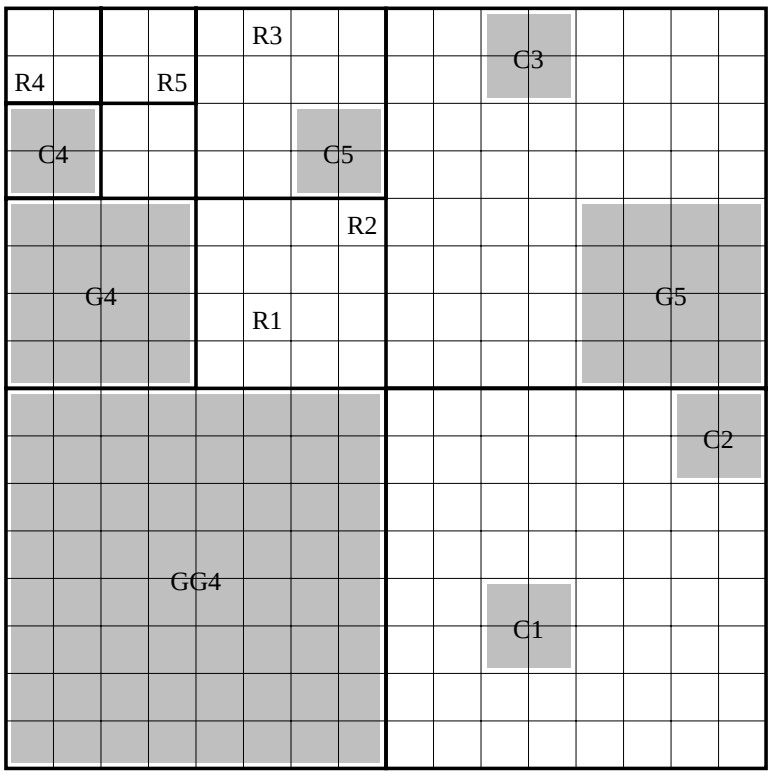

Figure 6.4: This three level dyadic wavelet decomposition shows spatial tree parent child relationships and examples of spatial trees. Thick lines show boundaries of the bands. Individual codevalues are separated by thin lines.

ues in all other quadrants of the DC band are designated as roots of the hierarchical tree following the parent-child relationship defined above and depicted in Fig. 6.4. This change slightly simplifies the algorithm because the equations defining the parent-child relationship are now the same for all bands. There is no special exception for the DC band.

Spatial trees are formed by recursively applying the parent-child relationship to build descendant trees. In Fig. 6.4, the codevalue labeled R4 has four children labeled C4. The four children labeled C4 collectively have 16 children, which are labeled G4 because they are the grandchildren of R4. The 64 great-grandchildren are labeled GG4. All of the codevalues labeled R4, C4, G4 and GG4, together form a spatial tree.

The SPIHT algorithm, at both the encoder and decoder, groups insignificant

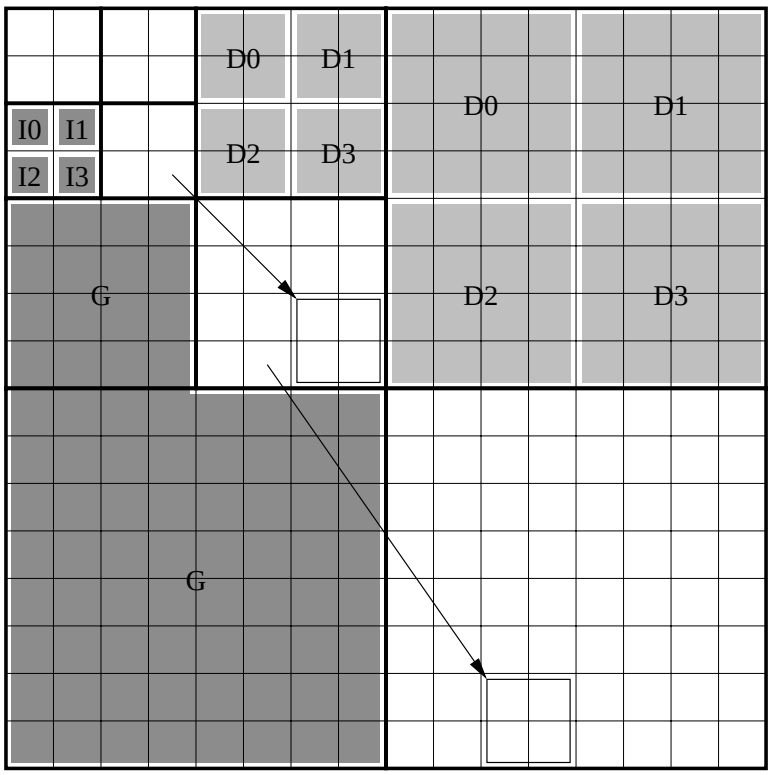

Figure 6.5: In this diagram, the dark gray pixels are all in a descendant set. When significant, this set is divided into the sets labeled I0, I1, I2, I3 and G. The light gray pixels are all in a granddescendant set. This set is divided into the four sets labeled D0, D1, D2 and D3. Also shown are two examples of the parent-child spatial relationship across subbands. In each case the arrow points from the parent pixel to the block of four child pixels.

codevalues into three types of sets: *descendant sets* (type A), *granddescendant sets* (type B), and *individual sets* (insignificant pixels). The names originally given to the set types are listed in parenthesis [16]. For notational consistency we will consider individual insignificant codevalues to be the sole member of an individual insignificant set. In Fig. 6.4, the descendant set rooted at R4 contains the codevalues labeled C4, G4 and GG4. All of the descendants of R4, but not R4 itself, are in that set. The granddescendant set rooted at R4 contains the codevalues labeled G4 and GG4. All of the granddescendants of R4 are in the set. An individual set contains a single codevalue.

A descendant set rooted at R that is found to be significant is partitioned into five new sets: the granddescendant set rooted at R and the four children of R, in four individual sets. A granddescendant set rooted at R is partitioned into four sets. The four new sets are four descendant sets, each one rooted at one of the children of R. A diagram showing these partitioning rules is in Fig. 6.5. If a descendant set is rooted in the second highest frequency band, its root codevalue has descendants, but no granddescendants. When such a set is found to be significant, it is partitioned into four individual sets. The granddescendant set is omitted because it would be empty. The descendant set rooted at R2 in Fig. 6.4 is an example of this situation.

The SPIHT algorithm begins with an initial set grouping. At initialization, in both the encoder and decoder, each codevalue in the DC band is placed in its own individual set. The rest of the codevalues are members of descendant sets. For each codevalue in the DC band, except for the upper left quadrant, a descendant set is formed with that codevalue as the root. Thus, at initialization, each codevalue is represented in exactly one set. Even as the algorithm progresses, this is invariant. Each codevalue will always be a member of one and only one set. Note that the decoder, just like the encoder, organizes the codevalues into sets. However, at the decoder, the codevalues are initially unknown, and during the progressive update will have only some of their bits known.

SPIHT uses three list data structures at both the encoder and decoder to keep track of these sets. The *list of insignificant pixels* (LIP) holds insignificant individual sets. The *list of insignificant sets* (LIS) holds insignificant descendant and granddescendant sets. The *list of significant pixels* (LSP) holds individual sets containing a single codevalue that is significant. These are the codevalues that will be refined after the significance passes find the newly significant codevalues in each plane. The names of these lists were defined in [16], which used the term pixel throughout. We have been referring to the pixels as codevalues to distinguish the quantized transform coefficients from the image data.

The LSP is actually two separate lists, the LSP and the New LSP (NLSP). The NLSP holds codevalues found significant in the current bitplane, b. These do not need to be refined for the current bitplane, since their bth bit is known to be 1.

After the refinement of the codevalues in the LSP and the conclusion of processing for a bitplane, the NLSP is appended to the LSP (emptying the NLSP), so that the codevalues that were on the NLSP will be refined in the next bitplane.

Each element of the LIP, LSP and NLSP lists is simply a pair of integers, the coordinates of the single codevalue in the set. Each element on these lists is called an individual set. As mentioned above, we will consider these lists to hold sets, with each set containing a single codevalue. Each element of the LIS represents a larger set, either a descendant set or a granddescendant set. Elements of the LIS are a pair of integers indicating the coordinates of the root of the set and another bit indicating whether the set is a descendant or granddescendant set.

At initialization, all of the initial individual sets are put on the LIP, all of the initial descendant sets are put on the LIS, and the LSP and NLSP are empty. The particular initial order of the sets on the lists in unimportant so long as it is consistent between the encoder and the decoder.

The SPIHT algorithm defines a particular order in which the lists of insignificant sets are tested and defines how newly created sets are stored on the lists. This process is defined by the following pseudo-code given for the both the encoder and the decoder. The differences between the encoder and decoder algorithms are minor. At the decoder, the bitstream bits are input instead of output, significance tests are not performed, and appropriate magnitude and sign bits are set when individual codevalues are found significant or refined. To be concise, the encoder and decoder can be, and often are, described by the same pseudo-code. In actual software implementations it is possible, and beneficial, to use the same code-base for both the encoder and the decoder. Here, however, to make the encoder and decoder algorithms more clear, they are defined by pseudo-code separately.

In each of the pseudo-code functions below, the boolean variable d represents whether a set is significant. In the encoder functions, set significance is determined from the known codevalues and placed on the bitstream. In the decoder functions, d is simply read from the bitstream.

After the header, the only data placed on the bitstream is the significance test bits and sign bits. The lists are not explicitly transmitted, but the decoder recreates

the lists of the encoder, guided by the significance test bits received.

The algorithm moves from bitplane to bitplane. For each plane there are two passes to find newly significant codevalues, and then a refinement pass. The encoder and decoder versions of the top level algorithm follow.

```
define encode_SPIHT
  write_header
  for b = b_ms, ..., 0                iterate from most to least significant plane
    encode_process_lip                find newly significant codevalues on LIP
    encode_process_lis                find newly significant codevalues on LIS
    encode_process_lsp                                refine codevalues on LSP

define decode_SPIHT
  read_header
  for b = b_ms, ..., 0                iterate from most to least significant plane
    decode_process_lip                find newly significant codevalues on LIP
    decode_process_lis                find newly significant codevalues on LIS
    decode_process_lsp                                refine codevalues on LSP
```

The encoder and decoder `process_lip` functions check each set on the LIP to see whether its lone member codevalue is newly significant. Newly significant codevalues have their signs transmitted and received and are moved to the NLSP. When `process_lip` is finished, there are no significant sets on the LIP. The encoder and decoder versions of `process_lip` are below.

```
define encode_process_lip
  for each S on LIP                                   iterate over each set on LIP
    d = (S is significant)        find whether the codevalue in the set is significant
    output(d)                                transmit significance test result, 0 or 1
    if d ≠ 0                                                  if the set is significant
      output(sign bit of codevalue in S)                                  send the sign
```

```
        remove S from LIP                          move the set from LIP to NLSP
        append S to NLSP

define decode_process_lip
   for each S on LIP                                iterate over each set on LIP
      input(d)                                  receive significance test result, 0 or 1
      if d ≠ 0                                          if the set is significant
         input(sign bit of codevalue in S)                 receive and set the sign
         remove S from LIP                          move the set from LIP to NLSP
         append S to NLSP
```

The `process_lis` function searches for significant sets on the LIS and partitions them when found. It must distinguish between descendant and granddescendant sets. When `process_lis` is finished, there are no significant sets on the LIS. In two places in the `process_lis` function, newly created sets are appended to the LIS, causing it to grow. The appended sets are tested later, but before the function completes for the current bitplane. The encoder and decoder versions of `process_lis` follow.

```
define encode_process_lis
   for each S on LIS                                iterate over each set on LIS
      (descendant and granddescendant sets are handled differently)
      if S is a descendant set
         d = (S is significant)                   find whether the set is significant
         output(d)                             transmit significance test result, 0 or 1
         if d ≠ 0                                           if the set is significant
            remove S from LIS
            partition S into 4 individual and 1 granddescendant sets
            (iterate over the newly created individual sets)
            for each child set S_c
               (newly created individual sets are processed immediately)
```

```
            encode_process_child(S_c)
         (the new granddescendant will be checked later)
         append the granddescendant set to LIS
      else                                   the set must be a granddescendant set
         d = (S is significant)                 find whether the set is significant
         output(d)                              transmit significance test result, 0 or 1
         if d ≠ 0                                                  if the set is significant
            remove S from LIS
            partition S into 4 descendant sets
            (the new descendants will be checked later)
            append each descendant set on LIS

define decode_process_lis
   for each S on LIS                                   iterate over each set on LIS
      (descendant and granddescendant sets are handled differently)
      if S is a descendant set
         input(d)                                 receive significance test result, 0 or 1
         if d ≠ 0                                                  if the set is significant
            remove S from LIS
            partition S into 4 individual and 1 granddescendant sets
            (iterate over the newly created individual sets)
            for each child set S_c
               (newly created individual sets are processed immediately)
               decode_process_child(S_c)
         (the new granddescendant will be checked later)
         append the granddescendant set to LIS
      else                                   the set must be a granddescendant set
         input(d)                                 receive significance test result, 0 or 1
         if d ≠ 0                                                  if the set is significant
            remove S from LIS
            partition S into 4 descendant sets
```

```
        (the new descendants will be checked later)
        append each descendant set on LIS
```

The `process_child` function is a helper for `process_lis` to handle newly created individual sets. These sets each contain a single codevalue. Below are the encoder and decoder versions.

```
define encode_process_child(S)
   d = (S is significant)          find whether the codevalue in the set is significant
   output(d)                       transmit significance test result, 0 or 1
   if d ≠ 0                        if the set is significant
      output(sign of codevalue in S)   send the sign of the codevalue
      append S to NLSP
   else
      append S to LIP

define decode_process_child(S)
   input(d)                        receive significance test result, 0 or 1
   if d ≠ 0                        if the set is significant
      input(sign of codevalue in S)   receive and set the sign
      append S to NLSP
   else
      append S to LIP
```

The `process_lsp` function handles the refinement pass. Each codevalue found significant in a previous plane is updated by the transmission of its bit from the current plane. The encoder and decoder versions follow

```
define encode_process_lsp
   for each S on LSP
      output(bth bit of the codevalue in S)   send the refinement bit
```

```
    append NLSP to LSP

define decode_process_lsp
    for each S on LSP
        input(bth bit of the codevalue in S)        receive and set the refinement bit
    append NLSP to LSP
```

6.9 SPECK Set Partitioning

The original SPECK algorithm was designed to operate on a full wavelet decomposition. In this work, SPECK will only be applied to subband blocks, which are independent blocks of codevalues within a single band of the wavelet decomposition. SPECK does not use spatial tree structured sets. A portion of the original algorithm is no longer needed for this case, and is not described here. The portion of the original SPECK algorithm that is no longer needed is that which deals with the L-shaped I set that extends over several bands.

There are K different types of insignificant sets used, where K can be selected to suit the application, and is usually around 5 or 6. The kth type of set is simply a 2^k by 2^k block, where $k = 0, \ldots, K-1$.

If a 2^k by 2^k insignificant set is found newly significant and $k > 0$, it is split into four 2^{k-1} by 2^{k-1} sets. This is called quad-tree splitting. Figure 6.6 shows several examples of this set splitting scheme. If a 1 by 1 set is found significant, no splitting is needed because a single newly significant codevalue has been found. The SWEET [66] image compression system uses a similar quad-tree splitting scheme.

To keep track of the current partitioning state, which sets have been tested and which codevalues have been refined, both the encoder and the decoder maintain K lists of insignificant sets, $\mathsf{LIS}(0)$, $\mathsf{LIS}(1)$, ..., $\mathsf{LIS}(K-1)$, and one list of significant codevalues, LSP. For each k, $\mathsf{LIS}(k)$ holds the insignificant 2^k by 2^k sets. Each element of a list is simply a row-column pair indicating the row-column index of the upper left corner codevalue in the set.

The subband block must be partitioned into initial sets. Each codevalue will

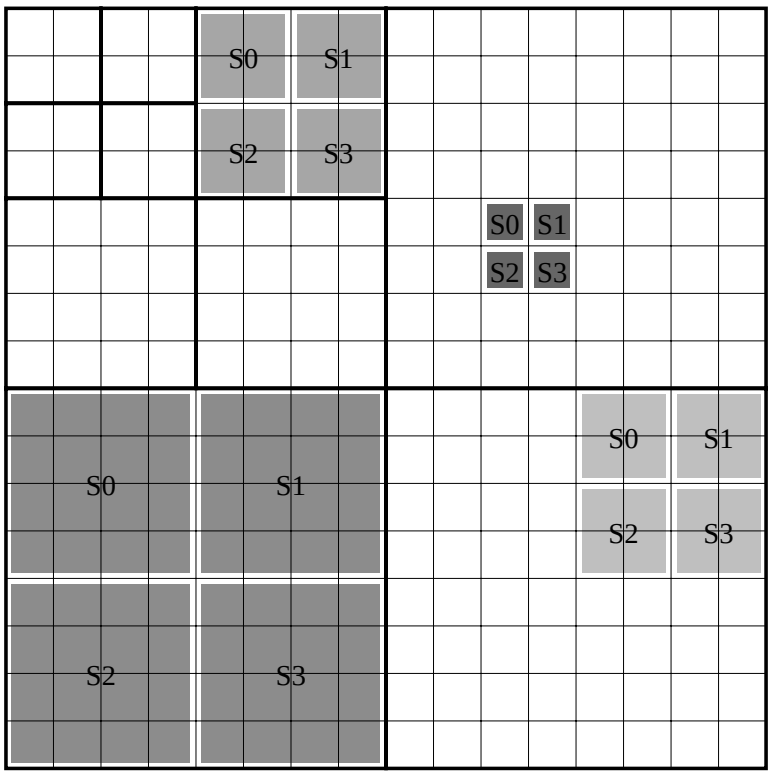

Figure 6.6: This diagram shows four separate examples of the quad-tree set splitting scheme used by the SPECK algorithm. A significant set is split into four new sets. In each case, the four new sets are labeled S0, S1, S2 and S3.

be in exactly one initial set. In the simplest case, the subband block to be encoded is 2^{K-1} by 2^{K-1}, so a set fits evenly. The initial set is then simply a single 2^{K-1} by 2^{K-1} set, and is placed on the $\mathsf{LIS}(K-1)$ list.

In general, whatever value is selected for K may not be large enough that the largest set size covers a subband block. Also, for a subband block of an unusual size, no set may cover it evenly. Such subband blocks frequently occur at the right and bottom edges of a subband. An algorithm is used to cover any size subband block with sets of different sizes, using as few sets as possible. First, as many non-overlapping adjacent 2^{K-1} by 2^{K-1} initial set blocks as possible are fit into the subband block working from the upper left of the subband block. Each of these sets is placed on the $\mathsf{LIS}(K-1)$ list. Then, as many 2^{K-2} by 2^{K-2} initial set blocks as possible are fit into the uncovered region at the right and bottom of the subband

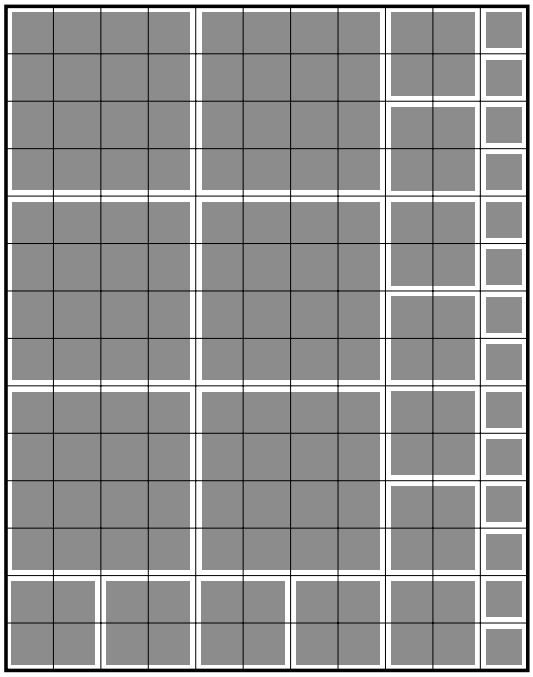

Figure 6.7: This figure shows the SPECK set initialization for a subband block that has 14 rows and 11 columns. A block of this size might occur in the lower right corner of a subband that has an unusual size. The largest set size is selected to be 4 by 4 ($K = 2$), which is small, but used for demonstration. At initialization, the subband block is covered by 6 sets that are 4 by 4, 11 sets that are 2 by 2 and 14 sets that are 1 by 1.

block. Each of these sets is placed on the $\mathsf{LIS}(K-2)$ list. This process continues until the entire subband block is covered with initial set blocks. Figure 6.7 gives an example of the result of this process.

At all times, each codevalue is represented in exactly one set on one of the lists, unless it is in a set currently being processed. As with SPIHT, this invariant is maintained.

Pseudo-code for the encoder significance passes is below. The decoder follows the same basic routine and is not given explicitly. The most important difference is that at points where the encoder outputs a bit to the bitstream, the decoder inputs a bit from the bitstream. Also, when a significant 1 by 1 set is found, the decoder sets the bth bit and then reads and sets the sign bit. This code defines the order in which the sets are tested for significance. Some changes can be made to make the

algorithm more efficient. This pseudo-code was written for ease of understanding. The top level algorithm is below.

```
define SPECK
  for b = b_ms, ..., 0                    iterate from most to least significant plane
    for k = 0, ..., K − 1                 loop over set type, small to large
      for each S on LIS(k)                iterate over each set on the list
        check_set(S, k, k)
    refine
```

The function `check_set`, which does most of the work, is separated from the top level algorithm because it calls itself recursively.

```
define check_set(S, k, k_0)               k_0 tells which LIS holds the original set
  d = (S is significant)                  find whether the set is significant
  output(d)                               transmit 0 or 1
  if d ≠ 0                                if the set is significant
    if k = k_0                            if this is the original set
      remove S from LIS(k)
    if k = 0                              if this is an individual set
      output(sign bit of codevalue in S)  send sign of the codevalue
      append S to NLSP
    else
      partition S into 4 new sets
      for each new set S_c
        check_set(S_c, k − 1, k_0)
```

In each plane, the 1 by 1 sets (single codevalue sets) on LIS(0) are tested first, then the sets on LIS(1), and up to LIS($K-1$). When a 2^k by 2^k set on LIS(k) is found to be significant, it is split into four 2^{k-1} by 2^{k-1} child sets. It is important to note that these sets are not simply placed, untested, on the LIS($k-1$) list.

They are immediately tested for significance and resplit as needed by the `check_set` function. When the algorithm is processing $\mathsf{LIS}(k)$, only insignificant sets are placed on $\mathsf{LIS}(k-1)$, ..., $\mathsf{LIS}(0)$.

It would be easier to describe and implement this algorithm if the LIS sets were processed in the reverse order, $\mathsf{LIS}(K-1)$, $\mathsf{LIS}(K-2)$, ..., $\mathsf{LIS}(0)$. Then, when a significant set is found on $\mathsf{LIS}(k)$, its four children could be append to the $\mathsf{LIS}(k-1)$ list and checked again after k is decremented. There would be no need for the recursion of the `check_set` function. However, this change would hurt the performance of the SPECK algorithm because it delays all opportunities to find newly significant codevalues. It is better to check the smaller insignificant sets first, because they provide an opportunity to improve the decoded image more quickly.

Pseudo-code for the refinement pass is identical to that of the SPIHT algorithm. The bth bit of each codevalue found significant in a previous plane is transmitted. For the codevalues found significant in this plane, the bth bit is not transmitted because it is known to be 1. This is why the temporary list NLSP is used to hold the codevalues found significant in the current plane.

```
define refine
   for each S on LSP
      output(bth bit of the codevalue in S)
   append NLSP to LSP
```

CHAPTER 7
Spatial Block SPIHT

This chapter introduces a low-memory cache efficient image compression system based on SPIHT, called Spatial Block SPIHT (SB-SPIHT) [67]. The same overall approach will be applied to develop a low-memory cache efficient compression system based on SPECK in Chapter 8 and then a generalized combination SPIHT and SPECK compression system in Chapter 9.

7.1 Motivation

The SPIHT algorithm is a fast and efficient technique for image compression [16]. Like EZW [15] and other embedded wavelet compression schemes [68, 69], SPIHT generally operates on an entire image at once. The whole image is loaded and transformed, and then the algorithm requires repeated access to all coefficient values. In order to efficiently code the image, the SPIHT algorithm accesses the coefficient values following a particular complex pattern as it searches for significant coefficients over the whole image.

The unstructured coefficient access requirement of the SPIHT algorithm hinders its use in certain memory constrained environments. Consider a processing architecture with two memory segments, a small fast memory and large slow memory. Depending on the application framework, these segments may be considered the on-chip cache, and external RAM of a microprocessor. In a virtual memory system, the segments would be external RAM and a hard disk swap space. Using full-image SPIHT in such a framework without sufficient fast memory for the image can cause time consuming memory swapping as data that must be accessed will frequently lie in the slower memory segment. It is more desirable to use a compression algorithm that can work with a portion of data that fits in the small fast memory, and only occasionally exchanges data between the memory segments.

Many cache systems, such as microprocessor caches and virtual memory systems are automatic. That is, the microprocessor hardware or the operating system

keeps track of memory accesses, anticipates what stored data will be required next, and automatically moves data between the cache and the slow memory segment. For image compression systems, such as those presented here, it is more likely that the compression system will manually move data between the cache and slow memory segment. The image compression system knows exactly what data segments need to be moved and can do this more efficiently than an external adaptive system.

The capability to encode a large image without storing the entire image in memory is an important feature in the JPEG 2000 requirements specification. The popular *bike*, *càfe* and *woman* JPEG 2000 test images are each 2560 by 2048 pixels and others are much larger. Since most wavelet based image compression techniques use 4 or 8 bytes (single or double precision floating point) of memory per pixel while computing the transform, a 5M pixel image can be challenging on most implementation platforms. The system described here has been proposed to the JPEG 2000 standardization committee. The encoder used to generate results for this work is implemented as a module in the JPEG 2000 verification model test software.

Simple tiling applied before a transform is another technique that could be used to apply SPIHT coding to large images with limited memory. Tiling applied before the transform tends to result in block artifacts at low rates. The Spatial Block SPIHT technique presented here has no block artifacts. An additional cost that must be endured by the SB-SPIHT scheme is memory required by the line-based wavelet transform engine [19, 20].

7.1.1 Prior Work

Rogers and Cosman [21] have also developed a codec in which EZW or SPIHT encoded bits for a given spatial tree are kept together in a packet. They effectively use very small spatial blocks and store the coded bits for as many spatial blocks as possible in each fixed size packet. Their goal is graceful degradation against packet loss.

Creusere has presented an adaptation of the EZW algorithm for parallel implementation [22]. The system is set up so that each processor in a Multiple Instruction Multiple Data (MIMD) array produces a spatial block in the transform domain and

then applies the EZW algorithm. Creusere find that computing the mean of the whole DC band and removing it results in better compression performance than carrying on this operation for each spatial block independently. It is showed that this reduces blocking at low bit rates as well. A global embedded bitstream can be generated by interleaving bits, but this corresponds to a a sub-optimal bit allocation. An alternative algorithm is presented that interleaves non-entropy coded EZW symbols so that they are ordered as they would be if generated by a full-image EZW encoder. Adaptive arithmetic coding is then applied to these symbols by a post-processor.

Lossless EZW has also been adapted for region of interest decoding [23]. A new symbol, *exact*, is added for transmitting the refinement pass. Small spatial blocks are EZW encoded independently. The final bitstream is constructed by concatenating the sub-bitstreams, preceded by a table of their lengths. The primary goal of this work is a system in which a low-bandwidth user can select a region of the image for refinement and to experiment with alternate refinement symbol encoding methods.

Creusere also presents an image compression algorithm somewhat similar to the SB-SPIHT algorithm described here [24]. Creusere chose the EZW algorithm for his system, and the motivation is error resilience, not memory efficiency. Subband coefficients are partitioned into spatial blocks. Each spatial block is encoded with EZW independently. This work, however, does not address the fact that not all sub-bitstreams should be the same length. For 32 by 32 spatial blocks he gets a PSNR loss greater than 3 dB with no channel errors on the *lena* and *barbara* images.

Independent spatial block coding is used for a parallel VLSI design in [70, 71]. The emphasis of this work is reduced processor memory achieved through independent spatial block coding with the SPIHT algorithm while still generating a global bitstream that is progressive in fidelity across the whole image.

7.2 Overview

To efficiently apply the SPIHT algorithm in two level memory environments, we have modified the overall algorithm so that the conversion of the subband coefficients to an encoded bitstream is done only for a small portion of the coefficients

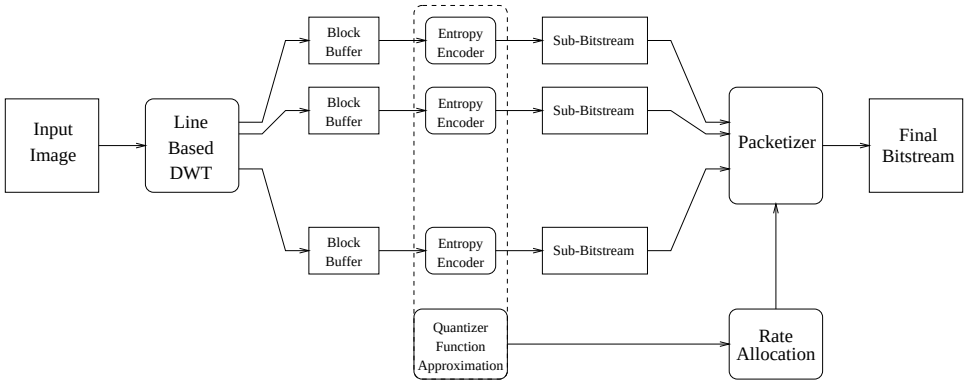

Figure 7.1: This diagram depicts the overall structure of the Spatial Block SPIHT encoder.

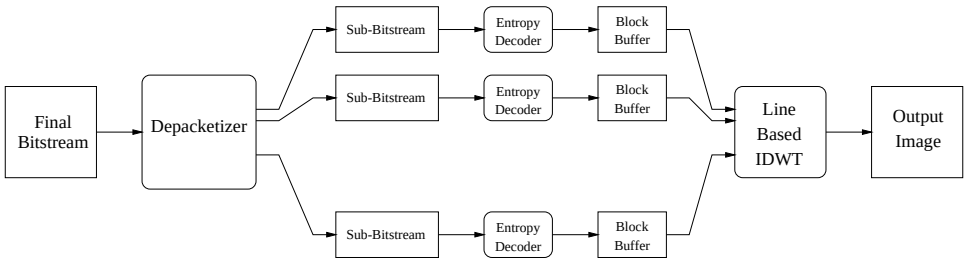

Figure 7.2: This diagram depicts the overall structure of the Spatial Block SPIHT decoder.

at a given time. Thus, only a small portion of the coefficients must be stored in fast cache memory.

The overall encoding system is diagrammed in Fig. 7.1, and the decoding system is in Fig. 7.2. Instead of placing quantized coefficients in one full-size subband decomposition, they are placed in many small spatial blocks. The partitioning is done in such a way that each hierarchical coefficient tree in the full-size subband decomposition appears in one of the spatial blocks.

The basic SPIHT algorithm is applied to each spatial block independently to produce a fidelity embedded sub-bitstream for each block. These independent embedded sub-bitstreams are then assembled to make a final bitstream that is either fixed rate or fidelity embedded. If it is fidelity embedded, it may be coarsely or finely embedded, depending on parameters chosen at the encoder.

The EZW [15] image coding algorithm could be used in place of SPIHT in

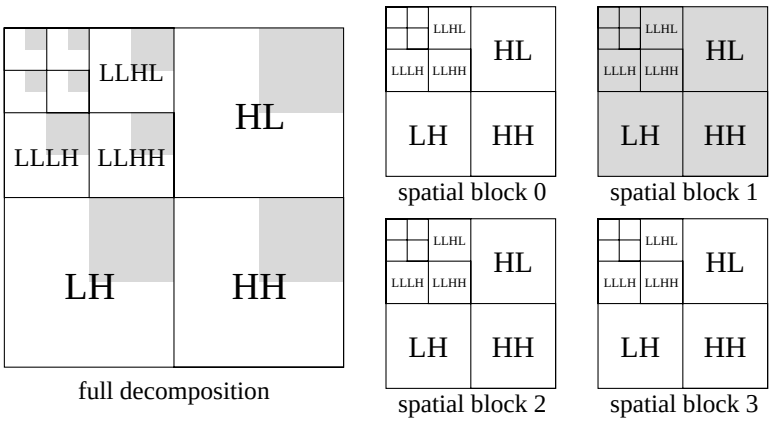

Figure 7.3: A full subband decomposition partitioned into 4 equal sized spatial blocks is depicted. The coefficients in the gray regions are all in spatial block 1.

the system presented here. EZW is also a bitplane codec with significance and refinement passes.

7.3 Spatial Blocks

To apply the SPIHT algorithm, or any tree-based codec, to small portions of the subband decomposition at a time, the decomposition coefficients $\mathcal{C}$ are partitioned into P spatial blocks, $\mathcal{C}_p$, $p = 0, \ldots, P-1$, as shown in Fig. 7.3. A spatial block is a subset of the coefficients consisting of one or more hierarchical trees. The number of trees is chosen to meet a desired block size, typically 64 by 64 or 128 by 128 coefficients. The trees in a spatial block are rearranged in a smaller 2-D array such that the usual hierarchical relationship between the coefficients is preserved. Due to the nature of the subband decomposition, each spatial block holds the coefficients needed to reconstruct a contiguous block of the original image, neglecting overlap stemming from the width of the wavelet filters.

7.4 Subband Transform

The first step in the SPIHT image compression algorithm is the dyadic subband transform. This transform is usually computed for the entire image at once. For our application it is important that the transform instead be implemented with

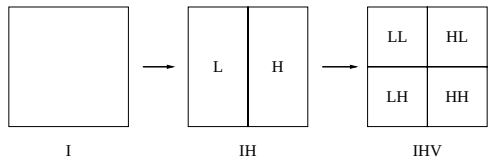

Figure 7.4: Computation stages for one level of a subband decomposition.

a rolling, or line-based, algorithm. Just as 1-D FIR filters can be implemented to produce filtered data after a lag in the time domain, the 2-D separable filters used for the subband decomposition can be implemented to produce the coefficients filling a spatial block after a lag in the spatial domain. With spatial blocks processed as they become available, and then released, the codec requires a fraction of the memory that would be needed to hold an entire image or subband transform.

A practical means to accomplish this is with the line-based transform described by Chrysafis and Ortega [19, 20]. Initially, consider just the first level of decomposition, and its two processing stages. The original image I is transformed horizontally to generate the IH image, and then vertically to generate the IHV image, seen in Fig. 7.4. IH and IHV are new terms representing intermediate data and not final subband coefficients.

After each row of I is moved into memory, wavelet filters are immediately applied to produce the corresponding row of IH . The rows of IHV depend on several adjacent rows of IH, with the exact number depending on the filter length, so the rows of IH are kept in a rolling buffer that moves down the image. As soon as the rolling buffer holds enough rows of IH a vertical wavelet filter is applied to generate a row of IHV . The rolling buffer releases memory for rows which will no longer be needed and thus uses memory proportional to the width of the image, but not the height. This procedure produces the exact same coefficient values that would be produced by a full-image transform. All that is different is the order in which the computations are performed.

To produce a dyadic transform, the above technique is applied recursively to the low-pass band. As soon as a row of the low-pass band is produced, it is passed to an independent transform engine operating on its level.

Completed transform coefficients generated by the line-based DWT engine are

moved to a series of P spatial block buffers, each of which holds the coefficients for one of the spatial blocks. For each block buffer, memory is allocated only when it is needed. A block buffer fills up as it receives its coefficients from the line-based transform engine. As soon as the block buffer for spatial block $\mathcal{C}_p$ is full, it is encoded using the SPIHT algorithm which generates the sub-bitstream $\mathcal{B}_p$ for that spatial block. The buffer memory used for coefficients of encoded spatial blocks is no longer needed and immediately released.

By the nature of the line-based technique, blocks corresponding to the top of the image will be ready for encoding first. They will be coded and released before any coefficients for lower blocks are generated. With only a portion of the block buffers allocated at any time, the memory requirements of the system are greatly reduced. On a MIMD multi-processor architecture, several of the spatial blocks can be processed by the SPIHT codec simultaneously. The SPIHT algorithm does not readily scale to a Single Instruction Multiple Data (SIMD) architecture.

7.5 SPIHT Encoding

Each spatial block of subband coefficients is moved to fast cache memory and processed by the SPIHT encoder once, as soon at it is available from the rolling wavelet transform engine. The encoder generates an independent, embedded in fidelity, sub-bitstream for each spatial block. These sub-bitstreams are stored in external slow memory until all spatial blocks have been encoded.

The SPIHT codec used for the spatial blocks has only a few minor differences from a full-image SPIHT codec. Spatial blocks typically have few DC coefficients. It is not effective to separate and transmit the DC mean, so this is not done. Instead, 128 is subtracted from each pixel value at the encoder before the wavelet transform, and added after the inverse transform at the decoder. Also, as the encoder processes, it records information that will be needed for rate allocation among the sub-bitstreams.

A full-image SPIHT encoder simply processes and outputs bits until the full-image bit budget is met. Since each spatial block is processed only once, in turn, by the encoder and at the time of processing it is not known what rate will later be

assigned to a particular block, over-coding, coding to a higher bit rate, is performed. Each block is encoded to 2.5 or 3 times the desired overall rate in bits per pixel. Unneeded portions of the sub-bitstreams will be pruned after rate allocation. An alternative system could avoid the over-coding at the expense of repeatedly swapping spatial block data into cache memory.

7.6 Rate Constraint

In this section the rate allocation problem is posed to motivate the need for an operational rate distortion characteristic, or quantizer function, for each spatial block. Then, the following sections detail how the quantizer functions are acquired and how they are used by the rate allocation procedure.

Given a total rate budget, the SB-SPIHT encoder allocates bits to each sub-bitstream such that the MSE between the original and reconstructed images is minimized. Let the total rate budget be B_t bits and let B_p, the elements of vector $\underline{B}$, be the number of bits assigned to spatial block p, $p = 0, \ldots, P-1$. The equations below do not explicitly show the dependence of $\hat{\mathcal{I}}$ and $\hat{\mathcal{C}}$ on $\underline{B}$ or the dependence of $\hat{\mathcal{C}}_p$ on B_p. The allocation problem may be stated,

$$\underline{B}^* = \underset{\underline{B},\, \sum B_p \leq B_t}{\operatorname{argmin}} \; d(\mathcal{I}, \hat{\mathcal{I}}). \tag{7.1}$$

This means we wish to allocate the bits across the spatial blocks such that the overall distortion of the reconstructed image is minimized. The wavelet transforms which we will use are biorthogonal, and thus not unitary. However, they are nearly unitary, so we have $d(\mathcal{I}, \hat{\mathcal{I}}) \approx d(\mathcal{C}, \hat{\mathcal{C}})$, and instead can solve the problem,

$$\underline{B}^* = \underset{\underline{B},\, \sum B_p \leq B_t}{\operatorname{argmin}} \; d(\mathcal{C}, \hat{\mathcal{C}}) = \underset{\underline{B},\, \sum B_p \leq B_t}{\operatorname{argmin}} \sum_{p=0}^{P-1} d(\mathcal{C}_p, \hat{\mathcal{C}}_p). \tag{7.2}$$

Now the distortion is measured in the transform domain. Evaluating distortion in the transform domain is critical for this system. This allows us to consider the distortion contribution of each spatial block independently. Because the wavelet transform is not perfectly unitary, this is an approximation.

Let $D_p(B_p)$ be the MSE distortion, $d(\mathcal{C}_p, \hat{\mathcal{C}}_p)$, in spatial block p when B_p bits are received for that spatial block. In this context, $D_p(B_p)$ is traditionally called a *quantizer function*. To solve the rate allocation problem we will need to know the slope of the quantizer function $D_p(B_p)$ for each spatial block [72, 73]. Each sub-bitstream $\mathcal{B}_p$ will be truncated to B_p bits which determines how accurately the transform coefficients for the spatial block $\mathcal{C}_p$ are reproduced as $\hat{\mathcal{C}}_p$ and hence the distortion contribution of the spatial block.

7.7 Quantizer Function

This section explains, in several distinct stages, how SB-SPIHT generates the quantizer function for each spatial block. After some introductory remarks, we will define what extra information must be found and stored by SPIHT while encoding each spatial block. Then, we will evaluate what is done at the decoder when it finds a newly significant coefficient, or receives refinement information and how much this reduces the distortion. It is then explained how SB-SPIHT estimates how much the distortion is reduced for each pass of the SPIHT algorithm. Two sections explain how to properly account for the extraction of the DC subband mean and other header information in this process. Then, the procedure for forming a convex piecewise linear quantizer function is given. At the end of this section some similar techniques used by others are described and referenced. It is much easier to describe those methods after understanding the SB-SPIHT method.

The rate-distortion performance of SPIHT is data dependent. For different spatial blocks, but the same bitplane and SPIHT algorithm pass, different levels of rate-distortion efficiency may be achieved. The rate allocation scheme of SB-SPIHT accounts for this by assigning bits to spatial blocks based on the actual performance of the algorithm in that block, as given by a quantizer function.

The quantizer function is so named because it often represents rate-distortion operating points available given a choice of quantizers, or the ability to select a parameter of the quantizer, such as step size. In many other image compression systems, distortion is introduced only by the quantizer, and the tradeoff between rate and distortion is made at the quantizer. For the systems being developed here, the

quantizer is a fixed uniform scalar quantizer and introduces little distortion. Instead, distortion is controlled by the number of bits used from the fidelity embedded sub-bitstreams. In a sense, the truncation of the fidelity embedded sub-bitstream selects an embedded quantizer for a spatial block, so we continue to use the term quantizer function.

We must know the slope of the quantizer function for the rate allocation scheme. For this system, a fast technique has been developed to approximate the slope of the quantizer function for each spatial block. For each spatial block, each bitplane, and each encoder processing pass, simply count the number of newly significant coefficients revealed (or the number refined if in a refinement pass) during the pass, and the number of bits placed on the sub-bitstream during the pass. This operation adds very little processing time or storage overhead to the algorithm. Then approximate the reduction in distortion gained by decoding the pass and form a convex piecewise linear approximation to the quantizer function. Details of the method and formulas used are in the following subsections.

7.7.1 Encoder Data Collected

The first step in this process is for the SPIHT encoder to record some additional performance data as it compresses each spatial block. Recall that within each bitplane, the SPIHT algorithm has three passes, corresponding to the processing of the LIP, LIS and LSP [16]. During the LIP and LIS passes, also called sorting passes, newly significant coefficients and their signs are revealed to the decoder. During the LSP, or refinement, pass coefficients found significant for previous bitplanes have their bit from the current plane sent to the decoder.

Let $B_{\mathsf{LIP},n}$ be the number of bits are placed on the sub-bitstream during the LIP pass operating in bitplane n with significance threshold $\tau = 2^n$. $B_{\mathsf{LIP},n}$ includes all significance test bits and all sign bits. Let $N_{\mathsf{LIP},n}$ be the number of newly significant coefficients found. The encoder records $B_{\mathsf{LIP},n}$ and $N_{\mathsf{LIP},n}$ for each pass.

The LIS pass is handled in nearly the same way. For each bitplane n, the encoder records $B_{\mathsf{LIS},n}$, the number of bits placed on the sub-bitstream, and $N_{\mathsf{LIS},n}$, the number of newly significant coefficients. Even though the LIS contains only

descendant and granddescendant sets, significant coefficients are found during the LIS pass. New individual sets, created by partitioning, are tested for significance during this pass.

Even though in both the LIP and LIS passes, the encoder finds newly significant coefficients, the encoder records its performance in each pass separately. The reason for this is that SPIHT is more effective in the LIP pass than in the LIS pass. This is why the passes are done separately. In any particular bitplane, the slope of the quantizer function $D_p(B_p)$ will generally be more negative (steeper) during the LIP pass than during the LIS pass.

For the refinement pass at bitplane n, the encoder records $B_{\mathsf{REF},n}$, the number of bits placed on the sub-bitstream, and $N_{\mathsf{REF},n}$, the number of coefficients refined. Without back-end entropy coding, these numbers are equal $B_{\mathsf{REF},n} = N_{\mathsf{REF},n}$.

7.7.2 Decoder Reconstruction

In order to determine how effective the SPIHT algorithm is at reducing the distortion of a spatial block we must first examine what effect finding a newly significant coefficient and refining a coefficient has on the reconstructed coefficients. When new information, from positive significance tests or refinement, is received for a coefficient, its reconstructed value at the decoder is updated. Whether a coefficient is found newly significant, or it is being refined, let α be its actual value, let α_0 be its reconstructed value before the update, and let α_1 be its reconstructed value after the update. If a coefficient is found to be newly significant at significance level τ, then before the update it is reconstructed as $\alpha_0 = 0$, its true value α lies in $[\tau, 2\tau)$ and after the update it is reconstructed, for now, at the midpoint $\alpha_1 = 1.5\tau$. We assume that the coefficient is positive without loss of generality.

If a coefficient is to be refined at significance level τ, then it is currently set to some value α_0 and α is known to lie in $[\alpha_0 - 0.5\tau, \alpha_0 + 0.5\tau)$. The refinement bit halves the size of this region of uncertainty and specifies whether α lies in $[\alpha_0 - 0.5\tau, \alpha_0)$ or $[\alpha_0, \alpha_0 + 0.5\tau)$ and hence whether $\alpha_1 = \alpha_0 - 0.25\tau$ or $\alpha_1 = \alpha_0 + 0.25\tau$ In either case, the width of the region in which α is known to lie is reduced from τ to 0.5τ

7.7.3 Coefficient Distortion Reduction

Now that we know what happens to individual reconstructed coefficients as each spatial block is decoded, we can determine how much the distortion of a reconstructed coefficient is reduced when it is found newly significant or refined. Assume that the unknown coefficient value α is a random variable, and, conditioned on being newly significant at significance level τ, is distributed uniformly between τ and 2τ. The expected squared error in reproducing the coefficient as $\alpha_0 = 0$ is,

$$D_0 = E\{(\alpha - \alpha_0)^2\} = E\{\alpha^2\} = \int_{\tau}^{2\tau} \frac{1}{\tau}\alpha^2 d\alpha = \frac{7}{3}\tau^2. \tag{7.3}$$

Reproducing the coefficient at $\alpha_1 = 1.5\tau$ gives the expected squared error,

$$D_\tau = E\{(\alpha - \alpha_1)^2\} = E\{(\alpha - 1.5\tau)^2\} = \int_{\tau}^{2\tau} \frac{1}{\tau}(\alpha - 1.5\tau)^2 d\alpha = \frac{1}{12}\tau^2. \tag{7.4}$$

This is also the expected squared error of a coefficient refined up to significance level τ. So, discovering a newly significant coefficient reduces the expected sum squared error by,

$$D_0 - D_\tau = \frac{7}{3}\tau^2 - \frac{1}{12}\tau^2 = \frac{27}{12}\tau^2. \tag{7.5}$$

If a coefficient is refined to significance level τ, its previous squared error contribution was $D_{2\tau}$ and its new squared error is D_τ. So, the reduction in squared error by the refinement is,

$$D_{2\tau} - D_\tau = \frac{1}{12}(2\tau)^2 - \frac{1}{12}\tau^2 = \frac{1}{4}\tau^2. \tag{7.6}$$

The reduction in distortion that comes with finding a newly significant coefficient is much greater than the reduction that comes with refining a coefficient. This is somewhat mitigated by the fact that the SPIHT algorithm produces only a single bit per refined coefficient, but produces two bits for each newly significant coefficient found. For each newly significant coefficient, a positive significance test bit and a sign bit are placed on the bitstream. Also, during the search for newly significant coefficients in the LIS pass, negative significance test bits are produced and must be transmitted.

7.7.4 Pass Distortion Reduction

Now we can find out how effective the SPIHT encoder is on a larger scale. This subsection details the method to estimate the reduction in distortion that occurs at the decoder for each bitplane and each pass of the SPIHT algorithm as it is applied to each spatial block. This process commits SB-SPIHT to truncating sub-bitstreams only at the ends of passes.

The quantizer function is approximated as a piecewise linear function. Its slope is piecewise constant. Its slope over the range corresponding to the processing of the LIP pass at bitplane n is,

$$S_{\mathsf{LIP},n} = -\frac{N_{\mathsf{LIP},n}}{B_{\mathsf{LIP},n}} \frac{27}{12}\tau^2. \tag{7.7}$$

and over the range for the LIS pass, the slope is,

$$S_{\mathsf{LIS},n} = -\frac{N_{\mathsf{LIS},n}}{B_{\mathsf{LIS},n}} \frac{27}{12}\tau^2. \tag{7.8}$$

Finally, over the range for the refinement pass at bitplane n, the slope is,

$$S_{\mathsf{REF},n} = -\frac{N_{\mathsf{REF},n}}{B_{\mathsf{REF},n}} \frac{1}{4}\tau^2. \tag{7.9}$$

Each of these slopes is the expected reduction in MSE per bit for the pass.

7.7.5 DC Subband Mean Distortion Reduction

In general, SPIHT codecs compute and subtract the mean in the DC subband before encoding coefficients. The mean is transmitted to the decoder which stores it and adds it back in after decoding the coefficients. The action reduces the distortion of the reconstructed image at some expense of bits and is handled by the length-slope framework.

Because the spatial blocks are small and have few DC coefficients it is not beneficial to extract the DC mean for separate transmission for each spatial block, so it is not done. For completeness, however, the effect of mean extraction on the length-slope pairs is described here.

If i ranges over the N_{DC} DC coefficients, and v_i is the value of the ith coefficient, the distortion due to the coefficients before extracting the mean is,

$$D_0 = \sum_i v_i^2, \tag{7.10}$$

because without any other information, they are reproduced as zero. The DC mean is,

$$m = \frac{1}{N_{\mathrm{DC}}} \sum_i v_i, \tag{7.11}$$

and the distortion after the mean is extracted is,

$$D_1 = \sum_i (v_i - m)^2. \tag{7.12}$$

The reduction in distortion by subtracting the mean is,

$$D_0 - D_1 = N_{\mathrm{DC}}\, m^2. \tag{7.13}$$

If B_{mean} bits are used to transmit the mean, then the slope of the quantizer function segment for this action is,

$$S_{\mathrm{mean}} = -\frac{N_{\mathrm{DC}}}{B_{\mathrm{mean}}}\, m^2. \tag{7.14}$$

7.7.6 Header Information

Some header information must be transmitted for each spatial block, such as the index of the most significant non-zero bitplane. Header information like this does not immediately reduce the distortion of the decoded image because if only the header is decoded, the reconstructed image will be the same as if no bits were decoded. The header information can, and must, be incorporated into the quantizer function. The length, L, is the number of header bits and the slope is zero, $S_{\mathrm{header}} = 0$. The header length-slope pair will be joined with at least one other pair because its slope is zero which guarantees non-convexity.

Later, the sub-bitstreams will be packetized to form the final bitstream. The headers of these packets do not need to be accounted for in the quantizer function because their size is the same for each spatial block.

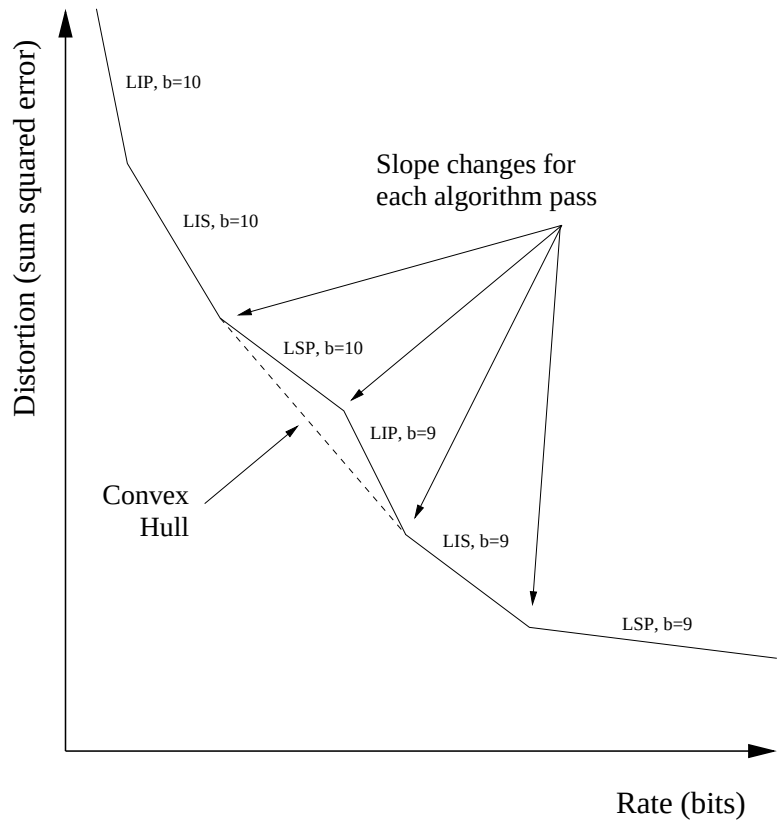

Figure 7.5: This figure depicts an example of a piecewise linear quantizer function with its convex hull. The slope is fixed over each pass, and each pass is labeled with the type of pass and bitplane.

7.7.7 Convex Quantizer Function

Now we have enough information to form a piecewise linear quantizer function for each spatial block at the encoder. This subsection defines how the function is stored. Then the process for extracting the convex hull of the quantizer function is given. Figure 7.5 gives an example of such a quantizer function with its convex hull.

The encoder stores data defining the quantizer function as an ordered sequence of *length-slope* pairs, (L_i, S_i). Here L_i is the length of a linear segment of the function in bits and S_i is the slope of the segment, from the appropriate formula above.

The length-slope pairs are processed to extract the convex hull of the quantizer function. If, for any i, $S_i > S_{i+1}$, then the i and $i+1$ segments are combined and

replaced by a new segment with slope,

$$S' = \frac{S_i L_i + S_{i+1} L_{i+1}}{L_i + L_{i+1}}, \tag{7.15}$$

and length,

$$L' = L_i + L_{i+1}. \tag{7.16}$$

A new (L', S') segment replaces the two segments (L_i, S_i) and (L_{i+1}, S_{i+1}). This conversion is accomplished with a single traversal of the sequence of length-slope pairs. The length-slope pairs are stored in a linked list data structure to enable removal of pairs from within the list.

7.7.8 Other Techniques

The EBCOT image coder [25], adopted as the basis for the JPEG 2000 standard, also partitions the wavelet coefficients into blocks for independent coding. EBCOT uses blocks within subbands instead of spatial blocks, but its rate allocation needs are the same. For the same reasons we have here, EBCOT forms a quantizer function for each block. Like SB-SPIHT, EBCOT uses the end of each processing pass in each bitplane as a candidate sub-bitstream truncation point and uses a piecewise linear quantizer function. EBCOT, however, finds the reduction in distortion in each pass explicitly based on the error between the original coefficient values and their expected reconstructed values. The method described here and used for SB-SPIHT is much less computationally complex. The procedure described here for extracting the convex hull of each quantizer function is the same as that used by EBCOT [25] and the JPEG 2000 verification model.

Caetano and da Silva also make use of a piecewise linear approximation to the rate distortion performance of a bitplane codec (EZW) in their rate control system for video [74, 75]. In their system, the image is decoded by the encoder at the bitstream points where the type of pass changes in order to determine the fidelity at that rate.

7.8 Rate Allocation

Given the bit budget and quantizer functions, we are now faced with a standard rate allocation problem which can be nearly optimally solved by iteratively increasing the number of bits taken from the sub-bitstream with the best cost-benefit ratio, that is, with the most negative rate-distortion slope. Using convex quantizer functions reduces the complexity of the algorithm greatly. The technique by Riskin [72] is followed after extracting the convex hull of the quantizer functions. This paper offers a simpler alternative to the more general application of the BFOS algorithm to optimal bit allocation [73].

The number of bits allocated to each spatial block is initialized to zero, $B_p = 0$, $p = 0, \ldots, P-1$. The number of bits left to be allocated, B_r, is initialized to the total bit budget, B_t, less the space needed for the global header and for P packet headers. The allocation is performed iteratively, in rounds, until $B_r = 0$. In each round, the head length-slope pair for each spatial block is examined. We find p^*, the index of the spatial block with the most negative slope S^*, and the corresponding length L^*. If $L^* \leq B_r$, there is room in the bit budget for all of the bits represented by this length-slope pair, so L^* is added to B_{p^*} and the initial length-slope pair for spatial blocks p^* is removed. If $L^* > B_r$, then this length-slope pair exceeds the bit budget. In this case, the initial length-slope pair is split into two pairs, each with slope S^*. The first new pair has length L^* and the second has length $L^* - B_r$. The first new pair is used as usual, and the second new pair remains on the list for possible allocation in a subsequent round.

The core SPIHT encoder produces a finely fidelity embedded bitstream for each spatial block. However, because of the way in which the sub-bitstreams will be packetized, the final bitstream will not automatically inherit this property. A fidelity embedded final bitstream will be accomplished via Q discrete rate allocation layers. Let R_q be the desired rate for layer q, where $q = 0, \ldots, Q-1$. Accounting for the global and fixed size packet headers, the rate allocation procedure is applied once for each layer. The result is $B_{p,q}$, the number of bits allocated to spatial block p for layer q. The rate allocation procedure described in the previous section was for the specific case with $Q = 1$.

EBCOT [25] and the JPEG 2000 verification model perform an iterative search for the quantizer function slope, or Lagrangian parameter, which represents the optimal bit allocation for some distortion. Both the Lagrangian technique and the technique used here result in the same allocation.

An alternative, and much simpler rate allocation policy is to allocate the same rate to each spatial block. With such a scheme, the rate assigned to each spatial block is known in advance, so over-coding is not necessary and it is not necessary to find the quantizer function. However, the appearance of the decompressed image will be less than ideal. Low and high activity regions of the image will have the same rate assignment. Low activity regions will be reconstructed to a fidelity that is too high and high activity regions will be reconstructed to a fidelity that is too low.

Our driving goal is to minimize image distortion given a fixed bit budget. An alternative to rate control, however, is distortion control. A distortion control mode could encode a predetermined set of bitplanes, up to a particular pass in the final plane, for fractional bitplane resolution. All encoded bits would then be transmitted, regardless of rate. In the context of our rate allocation and packetization scheme, this in effect sets the rate for each spatial block for the final rate layer. Additional intermediate rate layers may still be chosen using the rate allocation technique. If no intermediate layers are desired there is the added benefit that the sub-bitstreams need not be stored. As soon as they are produced, they can be placed in a packet, inserted into the final bitstream and transmitted. An important benefit of distortion control is that it eliminates the need to over-code each spatial block.

The packetization scheme that will be used allows for a great deal of flexibility in how the overall rate is allocated to each block and for more alternative rate allocation policies. If Region Of Interest (ROI) coding is needed, the spatial blocks with coefficients that the ROI depends on can have more rate allocated than other regions. All that needs to change at the encoder is the rate allocation strategy. No changes are needed at the decoder which determines how much rate was allocated to each block through the packet headers it receives.

In the next chapter, we will introduce an image coder called Subband Block

SPECK (SB-SPECK) that works by independently coding subband blocks instead of spatial blocks. Subband blocks are simply blocks of coefficients within a subband. SB-SPECK uses the same rate allocation and packetization schemes described here. If Human Visual System (HVS) weighting is desired, it is a simple matter to adjust the rate allocation strategy within SB-SPECK to allocate less rate in the high spatial frequency bands, which are generally less important visually.

7.9 Packetization

The final step is to combine the stored sub-bitstreams into a single final bitstream. The packets defined here are not necessarily meant as units for a transport mechanism, but as a way to organize data in the bitstream.

For each of the P spatial blocks we have generated a sub-bitstream $\mathcal{B}_p$, $p = 0, \ldots, P-1$. Each sub-bitstream is divided into Q segments, one for each fidelity layer. Segment q from $\mathcal{B}_p$ consists of bits $B_{p,q-1}+1$ through $B_{p,q}$, with $B_{p,-1} = 0$. These are the additional bits needed to get from fidelity layer $q-1$ to layer q. Let $\mathcal{D}_{p,q}$ denote the packet for fidelity layer q, spatial block p. The packet begins with a 4 byte header part that holds the length of the data part. The data part holds segment q from $\mathcal{B}_p$.

For any particular spatial block, p, the decoder can only use the packets for that spatial block $\mathcal{D}_{p,q}$, $q = 0, \ldots, Q-1$, in order. For any q, packet $\mathcal{D}_{p,q}$ is of no use without packets $\mathcal{D}_{p,0} \ldots \mathcal{D}_{p,q-1}$. So, on the final bitstream, the packets for each spatial block are kept in order.

The final bitstream, $\mathcal{B}$, consists of a global header, $\mathcal{G}$, followed by packets. First come the P packets for rate layer $q = 0$, in order. That is, $\mathcal{D}_{p,0}$, for $p = 0, \ldots, P-1$. Then come the P packets for rate layer $q = 1$, and so on, up to rate layer $Q-1$. We can represent the final bitstream as $\mathcal{B} = \mathcal{G}, \mathcal{D}_{0,0} \ldots \mathcal{D}_{P-1,0}\ \mathcal{D}_{0,1} \ldots \mathcal{D}_{P-1,1} \ldots \mathcal{D}_{0,Q-1} \ldots \mathcal{D}_{P-1,Q-1}$. This assembly is also depicted in Fig.7.6.

Suppose a 512 by 512 image is partitioned into 16 spatial blocks, each 64 by 64, and encoded to 1 bit per pixel with $Q = 1$ fidelity layer. The overhead of the 4 byte packet headers is 512 bits, or 0.20%. This 0.20% is incurred for each rate layer in fidelity embedded mode. Five layers would mean a packet header overhead

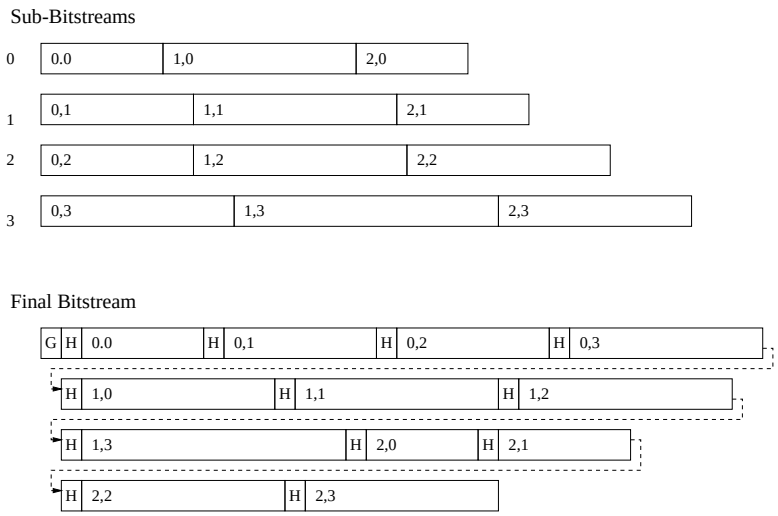

Figure 7.6: This diagram depicts sub-bitstreams and a final packetized bitstream for $P = 4$ spatial blocks and $Q = 3$ fidelity layers. Each segment of each sub-bitstreams is labeled with its layer index and spatial block index, q, p. These segments are moved intact to the final bitstream.

of 1%.

If channel error resilience is required, a logical approach is to protect early portions of each sub-bitstream more robustly than later parts, where errors will have less of an effect on the reconstructed image. Even if an information bit is corrupted, the effect on the reconstructed image is localized to the region affected by a single spatial block. The SB-SPIHT codec has some resilience to bitstream errors. Bit errors occurring in the data portion of packets have a detrimental effect but it is localized to a single spatial block.

A full-image SPIHT encoder without adaptive entropy coding will output the exact same decision bits as this SB-SPIHT encoder, though in a different order. With full-image SPIHT, all decision bits for each processing pass are placed on the bitstream before moving to the next processing pass, or bitplane. That is not necessarily the case for this packetizing encoder. The rate allocation mechanism may find that, in order to minimize the overall distortion, some spatial blocks should have enough bits allocated to them to decode to a lower bitplane, or different pass, than other blocks. This could happen in compound images where different sec-

tions of the image have radically different textures or other characteristics. The SPIHT algorithm may be much more effective in a rate-distortion sense in certain sections of the image than in others. The rate allocation procedure essentially finds a threshold quantizer function slope. With compound images, different spatial blocks may achieve this slope in different bitplanes, or in different passes within the same bitplane.

7.10 Buffer Memory

A line-based transform is used in these low memory systems because image data is generally acquired from a source and provided to a display device line-by-line. The line-based forward transform engine accepts image data line-by-line and produces transform coefficients line-by-line in each band. The line-based reverse transform engine demands coefficients line-by-line, as needed, in each band in order to produce the reconstructed image data line-by-line. There is a different lag in each band due to the FIR filter widths and the recursive application of the filters. The lag is greater in lower frequency bands and smaller in higher frequency bands.

The wavelet coefficient data must be organized into whole spatial blocks for coding by SPIHT or SPECK. Because of the band dependent lag, an additional block buffering stage is used to transfer the lines of transform coefficients within each band to and from spatial blocks. In this section we will determine how much memory is needed by the block buffer.

At the encoder, the block buffer receives the completed coefficients from the line-based transform and stores them in a series of small buffers, one per block, each holding the coefficients for one block. As soon as a block is complete, its data is passed on to the coding stage. The mechanism is similar, though reversed, at the decoder.

The overall approach of this section is to first define the dependencies between coefficients for one stage of a 1-D DWT. Then, these dependencies are used to determine the buffer requirement for producing a 1-D spatial block from a multiple level 1-D dyadic DWT. Then, the buffer requirements for the 2-D case are determined by extending the 1-D case to 2-D.

The line-based 2-D discrete wavelet transform releases completed coefficients after a lag due to FIR filtering in the vertical direction. Horizontal filtering is always performed immediately and does not affect the buffer memory requirement. Thus, much of the analysis needed to determine the block buffer memory requirement for the 2-D line-based system will come from analysis of the 1-D case.

7.10.1 1-D Discrete Wavelet Transform Coefficient Dependence

Consider the 1-D dyadic DWT. Let l_k and h_k represent the low- and high-pass sequences for level k. These sequences are each formed by filtering l_{k-1} and then downsampling by two. For notational consistency, l_0 will represent the original sequence.

The full dyadic transform is produced by transforming l_0 into l_1 and h_1, then l_1 into l_2 and h_2, and so on until we have l_K and h_K. The transform data of interest are the sequences h_1, h_2, ..., h_K and l_K. The sequences l_0, ..., l_{K-1} are temporary and are released as soon as their data is no longer needed.

In this section we are concerned with the processing dependencies between the individual sequence values of the various sequences. A dependency exists when a particular value of one sequence is needed to determine a value of some other sequence. We will start by looking into the dependencies between sequence l_0 and sequences l_1 and h_1 which are the same as the dependencies between sequence l_k and sequences l_{k+1} and h_{k+1}.

The sequence l_0 is normally written as,

$$l_0 : \; l_0[0], l_0[1], l_0[2], l_0[3], l_0[4], l_0[5], l_0[6], l_0[7], l_0[8], l_0[9], \ldots,$$

but we will use the following boxed format notation that will help clarify the alignment and coefficient dependency of filtered and downsampled sequences

$$l_0 : \boxed{l_0[0]}\boxed{l_0[1]}\boxed{l_0[2]}\boxed{l_0[3]}\boxed{l_0[4]}\boxed{l_0[5]}\boxed{l_0[6]}\boxed{l_0[7]}\boxed{l_0[8]}\boxed{l_0[9]} \cdots$$

The sequence l_0 is passed through a low-pass FIR filter and down-sampled by two to produce the sequence l_1 and also passed through a high-pass FIR filter and down-sampled by two to produce the sequence h_1, both of which are below.

l_1 :	$l_1[0]$	$l_1[1]$	$l_1[2]$	$l_1[3]$	$l_1[4]$	$\cdots$
h_1 :	$h_1[0]$	$h_1[1]$	$h_1[2]$	$h_1[3]$	$h_1[4]$	$\cdots$

The down-sampling procedure keeps the even indexed samples of l_1 and odd indexed samples of h_1. It will be useful to think of the l_1 and h_1 sequences with spaces where values have been eliminated by downsampling.

l_1 :	$l_1[0]$		$l_1[1]$		$l_1[2]$		$l_1[3]$		$l_1[4]$		$\cdots$
h_1 :		$h_1[0]$		$h_1[1]$		$h_1[2]$		$h_1[3]$		$h_1[4]$	$\cdots$

Here is where the box notation helps make the alignment clear. This notation keeps the transform coefficients aligned with the coefficients of l_0 as they were for the application of the FIR filters. This format makes it easier to understand the dependencies of values in sequences l_1 and h_1 on values of sequence l_0 during the filtering process.

The box notation also helps understanding of the dependencies of the inverse transform. For the inverse transform the l_1 and h_1 sequences are upsampled and interpolated by FIR filters. After upsampling the sequences look like this.

l_1 :	$l_1[0]$	0	$l_1[1]$	0	$l_1[2]$	0	$l_1[3]$	0	$l_1[4]$	0	$\cdots$
h_1 :	0	$h_1[0]$	0	$h_1[1]$	0	$h_1[2]$	0	$h_1[3]$	0	$h_1[4]$	$\cdots$

This diagram will help make clear which values of l_1 and h_1 are needed to produce a particular value of l_0.

We assume that all four of the wavelet filters have an odd number of taps, say $2S+1$, and are symmetric about the center tap. A symmetric filter with $2S+1$ has a lag of S. Let S_{LF} and S_{HF} represent the specific lags of the low- and high-pass filters for the forward transform. For the inverse transform the lags are S_{LR} and S_{HR}. We will assume the same filters are used for each level of the multi-level dyadic transform. For the 9/7 biorthogonal wavelet [76], the lags are $S_{LF} = S_{HR} = 4$ and $S_{LR} = S_{HF} = 3$.

Consider the coefficient dependence during the forward transform. Each coefficient from level $k+1$ depends on certain coefficients from level k. This relationship will be used to determine the buffer sizes needed to produce whole spatial blocks.

$$l_{k+1}[n] \leftarrow l_k[2n - S_{LF}], \ldots, l_k[2n + S_{LF}] \tag{7.17}$$

$$h_{k+1}[n] \leftarrow l_k[(2n+1) - S_{HF}], \ldots, l_k[(2n+1) + S_{HF}] \tag{7.18}$$

These relationships can be seen visually by using the box notation above and overlaying the filters. The two relations for the low- and high-pass are slightly different because the even coefficients are kept during downsampling of the low-pass signal and the odd coefficients are kept during downsampling of the high-pass signal.

Next, consider the coefficient effect during the forward transform. Each coefficient from level k affects certain coefficients from level $k+1$.

$$l_k[n] \rightarrow l_{k+1}[\left\lfloor \frac{n - S_{LF} + 1}{2} \right\rfloor], \ldots, l_{k+1}[\left\lfloor \frac{n + S_{LF}}{2} \right\rfloor] \tag{7.19}$$

$$l_k[n] \rightarrow h_{k+1}[\left\lfloor \frac{n - S_{HF}}{2} \right\rfloor], \ldots, h_{k+1}[\left\lfloor \frac{n + S_{HF} - 1}{2} \right\rfloor] \tag{7.20}$$

This relationship is in a sense the inverse of the dependence relationship. We will have no direct use for this relationship, but it is here for the sake of completeness.

There is a duality between the dependence and effect relationships for the forward and the reverse transforms. First, consider the coefficient dependence during the reverse transform. Each coefficient from level k depends on certain coefficients from level $k+1$.

$$l_k[n] \leftarrow l_{k+1}[\left\lfloor \frac{n - S_{LR} + 1}{2} \right\rfloor], \ldots, l_{k+1}[\left\lfloor \frac{n + S_{LR}}{2} \right\rfloor] \tag{7.21}$$

$$l_k[n] \leftarrow h_{k+1}[\left\lfloor \frac{n - S_{HR}}{2} \right\rfloor], \ldots, h_{k+1}[\left\lfloor \frac{n + S_{HR} - 1}{2} \right\rfloor] \tag{7.22}$$

Finally, consider the coefficient effect during the reverse transform. Each coefficient from level $k+1$ affects certain coefficients from level k during the reverse

transform process.

$$l_{k+1}[n] \rightarrow l_k[2n - S_{LR}], \ldots, l_k[2n + S_{LR}] \tag{7.23}$$

$$h_{k+1}[n] \rightarrow l_k[(2n+1) - S_{HR}], \ldots, l_k[(2n+1) + S_{HR}] \tag{7.24}$$

Forward dependence does not imply reverse effect, and vice versa.

It is interesting to consider the dependence and effect between the coefficients with the Haar transform as a baseline. Because of the simplicity of the Haar transform, the forward dependence and reverse effect are symmetric and as are the forward effect and reverse dependence. The forward dependence relations are

$$l_{k+1}[n] \leftarrow l_k[2n], l_k[2n+1] \tag{7.25}$$

$$h_{k+1}[n] \leftarrow l_k[2n], l_k[2n+1] \tag{7.26}$$

and the forward effect relations are

$$l_k[n] \rightarrow l_{k+1}[\left\lfloor \frac{n}{2} \right\rfloor] \tag{7.27}$$

$$l_k[n] \rightarrow h_{k+1}[\left\lfloor \frac{n}{2} \right\rfloor] \tag{7.28}$$

The reverse transform dependence relations are

$$l_k[n] \leftarrow l_{k+1}[\left\lfloor \frac{n}{2} \right\rfloor] \tag{7.29}$$

$$l_k[n] \leftarrow h_{k+1}[\left\lfloor \frac{n}{2} \right\rfloor] \tag{7.30}$$

and the reverse effect relations are

$$l_{k+1}[n] \rightarrow l_k[2n], l_k[2n+1] \tag{7.31}$$

$$h_{k+1}[n] \rightarrow l_k[2n], l_k[2n+1] \tag{7.32}$$

The coefficient dependence and effect with the Haar transform are minimal. With any more general useful filter, the dependence and effect are greater. With the Haar coefficient dependency and effect the dependency never crosses a spatial block

boundary.

The equations presented that precisely define the coefficient dependence and effect through the transform process are complicated to work with. Because of the different filter widths, and the special rounding rules it is difficult to use them to draw simple and precise conclusions about the buffer sizes.

The forward dependence can extend up to S_{LF} or S_{HF} coefficients beyond the minimal Haar dependence, or the spatially corresponding coefficients. The reverse dependence can extend $\lceil S_{LR}/2 \rceil$ or $\lceil S_{HR}/2 \rceil$ beyond the minimal Haar dependence. Whether these maximums are reached depends on whether the filter lags (the S variables) are even or odd and whether the particular coefficient under consideration is even or odd. In what follows, we will assume that the dependence and effect always reach these maximums, and will thus bound the buffer size. Further, we will assume that all filter widths are the same $S = S_{LF} = S_{HF} = S_{LR} = S_{HR}$ and $S > 0$.

7.10.2 Multi-level 1-D Dyadic DWT Buffering

In this section we will use the core 1-D DWT coefficient dependence results from the previous section to determine the buffer memory requirement for the 1-D system shown in Fig. 7.7. The 1-D system performs a K level dyadic DWT on a sequence. The sequence is input one value at a time, and after each input all filtering operations that are possible with the data received are completed, so this is analogous to the 2-D line-based system. The 1-D system buffers data in each band and releases whole spatial blocks of size $B2^K$, where B is the number of coefficients from l_K in the spatial block. For the specific case of $K = 5$ and $B = 2$, Table 7.1 shows some of the buffer size values computed in this section, and is referred to using column labels in several places below.

We will determine how much data is buffered when the first spatial block is ready. At that point the system is using the maximum amount of buffer memory. The driving factor in what coefficients are buffered in all bands is the coefficients needed for sequence l_K. The coefficients needed from l_K determine what is needed from sequence l_{K-1}, which determine what is needed from sequence l_{K-2}, and so on down to l_0. The approach taken is to start with the coefficients needed for sequence

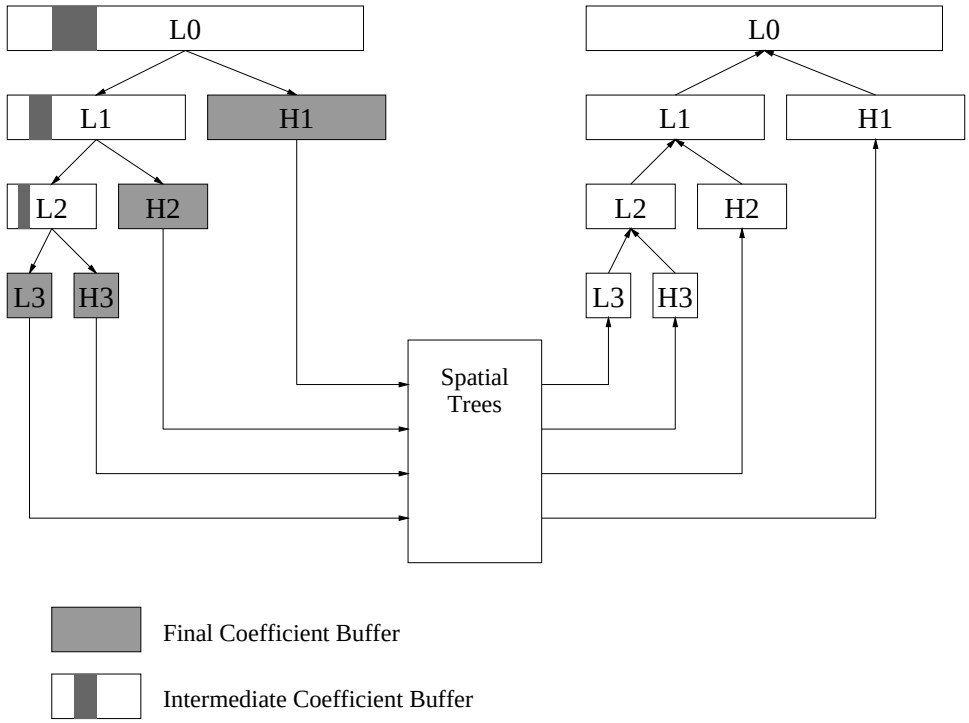

Figure 7.7: This diagram shows the buffering system for a 1-D dyadic wavelet transform.

a	b	c	d	e	f	g
	index of last coefficient in first spatial block		index of last coefficient buffered in each band		excess coefficients buffered	
decomp. level	low	high	low	high	low	high
0	0	–	$32+31S$	–		–
1	0	31	$16+15S$	$16+15S$		$15(S-1)$
2	0	15	$8+7S$	$8+7S$		$7(S-1)$
3	0	7	$4+3S$	$4+3S$		$3(S-1)$
4	0	3	$2+S$	$2+S$		$(S-1)$
5	1	1	1	1	0	0

Table 7.1: This table shows the coefficients needed for the first 1-D spatial block, the coefficients buffered when the first spatial block is ready and the excess coefficients for the specific case of $K = 5$ levels and $B = 2$ (64 coefficients per spatial block).

l_K, and successively determine what coefficients are needed from the low pass band in each lower level. Then, assuming those coefficients are available, verify that the high pass coefficients needed for the spatial block in each band are available.

There is a separate data buffer for each band. The band buffers for $l_0, \ldots, l_{K-1}$ are rolling buffers that hold $2S-1$ coefficients each, just the memory that is needed for the filters. As a new coefficient is added to one of these band buffers, an old one is removed. The $h_1, \ldots, h_K$ and l_K band buffers accumulate completed coefficients. Data is removed from these buffers only when all of the data for a spatial block is ready. A 1-D spatial block has B coefficients from both l_K and h_K, $2B$ coefficients from h_{K-1}, and so on, or in general $2^{K-k}B$ coefficients from h_k (columns b and c in Table 7.1).

Let n_k^l and n_k^h each represent the index of the last coefficient thus far known in l_k and h_k respectively. To produce the first spatial block, B coefficients are needed from sequence l_K, or, $n_k^l = B-1$. From the previous section, to determine coefficient n from sequence l_k, we must know up to coefficient $2n+S$ from sequence l_{k-1}. So, sequence l_{K-1} must be determined up to coefficient $n_{K-1}^l = 2(B-1)+S = 2B-2+S$. In general, sequence l_k must be determined up to $n_k^l = 2^{K-k}(B-1) + (2^{K-k}-1)S$ (column d in Table 7.1). The original sequence, l_0 must be input up to index $n_0^l = 2^K(B-1) + (2^K-1)S$.

Next we must verify the assumption that the required coefficients of l_K are the driving factor in the buffer requirements. Given the coefficients that have been determined for the low-pass sequences we must make sure that all of the dependencies of the high-pass coefficients in the spatial block are met. Given the assumptions, the dependencies of l_k and h_k on l_{k-1} are the same. So, the coefficients of h_k for $k = 1, \ldots, K$ will also be determined up to $n_k^h = 2^{K-k}(B-1)+(2^{K-k}-1)S$ (column e in Table 7.1). The bands $h_1, \ldots, h_K$ have the coefficients needed for the spatial block if $2^{K-k}B - 1 \leq n_k^h$, or the number of coefficients known for h_k is at least as many as are needed for the spatial block. This is in fact the case so long as $S > 0$. The h_k buffers are determined beyond what is needed for the spatial block. The excess is $M_{E,k}^l = n_k^h - (2^{K-k}B - 1) = (2^{K-k}-1)(S-1)$ (column g in Table 7.1).

Finally we can compute the total buffer memory required for this system. The

spatial block itself requires a $M'_S = B2^K$ coefficient memory. The K $l_0, \ldots, l_{K-1}$ intermediate band buffers each require a $2S+1$ coefficient memory to store a number of coefficients equal to the number of filter taps, so the total is $M'_B = K(2S+1)$. The h buffers have an excess of,

$$M'_E = \sum_{k=1}^{K} M'_{E,k} = \sum_{k=1}^{K} (2^{K-k} - 1)(S-1) = (2^K - K - 1)(S-1), \tag{7.33}$$

coefficients. The total buffer memory required is,

$$M' = M'_S + M'_B + M'_E = B2^K + K(2S+1) + (2^K - K - 1)(S-1). \tag{7.34}$$

So, for example, if $K = 5$ and $S = 4$, the excess is 78 coefficients and the total buffer memory required is $M' = M'_S + M'_B + M'_E = 64 + 45 + 78 = 187$

7.10.3 2-D DWT Buffering for Multi-Level Transform

From the results of the previous section the buffer size requirement for the full 2-D line-based transform follows readily because the rolling buffer technique is only used in the vertical direction. Each coefficient of a 1-D band buffer from the 1-D system corresponds to either one or three lines within a 2-D band buffer of the 2-D system.

Each coefficient in l_0 corresponds to one row of C coefficients in the LL0 band where C is the number of columns of the image. In general, each coefficient of l_k for $k = 0, \ldots, K$ now corresponds to a row of $C2^{-k}$ coefficients from the LLk band. Further, each coefficient of h_k corresponds to three rows of $C2^{-k}$ coefficients each, one in each of the bands LHk, HLk and HHk.

Let B_R be the number of rows from LLK in a spatial block. The number of columns in the spatial block does not affect the buffer memory requirement of this line-based system which produces an entire row of spatial blocks at a time, no matter what their width. The row of spatial blocks themselves require storage for $M_S = B_R 2^K C$ coefficients. The K LL0, ..., LL$K-1$ buffers each hold $2S+1$ rows of coefficients and thus requires a $(2S+1)C2^{-k}$ coefficient memory. So, the total

memory for these buffers is,

$$M_B = \sum_{k=0}^{K-1} (2S+1)C2^{-k} = 2(2S+1)C(2-2^{-K}). \tag{7.35}$$

The number of excess coefficients computed beyond the row of spatial blocks for each HL, LH and HH band is $M_{E,k} = M'_{E,k}C2^{-k}$. There are three bands per level so the total excess is

$$\begin{aligned} M_E &= \sum_{k=1}^{K} 3M_{E,k} = \sum_{k=1}^{K} 3(2^{K-k}-1)(S-1)C2^{-k} \\ &= (S-1)C(2^K - 3 + 2^{1-K}) \end{aligned} \tag{7.36}$$

The dominant term for M_E is $SC2^K$.

The total buffer memory required is

$$\begin{aligned} M &= M_S + M_B + M_E \\ &= B_R 2^K C + 2(2S+1)C(2-2^{-K}) + (S-1)C(2^K - 3 + 2^{1-K}) \end{aligned} \tag{7.37}$$

So, for example, for $K = 5$ levels of decomposition, a filter lag of $S = 4$, $B_R = 2$ for spatial blocks with $B_R 2^K = 64$ rows, and a $C = 1024$ column image, the total memory required for buffering in the transform is $M = M_S + M_B + M_E = 65536 + 36288 + 89280 = 191104$. If the image has 1024 rows as well, the memory requirement in coefficients is 18.2% of the number of pixels in the image.

7.11 Results and Conclusions

Coding results for the 512 by 512 grayscale *lena* and *goldhill* standard test images are plotted in Fig. 7.8. A five level decomposition using the 9/7 biorthogonal wavelet [76] was used for these tests. The solid lines show rate vs. distortion performance for full-image SPIHT without arithmetic coding. The dash-dot lines show the performance using independent 64 by 64 spatial blocks and packetization for fidelity embedding with only a few rate layers, at 0.1, 0.5 and 2 bpp. A scalloping effect is seen in these plots as the performance of the SB-SPIHT codec catches up to

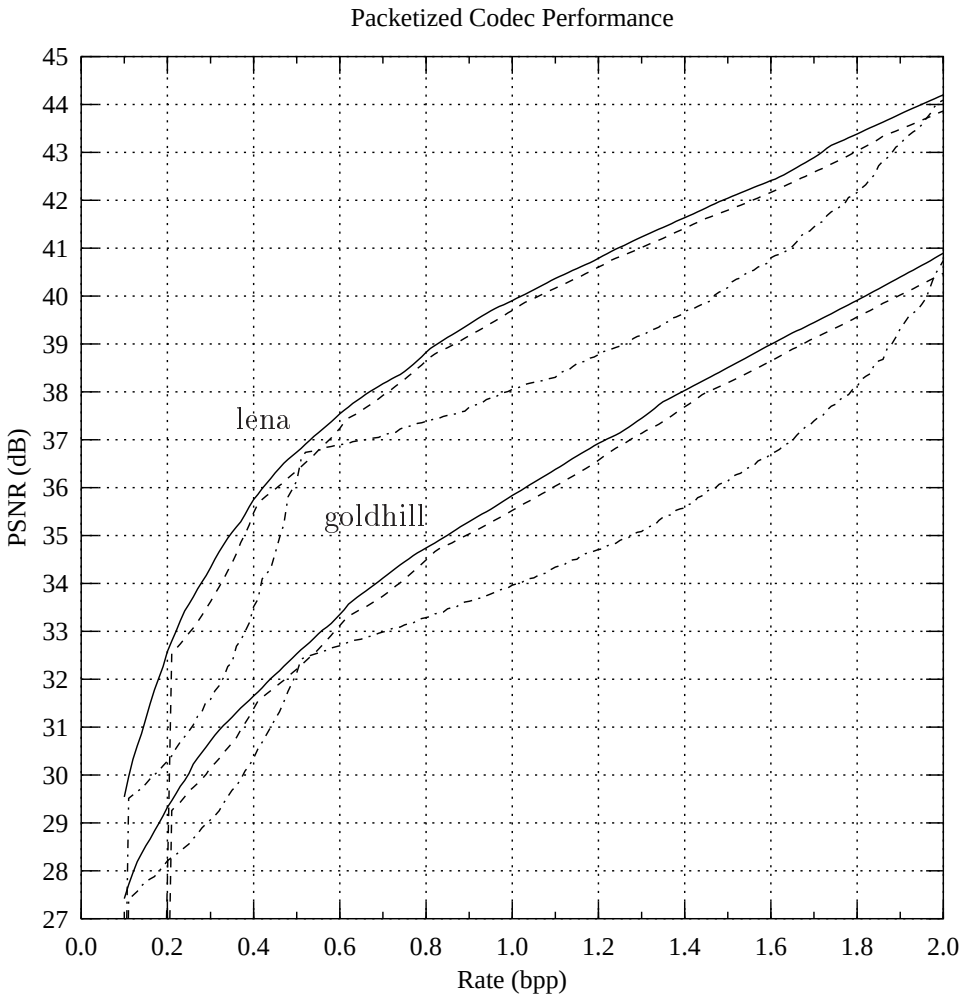

Figure 7.8: Coding performance for the *lena* and *goldhill* images. The solid lines are for full-image SPIHT; the dashed lines are for finely embedded Spatial Block SPIHT; the dash-dot lines are coarsely embedded Spatial Block SPIHT.

the full-image SPIHT codec at these rates. The dashed lines show the performance with rate layers at increments of 0.2 bpp. At rates beyond the first layer at 0.2 bpp, this performance plot follows full-image SPIHT closely, with some loss due to packet header overhead. On all plots for the block based codecs we observe a very low PSNR before the first layer is decoded. Before this point some spatial blocks receive zero encoded bits and contribute dramatically to the distortion. All decoded images in each plot were recovered from a single embedded encoded file, truncated at the desired rate.

image	rate	Full-Image SPIHT PSNR (dB)	1 Layer SB-SPIHT PSNR (dB)	Multi-Layer SB-SPIHT PSNR (dB)
lena	0.0625	27.5374	27.1532	14.4968
	0.125	30.4589	30.2541	26.6064
	0.25	33.5671	33.4623	31.7632
	0.5	36.7424	36.6838	35.9891
	1.0	39.9038	39.8765	39.5201
	2.0	44.2019	44.1542	43.8666
barbara	0.0625	22.9772	22.7355	13.2249
	0.125	24.2919	24.3913	22.4611
	0.25	27.0107	26.9234	25.5304
	0.5	30.7711	30.7150	29.8386
	1.0	35.7946	35.7809	35.1745
	2.0	41.9330	41.8246	41.5075
goldhill	0.0625	26.1908	25.9291	13.8611
	0.125	28.0879	27.9047	25.6165
	0.25	30.0314	29.9108	28.8105
	0.5	32.5305	32.3958	31.7893
	1.0	35.8367	35.6850	35.3289
	2.0	40.9011	40.8269	40.4758

Table 7.2: Spatial Block SPIHT coding results for standard test images.

Tables 7.2 and 7.3 also show performance results for several standard test images and rates. Results are given for full-image SPIHT, Spatial Block SPIHT with a single rate layer, and Spatial Block SPIHT with rate layers at increments of 0.2 bpp. The *lena*, *barbara* and *goldhill* standard grayscale test images are each 512 by 512 pixels. The *aerial2* JPEG 2000 grayscale test image is a 2048 by 2048 pixels. The other JPEG 2000 image, *bike*, *càfe* and *woman* are each 2560 by 2048 pixels in size. The performance of the multi-layer bitstream at 0.0625 bpp and 0.125 bpp is low because the first rate layer at 0.2 bpp has not been reached so some spatial blocks have no information to decode and contribute strongly to the distortion.

A couple of examples illustrate the nature of the layered fidelity embedding feature. Figure 7.9 shows a reconstructed *lena* image after encoding the image to 1 bpp in single layer mode, but then decoding only half of the bitstream, or 0.5 bpp. Because subband domain spatial blocks correspond to image domain blocks,

image	rate	Full-Image SPIHT PSNR (dB)	1 Layer SB-SPIHT PSNR (dB)	Multi-Layer SB-SPIHT PSNR (dB)
aerial2	0.0625	24.1796	23.8760	13.0988
	0.125	26.0006	25.8218	23.4358
	0.25	27.9151	27.8608	26.9994
	0.5	29.9984	29.9863	29.6231
	1.0	32.7264	32.7030	32.3786
	2.0	37.3166	37.2929	37.0190
bike	0.0625	22.6780	22.2797	9.03639
	0.125	25.1348	24.9500	21.8170
	0.25	28.4300	28.3117	26.6590
	0.5	32.2770	32.2288	31.3251
	1.0	36.8373	36.8388	36.3572
	2.0	42.7525	42.7270	42.4076
càfe	0.0625	18.5678	18.3640	10.4899
	0.125	20.1257	19.9666	18.0703
	0.25	22.2583	22.2005	21.1474
	0.5	25.7619	25.7131	24.8944
	1.0	30.7956	30.7655	30.1954
	2.0	37.8804	37.8564	37.4144
woman	0.0625	24.8436	24.6376	12.0574
	0.125	26.6630	26.5538	24.3574
	0.25	29.0794	29.0478	27.8124
	0.5	32.6608	32.6526	31.8187
	1.0	37.4866	37.5342	36.9804
	2.0	43.0429	43.0044	42.6355

Table 7.3: Spatial Block SPIHT coding results for JPEG 2000 test images.

it is clear that about half of the spatial blocks have been decoded, following a raster order. A few more than half are decoded because of the greater image activity in the lower half of this image. Figure 7.10 shows the *lena* image after encoding the image to 1 bpp in fidelity progressive mode with fidelity layers at 0.02 and 1 bpp, but then only decoding to 0.5 bpp. These layer rates were chosen for dramatic visual effect. Decoding to 0.5 bpp fully decodes all of the packets in the first rate layer at 0.02 bpp because they come first in the bitstream. Thus, all of the image is reconstructed to at least the fidelity achievable with 0.02 bpp. The 1 bpp layer is then partially

Figure 7.9: A fixed rate Spatial Block SPIHT decoded image after decoding half of the bitstream.

decoded and about half of the image is reconstructed to a higher fidelity.

7.11.1 Practical Demonstration

To demonstrate the peril in not heeding the size of cache memory we ran two codecs in a very large image using a general purpose 167 MHz Sun Ultra 1 computer with 64M RAM. In this experiment, consider RAM to be the fast cache memory and the hard disk swap space to be the external slow memory.

Both a full-image and independent Spatial Block SPIHT codec were applied to a 2k column by 5k row pixel image. This implementation is not heavily optimized. The SB-SPIHT codec encoded the image in 220 sec. and decoded in 43 sec. total with no significant disk swapping. The SPIHT algorithm itself, took only 20.90 sec. for encoding and 5.16 sec. for decoding. The full-image SPIHT encoder never finished encoding this large image. It was stopped manually after an hour of spending 97% of the time waiting for memory to be swapped from the disk.

The ability to decode a small region of interest of the image is a natural feature of the SB-SPIHT bitstream. To decode a subset of the spatial blocks, the decoder can drop packets it does not need.

Figure 7.10: A coarsely fidelity progressive Spatial Block SPIHT decoded image after decoding half of the bitstream.

7.12 Early System

An initial low-memory SPIHT system developed prior to the one described here used a different, processing pass based, rate allocation technique and fixed size packets. The initial system used the bitplane index, and the SPIHT pass (LIS, LIP, LSP) to assign a priority to segments of the sub-bitstreams. Let n be the bitplane index and s be the pass index. In the LIS pass, $s = 2$; in the LIP pass, $s = 1$; in the LSP pass, $s = 0$. Fixed size segments of each sub-bitstream were assigned a priority $Y = 3n + s$. The fixed size segments were packetized with a header consisting of the sub-bitstream index. The packets from all sub-bitstreams were interleaved onto the final bitstream according to their priority.

This system was abandoned for the rate-distortion motivated system for several reasons. The initial system was suited well to a finely fidelity progressive final bitstream, but not to a fixed rate final bitstream. If a fixed rate final bitstream is required, then the overhead of fixed rate packets is wasteful and variable length packets are more desirable.

The rate-distortion motivated rate allocation technique more readily extends to a similar image coding system based on SPECK which we will describe in the

following chapter. Further, it allows the independent Spatial Block SPIHT codec and the Subband Block SPECK codec to be joined in a generalized hybrid system which we will discuss in Chapter 9.

CHAPTER 8
Subband Block SPECK

8.1 Motivation

The original SPECK image compression algorithm [17, 18], like SPIHT, requires access to the wavelet coefficients following a complex pattern that is not easily predictable. This prevents the algorithm from using a cached memory system efficiently. In this chapter we present an enhanced SPECK compression algorithm that limits its access to small blocks of data at a time.

8.2 Overview

In the previous chapter the SPIHT algorithm was modified so that small portions of the wavelet decomposition are coded independently. The resulting sub-bitstreams are combined by packetization after rate allocation. With a change in structure, the same memory reduction process can be applied with the SPECK algorithm [17, 18]. In fact, the system diagrams in Figs. 7.1 and 7.2, on page 69 in the previous chapter, apply to the subband block codec just as well as they apply to the spatial block codec.

For SPECK to operate well on a small set of the transform coefficients at a time, the set should be a block of coefficients within one subband. Such a block is referred to as a subband block, as opposed to a spatial block, for the following reason. The coefficients from a subband block, like those from a spatial block, are all from the same spatial region of the image. However, a subband block has coefficients only from one subband, and thus does not have all of the coefficients needed to reproduce a region of the original image.

Islam has also applied SPECK independently to blocks within subbands [18]. In this work a much different technique is used for rate allocation among the blocks. The forthcoming JPEG 2000 image compression standard also partitions the wavelet decomposition in the same way described here. While JPEG 2000 applies the Embedded Block Coder with Optimal Truncation (EBCOT [25]) to each block, SPECK

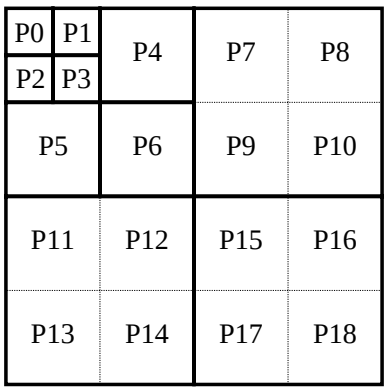

Figure 8.1: This diagram depicts a three level subband decomposition partitioned into square subband blocks. Some bands contain a single block, while larger bands contain more than one.

is applied in this system.

The SPECK algorithm is applied to each block independently, producing a sub-bitstream for each block. The progressive refinement of the SPECK algorithm can be viewed in the same framework that simplifies quantizer function generation and rate allocation amongst SPIHT encoded spatial blocks. Both algorithms generate a progressive bitstream that the rate allocation operation will truncate.

8.3 SPECK Coding in Subband Blocks

The Subband Block SPECK image codec is similar in structure to the Spatial Block SPIHT image codec of the previous chapter. For the description in this chapter we rely on the previous chapter as a foundation and emphasize the differences.

8.3.1 Subband Blocks

Each band of the dyadic wavelet transform is partitioned into N_{br} by N_{bc} blocks beginning in the upper left corner and using smaller blocks if necessarily at the right and bottom edges of the bands. N_{br} and N_{bc} are typically 32, 64 or 128. Figure 8.1 shows an example of subband block partitioning.

Each time the line-based wavelet transform engine produces a row of coefficients in a band, those coefficients are quantized by rounding to an integer and moved to the buffers for the subband blocks in which they belong.

8.3.2 SPECK Encoding

As soon as a subband block buffer is full, and $\mathcal{C}_p$ is available, it is encoded by the SPECK algorithm to generate the sub-bitstream $\mathcal{B}_p$. Each subband block is completely encoded through to the least significant bitplane. A nearly lossless sub-bitstream is generated.

The original SPECK system [17, 18] was designed to code full images, not independent subband blocks. Full image SPECK initially has a type S and an L-shaped type I set. The type I set spans all bands at all orientations above some level. To apply SPECK to a single block we initialize with the entire block represented in one or more square type S sets, and never use the L-shaped type I set.

The over-coding problem is a bit more complex with SB-SPECK than with SB-SPIHT. Each partition in the SB-SPECK system holds coefficients from a single subband. Due to the higher energy in lower frequency subbands, the rate allocation to lower frequency subbands is much greater than that to higher frequency subbands. The simple over-coding strategy used by SB-SPIHT is no longer sufficient. If each partition is coded to 2.5 to 3.0 times the overall desired rate, then the lower frequency subbands will not have enough rate allocated to them, while the higher frequency subbands will be over-coded much more than needed. The SB-SPECK system avoids this problem by completely encoding each partition. This leads to a greater computational burden at the encoder, but not at the decoder.

8.3.3 Rate Allocation

The SPECK decoder updates reproduced coefficients in the same manner as the SPIHT decoder. Bitplanes are scanned in order from most to least significant. In each bitplane, there are passes to find newly significant coefficients and then a refinement pass. Thus, the same technique of counting the coefficients that are modified in each pass in order to form the quantizer functions can be used by the SPECK encoder.

For each significance pass of the SPECK algorithm in each bitplane, the encoder records the number of bits placed on the sub-bitstream and the number of newly significant coefficients found. In the refinement pass, in each bitplane, the

encoder records the number of bits, and the number of coefficients refined.

SPIHT performs three separate passes at each significance level: the LIS, LIP and LSP passes. Bit and coefficient counts are reset before each pass because we expect rate distortion performance to change in each new pass.

In contrast, SPECK has two major types of passes, LIS and LSP. However, there are several stages of LSP passes and the bit and pixel counts are reset before each stage. The first LIS pass is only for 1 by 1 sets, the second LIS pass is only for 2 by 2 sets and for new smaller sets created by splitting, and so on. We expect each of the LIS passes to decrease in rate-distortion performance, which is why they are performed in this particular order in the first place.

Using this data, a convex piecewise linear quantizer function is formed. Rate allocation for any number of fidelity levels, Q, is performed using the quantizer functions, just as is done for SB-SPIHT.

8.3.4 Bitstream Assembly

After rate allocation and packetization, the packets may be placed on the bitstream in any order prearranged by the encoder and decoder. Two useful assembly modes provide fidelity progressive and resolution progressive final bitstreams.

For the resolution progressive mode, rate is allocated to each block in a single fidelity layer to meet the total rate budget. Packets are placed on the final bitstream in an order defined by the decomposition level. First, packets for the DC band are transmitted, then the next higher frequency bands, and so on up to the highest frequency bands. If this bitstream is truncated, all information for some number of resolution levels will still be available, and a lower resolution image can be reconstructed. A fidelity progressive bitstream is generated by allocating rate in $Q > 1$ fidelity layers and packetizing following the same procedure used for SPIHT in Section 7.8.

This framework is useful in a client server system in which custom compressed images are provided from a database upon demand. The server can hold the full embedded sub-bitstream for each block and precomputed rate allocation tables. When a client demands, an image at a certain rate and resolution, little processing

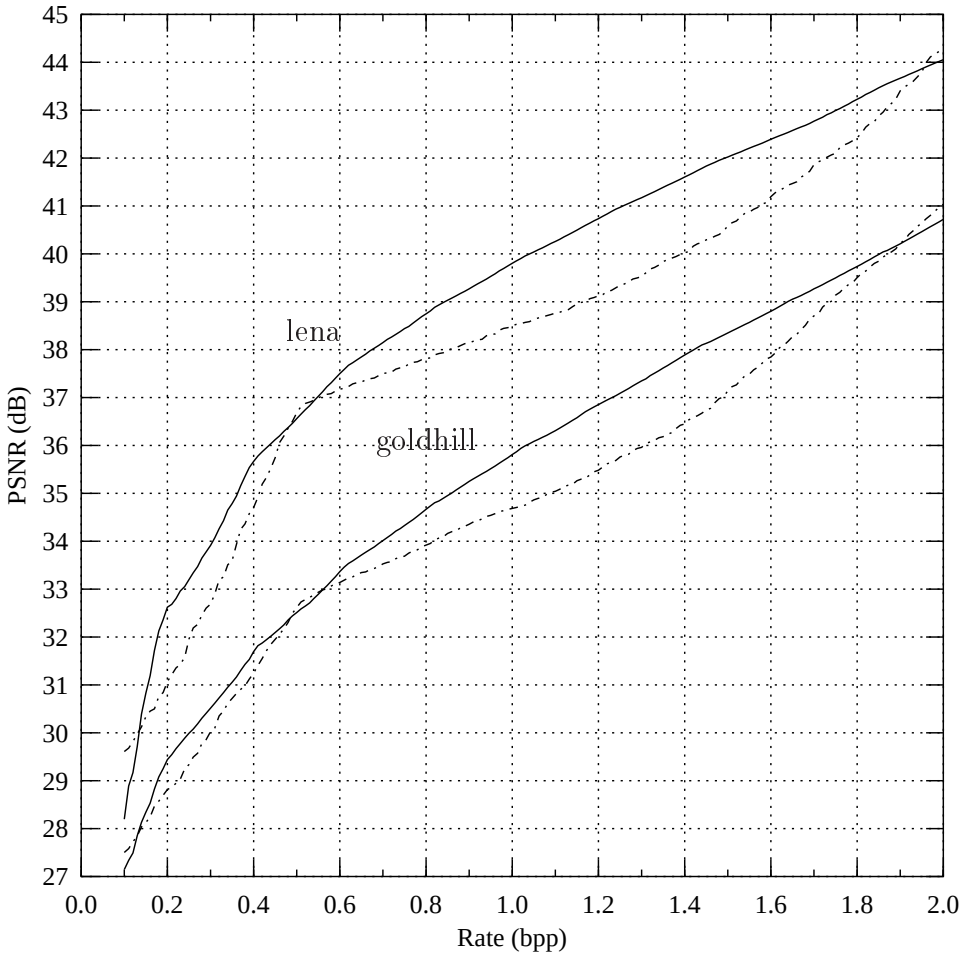

Figure 8.2: Coding performance for the *lena* and *goldhill* images. The solid lines are for finely embedded Subband Block SPECK; the dotted lines are for coarsely embedded Subband Block SPECK.

is needed to assemble the bitstream.

8.4 Results

Figure 8.2 shows coding results for the *lena* and *goldhill* 512 by 512 grayscale images. A five level decomposition using the 9/7 biorthogonal wavelet [76] was used in these tests. All decoded images in each plot were recovered from a single embedded encoded file, truncated at the desired rate. The solid lines show rate vs. distortion performance for finely embedded Subband Block SPECK. Here, in-

dependent 64 by 64 subband blocks are used with rate layers at increments of 0.2 bpp. The dotted lines show the performance for fidelity embedding with only a few rate layers, at 0.1, 0.5 and 2 bpp. A scalloping effect is seen in these plots as the performance of the block based SPECK codec is at its best at the rate layer points but is less efficient at rates in between the layers.

In the Spatial Block SPIHT results plotted in Fig. 7.8, on page 96, we observed a sharp decline in performance at decoded rates before the first rate layer of the bitstream. The reason was that before the first rate layer, some spatial blocks receive zero bits and significantly increase the overall image distortion. This does not generally happen with SB-SPECK because even before the first rate layer, the information received affects the entire reconstructed image. Usually, the DC band is in a single subband block and a portion of the sub-bitstream for that block is in the first packet. So, a rough reconstruction of the entire image is available even if only a portion of the first packet is received.

Performance results for this codec with 64 by 64 blocks and no entropy coding on standard and JPEG 2000 test images are given in Tables 8.1 and 8.2. Also in both tables are results for the SPECK codec described in [18] (Tables 7.1 and 7.2), where they are available. The results for the Subband Block SPECK system presented here are almost always better. For what it is worth, the "average" improvement in PSNR, taken over each result in the tables, is 0.234 dB. The major difference in the codecs that achieved these results is in the rate allocation technique.

Table 8.2 also shows results using the SBHP codec [77]. SBHP is similar to SPECK but uses back-end Huffman coding which increases its complexity while decreasing rate. Again, for what it is worth, the "average" improvement in PSNR, taken over each result in the table, that SBHP makes over the SB-SPECK is 0.36 dB.

8.5 Non-Unitary Integer Wavelet Transforms

We have been assuming that the wavelet transform is nearly unitary. When the filters are implemented with floating point operations, this assumption is usually true. However, for fast codec implementations, it is often desirable to use a

image	rate	PSNR (dB)	PSNR (dB) [18]	Δ PSNR (dB) [18]
lena	0.0625	27.1521		
	0.125	30.2827		
	0.25	33.4767	33.37	0.10
	0.5	36.7019	36.49	0.21
	1.0	39.8978	39.65	0.24
	2.0	44.2363		
barbara	0.0625	22.8542		
	0.125	24.6673		
	0.25	27.4009	26.96	0.44
	0.5	31.1614	30.79	0.37
	1.0	36.0753	35.57	0.50
	2.0	42.1177		
goldhill	0.0625	26.0237		
	0.125	27.9477		
	0.25	30.0819	30.21	-0.13
	0.5	32.6315	32.58	0.05
	1.0	35.9165	35.67	0.24
	2.0	40.9571		

Table 8.1: Subband Block SPECK coding results for standard test images.

transform that is implemented with integer operations only. Many integer based wavelet transforms used for image coding are nearly unitary except for different non-integer scale factors for the low and high pass bands. These scale factors can cause problems. For some integer filters, the scale factors can be compensated for images by bit shifting operations, but for others they cannot without resorting to costly multiplication or division operations to compensate for the uneven amplification. This section explains this problem and how it affects bitplane codecs like SPIHT and SPECK.

SPIHT is a bitplane codec and groups coefficients from different transform levels into the same set for significance testing. So, SPIHT relies on the uniform scaling of energy in each band. That is, a transform coefficient of some magnitude in some band and a transform coefficient of the same magnitude in another band each represent the same energy and presumably the same visual importance in the

image	rate	PSNR (dB)	PSNR (dB) [18]	Δ PSNR (dB) [18]	PSNR (dB) [77]	Δ PSNR (dB) [77]
aerial2	0.0625	24.0241	24.20	-0.18	24.453	-0.43
	0.125	26.0417	26.24	-0.20	26.382	-0.34
	0.25	28.1498	28.14	0.00	28.400	-0.25
	0.5	30.2695	30.12	0.14	30.436	-0.17
	1.0	32.8098	32.66	0.14	33.089	-0.28
	2.0	37.3773	37.32	0.05	37.807	-0.43
bike	0.0625	22.5815	22.53	0.05	23.092	-0.51
	0.125	25.1744	24.79	0.38	25.484	-0.31
	0.25	28.3882	27.96	0.42	28.671	-0.28
	0.5	32.2627	31.74	0.52	32.549	-0.29
	1.0	36.9437	36.24	0.70	37.272	-0.33
	2.0	42.8110	42.28	0.53	43.320	-0.51
càfe	0.0625	18.4629	18.51	-0.05	18.799	-0.34
	0.125	20.1888	20.28	-0.10	20.530	-0.34
	0.25	22.4285	22.15	0.27	22.721	-0.29
	0.5	25.7861	25.37	0.41	26.161	-0.37
	1.0	30.8299	30.17	0.65	31.295	-0.47
	2.0	37.8788	37.47	0.40	38.528	-0.65
woman	0.0625	24.9961	24.84	0.15	25.324	-0.33
	0.125	26.8408	26.91	-0.07	27.178	-0.34
	0.25	29.3959	29.10	0.29	29.701	-0.31
	0.5	32.9312	32.40	0.53	33.264	-0.33
	1.0	37.6920	37.02	0.67	38.025	-0.33
	2.0	43.2277	43.19	0.03	43.706	-0.48

Table 8.2: Subband Block SPECK coding results for JPEG 2000 test images.

original image. The original SPECK algorithm [17, 18] also groups coefficients from different transform levels into the same set early in its processing. So, SPECK also relies on uniform scaling. This restriction can be avoided with SB-SPECK because it groups coefficients only from the same subband for coding. This feature allows SB-SPECK to easily use integer based transforms that do not have uniform scaling. The method is explained below after examining the scale factors of a few popular wavelet filter sets.

In order for the wavelet transform to have uniform scaling, the L_2 norms of the filters must be 1. For the 9/7 filter set, this is nearly the case. The forward

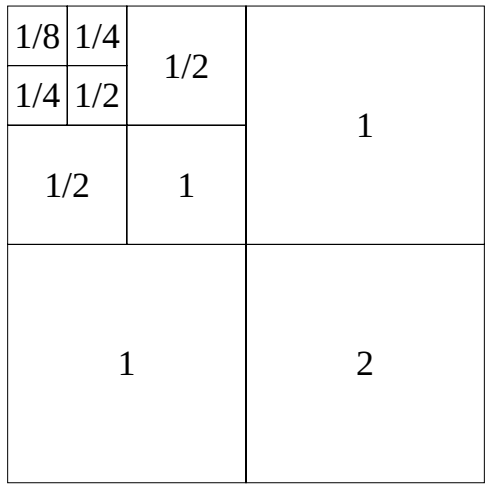

Figure 8.3: This diagram shows the scale factor in each band of a 3 level dyadic wavelet decomposition that occurs with the Haar transform.

low-pass filter has a norm of 1.0200 and the forward high-pass filter has a norm of 0.99144.

The Haar filter set is the simplest filter set that might be used for a wavelet transform. It can easily be implemented with integer operations. The S+P implementation technique [78], which is now also known as the lifting technique [79] offers a way to implement several filter sets, including the Haar set, such that no information is lost due to truncation, and thus the transform is a one-to-one mapping between integer vectors. With this implementation, the Haar filter taps are effectively [1/2, 1/2] for the forward low-pass filter and [1, -1] for the forward high-pass filter. The L_2 norms of these filters are $\sqrt{2}/2$ and $\sqrt{2}$ respectively. Since the norms are not 1, there is an apparent scaling problem, but there is a simple solution [78]. We are focusing on the norms of the 1-D filters, but it is important to consider their combined effect in the 2-D transform. Figure 8.3 shows the coefficient magnitude scaling that occurs with these filters. The product of the norms of any two of the filters (due to horizontal and vertical application) is an integer power of 2. So, the combined scale factors can be compensated for with simple bit shift operations.

A frequently used integer filter set for lossless coding is the 5/3 integer filter set. These filters are best implemented with the S+P, or lifting technique, but the

equivalent FIR coefficients are [-1/8, 1/4, 3/4, 1/4, -1/8] for the forward low-pass filter and [-1/2, 1, -1/2] for the forward high-pass filter. The L_2 norms of the filters are about 0.8478 and 1.2247 respectively.

The filter norms are no longer $\sqrt{2}/2$ and $\sqrt{2}$, so the magnitude scale factors in each band of the 2-D transform are not integer powers of 2. Multiplication or division operations are then necessary to compensate for the scale factors in each band. Even if the additional computation was not an issue, the multiplication makes it difficult to do efficient lossless coding because the least significant coefficient bits will not be 1 or 0 with equal likelihood.

This integer filter scale factor problem can be easily dealt with in an SB-SPECK system. Each partition is completely within a subband and thus affected by a single scale factor. So, each partition can be coded by SPECK, disregarding the scale factor. The scale factors cannot be completely ignored, however. They are accounted for in the rate allocation stage. The squared-error distortion estimates for each partition are simply scaled by the square of the scale factor for that partition. This feature is an important advantage that SB-SPECK has over SB-SPIHT for certain platforms and integer filter sets.

CHAPTER 9
Hybrid Block Coding

9.1 Motivation

The work presented in this chapter is motivated by compression efficiency, memory usage, memory cache efficiency, and the possibility of improved coding with a hybrid of SB-SPIHT and SB-SPECK. In previous reports, SPIHT [16] was generally superior to SPECK [17, 18] in rate vs. distortion performance. However, SPECK has been found to be most competitive in images with elevated high spatial frequency energy, such as the *barbara* image. The Hybrid Block Codec (HBC) presented here attempts to capitalize on this difference. HBC applies tree based SPIHT to low-frequency bands and block based SPECK to high-frequency bands [80].

As discussed in previous chapters, discrete wavelet transform based codecs like SPIHT and SPECK generally encode an entire image as one unit. The image is transformed as a whole, and then the algorithm encodes the coefficient values. Unfortunately, there is no structure to the order in which the coefficient values are accessed by the SPIHT and SPECK algorithms. With such an unstructured access requirement, memory cache systems are of no benefit because of frequent cache misses. Effective automatic memory caching relies on temporal and spatial correlation of memory accesses. In order to operate efficiently in a caching environment, wavelet coefficients are grouped into small blocks which are entropy coded independently, in turn. With blocks small enough to fit into cache memory, cache performance is greatly improved.

9.2 Overview

The Hybrid Block Codec is so named because it is a hybrid of the SB-SPIHT and SB-SPECK codecs. A new hybrid partitioning scheme divides the subband transform coefficients into a combination of spatial blocks and subband blocks. Given the new partitioning scheme, the overall coding system operates like the SB-SPIHT and SB-SPECK systems, as depicted in Figs. 7.1 and 7.2, on page 69.

The line-based transform engine feeds transform coefficients to a set of P buffers, one per block. As before, as soon as a block buffer is full it is encoded. Spatial blocks are coded with the SPIHT algorithm and subband blocks are coded with SPECK. As with the SB-SPIHT and SB-SPECK schemes, the encoder over-codes each block, and then truncates the sub-bitstream for each block after rate allocation. The sub-bitstreams are packetized and assembled into a final bitstream.

9.3 Hybrid Subband Partitioning

The hybrid partitioning scheme divides the subband transform coefficients into a combination of spatial blocks and subband blocks. The structure of the dyadic wavelet decomposition is diagrammed in Fig. 6.2, on page 44. An L level dyadic wavelet decomposition is partitioned into non-overlapping spatial and subband blocks, as shown by example in Fig. 9.1. The dividing point, between the spatial blocks and subband blocks is specified by K. For some decomposition level K, usually 1 or 2, all codevalues from band LLK are partitioned into spatial blocks. The hierarchical trees in these spatial blocks are truncated because the highest frequency coefficients are not included in the block. All bands not in LLK are partitioned into subband blocks, which are rectangular regions within a single subband. Thus, truncated spatial blocks are used for the lower frequency bands and subband blocks are used in the higher frequency bands.

Generally $K < L$, where L is the total number of decomposition levels, so the LLK band has been further subdivided by the transform. For example, with an $L = 5$ level decomposition, and $K = 2$, the LL2 band is no longer strictly present, but has been further divided by the dyadic decomposition. However, the coefficients that come from the LL2 band are partitioned into spatial blocks.

Each spatial block holds one or more hierarchical trees, which contain the tree based sets of the SPIHT algorithm, with high frequency bands pruned. Each subband block contains the square sets of the SPECK algorithm. Thus, the core SPIHT algorithm is readily modified to independently encode the spatial blocks and SPECK is also modified to encode the subband blocks.

If $K = 0$, all coefficients are in spatial blocks, resulting in a Spatial Block

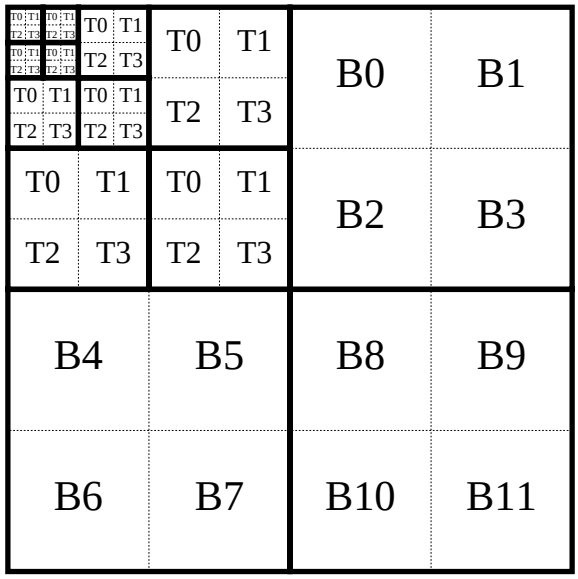

Figure 9.1: Shown is an example of hybrid partitioning with $L = 4$ decomposition levels and $K = 1$ levels in subband blocks. Bands from LL1 are partitioned into spatial blocks. The highest frequency bands (LH1, HL1, HH1) are partitioned into subband blocks. Thick solid lines separate subbands, and thin dotted lines separate spatial and subband blocks. Each section that is a member of a spatial block is labeled with a T followed by the index of the spatial block. The sections that comprise a spatial block are not contiguous. Each subband block is labeled with a B followed by the index of the subband block. There are 4 spatial blocks and 12 subband blocks. In this example, each block has the same number of coefficients.

SPIHT (SB-SPIHT) codec [67] from Chapter 7. If $K > L$, then all coefficients are in subband blocks, and we have a Subband Block SPECK (SB-SPECK) codec of Chapter 8.

This hybrid partitioning technique can be utilized with wavelet packet decompositions as well. Let $P1$ be the decomposition in which the 3 highest frequency bands are each further subdivided into 4 subbands; and $P2$ be the decomposition which starts with the $P1$ decomposition, but the 15 (level 1 and 2) highest frequency bands are further subdivided into 4 subbands each. Hybrid partitioning and the codec described here can be applied to these decompositions as well. The high frequency bands are partitioned into subband blocks and coded with SPECK.

Hybrid partitioning, with $K > 0$, facilitates resolution scalability. A smaller replica of the encoded image, resulting from the omission of high frequency bands, can be recovered from a truncated bitstream if the packetized bitstream is assembled properly. Strictly spatial tree based compression schemes, such as EZW and SPIHT, in general, do not lend themselves to this property. Because the trees extend to the highest frequency bands, and information bits for the different levels are intermixed in the bitstream, it is more difficult to extract just the information for low frequency bands from the bitstream.

9.4 Rolling Wavelet Transform

So that the overall system is cache efficient, and to reduce the memory requirements of this system, HBC employs a line-based transform engine [20], just as in the SB-SPIHT and SB-SPECK systems. At the encoder, the image is read line by line from the top. After each line, all possible computation with the data received thus far is completed. For each decomposition level, a buffer of F lines is required, where F is the vertical filter length. This buffer holds a sliding window of intermediate coefficients awaiting a vertical transform.

A memory buffer is created for each block, but not allocated until it is needed. The line-based transform process makes final coefficients ready one row at a time in each band. That data is moved to the appropriate block buffers. As soon as a block buffer is full, it is encoded with SPIHT or SPECK and the buffer memory is released.

By the nature of the line-based technique, blocks corresponding to the top of the image will be assembled and ready for encoding first. They will be coded and released before any coefficients for lower blocks are generated. With only a portion of the block buffers allocated at any time, the memory requirements of the system are greatly reduced.

9.5 Wavelet Coefficient Block Coding

The SPIHT and SPECK algorithms produce a fidelity embedded sub-bitstream for each block. Because we wish to visit each block with the encoder algorithm

once, use rate control, and minimize the overall reconstructed image distortion, over-coding is required at the encoder. This can be avoided with distortion control, and reduced by using a multi-pass procedure in which partially encoded blocks are revisited for additional coding based on feedback from the rate allocation process.

Like the SB-SPECK system, HBC does not use a rate multiplier to guide over-coding. Instead each block is completely encoded, to the least significant bitplane. The sub-bitstreams are stored until all blocks are encoded at which point they are packetized to form the final bitstream. Each block is fully encoded because the amount of rate to be allocated to a particular block cannot be determined before all blocks are encoded. The sub-bitstream $\mathcal{B}_p$ produced for each block is moved from cache memory to slow memory and the block buffer memory is released.

9.6 Rate Allocation

Rate control is applied to the sub-bitstreams to achieve an exact target bitrate and to minimize distortion in the reconstructed image. The rate allocated to each block is not yet known when the parts are encoded, so each block is fully encoded, as in the SB-SPECK system. It is critical that SPIHT and SPECK are fidelity embedded to be used in this way.

A piecewise linear quantizer function is produced for each block in the manner described in Section 7.7. The quantizer function represents the rate-distortion performance of the SPIHT or SPECK algorithm for that block. The technique used to generate the quantizer functions relies on the codec being a bitplane codec that has passes to find newly significant coefficients and a refinement pass in each bitplane. Because it makes no other assumptions regarding the entropy coding algorithm, it works just as well for SB-SPIHT, SB-SPECK, and this hybrid combination of the algorithms. The rate allocation algorithm can allocate rate amongst the combination of block types as well as just as well as if there were only one block type.

The nearly optimal greedy procedure [72] described in Section 7.8 is used to allocate Q fidelity layers at user selected rates. Let R_q be the desired rate for fidelity layer q, where $q = 0,\ldots,Q-1$. Accounting for the global and fixed size packet headers, the rate allocation procedure is applied once for each layer. This

process determines the number of bits, $B_{p,q}$, to use from $\mathcal{B}_p$, the sub-bitstream for block p, in order to achieve fidelity layer q.

The SPIHT and SPECK codecs produce a finely fidelity embedded bitstream for each block. However, because of the way in which the sub-bitstreams will be packetized, the final bitstream will not inherit this property. A fidelity embedded final bitstream will be produced via Q discrete rate allocation layers.

9.7 Final Bitstream Assembly

This independent block compression system offers many useful final bitstream assembly formats. The P blocks are ordered from the low to high frequency bands. If $p_1 < p_2$, then block p_1 is within the same level (or *levels* if it is a spatial block) as p_2, or in a level (or *levels*) lower in frequency. Ordering within a level for our purposes is arbitrary. Let P_l be defined such that the first P_l blocks consist of the minimal set of blocks holding all data from all bands up to decomposition level l.

Let $\mathcal{G}$ represent a series of bits forming the global header holding the image size and other parameters. Let $\mathcal{D}_{p,q}$ represent the packet for block p, layer q. The packet holds a fixed size header, and bits $B_{p,q-1}+1$ through $B_{p,q}$ from sub-bitstream $\mathcal{B}_p$, with $B_{p,-1} = 0$.

A simple assembly mode for $Q = 1$ with resolution embedding is $\mathcal{G}\ \mathcal{D}_{0,0}\ \mathcal{D}_{1,0}\ \ldots\ \mathcal{D}_{P-1,0}$. With $Q = 2$ rate layers, the bitstream can be assembled so that the entire image is reconstructed to layer 0, then layer 1 with $\mathcal{G}\ \mathcal{D}_{0,0}\ \mathcal{D}_{1,0}\ \ldots\ \mathcal{D}_{P-1,0}\ \mathcal{D}_{0,1}\ \mathcal{D}_{1,1}\ \ldots\ \mathcal{D}_{P-1,1}$. Or, the quarter resolution image can be completely transmitted first, with $\mathcal{G}\ \mathcal{D}_{0,0}\ \mathcal{D}_{1,0}\ \ldots\ \mathcal{D}_{P_{L-1}-1,0}\ \mathcal{D}_{0,1}\ \mathcal{D}_{1,1}\ \ldots\ \mathcal{D}_{P_{L-1}-1,1}\ \mathcal{D}_{P_{L-1},0}\ \mathcal{D}_{P_{L-1}+1,0}\ \ldots\ \mathcal{D}_{P-1,0}\ \mathcal{D}_{P_{L-1},1}\ \mathcal{D}_{P_{L-1}+1,1}\ \ldots\ \mathcal{D}_{P-1,1}$.

9.8 Experimental Results

Table 9.1 shows coding results for the 512 by 512 grayscale *lena*, *goldhill* and *barbara* images. Table 9.2 shows results for the JPEG 2000 grayscale test images *aerial2*, *bike*, *càfe* and *woman*. In each table, the reconstructed image distortion in dB is tabulated for several rates and images. In the tables, the codec names are SPIHT for HBC0 or SB-SPIHT ($K = 0$), HBC1 for the Hybrid Codec with

Codec, Decomposition	Rate (bpp) 0.0625	0.125	0.25	0.5	1.0	2.0
			lena			
SPIHT,D	27.15	30.25	33.46	36.68	39.88	44.15
HBC4,D	27.14	30.31	33.52	36.77	39.94	44.33
HBC4,P1	27.07	30.29	33.62	36.88	39.93	44.21
HBC3,D	27.16	30.32	33.54	36.77	39.97	44.36
HBC3,P1	27.10	30.30	33.63	36.87	39.96	44.24
HBC3,P2	26.93	30.19	33.43	36.55	39.62	43.87
SPECK,D	27.25	30.36	33.56	36.79	39.98	44.37
SPECK,P1	27.19	30.34	33.66	36.90	39.97	44.26
SPECK,P2	27.03	30.25	33.45	36.57	39.64	43.88
			barbara			
SPIHT,D	22.74	24.39	26.92	30.71	35.78	41.82
HBC4,D	22.72	24.49	27.25	31.07	36.05	42.16
HBC4,P1	22.74	24.92	27.86	31.92	36.89	42.52
HBC3,D	22.87	24.64	27.46	31.24	36.16	42.21
HBC3,P1	22.86	25.11	28.08	32.09	37.03	42.60
HBC3,P2	23.05	25.70	28.69	32.42	37.02	42.52
SPECK,D	22.89	24.70	27.48	31.26	36.18	42.22
SPECK,P1	22.90	25.19	28.10	32.12	37.05	42.61
SPECK,P2	23.08	25.76	28.73	32.45	37.03	42.54
			goldhill			
SPIHT,D	25.93	27.90	29.91	32.40	35.69	40.83
HBC4,D	25.91	27.93	30.08	32.61	35.95	41.06
HBC4,P1	25.89	27.93	30.11	32.78	36.13	41.14
HBC3,D	25.93	27.95	30.10	32.68	35.99	41.10
HBC3,P1	25.89	27.95	30.19	32.85	36.17	41.19
HBC3,P2	25.86	28.02	30.24	32.81	36.04	41.04
SPECK,D	26.07	28.00	30.13	32.71	36.01	41.13
SPECK,P1	26.02	28.00	30.24	32.87	36.19	41.20
SPECK,P2	25.96	28.06	30.30	32.84	36.07	41.07

Table 9.1: Hybrid Block Codec performance on standard test images.

$K = 1$, HBC2 for the Hybrid Codec with $K = 2$, and SPECK for SB-SPECK ($K > L$). Decomposition names are D for dyadic, and P1 and P2 for the wavelet packet decompositions defined in Sec. 9.3. For each rate and image the best result is highlighted. For convenience and clarity, Tables 9.3 and 9.4 give a subset of the results, showing the performance for just the dyadic decomposition.

Of the JPEG 2000 images, each is 2048 by 2560, except for *aerial2*, which is 2048 by 2048. A five level decomposition with the 9/7 biorthogonal wavelet [76] was

Codec, Decomposition	Rate (bpp) 0.0625	0.125	0.25	0.5	1.0	2.0
			aerial2			
SPIHT,D	23.88	25.82	27.86	29.99	32.70	37.29
HBC4,D	23.91	25.88	28.02	30.21	32.84	37.47
HBC4,P1	23.90	25.88	28.03	30.26	32.90	37.45
HBC3,D	23.95	26.00	28.15	30.30	32.87	37.51
HBC3,P1	23.95	26.00	28.16	30.36	32.94	37.49
HBC3,P2	23.96	26.05	28.24	30.36	32.87	37.36
SPECK,D	24.07	26.10	28.21	30.33	32.89	37.52
SPECK,P1	24.07	26.10	28.21	30.39	32.96	37.51
SPECK,P2	24.08	26.14	28.28	30.39	32.88	37.38
			bike			
SPIHT,D	22.28	24.95	28.31	32.23	36.84	42.73
HBC4,D	22.42	25.11	28.41	32.30	36.98	42.89
HBC4,P1	22.56	25.42	28.77	32.56	36.95	42.64
HBC3,D	22.49	25.16	28.42	32.32	37.02	42.94
HBC3,P1	22.65	25.49	28.79	32.58	37.00	42.69
HBC3,P2	22.88	25.58	28.60	32.10	36.39	42.11
SPECK,D	22.64	25.23	28.46	32.35	37.05	42.97
SPECK,P1	22.80	25.57	28.83	32.61	37.02	42.71
SPECK,P2	23.04	25.66	28.64	32.13	36.42	42.13
			càfe			
SPIHT,D	18.36	19.97	22.20	25.71	30.77	37.86
HBC4,D	18.39	20.09	22.33	25.80	30.88	37.98
HBC4,P1	18.40	20.15	22.55	26.07	30.97	37.63
HBC3,D	18.43	20.17	22.43	25.84	30.93	38.04
HBC3,P1	18.43	20.25	22.66	26.13	31.02	37.69
HBC3,P2	18.51	20.34	22.70	25.92	30.45	36.88
SPECK,D	18.50	20.25	22.50	25.89	30.97	38.07
SPECK,P1	18.51	20.33	22.74	26.17	31.06	37.72
SPECK,P2	18.59	20.42	22.77	25.96	30.48	36.90
			woman			
SPIHT,D	24.64	26.55	29.05	32.65	37.53	43.00
HBC4,D	24.71	26.72	29.27	32.87	37.69	43.25
HBC4,P1	24.71	26.78	29.44	33.07	37.77	43.24
HBC3,D	24.82	26.77	29.35	32.94	37.74	43.34
HBC3,P1	24.83	26.86	29.53	33.14	37.82	43.33
HBC3,P2	24.98	26.94	29.47	32.98	37.59	43.01
SPECK,D	25.03	26.90	29.48	33.04	37.80	43.40
SPECK,P1	25.04	26.98	29.66	33.24	37.87	43.39
SPECK,P2	25.21	27.06	29.60	33.07	37.64	43.06

Table 9.2: Hybrid Block Codec performance on JPEG 2000 test images.

Codec	Rate (bpp) 0.0625	0.125	0.25	0.5	1.0	2.0
	lena					
SPIHT	27.15	30.25	33.46	36.68	39.88	44.15
HBC4	27.14	30.31	33.52	36.77	39.94	44.33
HBC3	27.16	30.32	33.54	36.77	39.97	44.36
SPECK	27.25	30.36	33.56	36.79	39.98	44.37
	barbara					
SPIHT	22.74	24.39	26.92	30.71	35.78	41.82
HBC4	22.72	24.49	27.25	31.07	36.05	42.16
HBC3	22.87	24.64	27.46	31.24	36.16	42.21
SPECK	22.89	24.70	27.48	31.26	36.18	42.22
	goldhill					
SPIHT	25.93	27.90	29.91	32.40	35.69	40.83
HBC4	25.91	27.93	30.08	32.61	35.95	41.06
HBC3	25.93	27.95	30.10	32.68	35.99	41.10
SPECK	26.07	28.00	30.13	32.71	36.01	41.13

Table 9.3: Hybrid Block Codec performance on standard test images for the dyadic decomposition only.

used for these experiments. A single rate layer was used for each result presented here. All spatial and subband blocks are 64 by 64 coefficients. The codec used to generate these results is implemented as a module in the JPEG 2000 verification model test software, version 7.1.

In the results in Tables 9.1 and 9.2, the performance of Subband Block SPECK is better than Spatial Block SPIHT for all images and all rates. This unexpected result is due to binary uncoded SPECK compression that is better than previously reported [18]. The results for HBC fall close to, but slightly below the SPECK results. For the *barbara* image, which has greater than normal high frequency content, the wavelet packet decompositions offers a significant improvement. At low rates, for the JPEG 2000 test images, the wavelet packet decomposition offers a modest increase in PSNR.

No arithmetic coding was used on the significance test or any symbols produced by the SPIHT or SPECK algorithms for these results. Back-end arithmetic coding using contexts and joint encoding generally improves SPIHT and SPECK by about 0.5 dB [16].

Codec	Rate (bpp) 0.0625	0.125	0.25	0.5	1.0	2.0
	aerial2					
SPIHT	23.88	25.82	27.86	29.99	32.70	37.29
HBC4	23.91	25.88	28.02	30.21	32.84	37.47
HBC3	23.95	26.00	28.15	30.30	32.87	37.51
SPECK	24.07	26.10	28.21	30.33	32.89	37.52
	bike					
SPIHT	22.28	24.95	28.31	32.23	36.84	42.73
HBC4	22.42	25.11	28.41	32.30	36.98	42.89
HBC3	22.49	25.16	28.42	32.32	37.02	42.94
SPECK	22.64	25.23	28.46	32.35	37.05	42.97
	càfe					
SPIHT	18.36	19.97	22.20	25.71	30.77	37.86
HBC4	18.39	20.09	22.33	25.80	30.88	37.98
HBC3	18.43	20.17	22.43	25.84	30.93	38.04
SPECK	18.50	20.25	22.50	25.89	30.97	38.07
	woman					
SPIHT	24.64	26.55	29.05	32.65	37.53	43.00
HBC4	24.71	26.72	29.27	32.87	37.69	43.25
HBC3	24.82	26.77	29.35	32.94	37.74	43.34
SPECK	25.03	26.90	29.48	33.04	37.80	43.40

Table 9.4: Hybrid Block Codec performance on JPEG 2000 test images for the dyadic decomposition only.

9.9 Color Image Coding

The systems described in the previous three chapters, SB-SPIHT, SB-SPECK and HBC, were initially designed for and tested on grayscale images. With some modification, each system can also be applied to color images. This section will discuss the additions needed to work with color images and present color image coding results. With the right parameter selection, HBC is identical to SB-SPIHT or SB-SPECK, so all color image discussion and results are presented in this chapter.

A color image can be viewed as three component images, or an image in which each pixel value is a triad in some color space. Most uncompressed images are stored in the Red-Green-Blue (RGB) primary color space. A simple way to apply a grayscale codec to a color image is to simply encode each component independently as if it were a grayscale image. However, the red, green and blue components are generally highly correlated, so a decorrelating transform, or conversion to another

color space, is used to improve compression. Color spaces used for compression have a luminance and two chrominance components. The chrominance components tend to have much less information than the luminance component.

For the results in this section, the YCbCr color space that will be used by the JPEG 2000 standard is used, but any other luma-chroma color space could be used as well [59]. With R, G and B representing the reb, green and blue intensities of a pixel in the RGB color space, and Y, C_b and C_r representing the corresponding values in the YCbCr space, the following matrix equations define the conversions between the representations. The forward conversion is,

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.29900 & 0.58700 & 0.11400 \\ -0.16875 & -0.33126 & 0.50000 \\ 0.50000 & -0.41869 & -0.08131 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{9.1}$$

and the reverse is,

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & -0.00001 & 1.40200 \\ 1.0 & -0.34413 & -0.71414 \\ 1.0 & 1.77200 & 0.00004 \end{bmatrix} \begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix}. \tag{9.2}$$

None of the color components are subsampled. The YCbCr space has the important feature that its Y, or luminance, component is the compatible monochrome (grayscale) version of the color image.

After the color space conversion, each component is treated independently for most of the processing. Each component is treated just as a whole grayscale image in the HBC system during the wavelet transform, block partitioning, and encoding stages. Each component undergoes a subband transform, and each transformed component is partitioned into spatial and subband blocks just as if it were a complete image. Each block in each component is encoded using the SPIHT or SPECK algorithm, as appropriate. At the rate allocation stage, the components are no longer treated independently. Rate allocation and packetization are performed for all of the blocks just as if they all came from the same grayscale image.

With color comes a new consideration during the packetization stage. With

Codec, Decomposition	Rate (bpp) 0.0625	0.125	0.25	0.5	1.0	2.0
	color lena					
SPIHT,D	22.41	25.92	28.77	31.27	33.40	35.59
HBC4,D	22.48	25.92	28.83	31.35	33.50	35.68
HBC4,P1	22.16	25.80	28.84	31.37	33.54	35.66
HBC3,D	22.44	25.94	28.89	31.40	33.54	35.72
HBC3,P1	22.13	25.84	28.89	31.41	33.58	35.71
HBC3,P2	20.96	25.58	28.67	31.24	33.39	35.48
SPECK,D	22.87	26.18	29.00	31.46	33.56	35.74
SPECK,P1	22.56	26.06	29.01	31.45	33.60	35.73
SPECK,P2	21.58	25.81	28.80	31.28	33.41	35.51

Table 9.5: Hybrid Block Codec performance on the color lena image. For each rate the best performance is highlighted.

the grayscale image codecs, the final bitstream could be fidelity progressive, resolution progressive, or some combination. With color, there is the additional choice of color progression. The encoded data for the luminance (Y) component can be sent first, before the data for the chrominance (C_b, C_r) components. Then, a complete grayscale rendition of the image has precedence in the bitstream. Or, fidelity or resolution progression can be used, with no precedence given to particular components. An important feature of this system is that the information for the luminance component and for the chrominance components are kept in separate packets so it is easy to customize the final bitstream.

Table 9.5 shows results for coding the *color lena* image. As in the previous section, Table 9.6 shows just the results using the dyadic transform. Five decomposition levels, the 9/7 filter set, a single rate layer at the desired rate and 64 by 64 spatial blocks were used for these results. For the color image, as with the grayscale images, the performance of SB-SPECK is better than that of SB-SPIHT for all rates. The HBC results all fall between those of SB-SPIHT and SB-SPECK. No back-end entropy coding is used in HBC. The addition of arithmetic coding or Huffman coding would improve the results at a computational expense, and may reverse the trend of the results.

There is more than one color version of the *lena* image in use. In the version

Codec	Rate (bpp) 0.0625	0.125	0.25	0.5	1.0	2.0
	color lena					
SPIHT	22.41	25.92	28.77	31.27	33.40	35.59
HBC4	22.48	25.92	28.83	31.35	33.50	35.68
HBC3	22.44	25.94	28.89	31.40	33.54	35.72
SPECK	22.87	26.18	29.00	31.46	33.56	35.74

Table 9.6: Hybrid Block Codec performance on the color lena image for the dyadic decomposition only. For each rate the best performance is highlighted.

used for these results, the red component values of the first 4 pixels of the first row are 218, 218, 214, and 214. Another version having 226, 226, 223 and 223 for these values is used less often and tends to result in better compression performance.

CHAPTER 10
No List SPIHT

10.1 Introduction

The EZW [15] and SPIHT algorithms [16] are fast and effective techniques for image compression. Both use spatial trees which can exploit magnitude correlation across bands of the decomposition. Each generates a fidelity progressive bitstream by encoding, in turn, each bitplane of a quantized dyadic subband decomposition. Both use significance tests on sets of coefficients combined with set partitioning to efficiently isolate and encode high magnitude coefficients.

An important difference between EZW and SPIHT is in their set partitioning rules. SPIHT has an additional partitioning step in which a descendant (type A) set is split into four individual child coefficient sets and a granddescendant (type B) set.

EZW explicitly performs a breadth first search of the hierarchical trees, moving from the coarse to fine bands. Though it is not explicit, SPIHT does a roughly breadth first search as well. After partitioning a granddescendant set, SPIHT places the four new descendant sets at the *end* of the LIS, or list of insignificant sets. It is the appending to the LIS that results in the approximate breadth first traversal. Breadth first scanning, as opposed to depth first scanning, improves performance because coefficients more likely to be significant are tested first.

It is more complex to implement a zerotree codec that does a breadth first search for significant coefficients. The codec needs to determine whether each node of the tree encountered during the search must be tested and coded or can be skipped because it is a member of an insignificant set. It would be inefficient to repeatedly look up the tree for an ancestor that is the root of a zerotree. Thankfully, other means have been developed.

SPIHT uses list structures to keep track of which sets must be tested. However, the use of lists in SPIHT causes a data dependent, variable memory requirement, and the need for memory management as list nodes are added, removed and moved.

At low rates, and thus low fidelity, the lists are short and are an efficient means to store the state of the set partitioning. At high rates and high fidelity, there can be as many list nodes as coefficients. This is an inefficient use of memory.

This chapter describes a new image codec called No List SPIHT (NLS) that uses the set partitioning rules of SPIHT, and does an explicit breadth first search without using lists [81]. State information is kept in a fixed size array that corresponds to the array of coefficient values, with about four bits per coefficient to enable fast scanning of the bitplanes.

In NLS, instead of searching up the tree to find *predictable insignificance*, special markers are placed in the state table on certain lower nodes of insignificant trees when the trees are created. These sparse markers are updated when new insignificant trees are formed by partitioning. With this sparse marking scheme, large sections of the image are skipped at once as the breadth first scan moves through the lower nodes of the spatial trees.

Lin, Burgess, Ng and Bouzerdoum have developed listless zerotree codecs for images [28, 29] and video [30] that also make use of fixed size state tables. However, their codecs perform a depth first search of the trees. In the system presented here, a new set marking scheme is used to facilitate breadth first searching. Shively, Ammicht and Davis have also developed a similar flexible listless SPIHT implementation [31]. Their system relies on a bitstream reordering to avoid the need for an efficient scan order.

Järvi, Lehtinen and Nevalainen have developed a codec based on SPIHT called Variable Quality SPIHT (vqSPIHT) [32]. The vqSPIHT codec is designed for coding large digital mammography images and supports Region Of Interest (ROI) coding. In vqSPIHT, the state information about which coefficients would be on the LSP or LIP is kept in a single 2-D array, which is scanned. Information about the state of the insignificant sets is also kept in a separate 2-D array. However, a linked list structure is implemented in the memory space of the insignificant set 2-D array. The insignificant set 2-D array serves to preallocate all of the memory needed for each node along with a pointer to the next node in the list. While SPIHT has separate passes for the list of insignificant pixels and the list of significant pixels (refinement),

a	a_3	a_2	a_1	a_0				
b	b_3	b_2	b_1	b_0				
c	a_3	b_3	a_2	b_2	a_1	b_1	a_0	b_0

Figure 10.1: Bit interleaving for one-dimensional indexing.

vqSPIHT combines these operations for computational efficiency at some expense of coding efficiency between bitplanes.

The Distortion Limited Wavelet Image Codec (DLWIC) developed by Lehtinen [33] is based on the EZW algorithm and keeps all state information in an array instead of in lists. DLWIC uses a single depth first scan of the hierarchical trees for each bitplane, and thus does not perform separate sorting and refinement passes. Unlike EZW and SPIHT, DLWIC combines the three trees that are at different spatial orientations (LH, HL and HH), but are rooted at the same level and spatial location. The zerotrees are then three times as large. This new structure offers the possibility of exploiting magnitude correlation across bands at different spatial orientations. However, it is reported that this offers little improvement in performance [33].

In NLS, efficient skipping of blocks of insignificant coefficients is accomplished using a recursive zig-zag, or Morton scan [26], image indexing scheme. Instead of indexing the array of coefficients using two indices the two-dimensional array is moved to a one-dimensional array. This particular format offers several computational and organizational advantages. Seetharaman and Zavidovique [27] have used this same linear coefficient ordering in image segmentation applications.

Like SPIHT, NLS is a bitplane codec, and generates a fidelity embedded bitstream. As it receives and processes the bitstream, the decoder updates the coefficient values when newly significant coefficients are found, and when significant coefficients are refined. Because these features are maintained, NLS is a drop-in replacement for the core SPIHT algorithm in the SB-SPIHT codec of Chapter 7 and the HBC of Chapter 9.

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

Figure 10.2: Linear indexing with $R = 8$, $C = 8$, and two subband levels with bands delimited by thick lines.

10.2 Linear Indexing

The linear indexing technique uses a single number to represent the index of a coefficient instead of two. Let $R = C = 2^N$ be the number of rows and columns of the square image, and let r and c be zero-based row and column indices. Represent the row index in binary $r = [r_{L_r-1},\ldots,r_1,r_0]$, where each of the r_n is a bit, and do the same for the column index. For an index (r, c) the linear index is defined by $i = [r_{L_r-1},\ c_{L_r-1},\ \ldots,\ r_1,\ c_1,\ r_0,\ c_0]$. The bits of r and c are simply interleaved, as seen in Fig. 10.1. The linear index i ranges from 0 to $I - 1$, where $I = RC$. Figure 10.2 shows an example of this indexing scheme. Notice that the children on a spatial tree have 4 consecutive indices, the grandchildren have 16 consecutive indices, and so on. NLS will take advantage of this property when skipping past lower nodes of insignificant trees.

Another important property of the linear index is that it efficiently supports the operations on coefficient positions needed for tree-based algorithms with one operation instead of two, assuming the usual subband data arrangement of Fig. 10.2. Also, the linear index naturally facilitates a breadth first search of the hierarchical trees.

The linear index is also known as the Morton ordering [26] or recursive Z scan. Morton used this ordering instead of a raster scan in the Canada Geographic

Information System to visit each area of a map in sequence. The scan follows a Z pattern at various scales, from 2 by 2 groups of coefficient up to the quadrants of the whole image. This scan is better than the raster scan at preserving locality.

Given either coordinates (r, c) or i, suppose we must find the index of the first child (indicated with subscript c) in the spatial tree. Using row and column indices, we need $r_c = 2r$, and $c_c = 2c$. For the linear index the single operation is $i_c = 4i$. To find the location of the parent coefficient (indicated with subscript p) using row and column indices, we need $r_p = \lfloor r/2 \rfloor$, and $c_p = \lfloor c/2 \rfloor$. With the linear index, the single required operation is $i_p = \lfloor i/4 \rfloor$. Iterating over four siblings in a tree is another common operation. With row and column indices, the iteration can be represented by $r_n = r, r+1$ and $c_n = c, c+1$. Using the linear index, only a single level of iteration is needed, using $i_n = i, i+1, i+2, i+3$. All multiplication here (and throughout) is by integral powers of 2 and can be implemented by bit shifting.

Image data is usually stored in raster order, so index conversion must be done for each coefficient once by the encoder and once by the decoder. The interleaving and deinterleaving required to convert between index modes is trivial in custom hardware, but not directly supported by general purpose CPUs. For an efficient software conversion, NLS uses a fixed 256 entry lookup table that maps one byte to two bytes, with zero bits padded between the original bits. This bit spreader table combined with shift and bitwise OR operations makes the conversion simple and fast [27].

10.3 No List SPIHT Overview

NLS uses the same set structures and partitioning rules as SPIHT. The trees are tested for significance breadth first. Significance tests are made in a different order than SPIHT because SPIHT performs significance tests roughly breadth first, while NLS performs the tests strictly breadth first. Because the set splitting rules are the same, each encoder produces the exact same output bits, though in a different order. For the coefficients in the 8 by 8 example image given in [15], NLS and SPIHT happen to produce identical bitstreams.

There are three passes per bitplane. First, the insignificant pixel (IP) pass

which corresponds to the SPIHT LIP pass tests each lone insignificant pixel for significance. Then the significant set (IS) pass, like the SPIHT LIS pass, tests each multiple-pixel set for significance, splitting and repeating as needed. Finally, the refinement (REF) pass, like the SPIHT LSP pass, refines pixels found significant in previous bitplanes.

10.4 Storage

The number of coefficients in the DC band is $I_{dc} = R_{dc}C_{dc}$, where $R_{dc} = R\,2^{-L}$, $C_{dc} = C\,2^{-L}$, and L is the number of subband decomposition levels. The coefficients are stored in a single array of length I. For convenience, we will refer to the magnitude part with the array `val` and the sign part with array `sign`. The state table is an array, named `mark`, of length I, with 4 bits per coefficient. There is a one-to-one correspondence between `val` and `mark`.

Embedded bitplane zerotree encoders, like EZW and SPIHT, can optionally trade memory for computation by precomputing and storing the maximum magnitude of all possible descendant and granddescendant sets [82]. For NLS, the precomputed maximum descendant magnitude array, `dmax`, has length $I/4$, and the maximum granddescendant magnitude array, `gmax`, has length $I/16$. These arrays can be eliminated at the expense of repeated searching over insignificant trees for significant coefficients at the encoder.

If each subband coefficient is stored in W bytes, the total bulk storage memory needed is: RCW for the subband data, $RCW/4$ for the maximum descendant array, $RCW/16$ for the maximum granddescendant array, and $RC/2$ (half byte per pixel) for the state table. For a 512 by 512 image using 2 bytes per coefficient, and using the optional precomputed maximum descendant arrays, this is 800k bytes, or 56% more than is needed for the image alone. This is all of the significant memory needed for this algorithm. The amount of memory required is fixed given the size of the image. This does not count the memory required for the subband transform, but this part of the system can be handled efficiently by a rolling line-based transform [19, 20].

10.5 State Table Markers

The following markers are placed in the 4 bit per coefficient state table, `mark`, to keep track of the current set partitions. Each element of `mark`, if set, indicates something about the corresponding element in the `val` image array. Each marker and its meaning is listed below.

M_P Each of these first four markers of this form are for a lone pixel.

- MIP The pixel is insignificant or untested for this bitplane.
- MNP The pixel is newly significant so it will not be refined for this bitplane.
- MSP The pixel is significant and will be refined in this bitplane.
- MCP Like MIP, but applied during partitioning in the IS pass so the new pixel set will be tested for significance immediately during the same IS pass, while MIP pixels are skipped.

MD The pixel is the first (lowest index) child in a set consisting of all descendants of its parent.

MG The pixel is the first (lowest index) grandchild in a set consisting of all grand-descendants of its grandparent pixel, but not including the grandparent or the children of the grandparent.

MN_ The following markers of this form are used on the leading node of each lower level of an insignificant tree. As the image is scanned, these markers indicate that the next block of pixels is insignificant.

- MN2 The pixel is the first grandchild of a MD set. This pixel and its 16 (4 by 4) 1st cousins can be skipped.
- MN3 The pixel is the first great grandchild of a MD or MG set. This pixel and its 64 (8 by 8) 2nd cousins can be skipped.

  ...
- MN6 The pixel is the first 6th generation descendant of a MD or MG set. This pixel and its 4096 (64 by 64) 5th cousins can be skipped.

The markers MD and MG are similar in meaning to the SPIHT set types A and B. These set markers are associated directly with a pixel that is in the set, while in SPIHT, types A and B are associated with the root pixel of a set, which is not actually in the set.

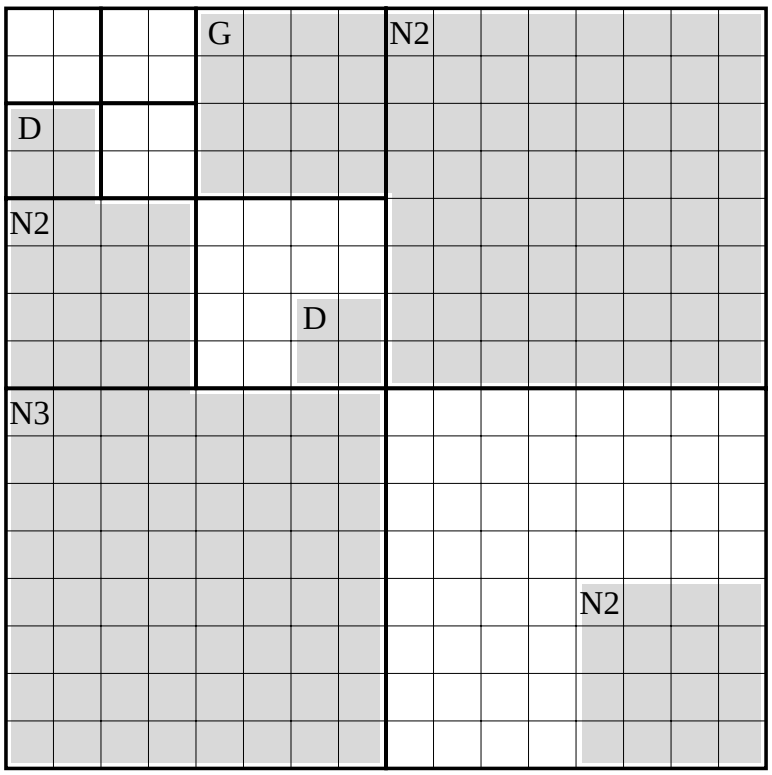

Figure 10.3: This diagram shows a partial example of an NLS state array. The initial "M" of each marker is omitted in the diagram for space. All of the pixels in two descendant sets and one grand-descendant set are shaded. The diagram shows all markers needed for these shaded pixels, but not any other markers. The first child pixel of each descendant set is labeled MD and the first grand-child pixel of the granddescendant set is labeled MG. Lower levels of the sets are labeled with MN2 and MN3 markers so that can be skipped by the scan.

The edge markers, MN_, are used to keep track of which pixels are members of insignificant sets with roots at higher levels of the tree. With linear indexing, when a MN2 marker is reached during a scan we simply advance the scan index by 16. When a MN3 marker is reached we advance 64, etc. An example of a state array partially populated with these markers is in Fig. 10.3. Figure 10.4 shows the scan order, which comes from the linear indexing, overlayed on the same basic diagram.

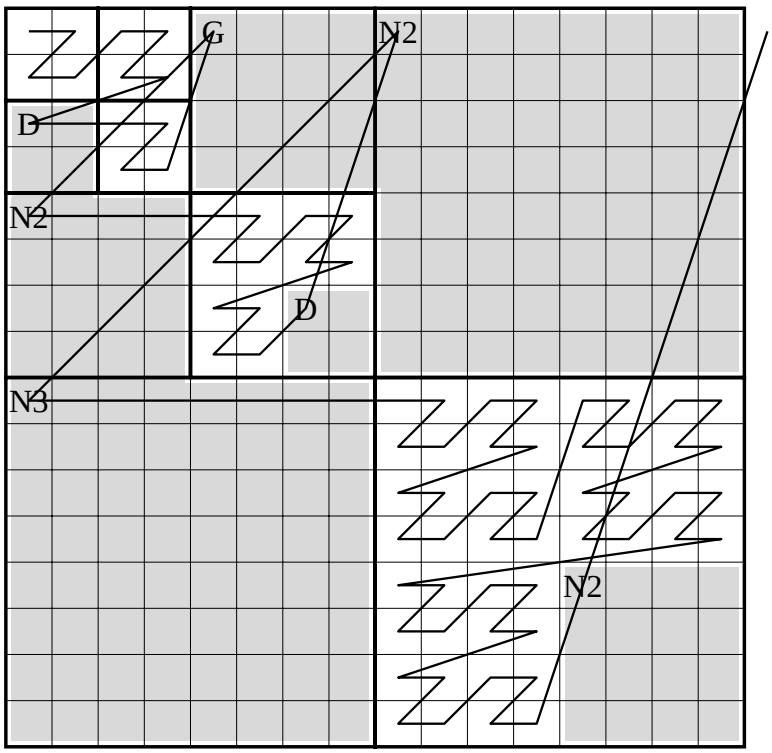

Figure 10.4: As in Fig. 10.3, this diagram shows a partial example of an NLS state array, but also depicts the scan order. The scan visits coefficients in the order defined by the linear index. Groups of predictably insignificant coefficients are efficiently skipped when MN type markers are encountered. For this example, the unshaded coefficients are assumed to have been split down to individual coefficients. They are no longer part of a larger set.

10.6 Initialization

A dyadic subband transform is performed on the image with about $L = 5$ levels, and the floating point transform coefficients are quantized to integers. Next, the transform coefficients are read into the linear array `val`. For each (r, c), find i by bit interleaving and move the coefficient to `val`$[i]$. The mean of the DC band is removed and transmitted.

If encoding, the two maximum descendant magnitude arrays are computed. The maximum magnitude of the descendant set rooted at coefficient i is `dmax`$[i]$, and the maximum magnitude of the granddescendant set rooted at coefficient i is `gmax`$[i]$.

This data is computed in advance by scanning the first quarter of the linear array backwards and using the equations $\texttt{gmax}[i] = \max(\texttt{dmax}[4i], \texttt{dmax}[4i+1], \texttt{dmax}[4i+2], \texttt{dmax}[4i+3])$ and $\texttt{dmax}[i] = \max(\texttt{val}[4i], \texttt{val}[4i+1], \texttt{val}[4i+2], \texttt{val}[4i+3], \texttt{gmax}[i])$. Only coefficients in the first quadrant have children in the trees and only coefficients in the first quadrant of the first quadrant have grandchildren. Because of the nature of the linear scan, the first $I/16$ coefficients in the linear array are the coefficients of the first quadrant of the first quadrant. So, zero is substituted for $\texttt{gmax}[i]$ when $i \geq I/16$ because these nodes have no granddescendants.

These arrays do not need to hold the true maximum magnitude. It suffices if the significance level of the array is the maximum significance level of any of the descendants. The significance level is always an integral power of two, so these arrays are actually computed via bitwise OR instead of the max function.

The most significant non-zero bitplane, B, is found by scanning the DC band and a small section of the array `dmax` and recording the maximum MSB index of each coefficient scanned. Only the first $I/4^L$ coefficients of the array `dmax` must be checked for the most significant bitplane because these coefficients each holds the maximum significant bitplane for a full size hierarchical tree and together these hierarchical trees cover all of the coefficients. The value B is transmitted to the receiver.

The function `push`, in pseudo-code below, is used to insert MN_ markers where needed to the lowest tree level after a new descendant set is created.

```
define push(i)
  mark[4i] = MN2;  mark[16i] = MN3;  ···
```

A 5 level descendant tree has 1364 pixels, but only 4 need to be marked by the `push` function. When scanning the image, only the top level MD marker and 4 MN_ markers associated with the tree will be encountered. So, a great number of predictably insignificant coefficients are skipped with little processing.

The state table is initialized by marking each DC coefficient with MIP, and each full-size spatial tree with MD. For each initial tree and each decomposition level, the leading pixel is marked with an appropriate MN_, as shown in the initialization pseudo-code below. Very few of the coefficients are marked during initialization.

```
for i = 0, ..., I_dc - 1
   mark[i] = MIP
for i = I_dc, ..., 4I_dc
   mark[i] = MD; push(i)
```

A small constant lookup table, `skip`, tells how many coefficients to skip if a marker is encountered and no processing needs to be done for that section in this pass. Values of `skip` are `skip`[M_P] = 1, `skip`[MD] = 4, `skip`[MG] = 16, `skip`[MN2] = 16, `skip`[MN3] = 64, `skip`[MN4] = 256, `skip`[MN5] = 1024, etc. Another table, `isskip`, is used in the IS pass. It is the same as `skip`, except for `skip`[MIP] = `skip`[MNP] = `skip`[MSP] = 4. Because of the partitioning rules, if a pixel with one of these markers is encountered during the IS pass, each of its siblings can be skipped.

10.7 No List SPIHT Main Algorithm

The main encoder algorithm in pseudo-code below is performed for each bitplane, b, starting with B and decrementing to 0, or until a bit budget is met. The significance level for each bitplane is $s = 2^b$. Significance checks are always done with bitwise AND instead of greater-than tests. In binary form, s has a single bit set, at bit position b. The test to determine whether the coefficient `val`$[i]$ is newly significant is d = `val`$[i]$ AND s. The coefficient is newly significant if and only if its bth bit is set. If d is non-zero, `val`$[i]$ significant. Tests for significant sets are also done with bitwise AND between s and `dmax`$[\lfloor i/4 \rfloor]$ for a descendant set or `gmax`$[\lfloor i/16 \rfloor]$ for a granddescendant set rooted at i.

The decoder follows the same overall procedure as the encoder with some low level changes. To decode, use `input` instead of `output`, and set the bits and signs of coefficients with bitwise OR instead of testing them with bitwise AND. For example, at some point in the algorithm, for some significance level $s = 2^b$ and coefficient index i, the encoder may find `val`$[i]$ significant because `val`$[i]$ AND s is non-zero. The coefficient `val`$[i]$ is never tested at significance level s if it was found significant at a previous significance level, so it is newly significant if and only if its bth bit is set. Only if this bit is set is $s \leq$ `val`$[i] < 2s$. The decoder will not test

the bth bit, but set it with $\texttt{val}[i] = \texttt{val}[i]$ OR s. Similarly, the refinement bits of $\texttt{val}[i]$ are extracted at the encoder with bitwise AND and set at the decoder with bitwise OR. The decoder performs midtread dequantization for coefficients that are not fully decoded.

Though the pseudo-code below for the main algorithm and above for the `push` function use multiplication and division for clarity, all multiplication and division operations are by integral powers of 2 and are implemented by bit shifting. Further, the addition operations needed during set partitioning are done with bitwise OR. Actual addition is only needed when advancing the main index i.

The `output` function always places a 0 or a 1, a single bit, on the bitstream. Sometimes, its argument variable, d, is not 0 or 1, but the decoder needs only know whether d is zero or non-zero.

INSIGNIFICANT PIXEL PASS

```
i = 0, while i < I                                  start the IP pass
   if mark[i] = MIP                                 insignificant pixel
      d = (val[i] AND s)                            test for significance
      output(d)                                     transmit 0 or 1
      if d ≠ 0
         output(sign[i])                            send the sign
         mark[i] = MNP                              pixel now newly significant
      i = i + 1                                     move to next pixel
   else
      i = i + skip[mark[i]]                         move past set/block
```

INSIGNIFICANT SET PASS

```
i = 0, while i < I                                  start the IS pass
   if mark[i] = MD                                  a set of descendants
      d = (dmax[⌊i/4⌋] AND s)                       test for significance
      output(d)                                     transmit 0 or 1
      if d ≠ 0                                      if the set is significant
         mark[i] = mark[i + 1] = MCP                split into 4 children
         mark[i + 2] = mark[i + 3] = MCP            (will test these next)
```

```
        mark[4i] = MG                                   and the granddescendants
        (no increment here so new MCP pixels processed next)
      else
        i = i + 4                                       move past the siblings in this set
    elseif mark[i] = MG                                 a set of granddescendants
      d = (gmax[⌊i/16⌋] AND s)                          test for significance
      output(d)                                         transmit 0 or 1
      if d ≠ 0                                          if the set is significant
        mark[i] = mark[i + 4] = MD                      split into 4 sets
        mark[i + 8] = mark[i + 12] = MD                 (will test these next)
        (mark the borders of these new sets)
        push(i), push(i + 4), push(i + 8), push(i + 12)
        (no increment here so new MD sets processed next)
      else
        i = i + 16                                      move past the cousins in this set
    elseif mark[i] = MCP                                an insignificant pixel
      d = (val[i] AND s)                                test for significance
      output(d)                                         transmit 0 or 1
      if d ≠ 0                                          if the pixel is significant
        output(sign[i])                                 send the sign of the pixel
        mark[i] = MNP                                   the pixel is now newly significant
      else
        mark[i] = MIP                                   the pixel is insignificant
      i = i + 1                                         move past this pixel
    else
      i = i + isskip[mark[i]]                           move past the set/block
```

REFINEMENT PASS

```
i = 0, while i < I                                      start the refinement pass
    if mark[i] = MSP                                    significant pixel
      output(val[i] AND s)                              transmit 0 or 1
      i = i + 1                                         move past the pixel
```

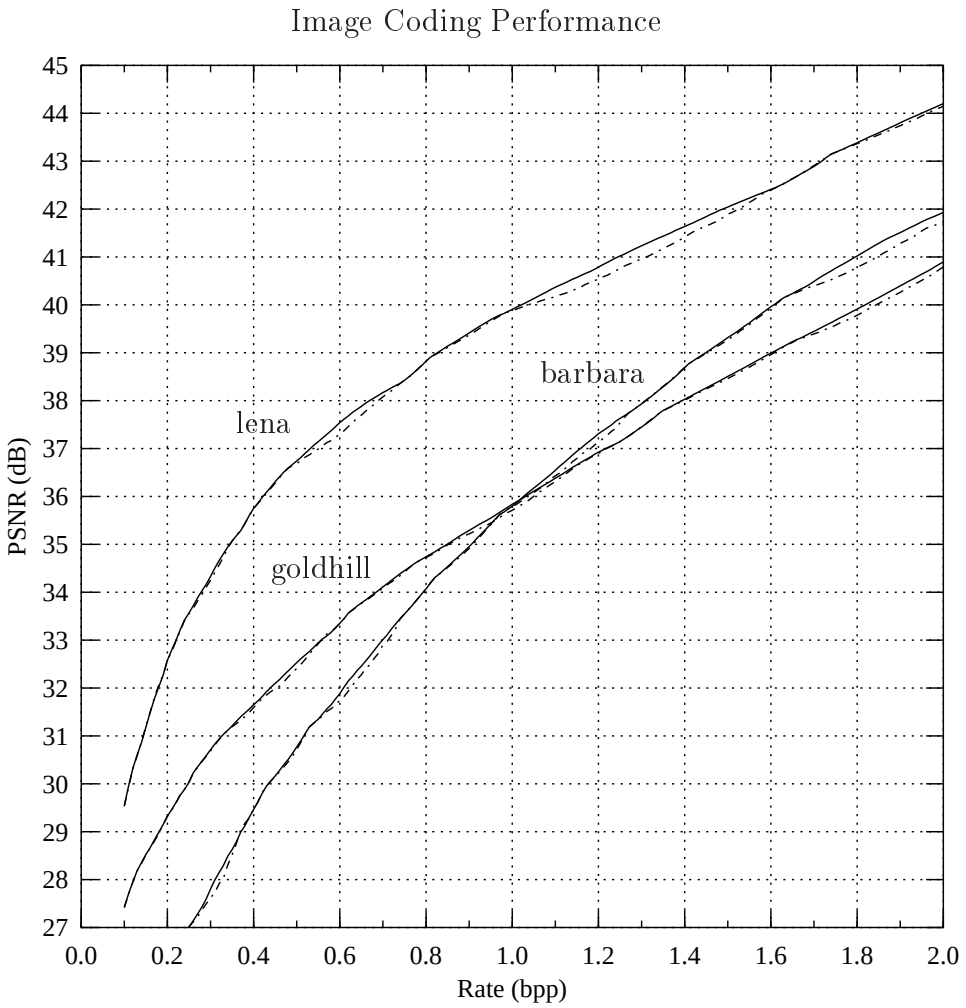

Figure 10.5: Coding performance for binary uncoded SPIHT and NLS on several images.

```
elseif mark[i] = MNP                          newly significant pixel
   mark[i] = MSP                              significant pixel in next plane
   i = i + 1                                  move past the pixel
else
   i = i + skip[mark[i]]                      move past the set/block
```

10.8 Results and Conclusions

Coding results for the 512 by 512 grayscale *lena*, *goldhill* and *barbara* images are plotted in Fig. 10.5. A five level decomposition with the 9/7 biorthogonal wavelet [76] was used for these tests. The solid lines show rate vs. distortion

performance for SPIHT without arithmetic coding. The dash-dot line shows the performance of NLS without arithmetic coding. All decoded images for each plot were recovered from a single fidelity embedded encoded file, truncated at the desired rate.

NLS and SPIHT produce bitstreams with the same bits, though in different order. The SPIHT ordering offers slightly better performance at certain points in the bitstream during the set significance (IS/LIS) passes. During refinement the results are practically identical. At the end of each bitplane, the results are precisely identical.

CHAPTER 11
Image Coding In Small Tiles

11.1 Motivation

In earlier chapters we have been concerned with the coding of very large images. It is practical to encode large images using a cached memory system and image coding systems were developed to take advantage of such an architecture.

Of course, sometimes it is necessary to encode small images as well. Images can be small themselves, or it may be required to break a large image into small tiles before coding. A large image may be broken into tiles before coding so that certain regions can be easily extracted as needed. Coding small tiles of a larger image is nearly the same problem as coding images which are small in the first place. It is convenient if an image compression system works well both on large images and on small images. With this in mind, this chapter investigates the effectiveness of spatial block and subband block codecs in small tiles.

11.2 Small Tiles

When an image $\mathcal{I}$ is to be coded in small tiles it is divided into T independent tiles $\mathcal{I}_t$, $t = 0, \ldots, T-1$. Each tile is a rectangular section of the original image. Each pixel of the original image is in exactly one of the tiles.

The important difference between tiling and spatial block or subband block partitioning, which has been our main focus, is when the division occurs. Tiling is separation performed before the transform, while partitioning is separation occurring after the transform. Because of the spatial domain division of tiling, one of the disadvantages is the appearance of blocking artifacts, especially at low rates.

Each tile is encoded as if it were an image unto itself. So, the first step is the wavelet transform. At this point a potential problem becomes apparent. Some of the subbands will be very small. If the tile size is 64 by 64, and 5 subband decomposition levels are used, then the smallest bands will be 2 by 2, there will be bands that are 4 by 4 and 16 by 16 as well. So, there will be independent subband

blocks of these sizes as well.

With the wavelet transform being performed on each tile instead of the whole image, any image codec will have reduced performance. The wavelet transform is useful because it decorrelates the image. If the transform is restricted to independent tiles then it cannot exploit self-similarity of the image across the tile boundaries.

Beyond the wavelet transform being less efficient, a tree based codec will not have any apparent additional difficulty in this situation. Even in the transform of a small tile, large spatial blocks of trees are present. Subband block codecs, however, are not as well suited in this situation.

There are two reasons that subband block codecs may not work well in small tiles. The first is the block overhead. One type of block overhead is initial information that must be placed on each the sub-bitstream for each block, such as the number of significant bitplanes. There is also packet header overhead. In the packetized systems we have been using, encoded bits from each subband block are placed in a packet with a fixed size header. Having smaller subband blocks means there will be more of them and more packet overhead for the whole image.

Another reason that subband block codecs may not work as well in tiles is they have less of a chance to adapt. Each subband block is coded independently, so back-end adaptive entropy coding source models are reset for each block. Adaptive models are less effective when there is less data to be coded.

11.3 Rate Allocation Across Tiles

The JPEG 2000 verification model test codec, version 5.2, in tiling mode, encodes blocks from all tiles and then allocates rate amongst the blocks to minimize distortion across the entire image, regardless of the tile boundaries. For comparison purposes, this is how the SB-SPIHT and SB-SPECK codecs operate as well. It should be noted, however, that this requires storing encoded data from tile to tile, which may defeat the purpose of tiling. This is done only at the encoder for the sake of rate allocation. The decoder can fully decode and release each tile before moving to the next tile.

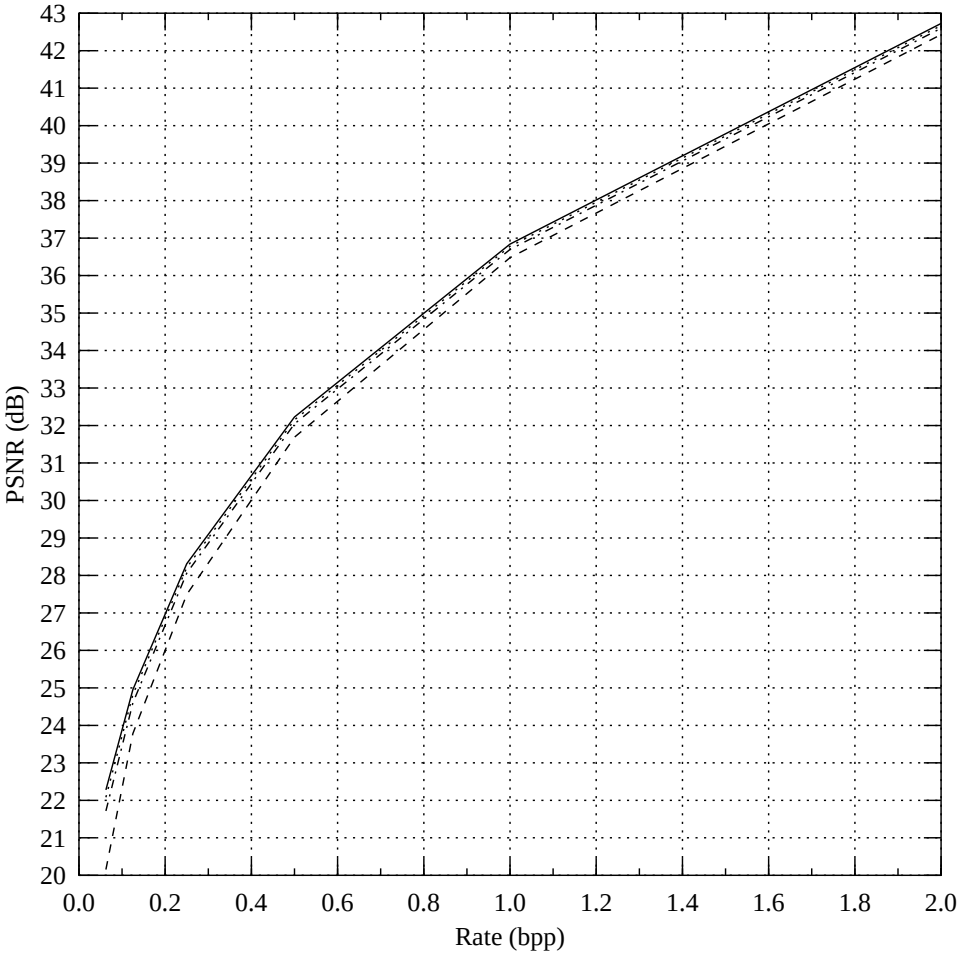

Figure 11.1: This plot shows the performance of SB-SPIHT on the *bike* image for various tile sizes. In order of performance, the solid line is for no tiling (entire image is a single tile); the dotted line is for 256 by 256 tiles; the dot-dash line is for 128 by 128 tiles; and the dashed line is for 64 by 64 tiles.

11.4 Results and Conclusions

The 2560 by 2048 pixel JPEG 2000 grayscale test image *bike* was encoded and decoded at the rates 0.0625, 0.125, 0.25, 0.5, 1.0 and 2.0 bits per pixel. Three codecs were tested, SB-SPIHT, SB-SPECK, and the JPEG 2000 verification model test codec version 5.2, which uses EBCOT [25], a subband block coder. All codecs were tested using a single full-image tile, 256 by 256 tiles, 128 by 128 tiles, and 64 by 64 tiles. In Figs. 11.1, 11.2 and 11.3 the results of these tests are plotted.

The PSNR performance of EBCOT in 64 by 64 tiles is 1–2 dB below its

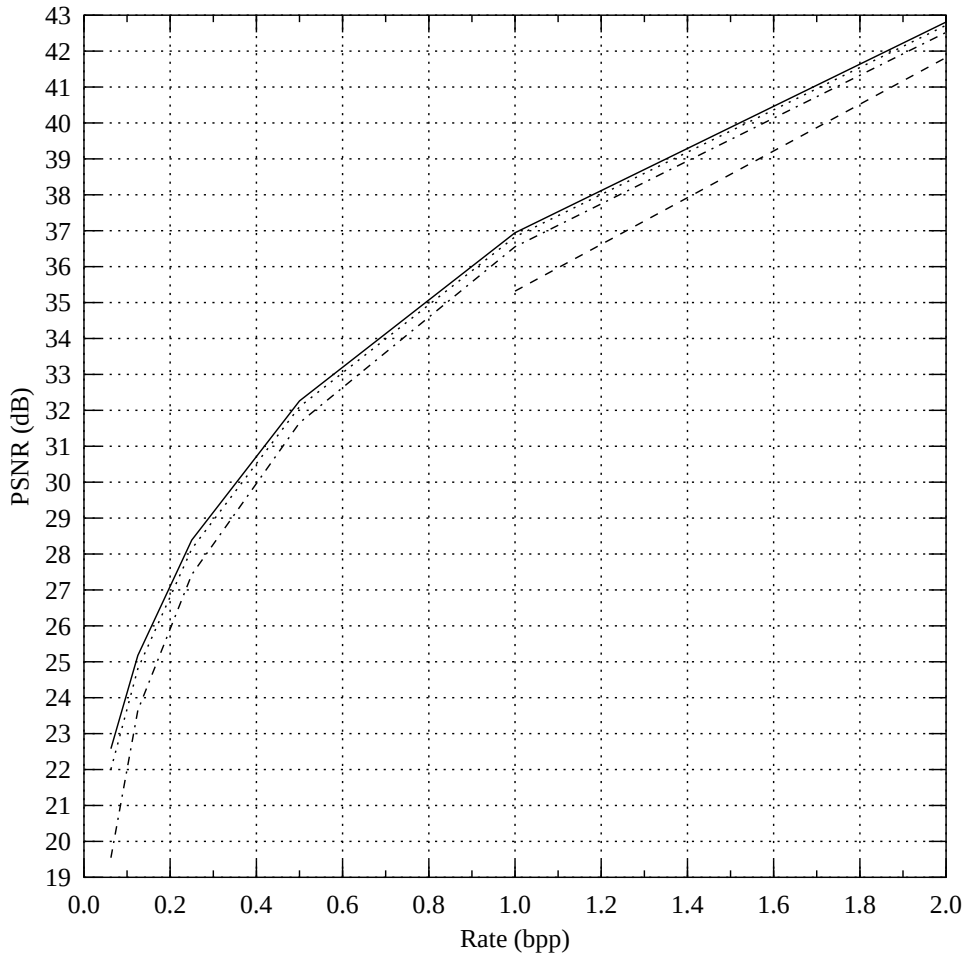

Figure 11.2: This plot shows the performance of SB-SPECK on the *bike* image for various tile sizes. In order of performance, the solid line is for no tiling (entire image is a single tile); the dotted line is for 256 by 256 tiles; the dot-dash line is for 128 by 128 tiles; and the dashed line is for 64 by 64 tiles.

performance with no tiling for the test image at 1.0 bpp. SB-SPECK shows a similar drop in performance. However, SB-SPIHT only loses about 0.4 dB in 64 by 64 tiles for the test images at 1.0 bpp. The performance of EBCOT in small tiles degrades more so than SB-SPIHT making binary uncoded SB-SPIHT as effective as EBCOT in small tiles. The main reason for this behavior is that, in small tiles, spatial blocks can remain full size, while subband blocks are forced to become small because the bands themselves are small. Adaptive entropy coders like EBCOT can not work as well applied in small blocks because they cannot adapt with a small

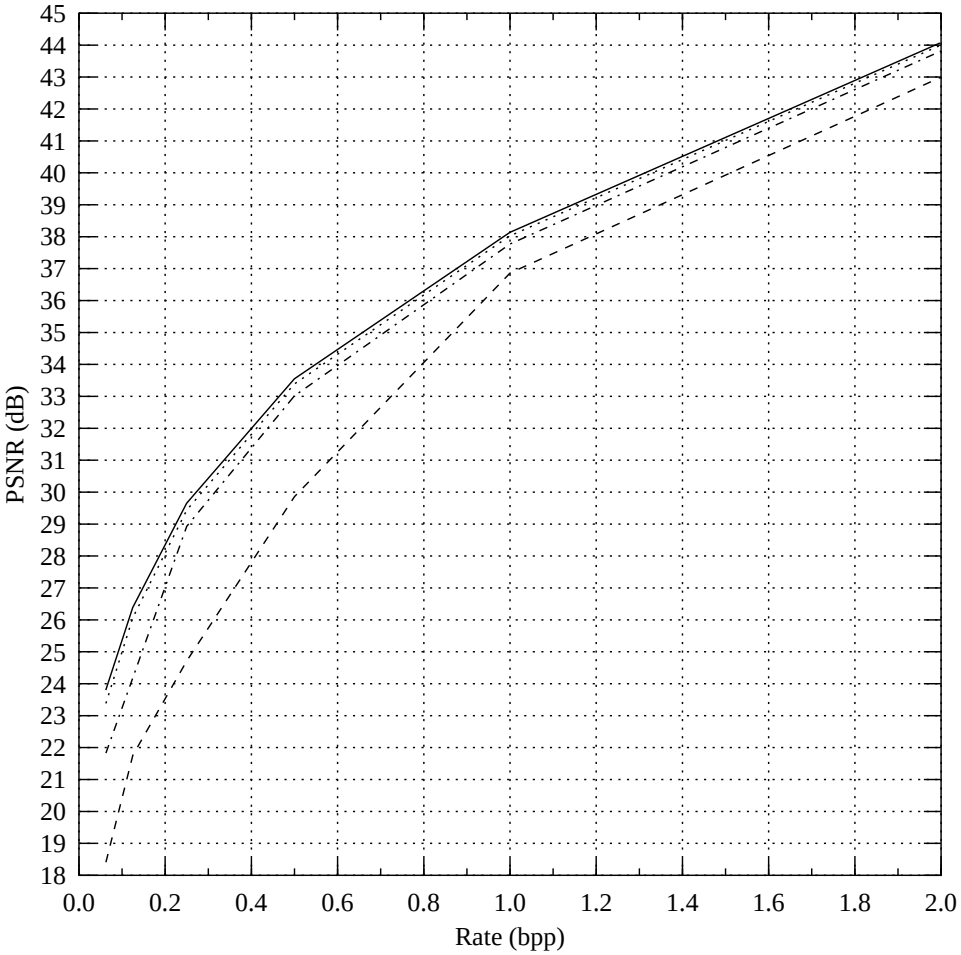

Figure 11.3: This plot shows the performance of the JPEG 2000 verification model test codec on the *bike* image for various tile sizes. In order of performance, the solid line is for no tiling (entire image is a single tile); the dotted line is for 256 by 256 tiles; the dot-dash line is for 128 by 128 tiles; and the dashed line is for 64 by 64 tiles.

amount of data. SB-SPECK also suffers from reduced performance in small tiles. SB-SPECK is not adaptive, but in small tiles there are many very small subbands, each of which has its own packet. With many small packets, the overhead of the fixed size packet headers becomes significant.

CHAPTER 12
Three-Dimensional Subband Block SPECK Coding

12.1 Introduction

The SB-SPECK image coding system can be extended from 2-D to 3-D for coding video and 3-D medical imagery. The resulting 3-D SB-SPECK is described in this chapter with emphasis on its application to medical image coding. After an overview of the nature of the 3-D data being coded, additions and changes to the wavelet transform and coding operations of the 2-D SB-SPECK codec are presented. The final section presents test results and conclusions.

The nature of video and medical data are of course quite different. Video is a sequence of images of a scene taken at increments in time. Objects in the scene appear in many frames, though their appearance changes as they move, rotate, change shape, become exposed, or become hidden. The 3-D medical images considered here are representations of a volume within a body. There are several means by which to acquire this data, including Magnetic Resonance Imaging (MRI), Computed Tomography (CT) and Positron Emission Tomography (PET).

For 2-D images, each dimension is a spatial dimension. For the medical image data we will consider here, the third dimension is also spatial, but for video, the third dimension is temporal. Increments in the third dimension of a video sequence are called frames, and for medical images they are usually called slices. Much of the explanation in this chapter applies to coding both video and 3-D medical imagery. So, in what follows, unless an exception is noted, the term *frame* may be replaced with *slice* and the term *temporal* may be replaced with the term *third spatial dimension*.

Video compression systems can use motion compensation to counteract the movement of objects in the video and improve compression, but at a significant computational burden at the encoder. Ordinary motion compensation would clearly not improve compression of 3-D medical imagery. In the interest of computational speed and performance on medical data, the 3-D SB-SPECK system does not use

motion compensation.

A video clip or 3-D medical image can be considered as a sequence of 2-D images, or frames. A simple way to apply SB-SPECK, or any image codec, to a sequence is to encode each frame independently, as if it were a stand-alone image. However, the frames are generally highly correlated, so a transform is used to decorrelate the data and improve compression.

There are different resolution requirements for the spatial and temporal dimensions of video. This usually results in more correlation in the spatial dimensions than in the temporal. For 3-D medical image data it is generally also the case that there is more correlation in the first two-dimensions than in the third spatial dimension, but for certain acquisition methods, each pixel represents a cubic or nearly cubic region.

Schelkens, Barbarien and Cornelis [34] have developed a technique for compressing medical data called Cube-Splitting (C-S) that is similar to 3-D SB-SPECK. Their system uses cube splitting, which is essentially a 3-D extension of quad-tree splitting, or SPECK, to 3-D blocks in a 3-D subband decomposition. The Cube-Splitting codec uses a context based arithmetic coder for significance test results and for refinement bits, significantly adding to its computational complexity. The 3-D transform is applied over the entire set of frames, as opposed to a smaller Group Of Frames (GOF), explained below.

Hsiang and Woods have also extended a quad-tree splitting algorithm similar to SPECK to 3-D, developing a codec called the 3-D Embedded Zero Block Coder (EZBC) [35, 36] that uses a motion compensated 3-D DWT. In this high performing system, the coding of subband coefficient significance tests is done with an adaptive arithmetic coder in a context developed from coefficients of other subbands. Thus, EZBC and 3-D EZBC do not have the same reduced memory and cache efficiency features of 3-D SB-SPECK.

12.2 Wavelet Transform

Before the wavelet transform, the frames are collected into a Group Of Frames (GOF) of F consecutive frames. Each GOF is independently encoded to the desired

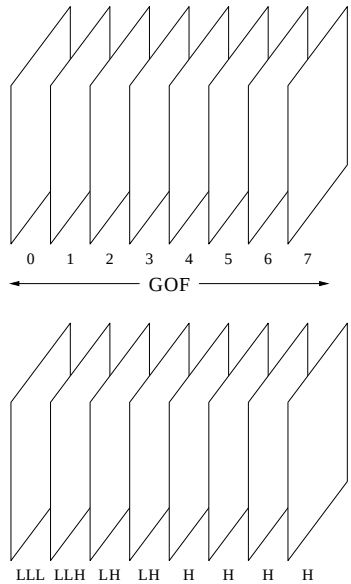

Figure 12.1: This diagram shows the temporal dyadic DWT of a GOF. A 3 level dyadic DWT converts the 8 original frames to 4 bands of transformed frames.

rate. The GOF size can be 1, essentially resulting in the 2-D SB-SPECK image codec.

Each GOF undergoes a dyadic DWT in the temporal direction first, as shown in Fig. 12.1. This step is analogous to the color transformation from Section 9.9, and is also separable from the 2-D spatial DWT to follow. The number of transform levels is maximized to $\log_2 F$. Because of the limited number of frames in a GOF, and the often lower temporal correlation, short filters are used. Tests reported below are performed with both the 2 tap Haar filter set and the integer 5/3 filter set, both of which were discussed in Section 8.5. Except for the fact that the GOF size is much smaller than the number of rows or columns and the filters are shorter, the temporal dyadic DWT procedure is identical to the spatial procedure. Just as in the two spatial dimensions, a reflection extension is used at the borders of the GOF. The reflection extension is needed only with the 5/3 filter set. The Haar filter set does not require extension because of its short filter widths.

The 3-D system has an increased memory requirement, but the same cache efficiency of the 2-D SB-SPECK codec. Since L frames are encoded at once, the

memory required for the 3-D system is at least L times more than for the 2-D SB-SPECK system. However, memory scaling by a factor of only L requires that the encoder can acquire and the decoder can release a line at a time from each frame in the GOF simultaneously. This is unlikely the case so most applications of this algorithm will need enough memory to store all coefficients for a GOF. The cache efficiency of the SB-SPECK system is preserved in the 3-D system because only small blocks of wavelet coefficients are encoded together.

12.3 Coding

After the temporal transform, each (transformed) frame is treated independently for most of the processing. This is what is done in the color system described in Section 9.9 as well. Each transformed frame is treated just as a whole grayscale image in the 2-D SB-SPECK system from Chapter 8 during the rest of the wavelet transform, the block partitioning, and the encoding stages. Each frame undergoes a 2-D dyadic subband transform, and each frame is partitioned into subband blocks just as if it were a complete image being coded by 2-D SB-SPECK. Each 2-D block in each frame is encoded using the core SPECK algorithm. At the rate allocation stage, the frames are no longer treated independently. Rate allocation and packetization is performed for all of the blocks from the GOF as if they all came from the same grayscale image. Thus, the rate budget is applied to each GOF.

Like the 2-D system, the 3-D system is scalable in resolution, and in fidelity with rate allocated in quality layers. With sequences, there is the additional option of frame rate progression. The final bitstream is scalable in many ways because of how the 3-D wavelet coefficients are grouped for independent coding.

A 3-D SB-SPECK encoded video stream is scalable in frame rate. If the GOF size is a multiple of 2, the frames of the highest temporal band can be dropped. This does not result in perfect frame downsampling. There is some averaging between the frames due to the temporal low-pass analysis filter.

For example, with the Haar filter set, the low pass filter will find the (scaled) average of two consecutive frames and the high pass filter will find their difference. To scale back the frame rate after coding, the high pass or difference frames are

dropped. The remaining frames, which will be viewed at the decoder, are each the average of two full rate frames. Of course, this extends to reduction of the frame rate by a factor of 4 if the GOF size is a multiple of 4, and to greater reduction as well.

12.4 Results and Conclusions

The 3-D SB-SPECK system was implemented by replacing and adding modules to the JPEG 2000 verification model test software, version 8.0. The system has been tested on three test data sets. First is the monochrome CCIR 601 format 30 frames/sec. *susie* sequence, which has 150 frames with 486 rows and 720 columns. Because we will be coding groups of up to 16 frames, only the first 144 frames are used. The second test data set is a 512 row by 512 column by 32 frame Computed Tomographic (CT) helical scan image of a normal human chest from the Digital Imaging and Communications in Medicine (DICOM) test set. The pixel resolution is 0.66 by 0.66 by 5–10 mm. The resolution in the third dimension is not fixed, but varies from frame to frame. The third test data set is a 256 row by 256 column by 192 frame Magnetic Resonance Image (MRI) of a normal human head, also from the DICOM test set. Each pixel in this image represents a 0.86 by 0.86 by 0.80 mm volume. Each original medical test data set uses 12 bits per pixel.

Each test uses four spatial decomposition levels and as many temporal decomposition levels as possible within the GOF. For lossy compression the 9/7 filter set is used in the two spatial dimensions and either the Haar or 5/3 integer filter is used in the temporal dimension. A single rate layer at the desired rate and 64 by 64 subband blocks were used for all of the results reported here.

To visually demonstrate the performance of this system some original and decoded groups of frames are shown below. The 13th through 16th frames of the CT test set were encoded as a GOF to an average rate of 0.15 bpp using the Haar filters in the temporal dimension and the 9/7 filters in the spatial dimensions. Figure 12.2 shows the original slices in the left, the decoded slices in th middle and the absolute difference image scaled by 16 on the right. The slices are in order from top to bottom. Also, the 61st through 64th frames of the MRI test set were encoded in the

same manner, but to a rate of 0.25 bpp. The original and decoded slices, and the absolute difference image, scaled by 16, are shown in Fig. 12.3. These slices were encoded at a rate low enough to make reconstruction artifacts visible. In normal use, medical imagery is almost always maintained at a very high fidelity.

Several plots show the distortion in PSNR vs. frame number for each of the test sequences. Each of these plots is reporting lossy compression results. For these results the 9/7 filter set is used in the spatial dimensions and the Haar filter set is used in the temporal dimension. Figure 12.4 shows results for the *susie* sequence encoded at an average rate of 0.5 bpp (5.18 Mbps). The PSNR vs. frame number plot for *susie* shows an anomaly in performance due to a repeated frame in the test data. Frames 96 and 97 of the *susie* sequence are identical.

For the CT test set, Fig. 12.5 shows the performance at 0.0625 bpp and Fig. 12.6 shows the performance at 0.125 bpp. For the MR test set, Fig. 12.7 shows the performance at 0.0625 bpp and Fig. 12.8 shows the performance at 0.125 bpp. In each plot, the results for several GOF sizes is shown. The results for a GOF size of 1 are a reference or baseline from which to evaluate the gain that comes from the 3-D transform with larger GOF sizes.

Table 12.1 also shows the lossy performance of 3-D SB-SPECK, with the distortion averaged over the entire sequence. The 9/7 filter set is used for the spatial directions for lossy encoding. For the *susie* sequence, a peak value of 255 ($= 2^8 - 1$) is used for computing PSNR. For the 12 bit medical images, 4095 ($= 2^{12} - 1$) is used. When the GOF size is 1 there is no difference between the Haar and 5/3 results because temporal filtering is not used. For the *susie* video sequence, encoded without motion compensation, the improvement in compression gained by using the 3-D transform is small. The improvement is generally less than 1 dB, and the gain as the GOF size increases beyond 4 is negligible.

The gain from using the 3-D transform for the medical data, however, is significant. Using the Haar filter in the temporal direction leads to better results for the CT and MR test sets for almost all rates and GOF sizes. The Haar filter results can be over 2 dB better than the 5/3 filter set results. Some exceptions are at high rates and large GOF sizes with the MR test set where the longer 5/3 filter gives

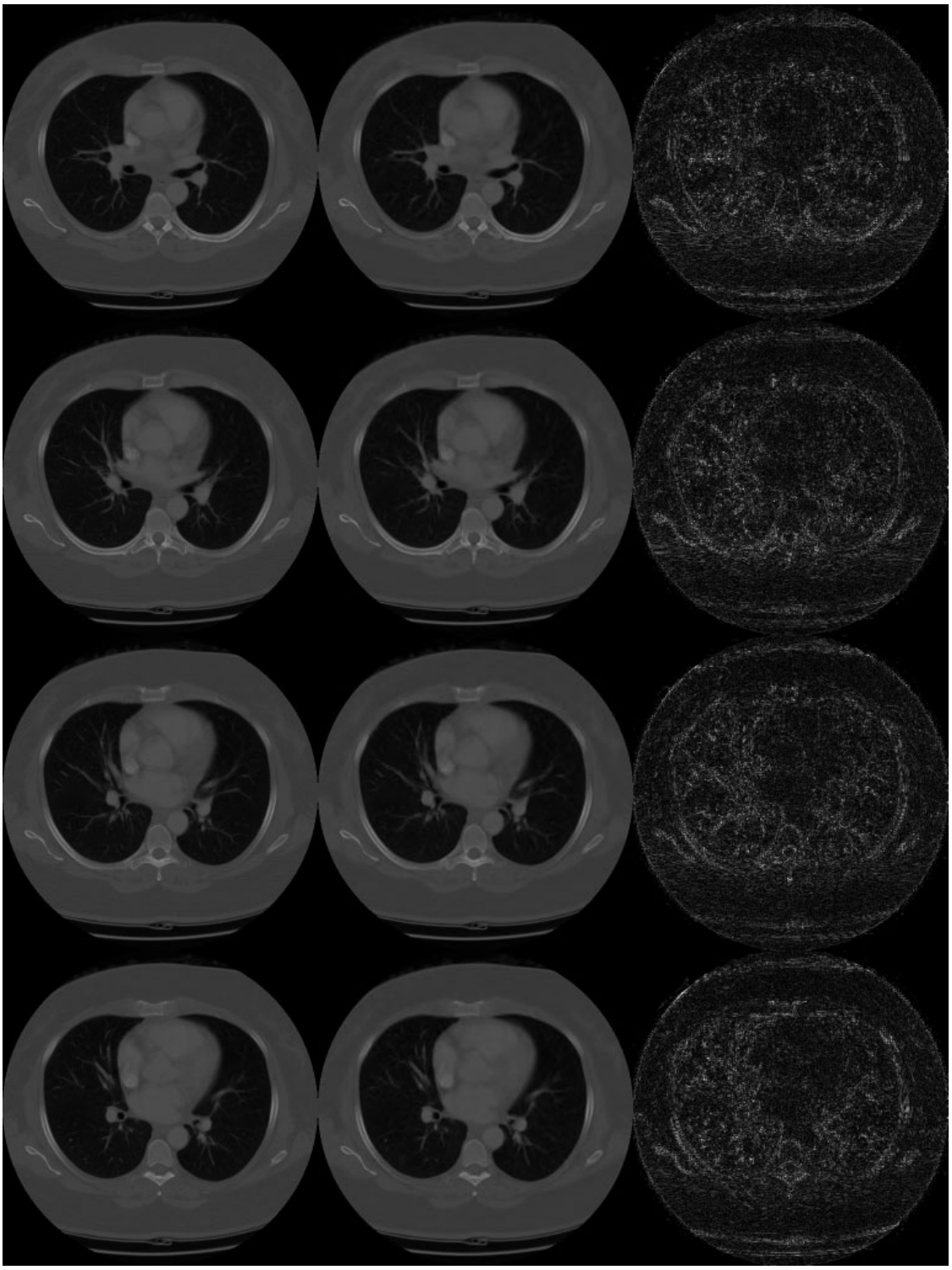

Figure 12.2: CT test set original and decoded slices and scaled absolute difference at 0.15 bpp.

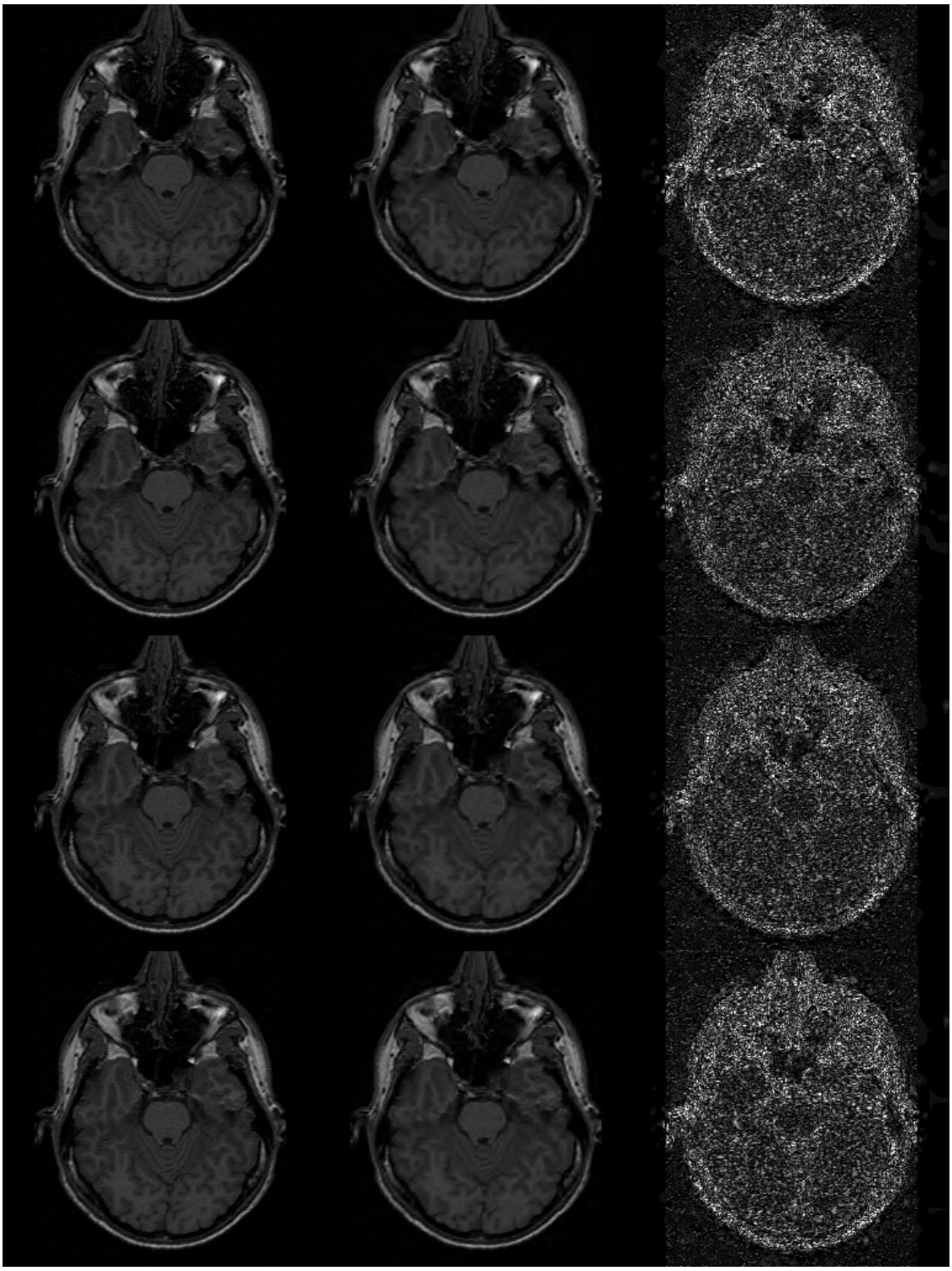

Figure 12.3: MRI test set original and decoded slices and scaled absolute difference at 0.25 bpp.

better results.

Results for the 3-D Cube-Splitting (C-S) technique by Schelkens, Barbarien and Cornelis [34] are also listed in Table 12.1 where they are available. For the CT test set, 3-D SB-SPECK performs much better than the Cube-Splitting codec. On each CT frame, pixels are 0.66 mm square. However, in the third dimension the frames are much further, 5–10 mm, apart, and so there is much less correlation in that direction. It seems that the Cube-Splitting technique is not appropriate where the correlation in the third dimension is reduced. Schelkens has also observed this problem on such data sets. For the MR test set, 3-D SB-SPECK with a GOF size of 16 offers a slight improvement over 3-D Cube-Splitting. 3-D SB-SPECK was used with floating point 9/7 filters for these lossy results while the 3-D Cube-Splitting technique used the integer 5/3 filter in all dimensions. This may account for some of the increased performance.

Table 12.2 shows lossless performance, averaged over the entire sequence, of the 3-D SB-SPECK codec. The reversible 5/3 filter set is used for the spatial dimensions for all lossless encoding. For the temporal dimension, both the Haar and 5/3 filter sets are tested. Again, for the *susie* video sequence the improvement in compression gained from the 3-D transform is small or non-existent. For 3-D medical data, 3-D transformation results in reduction of the compressed data size by up to 10%. When the GOF size is small there is not much improvement in performance over the 2-D baseline. For larger GOF sizes the 5/3 outperforms the Haar as a temporal filter, but both do well. Corresponding lossless results from the 3-D Cube-Splitting technique by Schelkens, Barbarien and Cornelis [34] are also listed in Table 12.2 where they are available. The large difference in performance of 3-D SB-SPECK over the Cube-Splitting technique for the CT test data is not seen with lossless coding. In fact, the Cube-Splitting technique achieves slightly better compression.

The 3-D SB-SPECK system is an effective technique for encoding medical image data. While the overall memory requirement reduction feature of 2-D SB-SPECK is not easily maintained, the cache efficiency feature is. The tests on MRI and CT data show a significant increase in fidelity for fixed rate coding and a

Sequence	Rate	Temporal Transform	GOF Size 1	2	4	8	16	C-S [34]
susie	0.25	Haar	37.90	38.70	38.93	38.97	38.97	
susie	0.25	5/3	37.90	37.87	38.32	38.32	38.44	
susie	0.5	Haar	40.64	41.28	41.44	41.47	41.46	
susie	0.5	5/3	40.64	40.58	40.81	40.69	40.76	
susie	1.0	Haar	43.72	44.18	44.30	44.32	44.30	
susie	1.0	5/3	43.72	43.69	43.59	43.46	43.49	
CT	0.0625	Haar	35.72	38.03	39.29	39.98	40.30	
CT	0.0625	5/3	35.72	35.82	37.96	39.01	39.70	34.34
CT	0.125	Haar	40.24	42.10	43.19	43.72	43.90	
CT	0.125	5/3	40.24	40.23	41.85	42.75	43.32	37.96
CT	0.25	Haar	44.56	46.16	46.77	47.05	47.14	
CT	0.25	5/3	44.56	44.54	45.82	46.28	46.53	42.51
CT	0.5	Haar	48.77	49.74	50.08	50.26	50.28	
CT	0.5	5/3	48.77	48.73	49.24	49.41	49.55	46.75
MR	0.0625	Haar	40.29	42.76	44.19	44.52	44.36	
MR	0.0625	5/3	40.29	40.60	43.21	44.23	44.60	43.06
MR	0.125	Haar	44.08	46.37	47.27	47.45	47.36	
MR	0.125	5/3	44.08	44.20	46.68	47.18	47.55	46.85
MR	0.25	Haar	47.67	49.79	50.42	50.51	50.45	
MR	0.25	5/3	47.67	47.70	50.06	50.36	50.70	48.34
MR	0.5	Haar	51.59	53.30	53.56	53.58	53.58	
MR	0.5	5/3	51.59	51.60	53.74	53.98	54.07	52.77

Table 12.1: For 3-D Subband Block SPECK, the distortion in PSNR (dB) is tabulated for each test data set, for several rates, for both the Haar and 5/3 temporal transform and for several GOF sizes. For comparison, results for the Cube-Splitting (C-S) [34] technique are given where available.

reduction in rate for lossless coding.

Sequence	Temporal Transform	GOF Size 1	2	4	8	16	C-S [34]
susie	Haar	3.834	3.834	3.773	3.774	3.782	
susie	5/3	3.834	3.833	3.838	3.856	3.863	
CT	Haar	5.302	5.301	5.197	5.157	5.143	
CT	5/3	5.302	5.301	5.220	5.189	5.173	5.033
MR	Haar	4.883	4.879	4.623	4.585	4.581	
MR	5/3	4.883	4.879	4.501	4.409	4.370	4.030

Table 12.2: For 3-D Subband Block SPECK, the amount of encoded data in bits per pixel for lossless encoding is tabulated for each test data set, for both the Haar and 5/3 temporal transform and for several GOF sizes. For comparison, results for the Cube-Splitting (C-S) [34] technique are given where available.

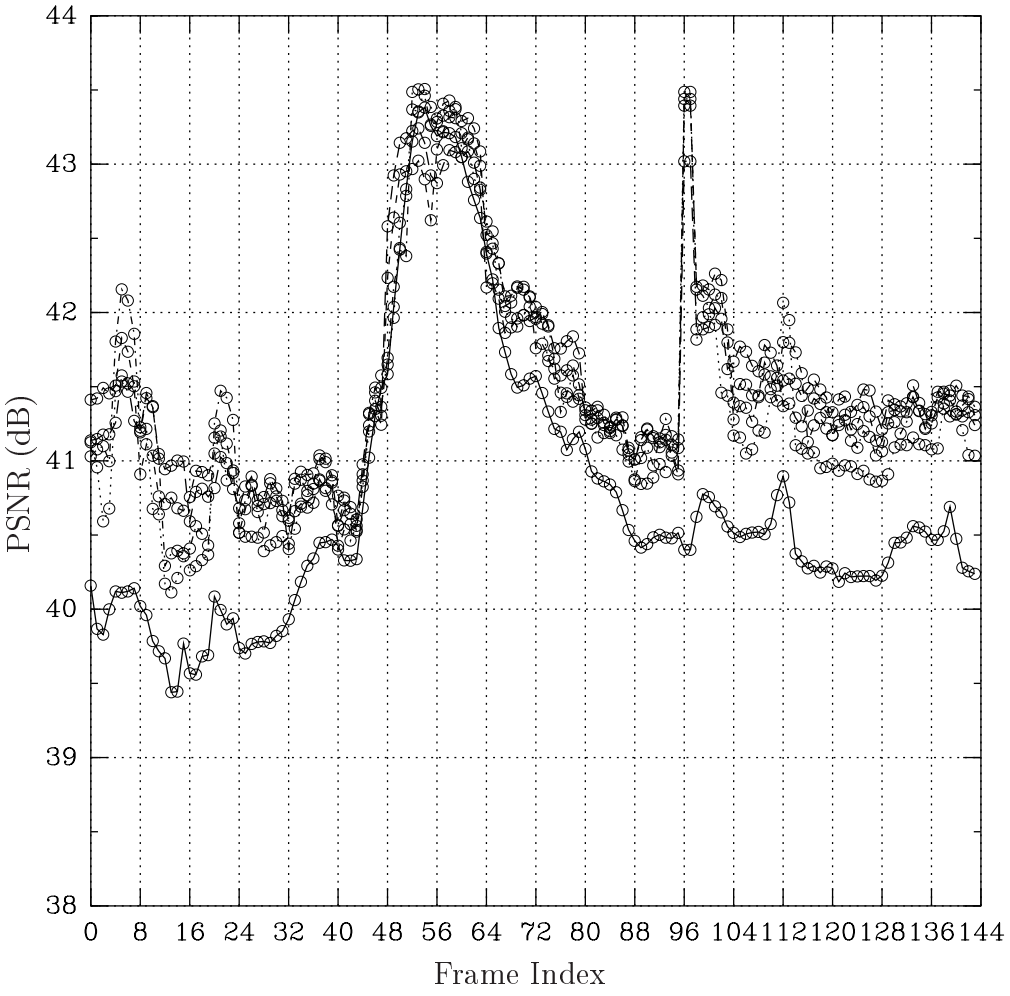

Figure 12.4: This plot shows the performance of the 3-D SB-SPECK algorithm on the *susie* video sequence for each frame. The overall rate is 0.5 bpp. Results for various GOF sizes are shown. The solid line is for just 1 frame in each GOF; the dotted line is for 2 frames per GOF; the dot-dashed line is for 4 frames per GOF; the short dashed line is for 8 frames per GOF; the long dashed line is for 16 frames per GOF.

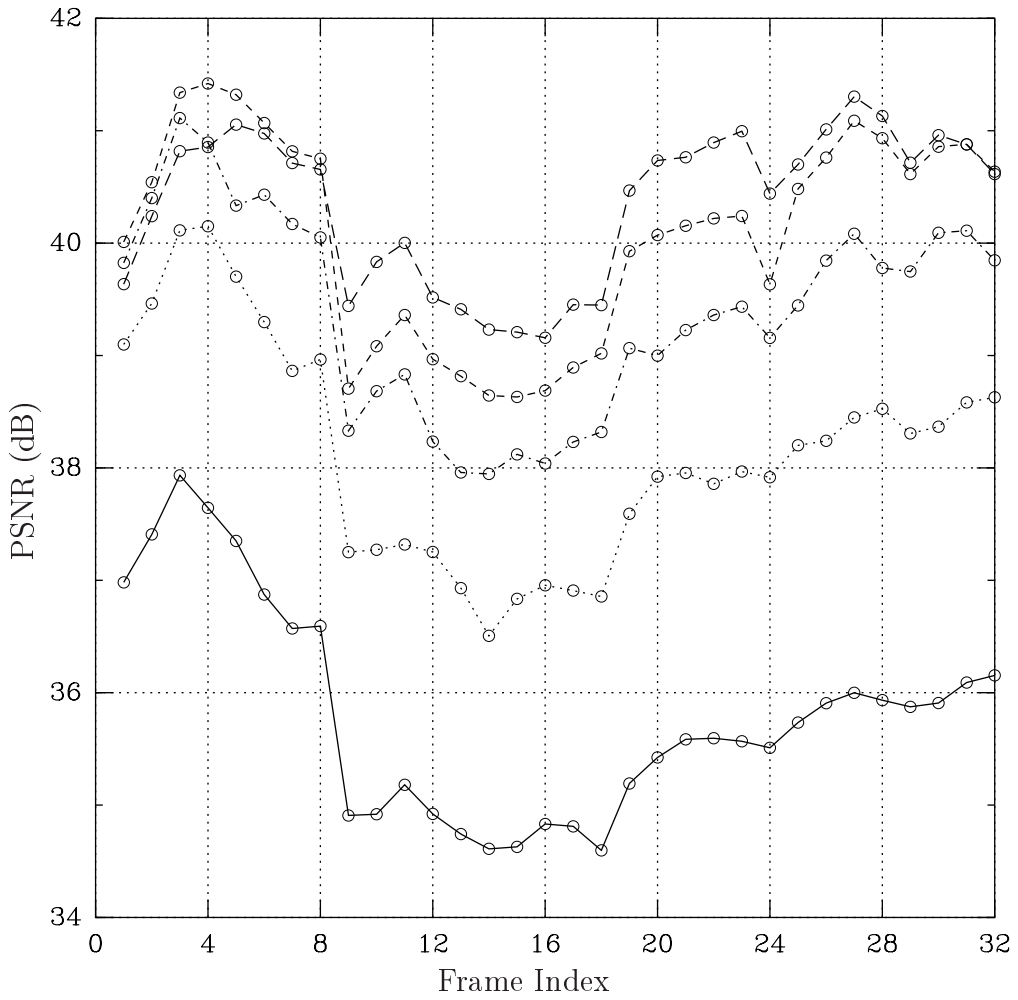

Figure 12.5: This plot shows the performance of the 3-D SB-SPECK algorithm on the CT test data. The overall rate is 0.0625 bpp. The plot legend is given with Fig. 12.4.

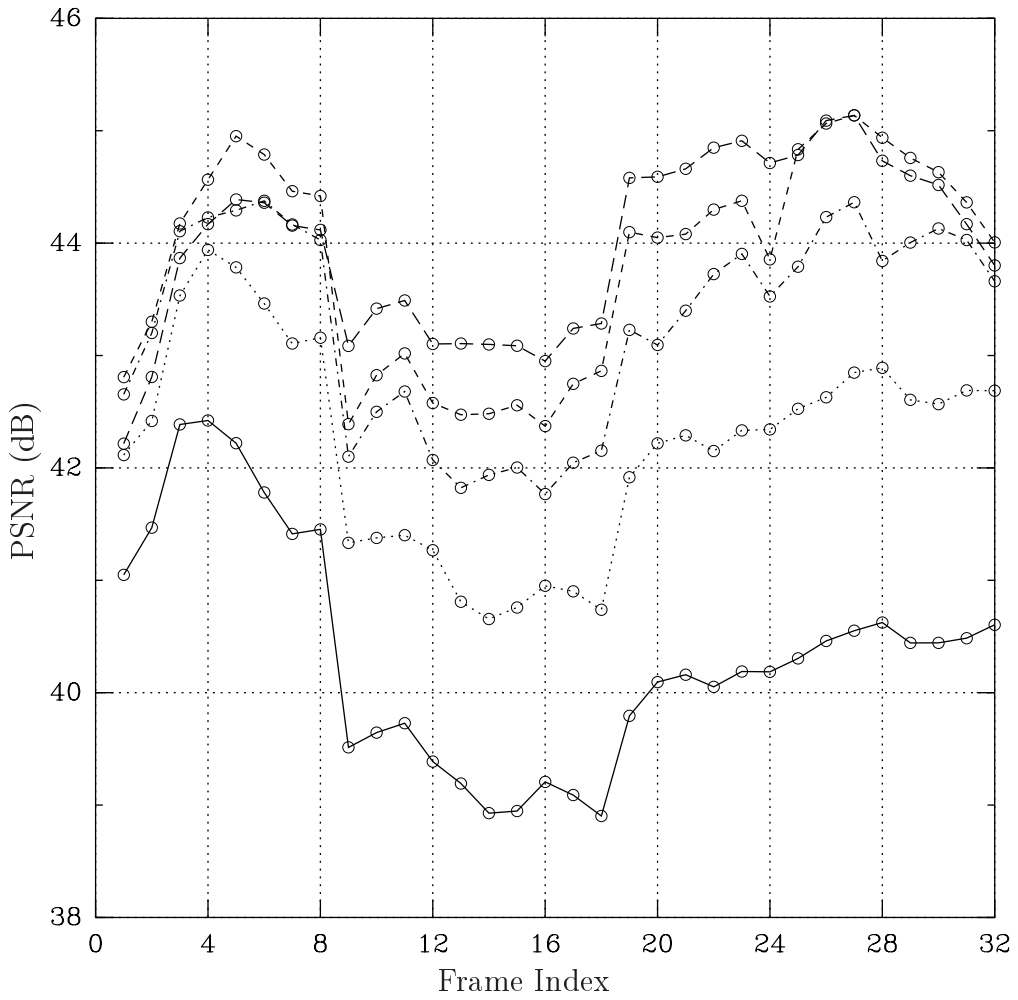

Figure 12.6: This plot shows the performance of the 3-D SB-SPECK algorithm on the CT test data. The overall rate is 0.125 bpp. The plot legend is given with Fig. 12.4.

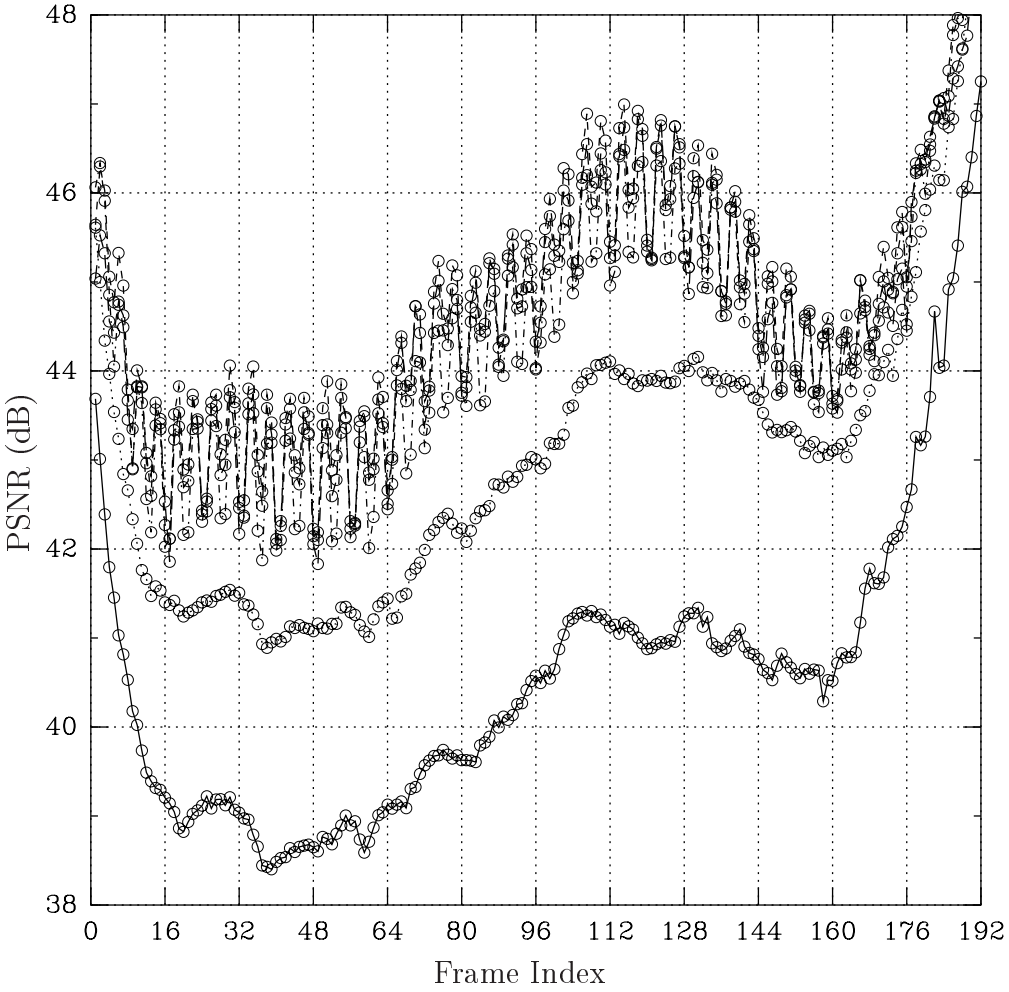

Figure 12.7: This plot shows the performance of the 3-D SB-SPECK algorithm on the MR test data. The overall rate is 0.0625 bpp. The plot legend is given with Fig. 12.4.

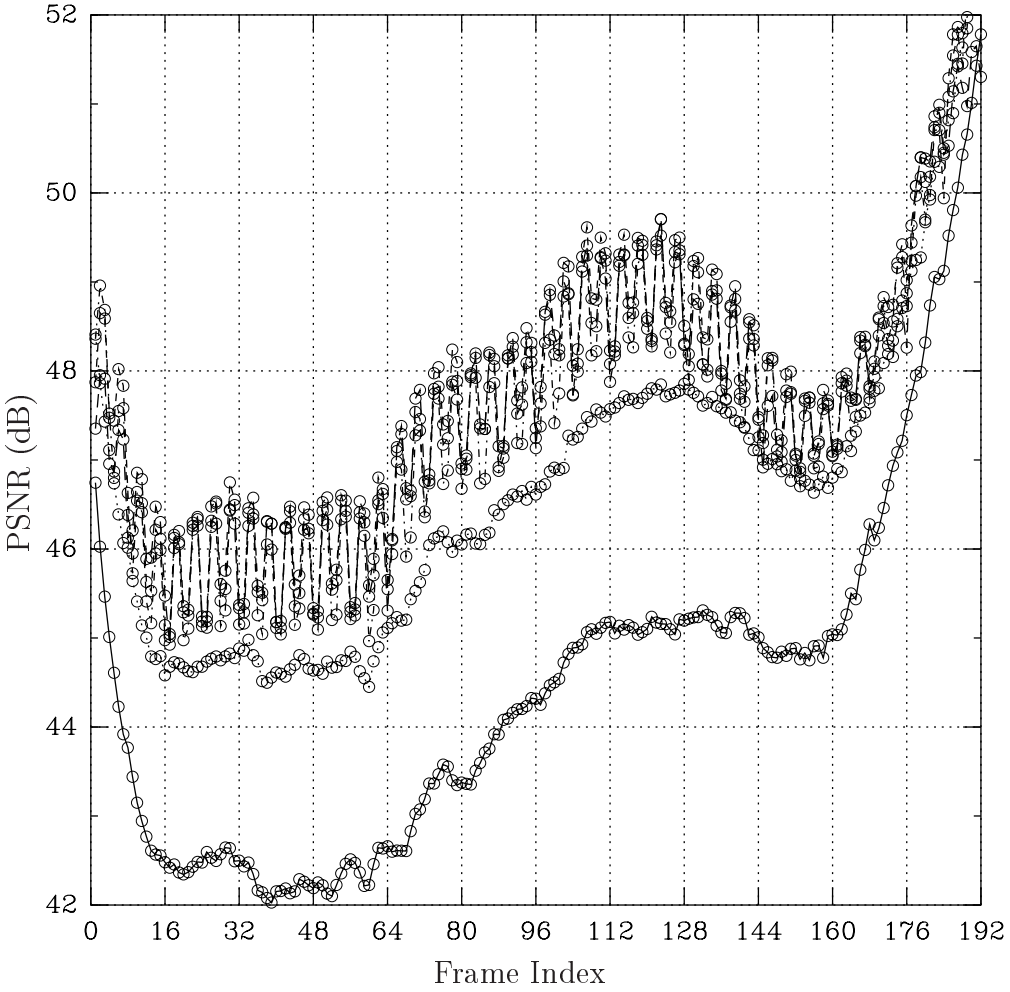

Figure 12.8: This plot shows the performance of the 3-D SB-SPECK algorithm on the MR test data. The overall rate is 0.125 bpp. The plot legend is given with Fig. 12.4.

CHAPTER 13
Conclusions

In this chapter the major technical contributions of this work are summarized and overall conclusions are drawn.

In the trellis coding part of this work, the primary contributions are two observations on the nature of the reproduction sequences produced by a trellis coded dequantizer. There are alternate non-TCM branch labelings that are slightly superior for ECTCQ source coding. Further, as the number of ECTCQ states increases, the reproduction sequences move toward the optimal reproduction sphere for Gaussian sources.

The application of the SPIHT tree based codec independently to spatial blocks of a wavelet decomposition (SB-SPIHT) for the purpose of efficient use of a cached memory environment in Chapter 7 is an important contribution of this work as is the memory requirement analysis. Also, the fast and effective technique to generate an approximate quantizer function for each spatial block introduced in Section 7.7 is a new and useful tool for practical image coding.

The application of the approximate piecewise linear quantizer function to Subband Block SPECK coding in Chapter 8 with the associated rate allocation scheme is another contribution of this work. This quantizer function based rate allocation method gives a significant improvement over previous techniques.

The generalized hybrid block coding (HBC) system of Chapter 9 and the color HBC system offer more efficient resolution scalability than SB-SPIHT, though their performance is not as high as SB-SPECK. Still, the hybrid block partitioning scheme used by HBC is a new and interesting approach to memory efficient image compression.

The NLS algorithm for implementing SPIHT without lists, presented in Chapter 10, is a new contribution. The performance of NLS is nearly identical to that of the original SPIHT algorithm. No List SPIHT has a fixed size memory requirement that is not as large as the requirement for original SPIHT. The memory require-

ment of NLS and the efficient breadth first scanning technique make it appealing for hardware based image coding.

In the development of the No List SPIHT scheme the utility of the linear index, or Morton ordering, for hierarchical tree based wavelet tree coding in general was recognized. This scan order will be quite useful in hardware based hierarchical tree scanning codecs.

The 3-D SB-SPECK codec of Chapter 12 is a new fast and efficient codec for video and 3-D medical imagery. The results show a large gain for coding medical images in three dimensions. It is reasonable to expect 3-D SB-SPECK to also work well on multi- and hyper-spectral imagery. Like 2-D SB-SPECK, this system encodes small blocks in turn for memory cache efficiency, which is important in practical applications.

The rolling line-based transform greatly reduces the memory required for wavelet based image compression. When spatial block or subband block partitioning is used, further reductions in memory are possible with a block-based transform implementation. Such a transform engine would apply the rolling technique in both the horizontal and vertical directions. Complete spatial and subband blocks can be produced after reading even less of the original image than the line-based engine reads.

ACRONYMS

Many of the acronyms used in this document are defined in this table.

ACRDT	Alphabet Constrained Rate Distortion Theory [4, 5]
BFOS	Breiman-Friedman-Olshen-Stone algorithm
CPU	Central Processing Unit
CREW	Compression with Reversible Embedded Wavelets [68]
CT	Computed Tomography
DCT	Discrete Cosine Transform
DICOM	Digital Imaging and Communications in Medicine
DWT	Discrete Wavelet Transform
EBCOT	Embedded Block Coding with Optimal Truncation [25]
ECSQ	Entropy-Constrained Scalar Quantization
ECTCQ	Entropy-Constrained TCQ
EZBC	Embedded Zero Block Coder [35, 36]
EZW	Embedded Zerotree Wavelets [15]
FIR	Finite Impulse Response
GOF	Group Of Frames
HBC	Hybrid Block Codec
HVS	Human Visual System
IP	Insignificant Pixel
IS	Insignificant Set
JPEG	Joint Photographic Experts Group
LBG	Linde-Buzo-Gray [1]
LIP	List of Insignificant Pixels
LIS	List of Insignificant Sets
LSB	Least Significant Bit
LSP	List of Significant Pixels
MIMD	Multiple Instruction Multiple Data
MPEG	Moving Picture Experts Group
MRI	Magnetic Resonance Imaging
MSB	Most Significant Bit
MSE	Mean Squared Error
NLS	No List SPIHT [81]
NLSP	New List of Significant Pixels
PET	Positron Emission Tomography
PSNR	Peak Signal to Noise Ratio
RAM	Random Access Memory
RGB	Red-Green-Blue
ROI	Region Of Interest
SB-SPECK	Subband Block SPECK

SB-SPIHT	Spatial Block SPIHT
SIMD	Single Instruction Multiple Data
SPECK	Set Partitioned Embedded Block Coder [17, 18]
SPIHT	Set-Partitioning In Hierarchical Trees [16]
TCQ	Trellis Coded Quantizer / Trellis Coded Quantization
TCM	Trellis Coded Modulator / Trellis Coded Modulation

LITERATURE CITED

[1] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. on Communications*, vol. 28, pp. 84–95, January 1980.

[2] L. C. Stewart, R. M. Gray, and Y. Linde, "The design of trellis waveform coders," *IEEE Trans. on Communications*, vol. 30, pp. 702–709, April 1982.

[3] G. H. Freeman, J. W. Mark, and I. F. Blake, "Trellis source codes designed by conjugate gradient optimization," *IEEE Trans. on Communications*, vol. 36, pp. 1–12, January 1988.

[4] W. A. Finamore and W. A. Pearlman, "Optimal encoding of discrete-time continuous-amplitude memoryless sources with finite output alphabets," *IEEE Trans. on Information Theory*, vol. 26, pp. 144–155, March 1980.

[5] W. A. Pearlman, "Sliding-block and random source coding with constrained size reproduction alphabets," *IEEE Trans. on Communications*, vol. 30, pp. 1859–1867, August 1982.

[6] M. W. Marcellin, *Trellis Coded Quantization: An Efficient Technique for Data Compression*. PhD thesis, Texas A&M University, November 1987.

[7] M. W. Marcellin, T. R. Fischer, and J. D. Gibson, "Predictive trellis coded quantization of speech," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 38, pp. 46–55, January 1990.

[8] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. on Information Theory*, vol. 28, pp. 55–67, January 1982.

[9] G. Ungerboeck, "Trellis-coded modulation with redundant signal sets part I: Introduction," *IEEE Communications Magazine*, vol. 25, pp. 5–11, February 1987.

[10] G. Ungerboeck, "Trellis-coded modulation with redundant signal sets part II: State of the art," *IEEE Communications Magazine*, vol. 25, pp. 12–21, February 1987.

[11] T. R. Fischer and M. Wang, "Entropy-constrained trellis-coded quantization," *IEEE Trans. on Information Theory*, vol. 38, pp. 415–426, March 1992.

[12] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Entropy-constrained vector quantization," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 37, pp. 31–42, January 1989.

[13] M. W. Marcellin, "On entropy-constrained trellis coded quantization," *IEEE Trans. on Communications*, vol. 42, pp. 14–16, January 1994.

[14] D. J. Sakrison, "A geometric treatment of the source encoding of a Gaussian random variable," *IEEE Trans. on Information Theory*, vol. 14, pp. 481–486, May 1968.

[15] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. on Signal Processing*, vol. 41, pp. 3445–3462, December 1993.

[16] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 6, pp. 243–250, June 1996.

[17] A. Islam and W. A. Pearlman, "An embedded and efficient low-complexity hierarchical image coder," in *Visual Communications and Image Processing, Proc. of SPIE 3653*, pp. 294–305, January 1999.

[18] A. Islam, *Set-Partitioned Image Coding*. PhD thesis, Rensselaer Polytechnic Institute, September 1999.

[19] C. Chrysafis and A. Ortega, "Line based reduced memory wavelet image compression," in *Proc. of the IEEE Data Compression Conference*, pp. 398–407, 1998.

[20] C. Chrysafis and A. Ortega, "Line-based, reduced memory, wavelet image compression," *IEEE Trans. on Image Processing*, vol. 9, pp. 378–389, March 2000.

[21] J. K. Rogers and P. C. Cosman, "Wavelet zerotree image compression with packetization," *IEEE Signal Processing Letters*, vol. 5, pp. 105–107, May 1998.

[22] C. D. Creusere, "Image coding using parallel implementations of the embedded zerotree wavelet algorithm," in *Symposium on Electronic Imaging, Proc. of IS&T/SPIE 2668*, 1996.

[23] C. D. Creusere, "Spatially partitioned lossless image compression in an embedded framework," in *Conf. Record of The Thirty-First Asilomar Conf. on Signals, Systems & Computers* (M. P. Fargues and R. D. Hippenstiel, eds.), vol. 2, (Pacific Grove, CA), pp. 1455–1459, November 1997.

[24] C. D. Creusere, "A new method of robust image compression based on the embedded zerotree wavelet algorithm," *IEEE Trans. on Image Processing*, vol. 6, pp. 1436–1442, October 1997.

[25] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. on Image Processing*, vol. 9, pp. 1158–1170, July 2000.

[26] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," tech. rep., IBM Ltd., Ottowa, Canada, 1966.

[27] G. Seetharaman and B. Zavidovique, "Z-trees: Adaptive pyramid algorithms for segmentation," in *Proc. of the International Conf. on Image Processing*, 1998.

[28] W.-K. Lin and N. Burgess, "Listless zerotree coding for color images," in *Conf. Record of The Thirty-Second Asilomar Conf. on Signals, Systems & Computers* (M. B. Matthews, ed.), vol. 1, (Pacific Grove, CA), pp. 231–235, November 1998.

[29] W.-K. Lin, W.-H. Ng, N. Burgess, and A. Bouzerdoum, "Reduced memory zerotree coding algorithm for hardware implementation," in *IEEE ICMCS '99*, June 1999.

[30] W.-K. Lin and N. Burgess, "3D listless zerotree coding for low bit rate video," in *Proc. of the International Conf. on Image Processing*, October 1999.

[31] R. R. Shively, E. Ammicht, and P. D. Davis, "Generalizing SPIHT: A family of efficient image compression algorithms," in *Proc. of the International Conf. on Acoustics, Speech and Signal Processing*, September 2000.

[32] A. Järvi, J. Lehtinen, and O. Nevalainen, "Variable quality image compression system based on SPIHT," *Signal Processing: Image Communications*, vol. 14, pp. 683–696, 1999.

[33] J. Lehtinen, "Distortion limited wavelet image codec," *Acta Cybernetica*, vol. 14, no. 2, pp. 341–356, 1999.

[34] P. Schelkens, J. Barbarien, and J. Cornelis, "Compression of volumetric medical data based on cube-splitting," in *Applications of Digital Image Processing XXIII, Proc. of SPIE 4115*, July 2000.

[35] S.-T. Hsiang and J. W. Woods, "Embedded image coding using zeroblocks of subband/wavelet coefficients and context modeling," in *MPEG-4 Workshop and Exhibition at ISCAS*, (Geneva, Switzerland), May 2000.

[36] S.-T. Hsiang and J. W. Woods, "Embedded video coding using motion compensated 3-D subband/wavelet filter bank," in *Proc. of the Packet Video Workshop*, (Sardinia, Italy), May 2000.

[37] J. G. Proakis, *Digital Communications.* McGraw Hill Book Company, 2nd ed., 1989.

[38] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* John Wiley and Sons, 1991.

[39] H. Gish and J. N. Pierce, "Asymptotically efficient quantizing," *IEEE Trans. on Information Theory*, vol. 14, pp. 676–683, September 1968.

[40] N. Farvardin and J. W. Modestino, "Optimum quantizer performance for a class of non-Gaussian memoryless sources," *IEEE Trans. on Information Theory*, vol. 30, pp. 485–497, May 1984.

[41] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Kluwer Academic Press, 1992.

[42] G. D. Forney, "The Viterbi algorithm," *Proc. of the IEEE*, vol. 61, pp. 268–278, March 1973.

[43] J. B. Anderson, "Limited search trellis decoding of convolutional codes," *IEEE Trans. on Information Theory*, vol. 35, pp. 944–955, September 1989.

[44] A. J. Viterbi and J. K. Omura, "Trellis encoding of memoryless discrete-time sources with a fidelity criterion," *IEEE Trans. on Information Theory*, vol. 20, pp. 325–332, May 1974.

[45] R. E. Blahut, "Computation of channel capacity and rate-distortion functions," *IEEE Trans. on Information Theory*, vol. 18, pp. 460–473, July 1972.

[46] W. A. Pearlman and A. Chekima, "Source coding bounds using quantizer reproduction levels," *IEEE Trans. on Information Theory*, vol. 30, pp. 559–567, May 1984.

[47] T. D. Lookabaugh and R. M. Gray, "High-resolution quantization theory and the vector quantizer advantage," *IEEE Trans. on Information Theory*, vol. 35, pp. 1020–1033, September 1989.

[48] M. V. Eyuboğlu and G. D. Forney, "Lattice and trellis quantization with lattice- and trellis-bounded codebooks–high-rate theory for memoryless sources," *IEEE Trans. on Information Theory*, vol. 39, pp. 46–59, January 1993.

[49] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, pp. 520–540, June 1987.

[50] J. Glen G. Langdon, "An introduction to arithmetic coding," *IBM Journal of Research and Development*, vol. 28, pp. 135–149, March 1984.

[51] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. on Information Theory*, vol. 28, pp. 129–137, March 1982.

[52] M. W. Marcellin and T. R. Fischer, "Trellis coded quantization of memoryless and Gauss-Markov sources," *IEEE Trans. on Communications*, vol. 38, pp. 82–93, January 1990.

[53] R. J. van der Vleuten, *Trellis-Based Source and Channel Coding.* PhD thesis, Delft University of Technology, Delft, The Netherlands, 1994.

[54] H. Jafarkhani, N. Farvardin, and C.-C. Lee, "Adaptive image coding based on the discrete wavelet transform," in *Proc. of the International Conf. on Image Processing*, vol. 3, pp. 343–347, 1994.

[55] C.-C. Lee and N. Farvardin, "Entropy-constrained trellis coded quantization: Implementation and adaption," in *Proc. of the Conf. on Information Sciences and Systems*, (Baltimore), March 1993.

[56] R. C. Gonzalez and P. Wintz, *Digital Image Processing.* Addison-Wesley Publishing Co., 2nd ed., 1987.

[57] A. K. Jain, *Fundamentals of Digital Image Processing.* Prentice-Hall, 1989.

[58] W. K. Pratt, *Digital Image Processing.* Wiley-Interscience, 1991.

[59] A. N. Netravali and B. G. Haskell, *Digital Pictures: Representation, Compression and Standards.* Plenum Press, 2nd ed., 1995.

[60] J. W. Woods and S. D. O'Neil, "Subband coding of images," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 34, pp. 1278–1288, October 1986.

[61] J. W. Woods, ed., *Subband Image Coding.* Kluwer Academic Publishers, 1991.

[62] S. G. Mallat, *A Wavelet Tour of Signal Processing.* Academic Press, 2nd ed., 1999.

[63] A. Munteanu, J. Cornelis, G. V. der Auwera, and P. Cristea, "Wavelet based lossless compression scheme with progressive transmission capability," *International Journal of Imaging Systems and Technology*, vol. 10, pp. 76–85, January 1999.

[64] A. Munteanu, J. Cornelis, G. V. der Auwera, and P. Cristea, "Wavelet image compression—the quadtree coding approach," *IEEE Transactions on Information Technology in Biomedicine*, vol. 3, pp. 176–185, September 1999.

[65] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," *ACM Transactions on Information Systems*, vol. 16, pp. 256–294, July 1998.

[66] J. Andrew, "A simple and efficient hierarchical image coder," in *Proc. of the International Conf. on Image Processing*, vol. 3, pp. 658–661, 1997.

[67] F. W. Wheeler and W. A. Pearlman, "Low-memory packetized SPIHT image compression," in *Conf. Record of The Thirty-Third Asilomar Conf. on Signals, Systems & Computers* (M. B. Matthews, ed.), vol. 2, (Pacific Grove, CA), pp. 1193–1197, October 1999.

[68] A. Zandi, J. D. Allen, E. L. Schwartz, and M. Boliek, "CREW: Compression with reversible embedded wavelets," in *Proc. of the Data Compression Conf.*, pp. 212–221, 1995.

[69] M. J. Gormish, E. L. Schwartz, A. Keith, M. Boliek, and A. Zandi, "Lossless and nearly lossless compression for high quality images," in *Proc. of SPIE 3025*, February 1997.

[70] J. Bae and V. K. Prasanna, "A fast and area-efficient VLSI architecture for embedded image coding," in *Proc. of the International Conf. on Image Processing*, pp. 452–454, 1995.

[71] N. Park, J. Bae, and V. K. Prasanna, "Synthesis of VLSI architectures for tree-structured image coding," in *Proc. of the International Conf. on Image Processing*, pp. 999–1002, 1996.

[72] E. A. Riskin, "Optimal bit allocation via the generalized BFOS algorithm," *IEEE Trans. on Information Theory*, vol. 37, pp. 400–402, March 1991.

[73] Y. Shoham and A. Gersho, "Efficient bit allocation for an arbitrary set of quantizers," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 36, pp. 1445–1453, September 1988.

[74] R. Caetano and E. A. B. da Silva, "A rate control strategy for embedded wavelet video coders," *Electronics Letters*, vol. 35, pp. 1815–1817, October 1999.

[75] E. A. B. da Silva and R. Caetano, "A rate control strategy for embedded wavelet video coders in an MPEG-4 framework," in *Proc. of IEEE Globecom*, (Rio de Janeiro, Brazil), 1999.

[76] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. on Image Processing*, vol. 1, pp. 205–220, April 1992.

[77] C. Chrysafis, A. Said, A. Drukarev, A. Islam, and W. A. Pearlman, "SBHP — A low complexity wavelet coder," in *Proc. of the International Conf. on Acoustics, Speech and Signal Processing*, June 2000.

[78] A. Said and W. A. Pearlman, "An image multiresolution representation for lossless and lossy compression," *IEEE Trans. on Image Processing*, vol. 5, pp. 1303–1310, September 1996.

[79] W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions," in *Wavelet Applications in Signal and Image Processing III, Proc. of SPIE 2569* (A. F. Laine and M. Unser, eds.), pp. 68–79, 1995.

[80] F. W. Wheeler and W. A. Pearlman, "Combined spatial and subband block coding of images," in *Proc. of the International Conf. on Image Processing*, vol. 3, (Vancouver, Canada), pp. 861–864, September 2000.

[81] F. W. Wheeler and W. A. Pearlman, "SPIHT image compression without lists," in *Proc. of the International Conf. on Acoustics, Speech and Signal Processing*, pp. 2047–2050, June 2000.

[82] J. M. Shapiro, "A fast technique for identifying zerotrees in the EZW algorithm," in *Proc. of the International Conf. on Acoustics, Speech and Signal Processing*, pp. 1455–1458, May 1996.