Foundations and Trends® in
Signal Processing
Vol. 2, No. 2 (2008) 95–180
© 2008 W. A. Pearlman and A. Said
DOI: 10.1561/2000000013

now
the essence of knowledge

Set Partition Coding: Part I of Set Partition Coding and Image Wavelet Coding Systems

William A. Pearlman[1] and Amir Said[2]

[1] *Department of Electrical, Computer and System Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, USA, pearlw@ecse.rpi.edu*
[2] *Hewlett-Packard Laboratories, 1501 Page Mill Road, MS 1203, Palo Alto, CA 94304, USA, Said@hpl.hp.com*

Abstract

The purpose of this two-part monograph is to present a tutorial on set partition coding, with emphasis and examples on image wavelet transform coding systems, and describe their use in modern image coding systems. Set partition coding is a procedure that recursively splits groups of integer data or transform elements guided by a sequence of threshold tests, producing groups of elements whose magnitudes are between two known thresholds, therefore, setting the maximum number of bits required for their binary representation. It produces groups of elements whose magnitudes are less than a certain known threshold. Therefore, the number of bits for representing an element in a particular group is no more than the base-2 logarithm of its threshold

rounded up to the nearest integer. SPIHT (Set Partitioning in Hierarchical Trees) and SPECK (Set Partitioning Embedded blocK) are popular state-of-the-art image coders that use set partition coding as the primary entropy coding method. JPEG2000 and EZW (Embedded Zerotree Wavelet) use it in an auxiliary manner. Part I elucidates the fundamentals of set partition coding and explains the setting of thresholds and the block and tree modes of partitioning. Algorithms are presented for the techniques of AGP (Amplitude and Group Partitioning), SPIHT, SPECK, and EZW. Numerical examples are worked out in detail for the latter three techniques. Part II describes various wavelet image coding systems that use set partitioning primarily, such as SBHP (Subband Block Hierarchical Partitioning), SPIHT, and EZBC (Embedded Zero-Block Coder). The basic JPEG2000 coder is also described. The coding procedures and the specific methods are presented both logically and in algorithmic form, where possible. Besides the obvious objective of obtaining small file sizes, much emphasis is placed on achieving low computational complexity and desirable output bitstream attributes, such as embeddedness, scalability in resolution, and random access decodability.

This monograph is extracted and adapted from the forthcoming textbook entitled *Digital Signal Compression: Principles and Practice* by William A. Pearlman and Amir Said, Cambridge University Press, 2009.

1

Introduction

The goal of this monograph is to elucidate the principles of a data compression method called *set partition coding,* mainly by showing its use in image compression. One of the most interesting aspects of this coding technique is that it is based on some very simple principles, but is also very effective, surpassing other coding methods that may seem theoretically much more sophisticated (and probably with a fraction of the computational complexity).

This technique is the primary entropy coding method in the well-known SPIHT (Set Partitioning In Hierarchical Trees) [18] and SPECK (Set Partitioning Embedded bloCK) [15] algorithms and is an auxiliary coding method in the JPEG2000 standard [11, 12]. It also has been successfully applied in the method of EZBC (Embedded Zero Block Coder) for video coding [9]. There are many books and articles explaining SPIHT and SPECK, and its many variations and modifications. These works state the algorithms and present worked-out examples, but none touches on the logic underlying these methods.

What had been missing is a proper explanation of why a method as simple as set partition coding can be so effective. A possible reason for this omission is that the usual approach for teaching data compression

is to start from information theory principles, and some simple statistical models, like i.i.d. sources, and then possibly move to somewhat more complicated cases. However, rarely there is any real intention of realism, since the data sources defined by the main multimedia compression applications (audio, images, and video) are quite complex. They can be extremely difficult to model accurately, and we may not even know what are all the aspects involved (e.g., if a Stradivarius violin sounds like no other, how to preserve that information?).

A superficial interpretation of information theory can also give the impression that increasingly more complex statistical models are the only thing needed to improve compression. Actually what is needed is the development of *workable* models (explicit or implicit) that are flexible enough to allow more *accurate* modeling for *coding,* without necessarily providing descriptive information. An *ad hoc* scheme based on empirical knowledge of the source's statistical properties can be much better than methods that in theory could be used for data sources of any complexity, but in practice need to be very strongly constrained to be manageable.

This monograph is an attempt to remedy this situation and help the reader understand and separate the various principles that are at play in effective coding systems. We chose an approach that keeps the presentation intuitive and simple, without formal statistical models, but always related to real data sources. The risk, in this case, is that the methods we present may seem at first too simplistic and limited, even naïve. The reader should keep in mind that even the simplest versions are quite effective, and could not be so if they were not able to exploit fundamental statistic properties of the data sources.

Set partition coding uses recursions of partitioning or splitting of sets of samples guided by a threshold test. The details will be explained at length in Part I of this monograph. There are different tests and different ways that these sets can be split. One unique feature is the presentation of the steps of the algorithms in separate boxes, in addition to the explanations. Contrary to what is written in many places, these methods are not married to bit-plane coding and do not by themselves produce embedded bitstreams. Bit-plane coding is an option and often an unnecessary one. Similarly, some authors assume that

these methods are meant to work only on wavelet transforms, without knowing that they had been successfully applied to other types of data.

Embedded coding within the framework of set partition coding will be carefully described. Integer data that tend to contain clusters of close value, such as those produced by certain mathematical transformations, are especially amenable to compression by most methods, including this method. Numerous examples of set partition coding of transformed data are presented. In particular, the SPIHT, SPECK and EZW (Embedded Zerotree Wavelet) [20] algorithms are explained thoroughly and compared in an actual example of coding a wavelet transform.

Part II describes current-day wavelet transform image coding systems. As before, steps of the algorithms are explained thoroughly and set apart. An image coding system consists of several stages of procedure: transformation, quantization, set partition or adaptive entropy coding or both, decoding including rate control, inverse transformation, de-quantization, and optional processing in certain stages. Wavelet transform systems can provide many desirable properties besides high efficiency, such as scalability in quality, scalability in resolution, and region-of-interest access to the coded bitstream. These properties are built into the JPEG2000 standard, so its coding will be fully described. Since JPEG2000 codes sub-blocks of subbands, other methods, such as SBHP (Subband Block Hierarchical Partitioning) [4] and EZBC [9], that code subbands or its subblocks independently are also described. The emphasis in this part is the use of the basic algorithms presented in the previous part in ways that achieve these desirable bitstream properties. In this vein, we describe a modification of the tree-based coding in SPIHT whose output bitstream can be decoded partially, corresponding to a designated region of interest and is simultaneously quality and resolution scalable.

Although not really necessary for understanding the coding methods presented here, we have included an appendix describing some mathematical transforms, including subband/wavelet transforms, used in coding systems. Some readers may feel more comfortable in reading this monograph with some introduction to this material.

This monograph is intended to be a tutorial and not a survey. Although many coding methods are described, some others that fit into the framework of this monograph are not, and the bibliography is certainly not comprehensive. We hope that any omissions will be forgiven. Nonetheless, we believe this tutorial is unprecedented in its manner of presentation and serves its stated purpose. We hope the readers agree.

2

Set Partition Coding

2.1 Principles

The storage requirements of samples of data depend on their number of possible values, called alphabet size. Real-valued data theoretically require an unlimited number of bits per sample to store with perfect precision, because their alphabet size is infinite. However, there is some level of noise in every practical measurement of continuous quantities, which means that only some digits in the measured value have actual physical sense. Therefore, they are stored with imperfect, but usually adequate precision using 32 or 64 bits. Only integer-valued data samples can be stored with perfect precision when they have a finite alphabet, as is the case for image data. Therefore, we limit our considerations here to integer data.

Natural representation of integers in a dataset requires a number of bits per sample no less than the base 2 logarithm of the number of possible integer values.[1] For example, the usual monochrome image has integer values from 0 to 255, so we use 8 bits to store every sample.

[1] When the lowest possible value in the dataset is not zero, then this offset from zero is stored as overhead.

Suppose, however, that we can find a group of samples whose values do not exceed 15. Then every sample in that group needs at most 4 bits to specify its value, which is a saving of at least 4 bits per sample. We of course need location information for the samples in the group. If the location information in bits is less than 4 times the number of such samples, then we have achieved compression. Of course, we would like to find large groups of samples with maximum values below a certain low threshold. Then the location bits would be few and the number of bits per sample would be small. The location information of each group together with its size and threshold determine the bit savings of every group and hence the compression compared to so-called raw storage using the full alphabet size for every sample. This is a basic principle behind set partition coding.

The reason why we discussed a group of samples and not individual samples is that location information for single samples would be prohibitively large. Ignoring for now the required location information, we can assess the potential compression gains just by adding the number of bits associated with the actual values of the samples. For example, the 512×512 monochrome image Lena, shown in Figure 2.1, is stored

Fig. 2.1 Lena image: dimension 512×512 pixels, 8 bits per sample.

as 8 bits for every sample. A histogram of this image is depicted in Figure 2.2. Note that moderate to large values are relatively frequent. In fact, given these values and their locations, the average number of minimal natural (uncoded) bits is 7.33 bits per sample. (What is meant by minimal natural bits is the base-2 logarithm of the value rounded up to the nearest integer.) Adding location information would surely exceed the 8 bits per sample for natural or raw storage. Clearly, set partition coding would not be a good method for this image. As this histogram is fairly typical for natural images, set partition coding should not be chosen for direct coding of natural images. For this method to be efficient, the source must contain primarily sets of samples with small maximum values. That is why we seek source transformations, such as discrete wavelet and cosine transformations, that have this property.[2] In Figure 2.3 is shown a histogram of an integer wavelet transform of the same source image Lena. It is evident that we have a large percentage of wavelet coefficients with small values and a low percentage with large values. In fact, the average number of minimal natural bits for the coefficients of this wavelet transform is 2.62 bits per sample (for 3 levels of decomposition). Clearly here, we have some room for location information to give us compression from the original 8 bits per sample. Set partition coding is a class of methods that can take advantage of the properties of image transforms. The members of this class differ in the methods of forming the partitions and the ways in which the location information of the partitions is represented.

2.1.1 Partitioning Data According to Value

Let us consider the source as either the original image or an unspecified transform of the image. To treat both image sources and transforms, let us assume that the values of the data samples are their magnitudes.

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_A\}$ be the finite set of all source samples. We postulate a set of increasing thresholds

$$\begin{aligned} v_{-1} = 0 < v_0 < v_1 < \cdots < v_N, \\ \max_k p_k < v_N \leq v_{N-1} + 2^{N-1}. \end{aligned} \tag{2.1}$$

[2] Appendix A contains an introduction to transforms.

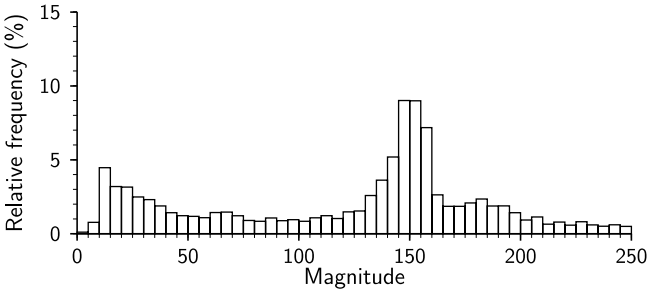


Fig. 2.2 Distribution of pixel values for the Lena image.

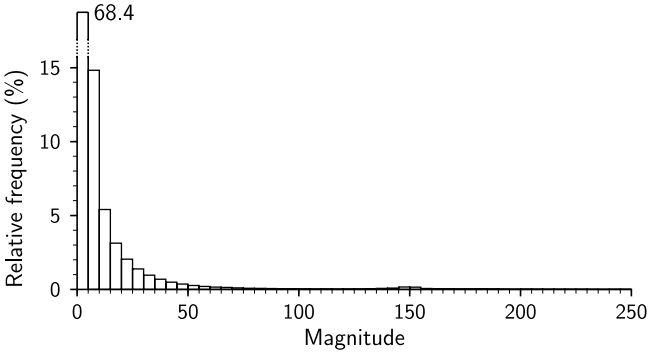


Fig. 2.3 Distribution of magnitudes of integer transform coefficients of the Lena image.

These thresholds are illustrated in Figure 2.4.

Consider the interval of values p between adjacent thresholds v_{n-1} and v_n and denote them as $\mathcal{P}_n$, where

$$\mathcal{P}_n = \{p : v_{n-1} \leq p < v_n\}. \tag{2.2}$$

Let us define the set of indices of the samples in an interval $\mathcal{P}_n$ as

$$S_n = \{k : p_k \in \mathcal{P}_n\}, \tag{2.3}$$

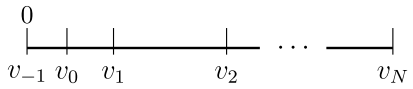


Fig. 2.4 Set of thresholds marking ranges of values in dataset.

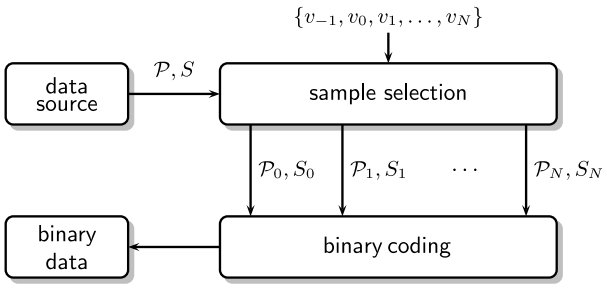


Fig. 2.5 Separation of values in dataset before coding.

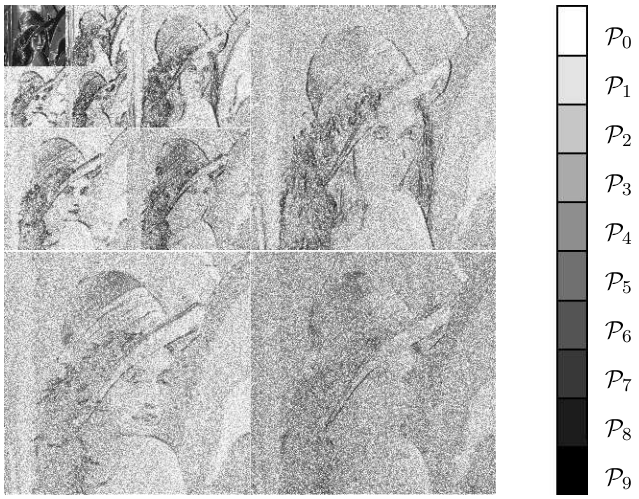


Fig. 2.6 Distribution of magnitudes of integer-wavelet-transform coefficients. Logarithmic scale, darker pixels representing larger values. ($\mathcal{P}_n$ notation explained in Section 2.1.1.)

Assume a procedure whereby the indices of the samples in $\mathcal{P}$ are sorted into a number of these index sets, $S_0, S_1, S_2, \ldots, S_N$. That is, indices of samples with values in interval $\mathcal{P}_0$ are put into S_0, indices of samples with values in interval $\mathcal{P}_1$ are put into S_1, and so forth. This procedure is what we call *set partitioning* of the data. A block diagram of a system that separates samples of the source into these sets is shown in Figure 2.5. Figure 2.6 shows an example of this type of data partitioning. In this figure, the magnitudes of coefficients of an integer wavelet transform, applied to image Lena (Figure 2.1) minus its mean,

are shown with pixel gray value defined by the set $\mathcal{P}_n$ to which the coefficient magnitudes belong. For instance, all pixels in $\mathcal{P}_0$ are shown in white, and those in $\mathcal{P}_9$ are shown in black.

Using $|S_n|$ to represent the number of elements (size) in the set S_n, it is easy to conclude that the minimal number of natural (uncoded) bits B to represent all these source values is

$$B = \sum_{n=0}^{N} |S_n| \left\lceil \log_2(v_n - v_{n-1}) \right\rceil . \tag{2.4}$$

If we use entropy coding, we may achieve lossless representation of the source with fewer than B bits, but we concentrate here on the principles to keep the discussion uncluttered. Therefore, in all that follows, except where especially noted, the code is natural binary (raw) representation.

We remind that B includes only the bits used to code the values of the samples. The information needed to determine the location of samples is conveyed by the indices in the sets S_n.

The coding process is simplified and more efficient when the intermediate thresholds are powers of two, i.e.,

$$v_n = 2^n, \quad n = 0, 1, \ldots, N-1 \tag{2.5}$$

with

$$N - 1 = \left\lfloor \max_{k \in \mathcal{P}} \{\log_2 p_k\} \right\rfloor \equiv n_{\max}. \tag{2.6}$$

We note that

$$v_n - v_{n-1} = 2^{n-1}$$

for $n \neq 0$ using these thresholds.

The exponent of two (n) is called the bit-plane index, and the largest index ($n_{\max}$) is called the most significant bit plane. It is possible to represent every data element perfectly using $n_{\max} + 1$ bits.[3] However, with

[3] Sometimes there is confusion about why $n_{\max}$ is defined in terms of the floor ($\lfloor\ \rfloor$) operator. ($\lfloor x \rfloor$ is defined as the largest integer not exceeding x.) If one recalls representation of an integer in a binary expansion, then $n_{\max}$ is the highest power of 2 in that expansion starting from 2^0.

the knowledge that $p_k \in \mathcal{P}_n$, $n \neq 0$, the number of bits to represent p_k perfectly is just $n-1$, because there are 2^{n-1} integer members of $\mathcal{P}_n$. The only value in $\mathcal{P}_0$ is 0, so knowing that an element is in $\mathcal{P}_0$ is sufficient to convey its value. Thus, Equation (2.4) for the minimum number of natural bits becomes

$$B = \sum_{n=1}^{N} |S_n|(n-1). \tag{2.7}$$

The indexing of thresholds and sets, although adopted for simplicity and consistency, may cause some confusion, so we illustrate with the following example.

Example 2.1. Suppose that the maximum value in our data set $\max_k p_k = 9$. Then $n_{\max} = 3$ and $N = 4$. Our intermediate thresholds are 2^n, $n = 0,1,2,3$. Table 2.1 shows the association of indices with the bitplanes, thresholds, and sets.

Three bits are needed to code $\max_k p_k = 9$, because eight elements are in its containing set $\mathcal{P}_4$.

As remarked earlier, the location information of the samples is contained in the elements of the sets S_n. We now turn our attention to methods of set partitioning that lead to efficient conveyance of these sets.

2.1.2 Forming Partitions Recursively: Square Blocks

An efficient method for generating partitions is through recursion. Here, we shall present a method where the partitions consist of square blocks of contiguous data elements. Since these elements are arranged as a

Table 2.1 Association of indices with various quantities.

n	Bit plane	v_n	2^n	$\mathcal{P}_n$
−1	—	0	—	—
0	0	1	1	{0}
1	1	2	2	{1}
2	2	4	4	{2,3}
3	3	8	8	{4,5,6,7}
4	4	9–16	16	{8,9,10,...,15}

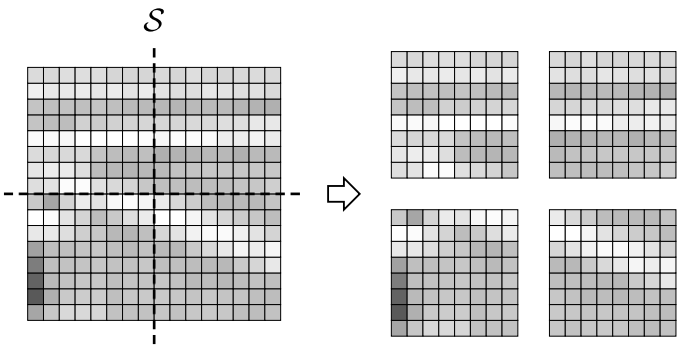


Fig. 2.7 Partitioning of an array of pixels $\mathcal{S}$ into four equal-sized subsets. Gray values are used for representing pixel values.

two-dimensional array, we shall call them *pixels* and suppose we have a square $2^m \times 2^m$ array of pixels.

First, the square array of source data is split into four $2^{m-1} \times 2^{m-1}$ quadrants, pictured in Figure 2.7. At least one of those quadrants contains an element $p_j \geq 2^{n_{\max}}$. Label those quadrants containing elements $p_j \geq 2^{n_{\max}}$ with "1", otherwise label those having no such elements with "0". The data elements in these quadrants labeled with "0" are seen to require at most $n_{\max}$ bits for lossless representation. Now, we split the "1" labeled quadrants into four $2^{m-2} \times 2^{m-2}$ element quadrants and test each of these four new quarter-size quadrants, whether or not all of its elements are smaller than $2^{n_{\max}}$. Again, we label these new quadrants with "1" or "0", depending whether any contained an element $p_j \geq 2^{n_{\max}}$ or not, respectively. Again any "0" labeled quadrant requires $n_{\max}$ bits for lossless representation of its elements. Any quadrant labeled "1" is split again into four equal parts (quadrisected) with each part again tested whether its elements exceed the threshold $2^{n_{\max}}$. This procedure of quadrisection and testing is continued until the '1'-labeled quadrants are split into 4 single elements, whereupon all the individual elements greater than or equal to $2^{n_{\max}}$ are located. These elements are known to be one of the $2^{n_{\max}}$ integers from $2^{n_{\max}}$ to $2^{n_{\max}+1} - 1$, so their differences from $2^{n_{\max}}$ are coded with $n_{\max}$ bits and inserted into the bitstream to be transmitted. The single elements less than $2^{n_{\max}}$ can be coded now with $n_{\max}$ bits. What also remains

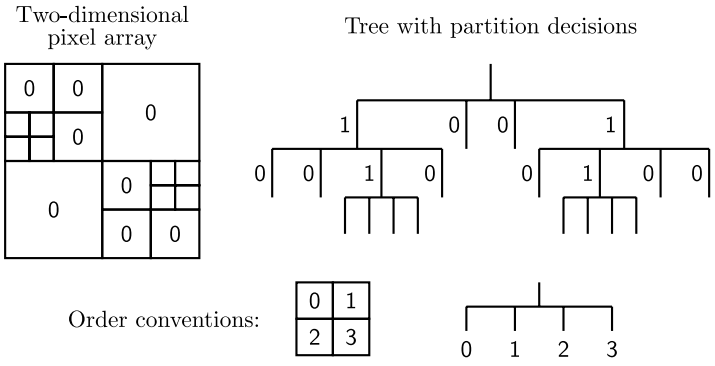


Fig. 2.8 Three levels of quadrisection of 8×8 block and associated quadtree and code.

are sets of sizes 2×2 to $2^{m-1} \times 2^{m-1}$ labeled with "0" to indicate that every element within these sets is less than $2^{n_{\max}}$. An illustration of three levels of splitting and labeling is shown in Figure 2.8.

We wish to do better than find sets requiring just one less bit for representation of its elements. So we lower the threshold by a factor of 2 to $2^{n_{\max}-1}$ and repeat the above procedure of quadrisection and labeling on the "0"-labeled sets already found. However, since we wish to identify and code individual elements as early as possible (the reason will become clearer later), these sets are tested in increasing order of their size. The result will emulate that of the previous higher threshold: individual elements labeled with "1" or "0" coded with $n_{\max}$ bits each as above and "0"-labeled sets of sizes 2×2 up to $2^{m-2} \times 2^{m-2}$. We then iterate this procedure on the previously found "0"-labeled sets by successively lowering the threshold by a factor of 2, quadrisecting when the maximum element in the set equals or exceeds the threshold, and labelling the new quadrants with "0" or "1" through the last threshold $v_1 = 2^0 = 1$.

The tests of comparing the maximum element of a set to a threshold are called *significance* tests. When the maximum is greater than or equal to a given threshold, we say the set is *significant*, with it being understood that it is with respect to this threshold. Otherwise, when the maximum is less than this threshold, the set is said to be *insignificant* (with respect to this threshold). The algorithm requires that sets

that test insignificant for a certain threshold be tested again at successively lower thresholds until they become significant. Therefore, we need a way to keep track of the insignificant sets that are constantly being created, so that they can be re-tested at lower thresholds. One way is to put them immediately onto an ordered list, which we call the LIS for List of Insignificant Sets. The coordinates of the top left corner of a set suffice to locate the entire set, since the size of the set is determined by the image size and the outcomes of the significance tests. Individual insignificant elements are also put onto the LIS through their coordinates. The steps are presented in detail in Algorithm 2.1.

In order to decode this bitstream, $n_{\max}$ and the labels "0" and "1" that are the outcomes of the threshold tests must also be sent to the coded bitstream. The decoder can thereby follow the same procedure as the encoder as it receives bits from the codestream. The order of visiting the quadrants and the "0"-labeled sets are understood by prior agreement at the encoder and decoder. To describe the decoding algorithm, the word "encode" is replaced by "decode" and the words "write to" are replaced by "read from" and the phrases in which they appear are reversed in order. (First "read" and then "decode.") In the last line, when $n = 0$, the "0"-labeled sets are decoded and all their elements are set to 0 value. Algorithm 2.2 shows the decoding algorithm in its entirety.

Algorithm 2.1. Algorithm for recursive set partitioning to encode a $2^m \times 2^m$ block of non-negative source elements.

(1) *Initialization*
   (a) Find $n_{\max}$, the most significant bit of the largest magnitude of the source.
   (b) Create a list LIS (List of Insignificant Sets), initially empty, to contain "0"-labeled (called insignificant) sets.
   (c) Put descriptor of $2^m \times 2^m$ source block onto LIS (upper left corner coordinates, plus size m).
   (d) Set $n = n_{\max}$. Encode $n_{\max}$ and write to codestream buffer.

(2) *Testing and partitioning*

(a) For each set on the LIS, do

i. If maximum element in set is less than 2^n, write "0" to codestream buffer.

A. If set has one element, write value using n bits to codestream buffer.

B. Remove set from LIS.

ii. Otherwise write "1" to codestream buffer and do the following:

A. if set has more than one element then divide the set into 4 equal quadrants, and add each quadrant to the end of the LIS.

B. If set has one element, then write its value minus 2^n using n bits to codestream buffer.

C. Remove set from LIS.

(3) If $n > 0$ then set $n = n - 1$ and return to 2a. Otherwise stop.

Algorithm 2.2. Algorithm for recursive set partitioning to decode a $2^m \times 2^m$ block of source elements.

(1) *Initialization*

(a) Create a list LIS (List of Insignificant Sets), initially empty, to contain "0"-labeled (called insignificant) sets.

(b) Put descriptor of $2^m \times 2^m$ source block onto LIS (upper left corner coordinates, plus size m).

(c) Read from codestream buffer and decode $n_{\max}$. Set $n = n_{\max}$.

(2) *Partitioning and recovering values*

(a) For each set on the LIS, do

i. Read next bit from codestream. If "0", do
   A. If set has more than one element, then return to 2a.
   B. If set has one element, read next n bits from codestream and decode value.
   C. Remove set from LIS.

ii. Otherwise, if "1," do the following:
   A. If set has more than one element, then divide the set into 4 equal quadrants, and add each quadrant to the end of the LIS.
   B. If set has one element, then read n bits from codestream to decode partial value. Value of "1"-associated element is decoded partial value plus 2^n.
   C. Remove set from LIS.

(3) If $n > 0$ then set $n = n - 1$ and return to 2a.

(4) Otherwise, set the value of all elements belonging to all sets remaining in the LIS to zero, and stop.

Clearly, this recursive set partitioning algorithm achieves the objective of locating different size sets of elements with upper limits of its elements in a defined range of values, so that they can be represented with fewer than $n_{\max}$ bits. The location information is comprised of the 4-bit binary masks that specifies whether or not a given set is quadrisected. These masks can be mapped onto a quadtree structure, as shown in the example of Figure 2.8. Therefore, this type of coding is often called quadtree coding. Note that the threshold lowering can be stopped before $2^0 = 1$, if we are content to use the number of bits corresponding to a higher threshold.

The performance of the algorithm in terms of compression ratio or bits per sample reduction depends very much on the characteristics of the source data. The input data must be a series of integers, so they can represent images, quantized images and transforms, and integer-valued transforms. Whatever the integer input data, they are encoded

Table 2.2 Recursive block partitioning in lossless coding of images.

Image	Attributes	Rate in bpp
Lena	512×512, 8 bpp	4.489
Goldhill	720×576, 8 bpp	4.839
Bike	2048×2560, 8 bpp	4.775
Barbara	512×512, 8 bpp	4.811

without loss by the algorithm. The coefficients of transforms can be negative, so they are represented with their magnitudes and signs. The threshold tests are carried out on the magnitudes, and when the individual elements are encoded, one more bit is added for the sign. We have programmed this algorithm and run it to compress an integer wavelet transform of a few images to assess its performance. For our purposes now, we can just consider that the original image array has been transformed to another array of integers with a different distribution of amplitudes.[4] The results for this perfectly lossless compression are presented in Table 2.2.

So we do get compression, since the amount of location information of the masks is less than the number of bits needed to represent the elements in the partitioned input. We can get further compression, if we entropy encode the masks, the individual coefficients, or the signs or any combination thereof. Of course, that requires gathering of statistics and substantial increase in computation. One can usually get results comparable to the state-of-the-art by adaptive entropy coding of the masks only.

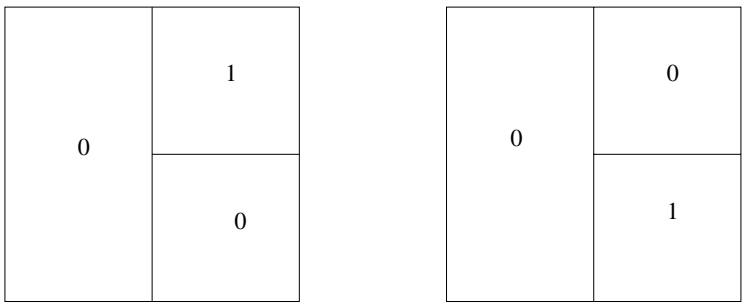
2.1.3 Binary Splitting

Quadrisection is not the only means of partitioning a square block. Another potentially effective procedure is bisection or binary splitting. The idea is to split into two in one dimension followed by a second split into two in the second dimension of any significant half. An example is shown in Figure 2.9(a) in which the significant square set is split into two vertically and the right hand significant half is split horizontally. We adopt the usual convention that a "0" signifies that a subset is

[4] For these tests, we have used the integer S+P filters [17] in five levels of wavelet decomposition.

insignificant and "1" that it is significant and the order is left to right in the vertical split and top to bottom in the horizontal split. Following this convention, the output bit sequence using binary splitting is 0110 for the significance pattern 0100 found by scanning the four quadrants in raster order. However, since we only split significant sets, the first "0" also signifies that the right half is significant, so the following "1" is redundant. Therefore, without loss of information, we can reduce the output bit sequence to 010. In quadsplitting, we would need four bits to represent the significance pattern 0100.

The conventional order for encoding bits in quadsplitting is also the raster order (left to right and then top to bottom), so that the significance pattern 0001 can be encoded using quadrisection to 000 without loss of information. In fact, that is the only reduction possible in quadsplitting. More are possible in binary splitting. It may take up to three binary splits to deduce the significance pattern of the four quadrants. Whenever the raw pattern "01" occurs in any of these binary splits, it can be reduced to "0". In the one case of the significance pattern 0001 (raster order), depicted in Figure 2.9(b), the raw output pattern "0101" reduces to only the two bits "00". However, there are significance patterns needing more than two binary splits to encode. For example, take the significance pattern and quadsplit code 1101. Both the left and right halves of the vertical split are significant, giving 11. The left half horizontal split yields 10 and the right one 11. This is an example of when



(a) Code 010 (b) Code 00

Fig. 2.9 Bisection codes of significance patterns: (a) 0100 and (b) 0001.

Table 2.3 Bisection vs. quadrisection of significant square array.

Significance pattern (raster order)	Quadrisection code	Bisection code
0001	000	00
0010	0010	100
0011	0011	1100
0100	0100	010
0101	0101	011
0110	0110	11010
0111	0111	11011
1000	1000	1010
1001	1001	11100
1010	1010	1011
1011	1011	11110
1100	1100	111010
1101	1101	111011
1110	1110	111110
1111	1111	111111

both halves of the first split are significant. Then we must split each of these subsets horizontally and send the additional bit patterns 10, 11, or 0 (for 01). (Pattern "00" can not occur.) In Table 2.3, we list the bisection and quadrisection codes for all significant quadrant patterns. In order to determine whether bisection or quadrisection is preferable, one must take into account the relative frequencies of the occurrences of the patterns of significance of the four quadrants. In other words, if the data favors more frequent occurrences of the significance patterns giving shorter bisection codewords, then bisection would produce a shorter average codeword length than would the quadrisection code. Entropy coding would yield the same codes with both methods, since the codewords then would be a function only of the probabilities of the significance patterns.

2.1.4 One-dimensional Signals

We have been using two-dimensional data arrays as our targets for encoding by set partitioning. However, bisection is a natural and efficient choice for encoding one-dimensional digital signal waveforms. A finite length signal then would be encoded by recursive binary splitting of segments that are significant for the given threshold until the significant single elements are located. The encoding algorithm is the

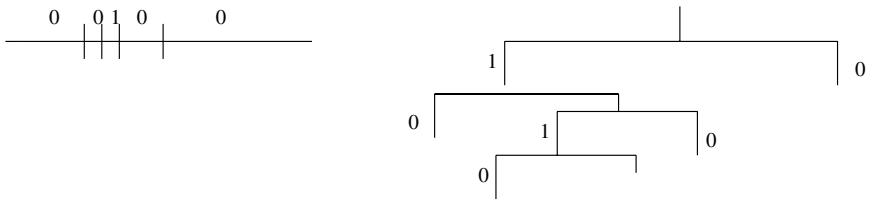


Fig. 2.10 Bisection of signal — left, splitting pattern 10011001; right, corresponding binary tree with bisection code 100100 labeled on branches.

same as Algorithm 2.1, except that there are 2^m single coordinates, sets are divided equally into two subsets, and the significance labeling of these subsets creates a 2-bit mask. The same changes obtain in the corresponding decoding algorithm, Algorithm 2.2. The splitting sequence can be mapped onto a binary tree, as shown in Figure 2.10. Note that when "01" reduces to "0", it results in a collapse of the "1" branch.

2.2 Tree-Structured Sets

The previously described sets were blocks of contiguous elements that form a partition of the source. Another kind of partition is composed of sets in the form of trees linking their elements. These so-called tree-structured sets are especially suitable for coding the discrete wavelet transform (DWT) of an image. The DWT is organized into contiguous groups of coefficients called subbands that are associated with distinct spatial frequency ranges.[5] One can view the DWT as a number of nonoverlapping spatial orientation trees. These trees are so called, because they branch successively to higher frequency subbands at the same spatial orientation. An example of a single spatial orientation tree (SOT) in the wavelet transform domain is shown in Figure 2.11. Note that the roots are the coefficients in the lowest frequency subband and branch by three to a coefficient in the same relative position in each of the other three lowest level subbands. All subsequent branching of each node is to four adjacent coefficients in the next higher level subband in the same relative position. The set of three or four nodes branching from

[5] See Appendix A for an explanation of subband transforms, including the wavelet transform.

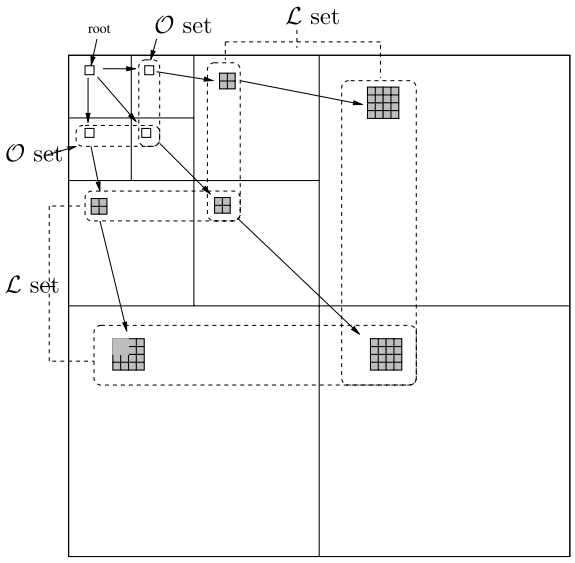


Fig. 2.11 A spatial orientation tree (SOT) of a discrete wavelet transform. $\mathcal{O}$ denotes offspring set and $\mathcal{L}$ denotes grand-descendant set. The full descendant set $\mathcal{D} = \mathcal{O} \cup \mathcal{L}$.

a single node are called the *offspring* set. The set of all descendants from nodes in the offspring set is called the *grand-descendant* set. The SOT's are nonoverlapping and taken altogether include all the transform coefficients. Therefore, the set of all SOT's partition the wavelet transform.

Although tree-structured partitions are used mostly with wavelet transforms, they can be used for the discrete cosine transform (DCT) [24] or even for the source itself, when the data organization seems appropriate. The DWT is used here to present motivation and an example of tree-structured sets. The focus here is the formation of trees and coding of the elements that are the nodes in these trees. It is immaterial as to how the dataset or transform was generated. Within these trees, we wish to locate subsets whose coefficients have magnitudes in a known range between two thresholds. We take our hint from the recursive block procedure just described.

Here we start with a number of these tree-structured sets, one for each coefficient in the lowest frequency subband. However, now we re-arrange the coefficients of each tree into a block, as shown in Figure 2.12. Let us call each such block a tree-block. Each tree-block

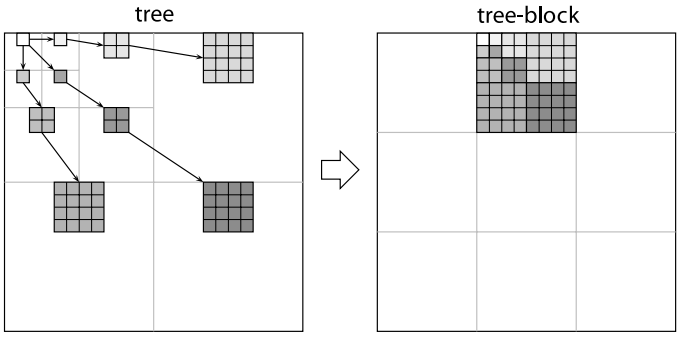


Fig. 2.12 Rearranging a spatial orientation tree into a tree-block placed in the position of the image region it represents.

is coded in turn as in Algorithm 2.1. However, we have the option of using either a single LIS and passing through all the blocks in turn at the same threshold or using an LIS for each block and staying within the same block as the threshold is successively lowered. For the latter option, we obtain essentially a separate codestream for each block, while in the former we encode all higher value coefficients across the blocks before the lower value ones.

Algorithm 2.3. Algorithm for encoding tree-blocks.

(1) Form the tree-blocks $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_K$.
(2) Create a list LIS (List of Insignificant Sets) initially empty. LIS will contain "0"-labeled (called insignificant) sets.
(3) Put coordinates of roots of tree-blocks on the LIS.
(4) Calculate $n_{\max}$, the highest bit-plane index among all the blocks.
(5) Set $n = n_{\max}$.
(6) For $k = 1, 2, \ldots, K$, set m=D and execute Step 2a of Algorithm 2.1 on $\mathcal{B}_k$.
(7) If $n > 0$ and there are multiple element sets on the LIS, set $n = n - 1$ and return to 6.
(8) If $n > 0$ and all the LIS sets comprise single elements, encode each with n bits, write to codestream, and stop.
(9) If $n = 0$, stop.

The procedure that uses an LIS for each tree-block can be summarized as follows. Assume the DWT is a square $M \times M$ array of coefficients $c_{i,j}, i, j = 1, 2, \ldots, M$. Let $\mathcal{H}$ denote the set of coefficients in the lowest spatial frequency subband. Every $c_{i,j}$ in $\mathcal{H}$ is the root of a spatial orientation tree (SOT). Group the coefficients of each SOT with root in $\mathcal{H}$ into a tree-block. For a dyadic DWT with D levels of decomposition, there will be $M^2/2^{2D}$ tree-blocks with dimensions $2^D \times 2^D$ elements. Denote these tree-blocks $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_K$, $K = M^2/2^{2D}$. They are usually ordered either by a raster scan or zig–zag scan of their roots in $\mathcal{H}$. For each tree-block $\mathcal{B}_k$, encode with the set partitioning algorithm, Algorithm 2.1, with $m = D$ while generating a corresponding LIS_k. Note that each tree-block has its own most significant bit plane $n_{\max}$. The codestream is a concatenation of the separate codestreams for each $\mathcal{B}_k$.

The alternative procedure uses a single LIS and passes through all the tree-blocks at the same threshold. It starts as above with the tree-blocks $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_K$, $K = M^2/2^{2D}$. When a tree-block is fully tested and partitioned at a certain threshold, the next tree-block is entered for testing and partitioning at the same threshold. Insignificant sets are put on the common LIS. The steps of this encoding algorithm are delineated in Algorithm 2.3. The decoding algorithm follows in the usual obvious way.

2.2.1 A Different Wavelet Transform Partition

Spatial Orientation Tree Encoding. Another method for partitioning the wavelet transform is the one introduced in the SPIHT coding algorithm [18], which will be visited later. Here we describe essentially the same method, but on a different configuration of the spatial orientation trees than defined in SPIHT. We consider here the same SOT's as before with the same significance test. However, we do not gather the coefficients of the trees into blocks. We test a given tree for significance at the current threshold and if significant, we divide the tree into its immediate offspring (child) coefficients and the set of all coefficients descending from the child coefficients, as shown in Figure 2.11. The latter set is called the grand-descendant set. We start with a

full tree rooted at a coefficient in $\mathcal{H}$, the lowest frequency subband. When this tree tests significant, it is split into child nodes (coefficients) and the grand-descendant set. The child coefficients are individually tested for significance, followed by testing of the grand-descendant test for significance at the same threshold. If the grand-descendant set is significant, it is split into its component sub-trees descended from the the child coefficients, again depicted in Figure 2.11. Now each of these sub-trees (minus their roots) is tested for significance and split as before into its child coefficients and their grand-descendant set, the set of all coefficients descended from these child coefficients. The testing of the individual child coefficients and the grand-descendant set of each of these subtrees continue in the same manner until there are empty grand-descendant sets at the terminal nodes of the trees.

In order to keep track of the various sets, we use here two lists: one for keeping track of single coefficients, called LIP (for List of Insignificant Pixels); and LIS, for keeping track of sets of more than one coefficient. All single coefficients not yet tested at the current threshold are put onto the LIP. All sets not tested at the current threshold are put onto the LIS. Therefore, when a tree is split, the coordinates of the child nodes (3 for the first split in a tree, four thereafter) go to the LIP and the locator of a grand-descendant set goes to the LIS. The coordinates of the tree root together with a Type B tag, indicating a grand-descendant set, constitute the locator. Now the LIP coefficients are tested at the same threshold. If an LIP member is significant, it is removed from the LIP, and a "1" and the code of its value (magnitude and sign) are written to the codestream.[6] If insignificant, it remains on the LIP and a "0" is written to the codestream. After all the LIP coefficients are tested, the LIS is tested. If insignficant, it remains on the LIS and a "0" is written to the codestream. If significant, it is removed from the LIS and partitioned into the subtrees descending from the child nodes (see Figure 2.11). The subtrees are put onto the LIS for immediate testing. They are designated by their roots, the child node

[6] The "1" indicates a value equal or above a known threshold, so only the code of the difference from threshold needs to be sent to the codestream.

coordinates, together with a Type A tag, to indicate all descendants of the child node root, but not including this root.

We see that there are two kinds of splitting, one that produces Type A LIS entries, indicating all descendants of the root, and one producing Type B LIS entries, indicating grand-descendant sets. The initial tree and the descendants of a child node are Type A sets. The use of an additional list, the LIP, for single insignificant coefficients, is convenient in this kind of splitting procedure that explicitly splits off single coefficients whenever any multiple element set is split. As with the previous procedures, the single elements on the LIP are always tested before the multiple element ones on the LIS. The justification is that the coefficients in an SOT generally decrease in magnitude from top to bottom and that the children of a significant root are highly likely to be significant either at the same or the next lower threshold.

Encoding SOT's Together. When we search through one tree at a time for significant pixels and sets, according to the procedure above, it is called a *depth-first* search. When we search across the trees for significant pixels and sets, it is called a *breadth-first* search. In such a search procedure, all the significant pixels at given threshold are determined before re-initiating the search at the next lower threshold. Therefore, we achieve an encoding that builds value in each threshold pass as quickly as possible. We now combine the above splitting and list management procedures to create the breadth-first SOT Encoding Algorithm. The following sets of coordinates are used to present this coding algorithm:

- $\mathcal{O}(i,j)$: set of coordinates of all offspring of node (i,j);
- $\mathcal{D}(i,j)$: set of coordinates of all descendants of the node (i,j);
- $\mathcal{H}$: set of coordinates of all spatial orientation tree roots (nodes in the lowest spatial frequency subband);
- $\mathcal{L}(i,j) = \mathcal{D}(i,j) - \mathcal{O}(i,j)$, the grand-descendant set.

For instance, except at the highest and lowest tree levels, we have

$$\mathcal{O}(i,j) = \{(2i,2j),(2i,2j+1),(2i+1,2j),(2i+1,2j+1)\}. \quad (2.8)$$

We also formalize the significance test by defining the following function

$$\Gamma_n(\mathcal{T}) = \begin{cases} 1, & \max\limits_{(i,j)\in\mathcal{T}} \{|c_{i,j}|\} \geq 2^n, \\ 0, & \text{otherwise}, \end{cases} \tag{2.9}$$

to indicate the significance of a set of coordinates $\mathcal{T}$. To simplify the notation of single pixel sets, we write $\Gamma_n(\{(i,j)\})$ as $\Gamma_n(i,j)$.

As stated previously, we use parts of the spatial orientation trees as the partitioning subsets in the sorting algorithm. The set partitioning rules are simply:

(1) the initial partition is formed with the sets $\{(i,j)\}$ and $\mathcal{D}(i,j)$, for all $(i,j) \in \mathcal{H}$;
(2) if $\mathcal{D}(i,j)$ is significant then it is partitioned into $\mathcal{L}(i,j)$ plus the four single-element sets with $(k,l) \in \mathcal{O}(i,j)$.
(3) if $\mathcal{L}(i,j)$ is significant then it is partitioned into the four sets $\mathcal{D}(k,l)$, with $(k,l) \in \mathcal{O}(i,j)$.

As explained before, to maintain proper order for testing significance of sets, the significance information is stored in two ordered lists, called *list of insignificant sets* (LIS), and *list of insignificant pixels* (LIP). In these lists each entry is identified by a coordinate (i,j), which in the LIP represents individual pixels, and in the LIS represents either the set $\mathcal{D}(i,j)$ or $\mathcal{L}(i,j)$. To differentiate between them we say that an LIS entry is of type A if it represents $\mathcal{D}(i,j)$, and of type B if it represents $\mathcal{L}(i,j)$.

All the pixels in the LIP — which were insignificant in the previous pass — are tested in turn, and those that become significant are encoded and outputted. Similarly, sets are sequentially evaluated following the LIS order, and when a set is found to be significant it is removed from the list and partitioned. The new subsets with more than one element are added back to the LIS, while the single-coordinate sets are added to the end of the LIP or encoded, depending whether they are insignificant or significant, respectively. Algorithm 2.4 delineates the entire procedure.

Algorithm 2.4. *Sot Encoding Algorithm*: Encoding an Image Wavelet Transform along SOTs.

(1) *Initialization*: output $n = \lfloor \log_2 \left(\max_{(i,j)} \{|c_{i,j}|\} \right) \rfloor$; add the coordinates $(i,j) \in \mathcal{H}$ to the LIP, and also to the LIS, as type A entries.
(2) *Sorting pass*
  (a) for each entry (i,j) in the LIP do:
    i. output $\Gamma_n(i,j)$;
    ii. if $\Gamma_n(i,j) = 1$ then output (write to code-stream) magnitude difference from 2^n (n bits) and sign of $c_{i,j}$;
  (b) for each entry (i,j) in the LIS do:
    i. if the entry is of type A then
      - output $\Gamma_n(\mathcal{D}(i,j))$;
      - if $\Gamma_n(\mathcal{D}(i,j)) = 1$ then
        — for each $(k,l) \in \mathcal{O}(i,j)$ do:
          * output $\Gamma_n(k,l)$;
          * if $\Gamma_n(k,l) = 1$, output the magnitude difference from 2^n (n bits) and sign of $c_{k,l}$;
          * if $\Gamma_n(k,l) = 0$ then add (k,l) to the end of the LIP;
        — if $\mathcal{L}(i,j) \neq \emptyset$ then move (i,j) to the end of the LIS, as an entry of type B; otherwise, remove entry (i,j) from the LIS;
    ii. if the entry is of type B then
      - output $\Gamma_n(\mathcal{L}(i,j))$;
      - if $\Gamma_n(\mathcal{L}(i,j)) = 1$ then
        — add each $(k,l) \in \mathcal{O}(i,j)$ to the end of the LIS as an entry of type A;
        — remove (i,j) from the LIS.
(3) *Threshold update*: decrement n by 1 and go to Step 2.

One important characteristic of the algorithm is that the entries added to the end of the LIS in Step 2(b)ii are evaluated before that same sorting pass ends. So, when we say "for each entry in the LIS" we also mean those that are being added to its end. Note that in this algorithm, all branching conditions based on the significance test outcomes Γ_n — which can only be calculated with the knowledge of $c_{i,j}$ — are output by the encoder. Thus, the decoder algorithm duplicates the encoder's execution path as it receives the significance test outcomes. Thus, to obtain the decoder algorithm, we simply have to replace the words *output* by *input*. The other obvious difference is that the value $n_{\max}$, which is received by the decoder, is not calculated.

2.2.2 Data-dependent Thresholds

Using a set of fixed thresholds obviates the transmission of these thresholds as side information. Choices of thresholds and their number have an impact on compression efficiency. One wants to identify as many newly significant sets as possible between successive thresholds, but at the same time have these sets be small and identify large insignificant sets. Choosing too many thresholds may result in sending redundant information, since too few changes may occur between two thresholds. Choosing too few thresholds will inevitably lead to large significant sets that require many splitting steps to locate the significant elements. The best thresholds for a given source are necessarily data-dependent and must decrease as rapidly as possible without producing too many large significant sets.

Consider the thresholds to be the maximum (magnitude) elements among the sets split at a given stage. For example, say we have found the maximum value in a set and split this set into four subsets. Find the maximum elements in each of these four subsets. Split into four any subset having an element whose value equals the set maximum. Such a subset is said to be significant. Now repeat the procedure for each of these four new subsets and so on until the final splits that produce four single elements. These single elements are then encoded and ouputted to the codestream. Clearly, we are creating subsets, the ones not further split, whose elements are capped in value by the full set

maximum. These may be called the insignificant sets. The maximum of each subset that is not further split is less than the previous maximum. Then each of these subsets is split, just as the full set, but with a lesser maximum value to guide further splitting. These subsets should be tested from smallest to largest, that is, in reverse of the order they were generated. The reason is that we can more quickly locate significant single elements, thereby finding earlier value information, when the sets are smaller and closer to those already found. Clearly, we are producing a series of decreasing thresholds, as so-called insignificant sets are generated. The steps in the modification of the fixed threshold algorithm, Algorithm 2.1, to an adaptive one are presented in detail in Algorithm 2.5.

One major difference from the previous preset thresholds is that the thresholds generated here depend on the original data, so have to be encoded. Prior to splitting of any set, its maximum is found. An efficient scheme is to generate a series of 4-bit binary masks, where a "1" indicates a significant set (having an element equal to the maximum) and "0" an insignificant set (no element within having a value equal to the maximum). A "1" in one of these masks specifies that the maximum is to be found in this set. A "0" indicates that the maximum element is not in the associated set and that a new (lesser) maximum has to be found and encoded. The masks themselves have to be encoded and outputted, so that the decoder can follow the same splitting procedure as the encoder. The maxima can be coded quite efficiently through differences from the global maximum or a similar differencing scheme. Each 4-bit mask can be encoded with the natural binary digits or with a simple, fixed Huffman code of 15 symbols (four "0"s cannot occur).

The above scheme works well when there are relatively large areas with 0-value elements, as found naturally in the usual image transforms. The compression results are comparable to those yielded from fixed thresholds. If the probabilities of values are approximately or partially known beforehand, it is advisable to index the values, so that the highest probability value has 0 index, the next highest 1, and so forth. Then this set partitioning and encoding procedure operates on these indices. The association of the indices to the actual values must be known beforehand at the decoder to reconstruct the source data.

Algorithm 2.5. Algorithm for recursive set partitioning with adaptive thresholds to encode a $2^m \times 2^m$ block of non-negative source elements.

(1) *Initialization*

  (a) Find $v_{\max}$, the largest magnitude among the source elements.

  (b) Create a list LIS (List of Insignificant Sets) initially empty. LIS will contain "0"-labeled (called insignificant) sets.

  (c) Put coordinates of $2^m \times 2^m$ source block onto LIS. (Only upper left corner coordinates are needed.) Label set with "0".

  (d) Set threshold $t = v_{\max}$ and bit-plane index $n = \lfloor \log_2 t \rfloor$. Encode t and write to codestream buffer.

(2) *Testing and partitioning*

  (a) For each multiple element set on the LIS (all "0"-labeled sets with more than one element), do

    i. if maximum element less than t, write "0" to codestream buffer, else write "1" to codestream buffer and do the following:

      A. Remove set from LIS and divide the set into 4 equal quadrants.

      B. Find maximum element of each quadrant $v_q, q = 1, 2, 3, 4$. For each quadrant, label with "1," if $v_q = t$, otherwise label with "0" if $v_q < t$. (This step creates a 4-bit binary mask.)

      C. Add coordinates of "0"-labeled quadrants onto LIS. Encode 4-bit binary mask and write to codestream buffer.

      D. For each quadrant of size 2×2 or greater labeled with "1", go to 2(a)iA.

E. When quadrants have been split into single elements, encode "0"-labeled elements with $n+1$ bits and write to codestream buffer. ("1"-labeled single elements have values equal to the known t.)

(3) If $t > 0$ and there are multiple element sets on the LIS, reset t equal to maximum of all the maxima of the LIS sets, encode t, set $n = \lfloor \log_2 t \rfloor$, and return to 2a.

(4) If $t = 0$, stop.

2.2.3 Adaptive Partitions

The methods described above always split the set into fixed-shape subsets, such as quadrants of a block or children and grand-descendant sets of a tree. Also the iterative splitting into subsets numbering four seems somewhat arbitrary and other numbers may prove better. Since the attempt is to find clusters of insignificant (or significant) elements, why not adjust the partition shape to conform better to the outer contours of the clusters? As stated before, methods whose sets are exactly these clusters need too much location information as overhead. But there are methods that try to vary the number of clusters and the splitting shape in limited ways that prove to be quite effective for compression.

One such method is a form of *group testing*. The basic procedure is to start with a group of k linearly ordered elements in the set K. We wish to test this set as before to locate significant elements and clusters of insignificant elements. Again, we envision a threshold test of a set that declares "1" as significant when there is a contained element whose value passes the test and declares "0" as insignificant when all contained elements fail the test. If the set K is significant, it is then split into two nearly equal parts, K_1 and K_2 of k_1 and k_2 elements, respectively. The smaller part, say K_1, is first tested for significance. If significant, K_1 is further split into two nearly equal subsets. If insignificant, then the other part, K_2, must be significant, so it is split into two nearly equal parts. This testing and splitting is repeated until the significant single elements are located. Whenever the threshold test is enacted, a "1"

Table 2.4 Group Testing of Binary Sequence 00000110.

Step number	Test elements (in braces)	Output code
1.	⌈00000110⌉	1
2.	⌈0000⌉0110	0
3.	0000⌈01⌉10	1
4.	0000⌈0⌉110	0
5.	000001⌈10⌉	1
6.	000001⌈1⌉0	1
7.	0000011⌈0⌉	0

or "0" is emitted to indicate a significant or insignificant result, respectively. We call this procedure a group iteration, where K is the group.

Consider the example of coding the binary sequence $K = \{00000110\}$ of $k = 8$ elements in Table 2.4. Here the threshold is 1, so "0" denotes an insignificant element and "1" a significant one. The test of K produces an output "1," because there is at least one "1" in the sequence. Therefore, it is split into the subsequences 0000 and 0110, whereupon the test of the first produces a "0" output, because it is insignificant. The second subsequence must be significant, so it is split into the dibits 01 and 10. The first dibit is significant, so "1" is the output, and it is split into 0 and 1. Since the 0 is insignificant, the second element must be significant, so only a "0" is the output for 0 and 1. The 10 is significant, so only a "1" is the output and it is likewise split into single digits 1 and 0, Since the first digit is 1, a "1" is the output, followed by the second digit's "0", since it too has to be tested. The resulting code is 1010110, seven digits compared to the original eight in this example.

Now when we have a large set of elements, we do not want to start the above iteration with the full set, because there is an optimum group size for the above iteration, based on the probability θ of a "0" or insignificant element. It turns out there is an equivalence between the encoding procedure above and the Golomb code [7], where it has been shown that a good rule for choosing group size k is k that satisfies[7]

$$\theta^k \approx \frac{1}{2},$$

but not less than $\frac{1}{2}$.

[7] A more precise statement of the rule is $\theta^k + \theta^{k+1} < 1 < \theta^k + \theta^{k-1}$ [6].

You usually do not know *a priori* the value of θ and it is often too inconvenient or practically impossible to pass through all the data to estimate it. Furthermore, this value may vary along the data set. Therefore, an adaptive method, based on a series of estimates of θ is a possible solution. One method that seems to work well is to start with group size $k = 1$ and double the group size on the next group iteration if no significant elements are found on the current iteration. Once significant elements are identified, we can estimate the probability θ as the number of insignificant elements identified so far divided by the total of elements thus far identified. For example, in the single group iteration of Table 2.4, the probability estimate would be $\frac{6}{8} = 0.75$. Therefore, we would take the next group size $k = 2$, according to the approximate formula above with $\theta = 0.75$. We would accumulate the counts of insignificant and total elements thus far identified and update these estimates as we pass through the data with these varying group sizes.

2.3 Progressive Transmission and Bit-Plane Coding

All the coding techniques so far described have in common that higher magnitude elements are encoded before lower value ones. Therefore, they provide a form of progressive transmission and recovery, since the reconstruction fidelity is increasing at the fastest possible rate as more codewords are received. One might call such encoding progressive in value. Another form of progressive encoding of finer granularity is achieved by progressive bit-plane coding, when the same bit planes of currently significant elements are sent together from higher to lower bit-planes. In order to realize such encoding, the significant elements cannot be immediately outputted, but instead are put onto another list, which is called the LSP for List of Significant Pixels. These elements are represented by magnitude in natural binary and sign. After the next lower threshold, the intermediate bit planes of all previously significant elements are output to the codestream. When the thresholds are successive integer powers of 2, then only the current bit-plane bits of previously significant elements are output. An illustration of the LSP with progressive bit-plane coding appears in Figures 2.13 and 2.14 for cases of successive and nonsuccessive integer powers of two thresholds.

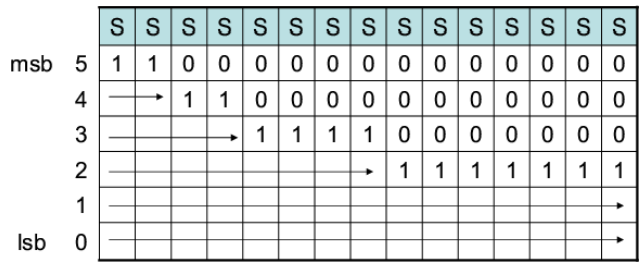


Fig. 2.13 Progressive bit-plane coding in the LIS for successive power of 2 thresholds. S above the most significant bit (msb) stands for the bit indicating the algebraic sign.

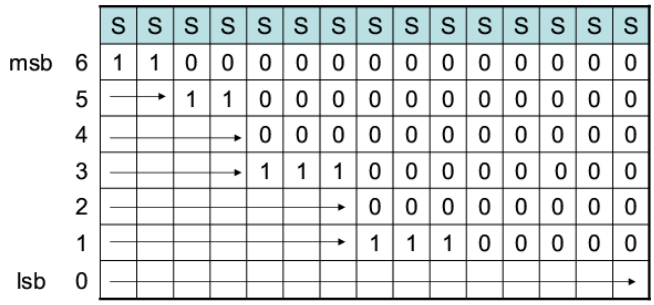


Fig. 2.14 Progressive bit-plane coding in the LIS for general thresholds. S above the most significant bit (msb) stands for the bit indicating the algebraic sign.

In the nonsuccessive case, where decreasing group maxima are the thresholds, it may be more efficient to set the thresholds to the largest power of two in the binary representation of magnitude, in other words, to the highest nonzero bit level. That is, for a given maximum V, set the threshold to 2^n, where $n = \lfloor \log_2 V \rfloor$. In this way, the thresholds will decrease more rapidly, giving fewer passes and fewer and more compact transmission of thresholds that are represented with small integers.

One of the benefits of progressive bit-plane coding is that the passes through bit planes can be stopped before the end of the bottom bit plane to produce a lossy source code that has close to the smallest squared error for the number of bits encoded up to the stopping point.[8] This means of lossy coding is called *implicit*

[8] The smallest squared error is obtained either when the stopping point is the end of a bit plane or the coefficients happen to be fully ordered by magnitude.

quantization, because its effect is a quantization step size one-half of the threshold corresponding to the bit plane where the coding stops. The partial ordering of LSP coefficients by bit plane significance and successive refinement from highest to lowest bit planes gives almost the smallest distortion possible for the bit rate corresponding to the stopping point. For a codestream of a given rate (number of bits), all the best lower rate codes can be extracted just by cutting the end of the codestream. Such a codestream is said to be *embedded* and, in this case, is said to be *bit embedded* or *finely embedded,* because the stream can be cut to any lower number of bits.

2.4 Applications to Image Transform Coding

Set partition coding methods are especially suitable for coding transforms of images. The reason is that transforms compact the energy of the source into a relatively small percentage of the transform coefficients, leaving relatively large regions of zero or small values. The two most prevalent transforms, the discrete cosine transform (DCT) and the discrete wavelet transform (DWT), are clear examples of this phenomenon. The typical DCT of an image contains large values at low spatial frequency coefficients and small or zero values otherwise. A DWT produces coefficients belonging to spatial frequency ranges called *subbands*. The size (number of coefficients) of subbands increases in octaves from low to high frequency. The typical DWT of an image contains coefficients of large values in the low spatial frequency subbands and decreasing values as the spatial frequency of its subbands increases. For a stationary Gauss–Markov image source, the DCT coefficients are Gaussian and asymptotically statistically independent [25]. Subbands of transforms generated by ideal ("brick-wall") filtering of a stationary Gaussian source are statistically independent, because they have no overlap in frequency. These asymptotic or ideal statistical properties of transforms have motivated much of the original coding methodology in the literature. Images, like most real-world sources, are neither stationary nor Gaussian and realizable filters are not ideal, so nonstatistically based techniques, such as set partition coding, have arisen to

treat these more realistic sources whose transforms have these energy clustering characteristics.

In this section, we shall describe the application of methods of set partition coding to image coding. However, because our focus is the particular coding algorithm in its use to take advantage of certain data characteristics, we deliberately do not describe here the mathematical operations or implementation of these transforms. We describe just the characteristics of the output of these transforms that are needed for proper application of the coding algorithms.

2.4.1 Block Partition Coding and Amplitude and Group Partitioning (AGP)

We consider, for the sake of simplicity, a square $M \times M$ image that has been transformed either blockwise by a $2^n \times 2^n, n = 2, 3, 4$ DCT or by a DWT yielding subbands of sizes $M2^{-m} \times M2^{-m}, m = 1, 2, \ldots, D$, where D is the number of stages of decomposition. For example, the JPEG standard calls for 8×8 DCT blocks and D is typically 3 to 6 in wavelet transform coders. In either case, we are faced with square blocks of transform coefficients.

One of the problems confronting us is that of large alphabet size. For almost any method of entropy coding, large alphabet means large complexity. For the block partition method (Algorithm 2.1) which uses group maxima as thresholds, this complexity is reflected in many closely spaced maxima and therefore many passes through the data. So before we enact this method, we reduce the alphabet size substantially using *alphabet partitioning*. Alphabet partitioning consists of forming groups of values and representing any value by two indices, one for the group, called the *group index*, and one for the position or rank within the group, called the *symbol index*. We itemize the steps in the explanation of alphabet partitioning below.

- The source alphabet is partitioned, before coding, into a relatively small number of groups;
- Each data symbol (value) is coded in two steps: first the group to which it belongs (called *group index*) is

coded; followed by the rank of that particular symbol inside that group (the *symbol index*);
- When coding the pair (group index, symbol index) the group index is entropy-coded with a powerful and complex method, while the symbol index is coded with a simple and fast method, which can be simply the binary representation of that index.

A theory and procedure for calculating optimal alphabet partitions has been presented by Said [19]. The advantage of alphabet partitioning comes from the fact that it is normally possible to find partitions that allow large reductions in the coding complexity and with a very small loss in the compression efficiency.

In the current case, the powerful entropy coding method is block partitioning as in Algorithm 2.1. The particular alphabet partitioning scheme to be used in the image coding simulations is shown in Table 2.5. The groups are adjacent ranges of magnitudes (magnitude sets) generally increasing in size and decreasing in probability from 0 to higher values. The groups are indexed so that 0 has the highest probability, 1 the next lower, and so forth, until the lowest probability has the highest index. These group indices are the symbols coded by block partitioning. The table indicates whether a sign bit is needed and lists the number of bits needed to code the values inside each group under the column marked "magnitude-difference bits." Knowledge of the group index reveals the smallest magnitude in the group and the symbol index is just the difference from this smallest magnitude. Notice that there are only 22 group indices associated with a 2^{16} range of magnitudes. This range is normally sufficient for a DCT or DWT of images with amplitude range 0 to 255.

The raw binary representation of the symbol indices results only in a small loss of coding efficiency and no loss at all when the values inside the groups are uniformly distributed.

2.4.2 Enhancements via Entropy Coding

We have already mentioned that the 4-bit masks can be simply and efficiently encoded with a fixed 15 symbol Huffman code. Another simple

Table 2.5 An example of an effective alphabet partitioning scheme for subsequent block partition coding.

Magnitude set (MS)	Amplitude intervals	Sign bit	Magnitude-difference bits
0	[0]	no	0
1	[−1], [1]	yes	0
2	[−2], [2]	yes	0
3	[−3], [3]	yes	0
4	[−5, −4], [4,5]	yes	1
5	[−7, −6], [6,7]	yes	1
6	[−11, −8], [8,11]	yes	2
7	[−15, −12], [12,15]	yes	2
8	[−23, −16], [16,23]	yes	3
9	[−31, −24], [24,31]	yes	3
10	[−47, −32], [32,47]	yes	4
11	[−63, −48], [48,63]	yes	4
12	[−127, −64], [64,127]	yes	6
13	[−255, −128], [128,255]	yes	7
14	[−511, −256], [256,511]	yes	8
15	[−1023, −512], [512,1023]	yes	9
16	[−2047, −1024], [1024,2047]	yes	10
17	[−4095, −2048], [2048,4095]	yes	11
18	[−8191, −4096], [4096,8191]	yes	12
19	[−16383, −8192], [8192,16383]	yes	13
20	[−32767, −16384], [16384,32767]	yes	14
21	[−65535, −32768], [32768,65535]	yes	15
⋮	⋮	⋮	⋮

entropy coding can be enacted once the block partitioning reaches the final 2×2 group. If the maximum of this group is small, then a Huffman code on the extension alphabet is feasible. For example, suppose the maximum is 2, then the extension alphabet has $3^4 - 1$ possible 4-tuples ((0,0,0,0) cannot occur), so is feasible for coding. For higher maxima, where the extension alphabet is considerably larger, the elements may be encoded individually, say with a simple adaptive Huffman code. This strategy of switching between Huffman codes of different alphabet sizes is made possible by calculating and sending the maximum values of the groups formed during group partitioning.

2.4.3 Traversing the Blocks

The coding of the blockwise DCT of an image usually proceeds in the usual raster scan of the blocks from top left to bottom right, where each

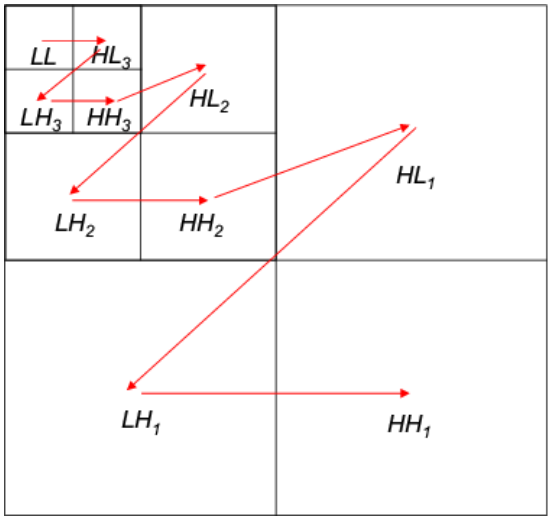


Fig. 2.15 Scanning order of subbands in a 3-level wavelet decomposition. Subbands formed are named for horizontal and vertical low- or high-passband and level of decomposition, e.g., LH_2 is horizontal low passband and vertical high passband at second recursion level.

DCT block is independently coded. Independent coding of the DCT blocks favors no scan mode over another. For the DWT, it matters how one scans the subbands when coding, because the subbands have different statistical properties, even for a stationary source. One should start with subbands of lowest spatial frequency, where the energy is highest and move next to a subband of same or higher spatial frequency. One should never move next to a subband of lower spatial frequency. The usual such scan is shown in Figure 2.15 for a three-level wavelet decomposition.

An inefficiency of the scanning mode in Figure 2.15 is that each subband has to be tested individually for significance, so that areas larger than a single subband can never be insignificant and signified with a single 0 bit. To remedy this situation, another layer of partitioning, called *octave band partitioning* precedes the partitioning of the coding procedure. The idea is to consider the DWT as initially consisting of two sets, one being the LL band marked as an $\mathcal{S}$ set and the remainder of the DWT marked as an $\mathcal{I}$ set, as depicted in Figure 2.16. The $\mathcal{S}$ set is coded by set partitioning, after which the $\mathcal{I}$ set is tested.

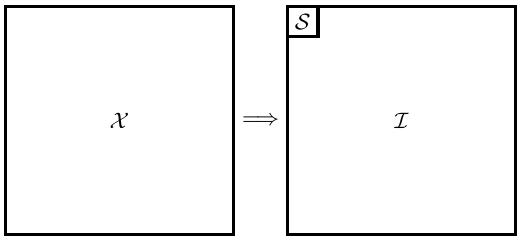


Fig. 2.16 Partitioning of image $\mathcal{X}$ into sets $\mathcal{S}$ and $\mathcal{I}$.

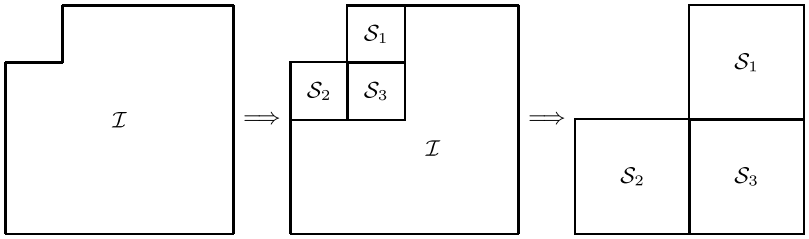


Fig. 2.17 Partitioning of set $\mathcal{I}$.

If not significant, a "0" is output and the threshold is lowered or a new maximum is calculated, depending on whether fixed or data-dependent thresholds are used. If $\mathcal{I}$ is significant, it is partitioned into three subbands marked as $\mathcal{S}$ at the same level adjacent to the previous $\mathcal{S}$ set and the remainder set marked as $\mathcal{I}$. This partitioning of $\mathcal{I}$ is illustrated in Figure 2.17. Testing and partitioning of $\mathcal{I}$ continues in the same manner until $\mathcal{I}$ becomes empty, as indicated in Figure 2.17. Octave band partitioning was first used in wavelet transform coding by Andrew [2] in the so-called SWEET method and then utilized in the embedded, set partition coding method SPECK [10], which will be described in the next section.

2.4.4 Embedded Block Coding of Image Wavelet Transforms

Using successive fixed powers of 2 thresholds, the octave band and block partitioning methods just described can be utilized together with progressive bit-plane coding to encode an image (integer)

wavelet transform. When a significant coefficient is found, instead of outputting a code for its value, only its sign is output and its coordinates are put onto an LSP (List of Significant Points). When a pass at a given n is completed, the LSP is visited to output the bits of previously significant coefficients in bit plane n. This step is just the progressive bit-plane coding depicted in Figure 2.13. This step is called the *refinement pass*, because its output bits refine the values of the previously significant coefficients. The LSP is populated and the bits are sent to the codestream sequentially from the highest to lowest bit plane. We see that this codestream is *embedded*, because truncation at any point gives the smallest distortion code corresponding to the rate of the truncation point.

We can state the steps of the full algorithm with the pseudo-code in Algorithm 2.6. The functions called by this algorithm are explained in the pseudo-code of Algorithm 2.7. The full algorithm is called SPECK for Set Partitioning Embedded bloCK.

Algorithm 2.6. The SPECK Algorithm.

(1) *Initialization*
- Partition image transform $\mathcal{X}$ into two sets: $\mathcal{S} \equiv$ root, and $\mathcal{I} \equiv \mathcal{X} - \mathcal{S}$ (Figure 2.16).
- Output $n_{\max} = \lfloor \log_2(\max_{(i,j)\in\mathcal{X}} |c_{i,j}|) \rfloor$.
- Add $\mathcal{S}$ to LIS and set LSP $= \phi$.

(2) *Sorting pass*
- In increasing order of size $|\mathcal{S}|$ of sets (smaller sets first).
  - — for each set $\mathcal{S} \in$ LIS do `ProcessS`($\mathcal{S}$).
- If $\mathcal{I} \neq \emptyset$, `ProcessI()`.

(3) *Refinement pass*
- For each $(i,j) \in$ LSP, except those included in the last sorting pass, output the nth MSB of $|c_{i,j}|$.

(4) *Threshold update*
- Decrement n by 1, and go to step 2.

After the splitting into the sets $\mathcal{S}$ and $\mathcal{I}$, the function ProcessS tests $\mathcal{S}$ and calls CodeS to do the recursive quadrisection of $\mathcal{S}$ to find the significant elements to put onto the LSP and populate the LIS with the insignificant quadrant sets of decreasing size. ProcessI tests $\mathcal{I}$ and calls CodeI to split $\mathcal{I}$ into three $\mathcal{S}$ sets and one new $\mathcal{I}$ set, whereupon ProcessS and CodeS are called to test and code the $\mathcal{S}$ sets as above.

Algorithm 2.7. The functions used by the SPECK Algorithm. Procedure `ProcessS`($\mathcal{S}$)

(1) output $\Gamma_n(S)$
(2) if $\Gamma_n(S) = 1$
- if $\mathcal{S}$ is a pixel, then output sign of $\mathcal{S}$ and add $\mathcal{S}$ to LSP
- else `CodeS`($\mathcal{S}$)
- if $\mathcal{S} \in$ LIS, then remove $\mathcal{S}$ from LIS

(3) else
- if $\mathcal{S} \notin$ LIS, then add $\mathcal{S}$ to LIS

(4) return

Procedure `CodeS`($\mathcal{S}$)

(1) Partition $\mathcal{S}$ into four equal subsets $\mathcal{O}(\mathcal{S})$ (see Figure 2.7)
(2) for each set $\mathcal{S}_i \in \mathcal{O}(\mathcal{S})$ $(i = 0, 1, 2, 3)$
- output $\Gamma_n(\mathcal{S}_i)$
- if $\Gamma_n(\mathcal{S}_i) = 1$
  - — if $\mathcal{S}_i$ is a pixel, output its sign and add $\mathcal{S}_i$ to LSP
  - — else `CodeS`($\mathcal{S}_i$)
- else
  - — add $\mathcal{S}_i$ to LIS

(3) return

Procedure `ProcessI`()

(1) output $\Gamma_n(\mathcal{I})$
(2) if $\Gamma_n(\mathcal{I}) = 1$
    - `CodeI`()
(3) return

Procedure `CodeI`()

(1) Partition $\mathcal{I}$ into four sets — three $\mathcal{S}_i$ and one $\mathcal{I}$ (see Figure 2.17)
(2) for each of the three sets $\mathcal{S}_i$ $(i = 0, 1, 2)$
    - `ProcessS`($\mathcal{S}_i$)
(3) `ProcessI()`
(4) return

2.4.5 A SPECK Coding Example

In order to clarify the SPECK coding procedure, we present an example of encoding data of the type resulting from an 8×8 two-level wavelet transform. Figure 2.18 depicts this data in the usual pyramidal subband structure. This is the same data example used by Shapiro to describe his EZW image coding algorithm [20].

The output bits, actions, and population of the lists are displayed in Table 2.6 for two full passes of the algorithm at bit planes $n = 5$ and $n = 4$. The following explains the notational conventions.

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

The maximum magnitude of the full transform is 63, so $n = 5$ is the initial bit-plane significance level. The initial $\mathcal{S}$ set is the top left

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

Fig. 2.18 Example of coefficients in an 8×8 transform used by example. The numbers outside the box are vertical and horizontal coordinates.

2×2 subband, so $\mathcal{S} = S^1(0,0)$ and (0,0)**1** initializes the LIS, and LSP is initially empty. The set $S^1(0,0)$ is tested and is significant, so it is quadrisected into four singleton sets added to the LIS and a "1" is output to the bitstream. These singleton sets (pixels) are tested in turn for significance. The point (0,0) with magnitude 63 is significant, so a "1" designating "significant" and a "+" indicating its sign is sent to the bitstream and it is moved to the LSP. Likewise, (0,1) is significant and negative, so a "1–" is output and its coordinate is moved to the LSP. Both (1,0) and (1,1) are insignificant with magnitudes below 32, so a "0" is output for each and they stay in the LIS. Next, the remainder set $\mathcal{I}$ is tested for significance, so a "1" is sent and it is partitioned into three new $\mathcal{S}$ sets and a new $\mathcal{I}$. Each of these new $\mathcal{S}$ sets, $S^1(0,2), S^1(2,0)$, and $S^1(2,2)$ are processed in turn as was $S^1(0,0)$. Of the three, only the first tests significant and is further quadrisected to four points, one of which, (0,2), moves to the LSP. The other three points, (0,3), (1,2), and (1,3) stay in the LIS and three "0"s are output to indicate their insignificance. The other two insignificant $\mathcal{S}$ sets, $S^1(2,0)$ and $S^1(2,2)$, are added to the LIS after the single point sets with bold suffixes **1**, since they are size 2×2, larger than the single point sets. Three "0"s are also output to indicate the insignificance of each.

Table 2.6 Example of SPECK coding of wavelet transform, From bit-plane $n = 5$.

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

(*Continued*)

Table 2.6 (*Continued*)

Comment	Point or set	Output bits	Action	Control lists
Test $\mathcal{I}$	$\mathcal{S}(\mathcal{I})$	1	split to 3 $\mathcal{S}$'s	
	$S^2(0,4)$	0	add to LIS(1)	LIS = {(1,0)**0**,(1,1)**0**},(0,3)**0**}(1,2)**0**,(1,3)**0**,(2,0)**1**,(2,2)**1**,(0,4)**2**}
	$S^2(4,0)$	1	quad split, add to LIS(1)	LIS = {(1,0)**0**,(1,1)**0**,(0,3)**0**,(1,2)**0**,(1,3)**0**,(2,0)**1**,(2,2)**1**,(4,0)**1**, (4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**}
	$S^1(4,0)$	0	none	
	$S^1(4,2)$	1	quad split, add to LIS(0)	LIS = {(1,0)**0**,(1,1)**0**,(0,3)**0**,(1,2)**0**,(1,3)**0**,(4,2)**0**,(4,3)**0**,(5,2)**0**, (5,3)**0**,(2,0)**1**,(2,2)**1**,(4,0)**1**,(4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**}
	(4,2)	0	none	
	(4,3)	1+	move (4,3) to LSP	LSP = {(0,0),(0,1),(0,2),(4,3)} LIS = {(1,0)**0**,(1,1)**0**,(0,3)**0**}(1,2)**0**,(1,3)**0**,(4,2)**0**,(5,2)**0**,(5,3)**0**, (2,0)**1**,(2,2)**1**,(4,0)**1**,(4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**}
	(5,2)	0	none	
	(5,3)	0	none	
	$S^1(6,0)$	0	none	
	$S^1(6,2)$	0	none	
	$S^2(4,4)$	0	add to LIS(2)	LIS = {(1,0)**0**,(1,1)**0**,(0,3)**0**,(1,2)**0**,(1,3)**0**,(4,2)**0**,(5,2)**0**,(5,3)**0**,
End $n = 5$				(2,0)**1**,(2,2)**1** ,(4,0)**1**, (4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**, (0,4)**2**}
Sorting				LSP = {(0,0),(0,1),(0,2),(4,3)}

Table 2.7 Example of SPECK coding of wavelet transform, continued to bit-plane $n = 4$.

Comment	Point or set	Output bits	Action	Control lists
$n = 4$ Sorting				LIS = {(1,0)**0**,(1,1)**0**,(0,3)**0**,(1,2)**0**,(1,3)**0**,(4,2)**0**,(5,2)**0**,(5,3)**0**, (2,0)**1**,(2,2)**1**,(4,0)**1** , (4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**, (0,4)**2**} LSP = {(0,0),(0,1),(0,2),(4,3)}
Test LIS(0)	(1,0)	1–	(1,0) to LSP	
	(1,1)	1+	(1,1) to LSP	LIS = {(0,3)**0**,(1,2)**0**,(1,3)**0**,(4,2)**0**,(5,2)**0**,(5,3)**0**,(2,0)**1**, (2,0)**1**,(2,2)**1**,(4,0)**1**, (4,2)**1**,(6,0)**1**,(6,2)**1**,(0,4)**2**, (0,4)**2**} LSP = {(0,0),(0,1),(0,2),(4,3),(1,0),(1,1)}
	(0,3)	0	none	
	(1,2)	0	none	
	(1,3)	0	none	
	(4,2)	0	none	
	(5,2)	0	none	
	(5,3)	0	none	
Test LIS(1)	$S^1(2,0)$	0	none	
	$S^1(2,2)$	0	none	
	$S^1(4,0)$	0	none	
	$S^1(6,0)$	0	none	
	$S^1(6,2)$	0	none	
Test LIS(2)	$S^2(0,4)$	0	none	
	$S^2(4,4)$	0	none	
Refinement	(0,0)	1	decoder adds 2^4	
	(0,1)	0	decoder subtracts 0	
	((0,2)	1	decoder adds 2^4	
	(4,3)	0	decoder adds 0	
End $n = 4$				

The algorithm continues in this way until the $n = 5$ sorting pass is complete. Note that SPECK puts out 29 raw (uncoded) bits in this first ($n = 5$) pass. Subsequent entropy coding can reduce this number of bits.

The significance threshold is now lowered to $n = 4$. The LIS at the end of the previous pass is the initial list for this pass. It turns out that only two single points are significant for $n = 4$, (1,0) and (1,1). They are accordingly moved to the LSP and removed from the LIS. Outputs are "1−" and "1+", respectively. All other LIS sets are insignificant, so stay in the LIS with "0" emitted for each to indicate insignificance. Note that there is no octave band partitioning for this nor any other lower threshold. The LSP coefficients significant for the $n = 5$ pass are now visited and the $n = 4$ bits in the binary expansion of their magnitudes are now sent to the code bitstream.

The decoder will duplicate the action of the encoder when receiving the code bitstream. In fact, if you replace the words in the column "Output Bits" by "Input Bits" in Table 2.6, the same exact table will be built from the codestream.

2.4.6 Embedded Tree-Based Image Wavelet Transform Coding

In previous sections, we have described coding of the wavelet transform of an image by set partitioning along spatial orientation trees (SOT's) and progressive bit-plane coding of significant coefficients. When putting these two coding procedures together, we obtain what is essentially the SPIHT (Set Partitioning in Hierarchical Trees) coding algorithm [18].

We start with an SOT partition of the wavelet transform, such as that shown in Figure 2.11 and initialize, test, and partition the SOT's using the SOT Encoding Algorithm 2.4. However, instead of immediately outputting the values of coefficients just found significant, we merely output their signs and put their coordinates on the LSP (list of significant pixels). When a pass at a given n is completed, the LSP is visited to output the bits of previously significant coefficients in bit plane n. This step is just the progressive bit-plane coding depicted in Figure 2.13. This step is called the *refinement pass*, because its output

bits refine the values of the previously significant coefficients. Therefore the bits are sent to the codestream sequentially from the highest to lowest bit plane. We see that this codestream is *embedded*, because truncation at any point gives the smallest distortion code corresponding to the rate of the truncation point.

The detailed description of the Embedded SOT Encoding Algorithm is the same as the SOT Encoding Algorithm, except for the additions of an LSP list and *refinement pass*, and changes to two places, Steps 2(a)ii and 2(b)i, when significant coefficients are found. The details of the algorithm are given in Algorithm 2.8.

Algorithm 2.8. *Embedded Sot Encoding Algorithm:* Embedded Coding in Spatial Orientation Trees–The SPIHT Algorithm.

(1) *Initialization*: output $n = \lfloor \log_2 \left(\max_{(i,j)} \{|c_{i,j}|\} \right) \rfloor$; set the LSP as an empty list, and add the coordinates $(i,j) \in \mathcal{H}$ to the LIP, and only those with descendants also to the LIS, as type A entries.

(2) *Sorting pass*

  (a) for each entry (i,j) in the LIP do:

    i. output $\Gamma_n(i,j)$;

    ii. if $\Gamma_n(i,j) = 1$ then move (i,j) to the LSP and output the sign of $c_{i,j}$;

  (b) for each entry (i,j) in the LIS do:

    i. if the entry is of type A then

    - output $\Gamma_n(\mathcal{D}(i,j))$;
    - if $\Gamma_n(\mathcal{D}(i,j)) = 1$ then
      — for each $(k,l) \in \mathcal{O}(i,j)$ do:
        * output $\Gamma_n(k,l)$;
        * if $\Gamma_n(k,l) = 1$ then add (k,l) to the LSP and output the sign of $c_{k,l}$;
        * if $\Gamma_n(k,l) = 0$ then add (k,l) to the end of the LIP;
      — if $\mathcal{L}(i,j) \neq \emptyset$ then move (i,j) to the end of the LIS, as an entry

of type B; otherwise, remove entry (i,j) from the LIS;

ii. if the entry is of type B then
- output $\Gamma_n(\mathcal{L}(i,j))$;
- if $\Gamma_n(\mathcal{L}(i,j)) = 1$ then
  - — add each $(k,l) \in \mathcal{O}(i,j)$ to the end of the LIS as an entry of type A;
  - — remove (i,j) from the LIS.

(3) *Refinement pass*: for each entry (i,j) in the LSP, except those included in the last sorting pass (i.e., with same n), output the nth most significant bit of $|c_{i,j}|$;

(4) *Threshold update*: decrement n by 1 and go to Step 2.

The steps of the Embedded SOT Algorithm are exactly those of the SPIHT Algorithm. In the initialization step, all coordinates in $\mathcal{H}$, the lowest frequency LL subband, are put on the LIP (list of insignificant pixels) and only those with descendants are tree roots to be put onto the LIS (list of insignificant sets). For trees as defined in Figure 2.11, all coordinates in $\mathcal{H}$ are tree roots, so are put onto the LIS. However, in SPIHT, the trees are configured slightly differently. For each 2×2 block in the LL subband, the upper left point has no descendant tree. Other points in each 2×2 block are roots of trees branching along the corresponding spatial orientations. The first branching is four to the same resolution level and then also four to each subsequent level until the leaves (termini) at the highest resolution level. The branching rules or parent-offspring dependencies are illustrated in Figure 2.19. Therefore, in the initialization step for these trees, only 3/4 of the points in the LL subband are put onto the LIS.

This algorithm provides very efficient coding with low computational complexity. Small improvements in efficiency may be achieved through adaptive entropy coding of the binary significance decisions (called the *significance map*), and the refinement and sign bits. In this chapter, our objective is to present fundamental principles of theory and practice in set partition coding. Therefore, we leave to a later chapter the discussion of complete coding systems.

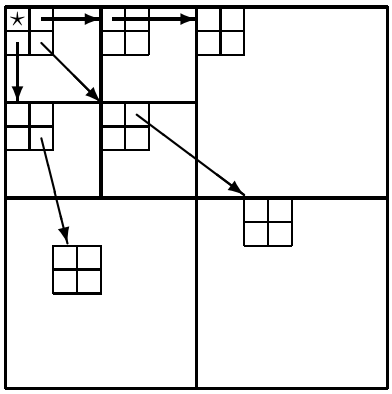


Fig. 2.19 Examples of parent-offspring dependencies in the spatial-orientation trees. Coefficients in the LL band marked "*" have no descendants.

2.4.7 A SPIHT Coding Example

We describe now in detail an example of SPIHT coding of the 8×8, three-level wavelet transform in Figure 2.18.

We applied the SPIHT algorithm to this set of data, for one pass. The results are shown in Table 2.8, indicating the data coded and the updating on the control lists (to save space only the modifications are shown). The notation is defined in the previous description of the algorithm. For a quick reference, here are some of the important definitions.

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

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

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

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

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

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

Table 2.8 Example of image coding using the SPIHT method.

Comm.	Pixel or set tested	Output bit	Action	Control lists
(1)				LIS = {(0,1)A,(1,0)A,(1,1)A}
				LIP = {(0,0),(0,1),(1,0),(1,1)}
				LSP = $\emptyset$
(2)	(0,0)	1+	(0,0) to LSP	LIP = {(0,1),(1,0),(1,1)}
				LSP = {(0,0)}
	(0,1)	1–	(0,1) to LSP	LIP = {(1,0),(1,1)}
				LSP = {(0,0),(0,1)}
	(1,0)	0	none	
	(1,1)	0	none	
(3)	$\mathcal{D}(0,1)$	1	test offspring	LIS = {**(0,1)**A,(1,0)A,(1,1)A}
	(0,2)	1+	(0,2) to LSP	LSP = {(0,0),(0,1),(0,2)}
	(0,3)	0	(0,3) to LIP	LIP = {(1,0),(1,1),(0,3)}
	(1,2)	0	(1,2) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2)}
	(1,3)	0	(1,3) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3)}
(4)			type changes	LIS = {(1,0)A,(1,1)A,(0,1)B}
(5)	$\mathcal{D}(1,0)$	1	test offspring	LIS = {**(1,0)**A,(1,1)A,(0,1)B}
	(2,0)	0	(2,0) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0)}
	(2,1)	0	(2,1) to LIP	LIP = {(1,0), (1,1), (0,3), (1,2), (1,3), (2,0), (2,1)}
	(3,0)	0	(3,0) to LIP	LIP = {(1,0), (1,1), (0,3), (1,2), (1,3), (2,0), (2,1), (3,0)}
	(3,1)	0	(3,1) to LIP	LIP = {(1,0), (1,1), (0,3), (1,2), (1,3), (2,0), (2,1), (3,0), (3,1)}
			type changes	LIS = {(1,1)A,(0,1)B,(1,0)B}
(6)	$\mathcal{D}(1,1)$	0	none	LIS = {**(1,1)**A,(0,1)B,(1,0)B}
(7)	$\mathcal{L}(0,1)$	0	none	LIS = {(1,1)A,**(0,1)**B,(1,0)B}
(8)	$\mathcal{L}(1,0)$	1	add new sets	LIS = {(1,1)A, (0,1)B, (2,0)A, (2,1)A, (3,0)A, (3,1)A}
(9)	$\mathcal{D}(2,0)$	0	none	LIS = {(1,1)A, (0,1)B, **(2,0)**A, (2,1)A, (3,0)A, (3,1)A}
(10)	$\mathcal{D}(2,1)$	1	test offspring	LIS = {(1,1)A, (0,1)B, (2,0)A, **(2,1)**A, (3,0)A, (3,1)A}
	(4,2)	0	(4,2) to LIP	LIP = {(1,0), (1,1), (0,3), (1,2), (1,3), (2,0), (2,1), (3,0), (3,1), (4,2)}
	(4,3)	1+	(4,3) to LSP	LSP = {(0,0), (0,1), (0,2), (4,3)}
	(5,2)	0	(5,2) to LIP	LIP = {(1,0), (1,1), (0,3), (1,2), (1,3), (2,0), (2,1), (3,0), (3,1), (4,2), (5,2)}
	(5,3)	0	(5,3) to LIP	LIP = {(1,0), (1,1), (0,3), (1,2), (1,3), (2,0), (2,1), (3,0), (3,1), (4,2), (5,2), (5,3)}
(11)			(2,1) removed	LIS = {(1,1)A, (0,1)B, (2,0)A, (3,0)A, (3,1)A}
(12)	$\mathcal{D}(3,0)$	0	none	LIS = {(1,1)A, (0,1)B, (2,0)A,**(3,0)**A, (3,1)A}
	$\mathcal{D}(3,1)$	0	none	LIS = {(1,1)A, (0,1)B, (2,0)A, (3,0)A,**(3,1)**A}
(13)				LIS = {(1,1)A, (0,1)B, (2,0)A, (3,0)A, (3,1)A}
				LIP = {(1,0), (1,1), (0,3), (1,2), (1,3), (2,0), (2,1), (3,0), (3,1), (4,2), (5,2), (5,3)}
				LSP = {(0,0), (0,1), (0,2), (4,3)}

The following refer to the respective numbered entries in Table 2.8:

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

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

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

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

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

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

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

(8) Same as above, but $\mathcal{L}(1,0)$ is significant, so the set is partitioned in $\mathcal{D}(2,0)$, $\mathcal{D}(2,1)$, $\mathcal{D}(3,0)$, and $\mathcal{D}(3,1)$, and the

corresponding entries are added to the LIS. At the same time, the entry (1,0)B is removed from the LIS.

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

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

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

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

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

Without using any other form of entropy coding, the SPIHT algorithm used 29 bits in this first pass, exactly the same number of bits as SPECK in its first pass for the same data. The initial lists for the second pass of SPIHT at $n = 4$ are those at the end of the first pass in the last line (13) of Table 2.8.

2.4.8 Embedded Zerotree Wavelet (EZW) Coding

An embedded SOT coding method that is not strictly a set partioning coder is the Embedded Zerotree Wavelet (EZW) Coder [20]. It is described in this section, because it inspired the creation of SPIHT and shares many of its characteristics. It operates on SOT's of a wavelet transform like SPIHT, but locates only significant coefficients and trees, all of whose coefficients, including the root, are insignificant. When the thresholds are integer powers of 2, insignificant trees means that all the bits in the corresponding bit-plane located at the coordinates of the tree nodes are 0s. Hence arises the term *zerotree*. SPIHT has also been called a *zerotree* coder, but it locates zerotrees in a broader sense than the EZW zerotrees [3]. For our purposes here, it is better to view SPIHT as a set partitioning coder.

The structure of a full spatial orientation tree used in EZW is shown in Figure 2.11. The EZW algorithm determines certain attributes of

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

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

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

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

$$\mathcal{T}(i,j) = (i,j) \bigcup \mathcal{D}(i,j), \tag{2.10}$$

the union of the root coordinate and the set of its descendants. The branching policy is the one shown in Figure 2.11. If (i, j) has descendants, we test both the root coefficient $c(i, j)$ and all the coefficients in $\mathcal{T}(i, j)$ for significance. Referring to (2.9), the outcomes are

as follows:

$$\begin{aligned}
\Gamma_n(\mathcal{T}(i,j)) &= 0, \text{ output ZTR} \\
\Gamma_n(\mathcal{T}(i,j)) &= 1 \text{ AND } \Gamma_n(i,j) = 0, \text{ output IZ} \\
\Gamma_n(i,j) &= 1 \text{ AND } c(i,j) > 0, \text{ output POS} \\
\Gamma_n(i,j) &= 1 \text{ AND } c(i,j) < 0, \text{ output NEG}
\end{aligned} \tag{2.11}$$

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

$$\begin{aligned}
\Gamma_n(i,j) &= 0, \text{ output } Z \\
\Gamma_n(i,j) &= 1 \text{ AND } c(i,j) > 0, \text{ output POS} \\
\Gamma_n(i,j) &= 1 \text{ AND } c(i,j) < 0, \text{ output NEG}
\end{aligned} \tag{2.12}$$

We maintain two ordered control lists, the dominant list (DL) and the subordinate list (SL).[9] The dominant list keeps track of coordinates of subtree roots that are to be examined at the current and lower thresholds. The subordinate list (SL) stores coordinates of significant coefficients, so is identical to the LSP in the previous algorithms. The dominant list (DL) is initialized with the coordinates in the lowest frequency subband. The algorithm first tests entries in order from the DL. When the outcome is not ZTR and there are offspring, the coordinates of the offspring are added to the end of the list DL. When the coefficient for the entry is significant, its coordinates are added to the end of the list SL. These actions for populating the lists may be summarized as follows:

for (i,j) in DL,

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

Once a coefficient tests significant, it should not be tested again. It is customary to keep it on the DL, but mark it with an internal flag to skip over it. Putting the previous procedures together leads us to the

[9] These lists keep track of the actions of the algorithm and point to future actions. Other means that do not require these lists can be employed to achieve the same actions. For example, see Taubman and Marcellin [22]. SPIHT can also be implemented without lists.

full description of the EZW algorithm. As before, we calculate $n_{\max}$, initialize DL with the coordinates of $\mathcal{H}$, and set SL to empty. The order of scanning of the subbands follows the zigzag shown in Figure 2.15. Coordinates within a subband are visited in raster scan order from the top to the bottom row. We then visit the coordinates in the DL in order starting at $n = n_{\max}$. We test these coordinates and output the symbols according to (2.11) and re-populate the DL and SL by the rules above. We continue through the subbands in the prescribed scan order at the same threshold. When we reach DL coordinates belonging to the leaf node set in the Level 1 subbands, we test and output symbols by (2.12), but nothing more is added to the DL, since there are no offspring. The so-called *dominant pass* is complete, once all the DL coordinates have been visited at a given threshold n. Then the SL is entered to output the nth bit of coefficients found significant at previous higher values of n. This pass through the SL, called in EZW the *subordinate pass*, which is identical to the *refinement pass* previously described for progressive bit-plane coding, is omitted for coefficients found significant on the current pass, because the output symbols, POS and NEG, already signify that the nth bit is 1. When the subordinate pass is complete, the threshold is lowered by a factor of 2 (n is decremented by 1) and the dominant pass at this new threshold is executed on the coordinates in the DL left from the previous pass. The coordinates belonging to coefficients that have become significant on previous passes are removed or marked or set to 0 to be skipped. This process continues either until completion of the $n = 0$ passes or the bit budget is exhausted.

2.4.8.1 An EZW Coding Example

We now present an example of EZW coding of the same 8×8 wavelet transform shown in Figure 2.18. The notation of **F** following a co-ordinate on the dominant list means that an internal flag is set to indicate "significant" on that pass and its magnitude on the dominant list is set to 0 for subsequent passes. Recall that i signifies row index and j column index. We display only the actions and outputs for the first dominant pass at the top bit plane $n = 5$. Table 2.9 shows the results.

Table 2.9 Example of image coding using Shapiro's EZW method.

Tree root	Output symbol	DL: Dominant list SL: Subordinate list
		DL = {(0,0)}
		SL = ∅
(0, 0)	POS	DL = {(0,0)**F**, (0,1), (1,0), (1,1)}
		SL = {(0,0)}
(0,1)	NEG	DL = {(0,0)**F**, (1,0), (1,1), (0,2), (0,3), (1,2), (1,3), (0,1)**F**}
		SL = {(0,0), (0,1)}
(1,0)	IZ	DL = {(0,0)**F**, (1,0), (1,1), (0,2), (0,3), (1,2), (1,3), (0,1)**F**, (2,0), (2,1), (3,0), (3,1)}
(1,1)	ZTR	
(0,2)	POS	DL = {(0,0)**F**, (1,0), (1,1), (0,3), (1,2), (1,3), (0,1)**F**, (2,0), (2,1), (3,0), (3,1), (0,4), (0,5), (1,4), (1,5), (0,2)**F**}
		SL = {(0,0), (0,1), (0,2)}
(0,3)	ZTR	
(1,2)	ZTR	
(1,3)	ZTR	
(2,0)	ZTR	
(2,1)	IZ	DL = {(0,0)**F**, (1,0), (1,1), (0,3), (1,2), (1,3), (0,1)**F**, (2,0), (2,1), (3,0), (3,1), (0,4), (0,5), (1,4), (1,5), (0,2)**F**, (4,2), (4,3), (5,2), (5,3)}
(3,0)	ZTR	
(3,1)	ZTR	
(0,4)	Z	
(0,5)	Z	
(1,4)	Z	
(1,5)	Z	
(4,2)	Z	
(4,3)	POS	DL = {(0,0)**F**, (1,0), (1,1), (0,3), (1,2), (1,3), (0,1)**F**, (2,0), (2,1), (3,0), (3,1), (0,4), (0,5), (1,4), (1,5), (0,2)**F**, (4,2), (5,2), (5,3), (4,3)**F**}
		SL = {(0,0), (0,1), (0,2), (4,3)}
(5,2)	Z	
(5,3)	Z	
		DL = {(0,0)**F**, (1,0), (1,1), (0,3), (1,2), (1,3), (0,1)**F**, (2,0), (2,1), (3,0), (3,1), (0,4), (0,5), (1,4), (1,5), (0,2)**F**, (4,2), (5,2), (5,3), (4,3)**F**}
		SL = {(0,0), (0,1), (0,2), (4,3)}

Assuming the EZW uses (at least initially) two bits to code symbols in the alphabet {POS, NEG, ZTR, IZ}, and one bit to code the symbol Z, the EZW algorithm used $26 + 7 = 33$ bits in the first pass. Since the SPECK, SPIHT, and EZW methods are coding the same bit-plane defined by the threshold 32, both find the same set of significant coefficients, yielding images with the same mean squared error. However, SPIHT and SPECK used 29 bits, about 10% fewer bits to obtain the same results, because they coded different symbols.

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

2.4.9 Group Testing for Image Wavelet Coding

Group testing has been applied successfully to embedded coding of image wavelet transforms. The framework is the same as that in the Embedded SOT Algorithm 2.8, but the sorting pass is replaced by the method of group testing. Specifically, one encodes SOT's and starts testing for significance at threshold $2^{n_{\max}}$, successively lowers the thresholds by factors of 2, and sends significant coefficients to the LSP, where they are refined using progressive bit-plane coding (Figure 2.13). One might consider ordering the coefficients in an SOT from lowest to highest frequency subband and use the adaptive method of setting the group sizes in a series of group iterations. However, underlying assumptions in group testing are independence and equal probabilities of significance among coefficients in the group. These assumptions are not valid for coefficients in an SOT. The solution is to divide the SOT coefficients into different classes, within which independence and equal probabilities of significance are approximately true. The details of how these classes are formed and other aspects of applying group testing in this framework are described in [8]. Here, we just want to indicate how group testing can be applied to wavelet transform coding to produce an embedded codestream.

2.5 Conclusion

In this part, we have explained principles and described some methods of set partition coding. We have also shown how these methods naturally extend to produce embedded codestreams in applications to transform coding. Although we have described a single partitioning method to a given transform, different methods can be mixed within such a transform. For example, subbands of a wavelet transforms have different statistical characteristics, so that different partitioning methods

may be utilized for different subbands of the same transform. For example, the SOT partitioning becomes inefficient for transforms with substantial energy in the highest spatial or temporal frequency subbands, because it produces lots of LIP entries at several thresholds as it proceeds toward the significant coefficients at the end of the tree. Once a coefficient is listed on the LIP, one bit for each threshold pass is required to encode it. Therefore, you want an LIP entry to become significant as early as possible. One solution is to terminate the trees before the highest frequency subbands. Then the descendant sets are no longer significant, so they are not further partitioned and are put onto the LIS and represented with a single "0" bit. To maintain embedded coding, for a given threshold, one could encode these high frequency subbands by block partitioning, using quadrisection or bisection, after the pass through the SOT's at the same threshold is complete. No doubt there are other scenarios when a hybrid of two (or more) partitioning methods becomes potentially advantageous. In the next part, we shall explain further the advantageous properties of set partition coding as it is used in modern coding systems.

A

Mathematical Transformations

In this appendix, we present a brief introduction to the subject of linear transforms. We shall describe the operations and salient characteristics of the Karhune–Loeve Transform (KLT), the discrete cosine transform, and subband transforms, including the wavelet transform. For a thorough exposition, especially for subband and wavelet transforms, we refer the reader to the excellent textbook by Vetterli and Kovačević [23], whose notation we have adopted. Another useful reference, especially for wavelet transforms, is the textbook by Rao and Bopardikar [16].

A general linear transform of a vector $\mathbf{x}$ to a vector $\mathbf{y}$ is a representation of its components in a different basis. Consider the basis of $\mathbf{x}$ to be the normal basis with basis vectors $\mathbf{n}_1, \mathbf{n}_2, \ldots, \mathbf{n}_N$, such that

$$\mathbf{n}_j = (\delta_{j,1}, \delta_{j,2}, \ldots, \delta_{j,N})^t, \quad j = 1, 2, \ldots, N, \tag{A.1}$$

where

$$\delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{A.2}$$

In words, the basis vectors $\mathbf{n}_j$ have 1 in the jth component and 0 in all the other components. (We have adopted the convention here that

vectors are column vectors.) Note that these vectors are orthogonal (dot or inner product equals 0) and have length 1. Such a basis is said to be orthonormal. Then $\mathbf{x}$ can be written as

$$\mathbf{x} = \sum_{j=1}^{N} x_j \mathbf{n}_j.$$

Suppose we wish to express $\mathbf{x}$ in terms of another orthonormal basis $\psi_1, \psi_2, \ldots, \psi_N$ according to

$$\mathbf{x} = \sum_{j=1}^{N} y_j \psi_j.$$

Equating these two representations and forming matrices whose columns are the basis vectors or vector components, we obtain

$$\begin{aligned}(x_1 x_2 \cdots x_N)[\mathbf{n}_1|\mathbf{n}_2|\cdots|\mathbf{n}_N] &= (y_1 y_2 \ \cdots \ y_N)[\psi_\mathbf{1}|\psi_2|\cdots|\psi_N] \\ \mathbf{xI} &= \mathbf{y}\boldsymbol{\Psi} \\ \mathbf{y} &= \boldsymbol{\Psi}^{-1}\mathbf{x}, \end{aligned} \tag{A.3}$$

where $\boldsymbol{\Psi}$ is the matrix having the new basis vectors as columns.[1] (Enclosure of a sequence of vectors with brackets ([ ]) denotes formation of a matrix.) The vector $\mathbf{y}$ is the transform of $\mathbf{x}$ and is produced by the matrix $\boldsymbol{\Psi}^{-1}$.

We prove now that the mean squared quantization error equals the mean squared reconstruction error, if the transform matrix is unitary. Any matrix whose columns are comprised of orthonormal basis vectors is unitary. Assume that $\hat{\mathbf{y}}$ is a corrupted version of the transform vector $\mathbf{y}$. Its source domain reproduction $\hat{\mathbf{x}} = \boldsymbol{\Psi}\hat{\mathbf{y}}$ and the original source vector $\mathbf{x} = \Psi\mathbf{y}$. Then the reconstruction sum squared error is

$$\begin{aligned}||\mathbf{x} - \hat{\mathbf{x}}||^2 &= ||\boldsymbol{\Psi}\mathbf{y} - \boldsymbol{\Psi}\hat{\mathbf{y}}||^2 \\ &= ||\boldsymbol{\Psi}(\mathbf{y} - \hat{\mathbf{y}})||^2 = (\boldsymbol{\Psi}(\mathbf{y} - \hat{\mathbf{y}}))^t \boldsymbol{\Psi}(\mathbf{y} - \hat{\mathbf{y}}) \\ ||\mathbf{x} - \hat{\mathbf{x}}||^2 &= (\mathbf{y} - \hat{\mathbf{y}})^t \boldsymbol{\Psi}^t \boldsymbol{\Psi}(\mathbf{y} - \hat{\mathbf{y}}). \end{aligned} \tag{A.4}$$

Since the matrix $\boldsymbol{\Psi}$ is unitary, $\boldsymbol{\Psi}^t\boldsymbol{\Psi} = \mathbf{I}$, the identity matrix. Therefore, we conclude that

$$||\mathbf{x} - \hat{\mathbf{x}}||^2 = (\mathbf{y} - \hat{\mathbf{y}})^t \mathbf{I}(\mathbf{y} - \hat{\mathbf{y}}) = ||\mathbf{y} - \hat{\mathbf{y}}||^2. \tag{A.5}$$

[1] We use boldface type to signify matrices and vectors.

The above equation states that the sum squared reconstruction error equals the sum squared quantization error. Therefore, the MSE errors of the reconstruction and quantization are the same when the transform is unitary. Another notable property under the same conditions, arrived at through a similar argument, is that the squared lengths (powers) of the source and transform vectors are equal, i.e.,

$$||\mathbf{x}||^2 = ||\mathbf{y}||^2. \tag{A.6}$$

A.1 Karhunen–Loeve Transform

Now we present the optimal transform, the KLT, and its development. Let the source be the vector $\mathbf{X} = (X_0, X_1, \ldots, X_{N-1})$ have zero mean and N-dimensional covariance matrix $\mathbf{\Phi}_N = E[\mathbf{X}\mathbf{X}^t]$.[2] The elements of this matrix, $[\mathbf{\Phi}_N]_{i,j}$ may be expressed as

$$[\mathbf{\Phi}_N]_{i,j} = E[X_i X_j] = R_X(i,j), \quad i,j = 0,1,\ldots,N-1,$$

where $R_X(i,j),\ i,j = 0,1,2,\ldots,N-1$ is called the autocorrelation function of the source.[3] $X_i, i = 0,1,2,\ldots,N-1$ are the individual components of the vector X. This (Toeplitz) matrix is real, symmetric, and non-negative definite. There exists a set of N orthonormal eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N$ and a corresponding set of N non-negative, not necessarily distinct, eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_N$ for Φ_N. Expressed mathematically, the properties of the eigenvalues and eigenvectors are

$$\mathbf{\Phi}_N \mathbf{e_j} = \lambda_j \mathbf{e_j}, j = 0,1,\ldots,N-1, \tag{A.7}$$

where $||\mathbf{e_j}||^2 = 1$ for all j. A given source sequence $\mathbf{X} = \mathbf{x}$ can be represented with respect to the eigenvector basis by

$$\mathbf{x} = \mathbf{P}\mathbf{y}, \quad \mathbf{P} = [\mathbf{e}_1|\mathbf{e}_2|\cdots|\mathbf{e}_N], \tag{A.8}$$

where $\mathbf{P}$ is a unitary matrix, $(\mathbf{P}^{-1} = \mathbf{P}^t)$.

[2] We use capital letters to signify a random variable and lower case to denote a particular value of the random variable. In the parlance of information or communications theory, a random variable together with its probability distribution is often called an *ensemble*.

[3] The autocorrelation and covariance functions are identical in the case of zero mean assumed here.

The vector $\mathbf{y}$ is called the transform of the source vector $\mathbf{x}$. The covariance matrix of the transform vector ensemble $\mathbf{Y}$ is

$$\begin{aligned}\mathbf{\Lambda} &= E[\mathbf{Y}\mathbf{Y}^t] = E[\mathbf{P}^{-1}\mathbf{X}\mathbf{X}^t\mathbf{P}] \\ &= \mathbf{P}^{-1}\mathbf{\Phi}_N\mathbf{P} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_N \end{bmatrix}.\end{aligned} \tag{A.9}$$

This transformation of $\mathbf{x}$ to $\mathbf{y}$ is called the (discrete) KLT, Hotelling, or *principal components* transform. Significant characteristics of the KLT are enumerated below. Consult the textbook by Jayant and Noll [13] for further details.

(1) The components of $\mathbf{Y}$ are uncorrelated with variances equal to the eigenvalues.
(2) When the source vector ensemble is Gaussian, the ensemble $\mathbf{Y}$ is also Gaussian with statistically independent components.
(3) Truncation of the KLT to $K < N$ components yields the least mean squared error of any N-point unitary transform trucated to K components.
(4) The KLT has the smallest geometric mean of the component variances of any unitary transform of the source vector $\mathbf{x}$.

A.2 The Discrete Cosine Transform

The discrete KLT requires knowledge of the covariance function of the source and a solution for the eigenvectors of the $N \times N$ covariance matrix. In general, especially for large N, the solution and the transform are computationally burdensome procedures with no fast algorithms for their execution. Instead, one uses almost always a source-independent transform with a fast execution algorithm. One such transform is the discrete cosine transform (DCT), that, although suboptimal, seems to give nearly as good performance as the optimal KLT.

The unitary DCT of the sequence $x(0), x(1), \ldots, x(N-1)$ is given by

$$y(n) = \begin{cases} \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x(k), & n = 0 \\ \frac{2}{\sqrt{N}} \sum_{k=0}^{N-1} x(k) \cos \frac{n\pi}{2N}(2k+1), & n = 1, 2, \ldots, N-1 \end{cases} \tag{A.10}$$

The inverse is the same form with k and n interchanged. The forward and inverse DCT are often expressed more compactly as

$$y(n) = \frac{2}{\sqrt{N}} \alpha(n) \sum_{k=0}^{N-1} x(k) \cos \frac{n\pi}{2N}(2k+1), \quad n = 0, 1, \ldots, N-1 \tag{A.11}$$

$$x(k) = \frac{2}{\sqrt{N}} \alpha(k) \sum_{n=0}^{N-1} y(n) \cos \frac{k\pi}{2N}(2n+1), \quad k = 0, 1, \ldots, N-1 \tag{A.12}$$

The weighting function $\alpha(n)$ is defined as

$$\alpha(n) = \begin{cases} \frac{1}{\sqrt{2}}, & n = 0 \\ 1, & n \neq 0. \end{cases} \tag{A.13}$$

The matrix transformation is

$$\mathbf{C} = \{C_{n,k}\}_{n,k=0}^{N-1}, \; C_{n,k} = \begin{cases} \frac{1}{\sqrt{N}} & n = 0 \\ & k = 0, 1, \ldots, N-1 \\ \sqrt{\frac{2}{N}} & \cos \frac{n\pi}{2N}(2k+1), \\ & n = 1, 2, \ldots, N-1 \\ & k = 0, 1, 2, \ldots, N-1 \end{cases} \tag{A.14}$$

and the unitary property and real symmetry of the forward and inverse make

$$\mathbf{C}^{-1} = \mathbf{C}^t = \mathbf{C}.$$

Salient properties of the DCT are as follows:

(1) For a stationary, finite order Markov process, the DCT is asymptotically optimal in a distributional sense, i.e., for a function f, $f(E[|y(n)|^2], \; n = 0, 1, \ldots, N-1)$, converges

to $f(\lambda_o, \lambda_1, \ldots, \lambda_{N-1})$ as $N \to \infty$, if the eigenvalues $\lambda_o, \lambda_1, \lambda_2, \ldots, \lambda_{N-1}$ are bounded [25]. That is, the DCT and KLT are asymptotically equivalent for all finite order Markov processes.

(2) Ahmed et al. [1] gave empirical evidence that DCT performance is close to KLT even for small values of N in the case of stationary Markov-1 signals.

You can evaluate the DCT through the DFT (discrete Fourier transform), which has a fast algorithm. First, express the DCT as

$$y(n) = \sqrt{\frac{2}{N}} \mathrm{Re} \left\{ e^{-j\frac{n\pi}{2N}} \alpha(n) \sum_{k=0}^{N-1} x(k) e^{-j\frac{2\pi kn}{2N}} \right\}.$$

Note that the summation is a $2N$-point DFT with $x(k) = 0, \quad N \leq k \leq 2N-1$. The $2N$-point DFT is a slight disadvantage along with the exponential (or sine-cosine) multipliers. If the sequence is re-ordered as

$$\left.\begin{array}{l} w(k) = x(2k) \\ w(N-1-k) = x(2k+1) \end{array}\right\} \quad k = 0, 1, \ldots, \frac{N}{2} - 1$$

$$y(n) = \sqrt{\frac{2}{N}} \alpha(n) \mathrm{Re} \left\{ e^{j\frac{\pi n}{2N}} \sum_{k=0}^{N-1} w(k) e^{j2\pi nk/N} \right\}$$

the summation is an N-point DFT of the re-ordered sequence [14]. Similar procedures apply to the inverses. There are also direct and fast algorithms for computing the DCT that are beyond the scope of this discussion.

A.3 Subband Transforms

Transforms that decompose the source into nonoverlapping and contiguous frequency ranges called *subbands* are called *subband transforms*. A *wavelet transform*, as we shall see, is just a particular kind of subband transform. The source sequence is fed to a bank of bandpass filters which are contiguous and cover the full frequency range. The set of output signals are the subband signals and can be recombined

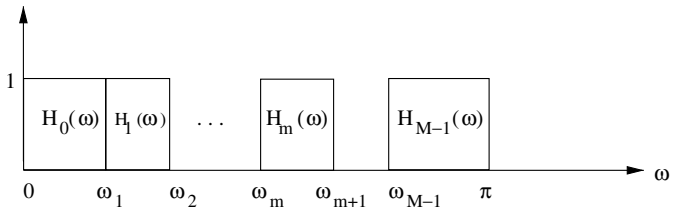


Fig. A.1 Subband filter transfer functions.

without degradation to produce the original signal. Let us assume first that the contiguous filters, each with bandwidth $W_m, m = 1,2,\ldots,M$, are ideal with zero attenuation in the pass-band and infinite attenuation in the stop band and that they cover the full frequency range of the input, as depicted in Figure A.1.

The output of any one filter, whose lower cutoff frequency is an integer multiple of its bandwidth W_m, is subsampled by a factor equal to $V_m = \pi/W_m$ and is now a full-band sequence in the frequency range from $-\pi$ to π referenced to the new, lower sampling frequency. This combination of filtering and subsampling (often illogically called *decimation*) is called M-channel filter bank analysis and depicted in Figure A.2. We shall assume that the integer bandwidth to lower frequency relationship holds for all filters in the bank so that

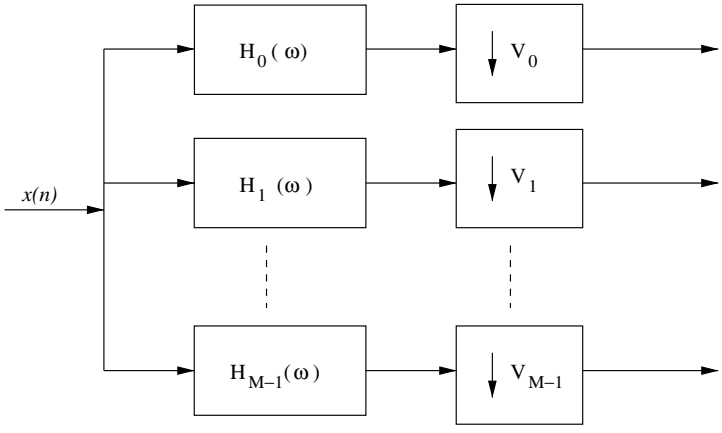


Fig. A.2 M-channel filter bank analysis of source into subbands.

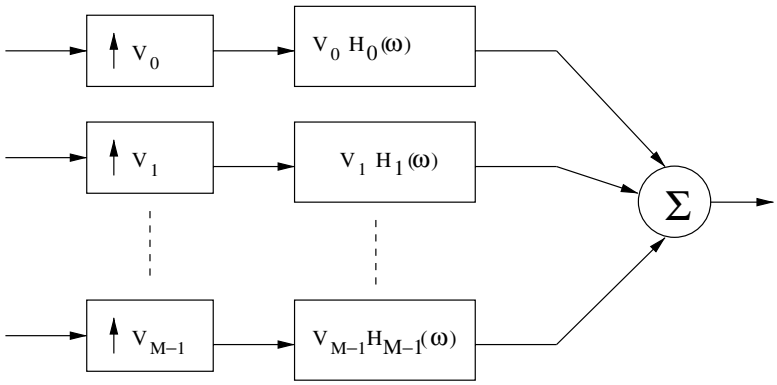


Fig. A.3 M-channel filter bank synthesis of source from subbands.

all outputs are decimated by the appropriate factor. These outputs are called the subband signals or waveforms and their aggregate number of samples equals that in the original input waveform. The original input can be reconstructed exactly from the subband signals. The sampling rate of each subband signal is increased to that of the original input by filling in the appropriate number of zero samples and the zero-filled waveform is fed into an ideal filter with gain equal to the sub-sampling factor covering the original pass-band (called interpolation). The sum of these interpolated subband signals equals the original input signal. Figure A.3 depicts this M-channel filter bank synthesis of the subband signals to reproduce the original signal.

When the input to a bank of linear, shift-invariant filters is a Gaussian random sequence, the output subband signals are Gaussian and are also statistically independent when the filters are ideal and nonoverlapping. Because of the subsampling, however, the subband signals are not shift-invariant.

Most often, however, a transform of M subbands is generated by successive application of two-channel filter banks, consisting of half-band filters, to certain subband outputs. The subbands of half-band filters occupy the frequency intervals $[0, \pi/2)$ and $[\pi/2, \pi]$. Figure A.4 depicts the two-channel analysis and synthesis filter banks. For example, two stages of applying the two-channel analysis filter bank on the ouput subband signals creates $M = 4$ subbands of equal

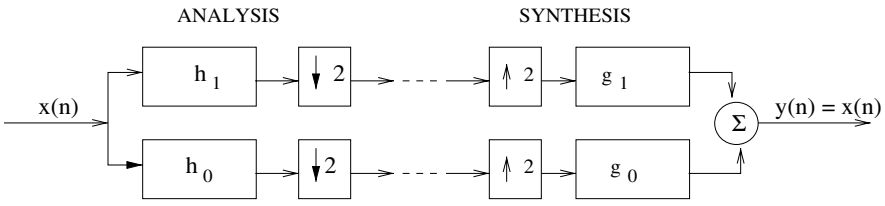


Fig. A.4 Two-channel analysis and synthesis filter banks. h_0 and g_0 denote impulse responses of the lowpass filters; h_1 and g_1 those of the highpass filters.

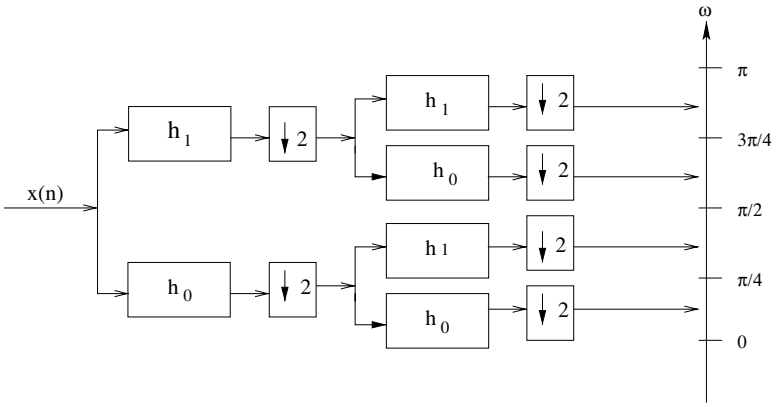


Fig. A.5 Two stages of two-channel analysis for four equal-size subbands.

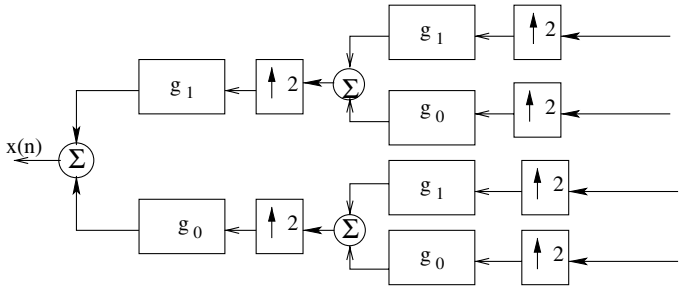


Fig. A.6 Two stages of two-channel synthesis to reconstruct signal from four subbands.

width, as shown in Figure A.5. Synthesis is accomplished by successive stages of application of the two-channel synthesis filter bank. The corresponding synthesis of these $M = 4$ subbands to reconstruct the original signal is shown in Figure A.6.

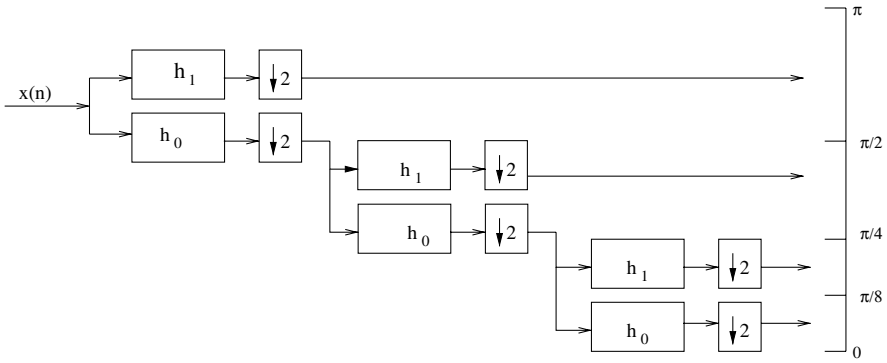


Fig. A.7 Three-level multiresolution analysis.

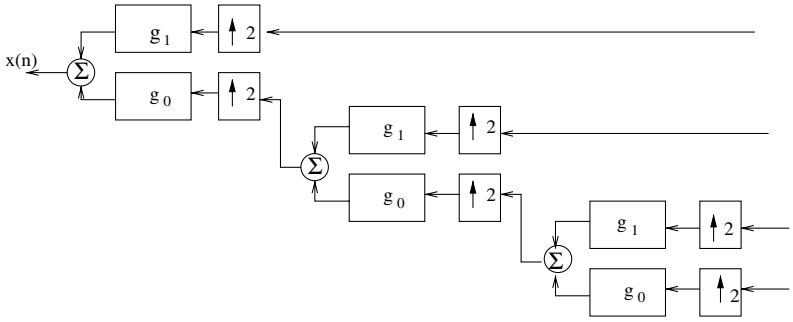


Fig. A.8 Three-level multi-resolution synthesis.

A multi-resolution analysis is created by successive application of the two-channel filterbank only to the lowpass subband of the previous stage. Figure A.7 shows three stages of two-channel filter bank analysis on the lowpass outputs that generates $M = 4$ subbands. Shown also are the frequency occupancies of these subbands, which are of widths $\pi/8, \pi/8, \pi/4$, and $\pi/2$ in order of low to high frequency. The lowest frequency subband signal is said to be the coarsest or lowest resolution. The subbands are recombined in three stages of two-channel synthesis in the reverse order of the analysis stages, as depicted in Figure A.8. The two lowest frequency subbands are first combined in the two-channel synthesizer to create the lowpass subband $[0, \pi/4)$ that is the next higher level of resolution. This subband is then combined

with the subband $[\pi/4, \pi/2)$ to create the next higher level of resolution. Then this $[0, \pi/2)$ subband is combined with the $[\pi/2, \pi]$ high frequency subband to produce the full resolution original signal. So, the subbands that are the output of three stages of two-channel filterbank decomposition are said to contain four levels of resolution. The lowest level is 1/8 the original sequence length, the next level 1/4, the next 1/2, and the final one $1\times$ original length.

A.3.1 Realizable Perfect Reconstruction Filters

The concept of subband transforms and multiresolution transforms was presented with ideal filters for the sake of conceptual simplicity. However, ideal filters with zero transition length between pass band and stop band are not physically realizable. Normally, use of realizable filters will cause aliasing (spectral overlap) due to the finite length in this transition region. However, there is a class of filters with frequency response in the transition region that cancels aliasing. A depiction of two such half-band filter responses is shown in Figure A.9. The mirror symmetry of the magnitudes of the frequency response about $\pi/2$ is called *quadrature mirror* symmetry and the filters are called quadrature mirror or qmf filters. The two-channel case with half-band filters being a basic building block allows us to focus only on this case. In order to fulfill the requirement of perfect reconstruction, we choose linear phase, finite impulse response (FIR) filters. The impulse responses of such filters are either symmetric or anti-symmetric about the midpoint of their support. For an arbitrary FIR filter, $h(n)$, $0 \leq n \leq L-1$,

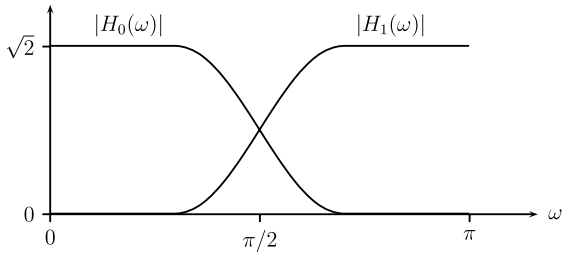


Fig. A.9 Alias-cancelling filters.

linear phase dictates that

$$h(n) = \pm h(L-1-n) \tag{A.15}$$

in the time domain and

$$H(\omega) = \pm e^{-j(L-1)\omega} H^*(\omega) \tag{A.16}$$

in the frequency domain. Omitting for now the linear phase condition, the frequency responses of the two analysis filters must obey the following relationships for perfect reconstruction:

$$H_1(\omega) = H_0(\omega + \pi) \tag{A.17}$$

$$G_0(\omega) = H_0(\omega), \quad G_1(\omega) = -H_1(\omega) \tag{A.18}$$

$$H_0^2(\omega) - H_0^2(\omega + \pi) = 2e^{-j(L-1)\omega}, \tag{A.19}$$

where L is even to make the delay odd. The first relationship expresses the quadrature mirror property, which corresponds to $h_1(n) = (-1)^n h_0(n)$ in the time domain. The next relationships in (A.18) assure reconstruction free of aliasing. The last relationship is the power complementarity or all-pass property. When we impose the linear phase requirement in (A.16) on the filters, (A.19) becomes

$$|H_0(\omega)|^2 + |H_0(\omega + \pi)|^2 = 2. \tag{A.20}$$

Actually these properties can only be satisfied exactly only for the trivial case of length $L = 2$ FIR filters. However, there are fairly accurate approximations developed for longer odd lengths.

The symmetry and equal lengths of the analysis and synthesis filters are only sufficient, but not necessary for perfect reconstruction. This symmetry makes the low and high pass filters orthogonal.[4] One can achieve perfect reconstruction using biorthogonal filters.[5] One such solution follows.

Let the time-reversal of the impulse response $h(n)$ be denoted $\tilde{h}(n) = h(-n)$. The corresponding discrete-time Fourier transform relationship is $\tilde{H}(\omega) = H^*(\omega)$, with $H(\omega)$ and $\tilde{H}(\omega)$ being the discrete-time

[4] Linear phase, FIR filters of even length are necessary to realize orthogonality.

[5] The coefficients in an orthogonal basis expansion are found by projection on the same orthogonal basis, whereas coefficients in a biorthogonal basis expansion are found by projection onto another basis, orthogonal to the expansion basis.

Fourier transforms of $h(n)$ and $\tilde{h}(n)$, respectively. The following filter relationships also guarantee perfect reconstruction.

$$H_1(\omega) = e^{-j\omega}\tilde{G}_0^*(\omega + \pi) \tag{A.21}$$

$$\tilde{G}_1(\omega) = e^{-j\omega}H_0^*(\omega + \pi). \tag{A.22}$$

You may check that power complementarity in (A.19) is satisfied. The corresponding impulse response relationships are

$$h_1(n) = (-1)^{1-n}g_0(1-n) \quad \text{and} \quad \tilde{g}_1(n) = (-1)^{1-n}h_0(1-n). \tag{A.23}$$

If we remove the tildes, or equivalently, replace the time-reversals by their original forms, we revert to the quadrature mirror filters.

A.3.2 Orthogonal Wavelet Transform

In a two-channel, perfect reconstruction filter bank with orthogonal filters as just above, the only flexibility left is the choice of the prototype lowpass analysis or synthesis filter. The demand of perfect reconstruction defines the other three filters. The orthogonal wavelet filters constitute a certain class of these prototype filters. We shall review the derivation that a wavelet transform and its inverse are realized using a two-channel, perfect reconstruction filter bank composed of these so-called wavelet filters.

The idea is first to consider the discrete-time input signal to the filter bank as coefficients in an orthonormal expansion of a continuous-time signal $x(t)$. Expressed mathematically, let $\theta_n(t), n = \ldots, -1, 0, 1, \ldots$ be an orthonormal set of basis functions, meaning[6]

$$\langle \theta_m(t), \theta_n(t) \rangle = \delta(n - m),$$

where $\delta(n)$ is the Kronecker delta function, defined by

$$\delta(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n \neq 0. \end{cases} \tag{A.24}$$

[6] The notation $\langle f(t), g(t) \rangle$ means the inner product of the functions $f(t)$ and $g(t)$, which in this case is $\int_{-\infty}^{+\infty} f(t)g^*(t)dt$.

The particular basis of interest here is derived from a so-called scaling function $\phi(t)$ such that

$$\langle \phi(2^{-k}t), \phi(2^{-k}t - m) \rangle = 2^k \delta(m). \tag{A.25}$$

The quantity k is an integer called the scale or dilation level, and m is an integer shift or translation. The meaning of Equation (A.25) is that different shifts of dilations of $\phi(t)$ by factors of 2^k are orthogonal. Furthermore, the basis for $\phi(2^{-(k-1)}(t)$ is the set $\{\phi(2^{-k}t - n)\}$. In particular, for $k = -1$, the set $\{\phi(2t - n)\}$ is an orthogonal basis for $\phi(t)$. Therefore $\phi(t)$ can be expressed as a linear combination of these basis functions, according to

$$\phi(t) = \sum_n q_0(n)\phi(2t - n). \tag{A.26}$$

This expression is called a dilation equation. We therefore choose to express the input continuous-time signal in terms of the orthonormal basis $\{\phi(2t - n)\}$ as

$$x(t) = \sum_m x(m)\phi(2t - m). \tag{A.27}$$

Reproducing $x(n)$ for all n is our objective for the cascade of the analysis and synthesis filter banks.

There is a wavelet function $\psi(t)$ associated with the scaling function $\phi(t)$ that is orthonormal to all shifts of itself and orthogonal to all shifts of the scaling function according to

$$\langle \psi(t), \psi(t - m) \rangle = \delta(m) \tag{A.28}$$

$$\langle \psi(t), \phi(t - m) \rangle = 0 \text{ for all } m. \tag{A.29}$$

In addition, it must integrate to zero, i.e., $\int_{-\infty}^{+\infty} \psi(t)dt = 0$. The wavelet function also satisfies a similar dilation equation

$$\psi(t) = \sum_n q_1(n)\phi(2t - n), \tag{A.30}$$

This wavelet function together with its integer shifts generates an orthogonal basis for the space orthogonal to that generated by $\phi(t)$ and its integer shifts. These two spaces together constitute the space of

the next lower scale $k = -1$. Therefore, since $x(t)$ belongs to this scale, it may be expressed as a linear combination of the shifts of $\phi(t)$ and $\psi(t)$.

$$x(t) = \sum_n x_0(n)\phi(t-n) + \sum_n x_1(n)\psi(t-n), \tag{A.31}$$

where $x_0(n)$ and $x_1(n)$ are the sample values at time n of the lowpass and highpass subbands, respectively. Now to determine these subband sample values, we reconcile Equations (A.27) and (A.31) for $x(t)$. In (A.31), $x_0(n)$ and $x_1(n)$ must be the orthogonal projections of $x(t)$ onto $\phi(t-n)$ and $\psi(t-n)$, respectively. Taking $x_0(n)$ first,

$$\begin{aligned} x_0(n) &= \langle x(t), \phi(t-n)\rangle && \text{(A.32)}\\ &= \left\langle \sum_m x(m)\phi(2t-m), \phi(t-n)\right\rangle && \text{(A.33)}\\ &= \sum_m x(m)\langle \phi(2t-m), \phi(t-n)\rangle && \text{(A.34)}\\ &= \frac{1}{2}\sum_m x(m) q_0(m-2n). && \text{(A.35)} \end{aligned}$$

The last step follows from substitution of the dilation equation (A.27) and the orthogonality of different shifts of $\phi(2t)$. A similar calculation for $x_1(n) = \langle x(t), \psi(t-n)\rangle$ yields

$$x_1(n) = \frac{1}{2}\sum_m x(m) q_1(m-2n). \tag{A.36}$$

Recognizing these equations for the subband sequences as correlations, we define

$$h_0(n) = \frac{1}{2}q_0(-n) \quad \text{and} \quad h_1(n) = \frac{1}{2}q_1(-n). \tag{A.37}$$

Therefore, the subband sequences may be expressed as convolutions

$$x_0(n) = \sum_m x(m) h_0(2n-m), \tag{A.38}$$

$$x_1(n) = \sum_m x(m) h_1(2n-m). \tag{A.39}$$

They may be implemented by feeding the input signal $x(n)$ into the lowpass filter with impulse response $h_0(n)$ and the highpass filter with impulse response $h_1(n)$ and retaining only the even-indexed time samples.

The requirement that $\psi(t)$ and $\phi(t-n)$ must be orthogonal for all n establishes the relationship between $h_0(n)$ and $h_1(n)$. Substituting the dilation equations (A.26) and (A.30) into $\langle \psi(t), \phi(t-n) \rangle = 0$ yields

$$\frac{1}{2} \sum_{\ell} q_1(\ell) q_0(\ell - 2n) = 0,$$

which deems that the crosscorrelation between the $q_1(n)$ and $q_0(n)$ sequences must be zero at even lags. The equivalent Fourier transform relationship is

$$Q_1(\omega) Q_0^*(\omega) + Q_1(\omega + \pi) Q_0^*(\omega + \pi) = 0, \tag{A.40}$$

where $Q_0(\omega)$ and $Q_1(\omega)$ are the respective discrete-time Fourier transforms of $q_0(n)$ and $q_1(n)$. A solution to this equation is

$$Q_1(\omega) = e^{-j\omega} Q_0^*(\omega + \pi). \tag{A.41}$$

In the time domain, this condition means

$$q_1(n) = (-1)^{1-n} q_0(1-n), \tag{A.42}$$

or equivalently in terms of the filter impulse and frequency responses,

$$h_1(n) = (-1)^{n-1} h_0(n-1), \tag{A.43}$$

$$H_1(\omega) = e^{-j\omega} H_0(\omega + \pi). \tag{A.44}$$

The latter equation (A.44) expresses the qmf property.

In order to reconstruct $x(t)$ (and hence $x(n)$) from its subband sequences, we look back to $x(t)$ expressed as in (A.31) and substitute the dilation equations for $\phi(t)$ and $\psi(t)$. Doing so with a few variable replacements and collection of terms yields the following formula:

$$x(t) = \sum_{n} \left[\sum_{\ell} x_0(\ell) q_0(n - 2\ell) + \sum_{\ell} x_1(\ell) q_1(n - 2\ell) \right] \phi(2t - n).$$

The bracketed sums in the above expression must amount to $x(n)$. To prove the required filter bank operations, we separate the two sums within these brackets and define

$$\tilde{x}_0(n) = \sum_\ell x_0(\ell) q_0(n - 2\ell) \tag{A.45}$$

$$\tilde{x}_1(n) = \sum_\ell x_1(\ell) q_1(n - 2\ell). \tag{A.46}$$

Let us define the impulse responses

$$\tilde{h}_0(n) = h_0(-n) \quad \text{and} \quad \tilde{h}_1(n) = h_1(-n).$$

Using these definitions and substituting the filter impulse responses in (A.37) yield

$$\tilde{x}_0(n) = 2\sum_\ell x_0(\ell) \tilde{h}_0(n - 2\ell) \tag{A.47}$$

$$\tilde{x}_1(n) = 2\sum_\ell x_1(\ell) \tilde{h}_1(n - 2\ell). \tag{A.48}$$

We can interpret these equations as inserting zeroes between samples of $x_0(n)$ and $x_1(n)$ and filtering the resulting sequences with $\tilde{h}_0(n)$ and $\tilde{h}_1(n)$, respectively. The sum of the ouputs reveals $x(n)$, i.e., $x(n) = \tilde{x}_0(n) + \tilde{x}_1(n)$. Therefore the synthesis filters in Figure A.4 are

$$g_0(n) = \tilde{h}_0(n) \quad \text{and} \quad g_1(n) = \tilde{h}_1(n). \tag{A.49}$$

A.3.3 Biorthogonal Wavelet Transform

The orthogonal wavelet transform just described used the same orthogonal basis functions for decomposition (analysis) and reconstruction (synthesis). A biorthogonal wavelet tranform results when we have related, but different bases for decomposition and reconstruction. A scaling function $\phi(t)$ and its so-called dual scaling function $\tilde{\phi}(t)$ are needed. The shifts of these functions individually are linearly independent and together form a basis for the space of $x(t)$. But now the basis functions are linearly independent and not orthogonal. These functions

also obey dilation relations:

$$\phi(t) = \sum_n q_0(n)\phi(2t-n), \tag{A.50}$$

$$\tilde{\phi}(t) = \sum_n \tilde{q}_0(n)\tilde{\phi}(2t-n). \tag{A.51}$$

Furthermore, the two scaling functions must satisfy the biorthogonality condition

$$\langle \phi(t), \tilde{\phi}(t-k) \rangle = \delta(k). \tag{A.52}$$

We also need two wavelet functions $\psi(t)$ and $\tilde{\psi}(t)$ which are duals of each other. They must meet the following conditions:

$$\begin{aligned} \langle \psi(t), \tilde{\psi}(t-k) \rangle &= \delta(k), \\ \langle \psi(t), \tilde{\phi}(t-k) \rangle &= 0, \\ \langle \tilde{\psi}(t), \phi(t-k) \rangle &= 0. \end{aligned} \tag{A.53}$$

Since $\psi(t)$ and $\tilde{\psi}(t)$ are in the respective spaces spanned by $\{\phi(2t-n)\}$ and $\{\tilde{\phi}(2t-n)\}$, they can be expressed by dilation equations

$$\psi(t) = \sum_n q_1(n)\phi(2t-n), \tag{A.54}$$

$$\tilde{\psi}(t) = \sum_n \tilde{q}_1(n)\tilde{\phi}(2t-n). \tag{A.55}$$

Similar to before, we regard the input sequence $x(n)$ to be coefficients in the expansion of $x(t)$ in the basis $\{\phi(2t-n)\}$:

$$x(t) = \sum_m x(m)\phi(2t-m). \tag{A.56}$$

Although the spaces spanned by $\{\phi(t-n)\}$ and $\{\psi(t-n)\}$ are not orthogonal, we can still write $x(t)$ as

$$x(t) = \sum_n x_0(n)\phi(t-n) + \sum_n x_1(n)\psi(t-n), \tag{A.57}$$

because the second term is the difference between $x(t)$ in the original, finer space expressed in (A.56) and its approximation in the coarser

space represented by the first term. The discrete-time subband signals $x_0(n)$ and $x_1(n)$ are now found as projections onto the dual biorthogonal basis as

$$x_0(n) = \langle x(t), \tilde{\phi}(t-n) \rangle, \tag{A.58}$$

$$x_1(n) = \langle x(t), \tilde{\psi}(t-n) \rangle. \tag{A.59}$$

Again, substituting $x(t)$ in (A.56) and the dilation equations in (A.55) into the above equations for the subband signals give

$$x_0(n) = \sum_m \sum_\ell x(m) \tilde{q}_0(\ell) \langle \tilde{\phi}(2t-2n-\ell), \phi(2t-m) \rangle, \tag{A.60}$$

$$x_1(n) = \sum_m \sum_\ell x(m) \tilde{q}_1(\ell) \langle \tilde{\phi}(2t-2n-\ell), \phi(2t-m) \rangle. \tag{A.61}$$

Invoking the biorthogonality properties in (A.54) yields

$$x_0(n) = \frac{1}{2} \sum_m x(m) \tilde{q}_0(m-2n), \tag{A.62}$$

$$x_1(n) = \frac{1}{2} \sum_m x(m) \tilde{q}_1(m-2n). \tag{A.63}$$

By defining

$$\tilde{h}_0(n) \equiv \frac{1}{2} \tilde{q}_0(-n) \quad \text{and} \quad \tilde{h}_1(n) \equiv \frac{1}{2} \tilde{q}_1(-n), \tag{A.64}$$

we can express $x_0(n)$ and $x_1(n)$ as convolutions

$$x_0(n) = \sum_m x(m) \tilde{h}_0(2n-m), \tag{A.65}$$

$$x_1(n) = \sum_m x(m) \tilde{h}_1(2n-m). \tag{A.66}$$

Therefore, $x_0(n)$ and $x_1(n)$ are the outputs of a two-channel filter bank with filter impulse responses $\tilde{h}_0(n)$ and $\tilde{h}_1(n)$ followed by downsampling by a factor of two, which is depicted in Figure A.10.

The reconstruction or synthesis follows the same procedure as in the orthogonal case. It does not involve the dual functions. We start with the expression in (A.57) for $x(t)$, substitute the dilation equations

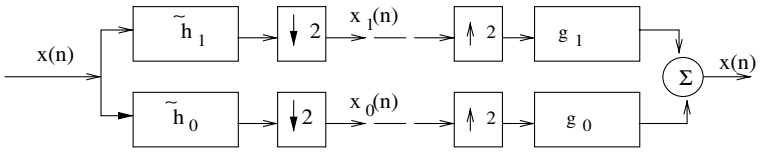


Fig. A.10 Two-channel analysis and synthesis biorthogonal filter banks. $\tilde{h}_0$ and g_0 denote impulse responses of the lowpass filters; $\tilde{h}_1$ and g_1 those of the highpass filters.

Property	Constraint
$\langle \phi(t), \tilde{\phi}(t-n) \rangle = \delta(n)$	$\sum_\ell q_0(\ell)\tilde{q}_0(\ell - 2n) = 2\delta(n)$
$\langle \psi(t), \tilde{\psi}(t-n) \rangle = \delta(n)$	$\sum_\ell q_1(\ell)\tilde{q}_1(\ell - 2n) = 2\delta(n)$
$\langle \psi(t), \tilde{\phi}(t-n) \rangle = 0$	$\sum_\ell q_1(\ell)\tilde{q}_0(\ell - 2n) = 0$
$\langle \tilde{\psi}(t), \phi(t-n) \rangle = 0$	$\sum_\ell \tilde{q}_1(\ell)q_0(\ell - 2n) = 0$

in (A.26) and (A.30), collect terms and change variables to determine the outputs $\hat{x}_0(n)$ corresponding to the input $x_0(n)$ and $\hat{x}_1(n)$ corresponding to the input $x_1(n)$ to reveal

$$\hat{x}_0(n) = \sum_\ell x_0(\ell) q_0(n - 2\ell) \tag{A.67}$$

$$\hat{x}_1(n) = \sum_\ell x_1(\ell) q_1(n - 2\ell) \tag{A.68}$$

and that

$$x(n) = \hat{x}_0(n) + \hat{x}_1(n). \tag{A.69}$$

Using the filter definitions

$$h_0(n) \equiv q_0(n) \quad \text{and} \quad g_1(n) \equiv q_1(n), \tag{A.70}$$

we see that the subband reconstructions are produced by upsampling by 2 and filtering with either the impulse response $g_0(n)$ or $g_1(n)$ and then added to give perfect reconstruction of $x(n)$. Figure A.10 depicts also this synthesis filter bank.

The relationships between the various filters are dictated by the biorthogonal properties of the scaling and wavelet functions and their duals. These relationships are proved by substituting the appropriate dilation equations into a condition in (A.54) or (A.52). These manipulations yield the following correspondences between the property and its

constraint on the coefficients of the dilation equations: The constraint equations are usually expressed in terms of the discrete-time Fourier transforms of the coefficient sequences. We capitalize the letter symbol to denote its Fourier Transform, e.g., $q_0(n) \Leftrightarrow Q_0(\omega)$. The discrete Fourier transforms of the constraint equations in the table are listed below.

$$\begin{aligned} Q_0(\omega)\tilde{Q}_0^*(\omega) + Q_0(\omega+\pi)\tilde{Q}_0^*(\omega+\pi) &= 2, \\ Q_1(\omega)\tilde{Q}_1^*(\omega) + Q_1(\omega+\pi)\tilde{Q}_1^*(\omega+\pi) &= 2, \\ Q_1(\omega)\tilde{Q}_0^*(\omega) + Q_1(\omega+\pi)\tilde{Q}_0^*(\omega+\pi) &= 0, \\ \tilde{Q}_1(\omega)Q_0^*(\omega) + \tilde{Q}_1(\omega+\pi)Q_0^*(\omega+\pi) &= 0. \end{aligned} \tag{A.71}$$

The solutions are not unique, but the following relationships satisfy the above constraint equations.

$$\begin{aligned} \tilde{Q}_1(\omega) &= e^{-j\omega}Q_0^*(\omega+\pi) \\ Q_1(\omega) &= e^{-j\omega}\tilde{Q}_0^*(\omega+\pi). \end{aligned} \tag{A.72}$$

Expressing these solutions in terms of the frequency responses of the filters corresponding to these dilation coefficient transforms reveals

$$\begin{aligned} G_0(\omega+\pi) &= 2e^{-j\omega}\tilde{H}_1(\omega), \\ G_1(\omega) &= 2e^{-j\omega}\tilde{H}_0(\omega+\pi). \end{aligned} \tag{A.73}$$

The above frequency response relationships of the filters guarantee cancellation of aliasing. These relationships translate to the following filter impulse response relationships in the time domain:

$$\begin{aligned} g_0(n) &= -2(-1)^{n-1}\tilde{h}_1(n-1), \\ g_1(n) &= 2(-1)^{n-1}\tilde{h}_0(n-1). \end{aligned} \tag{A.74}$$

This still leaves open the impulse responses of the filters $g_0(n)$ and $g_1(n)$ that satisfy the constraints. One common solution that gives equal lengths of these filters is

$$Q_0(\omega) = -e^{-j\omega}Q_1^*(\omega+\pi). \tag{A.75}$$

or in terms of the frequency responses,

$$G_0(\omega) = -e^{-j\omega}G_1^*(\omega+\pi). \tag{A.76}$$

The time domain equivalent is

$$g_0(n) = -(-1)^{1-n}\tilde{g}_1(1-n). \tag{A.77}$$

The relationship between $\tilde{h}_0(n)$ and $\tilde{h}_1(n)$, implied by (A.73) and (A.76), is

$$\tilde{h}_0(n) = -(-1)^{1-n}\tilde{h}_1(1-n). \tag{A.78}$$

Solutions exist for unequal lengths of the synthesis or analysis filters. However, the lengths of the two filters must be both even or both odd. In fact, the most common filters used in wavelet transform image coding are biorthogonal filters of lengths 9 and 7, the so-called biorthogonal CDF(9,7) filter pair [5]. As the construction of filters is beyond the scope of this appendix, we urge the interested reader to consult one or more of many excellent textbooks, such as [16, 23], and [21].

These two-channel analysis filterbanks can be repeatedly applied to either of the two outputs to divide further into two half-width subbands. The reason is that the property of a single dilation of the scaling and wavelet functions, such as that expressed in (A.26) or (A.30), implies the same property for succeeding stages of dilation. Therefore, subband decompositions, such as those in Figures A.5 and A.7, can be created. Analogously, the corresponding synthesis filterbanks can be configured in successive stages of two-channel synthesis filterbanks.

References

[1] N. Ahmed, T. R. Natarajan, and K. R. Rao, "On image processing and a discrete cosine transform," *IEEE Transactions on Computers*, vol. 23, pp. 90–93, January 1974.

[2] J. Andrew, "A simple and efficent hierarchical coder," *Proceedings of IEEE International Conference on Image Processing '97*, vol. 3, pp. 658–661, 1997.

[3] Y. Cho and W. A. Pearlman, "Quantifying the performance of zerotrees of wavelet coefficients: Degree-k zerotree model," *IEEE Transactions on Signal Processing*, vol. 55, no. 6.1, pp. 2425–2431, 2007.

[4] C. Chrysafis, A. Said, A. Drukarev, A. Islam, and W. A. Pearlman, "SBHP — A low complexity wavelet coder," *IEEE International Conference Acoustics, Speech and Signal Processing* (*ICASSP2000*), vol. 4, pp. 2035–2038, 2000.

[5] A. Cohen, I. Daubechies, and J.-C. Feauveau, "Biorthogonal bases of compactly supported wavelets," *Communications of Pure and Applied Mathematics*, vol. 45, no. 5, pp. 485–560, June 1992.

[6] R. G. Gallager and D. V. Vorrhis, "Optimal source codes for geometrically distributed integer alphabets," *IEEE Transactions on Information Theory*, vol. 21, pp. 228–230, March 1975.

[7] S. W. Golomb, "Run-length encodings," *IEEE Transactions on Information Theory*, vol. 12, pp. 399–401, July 1966.

[8] E. H. Hong and R. E. Ladner, "Group testing for image compression," *IEEE Transactions on Image Processing*, vol. 11, no. 8, pp. 901–911, 2002.

[9] S.-T. Hsiang and J. W. Woods, "Embedded image coding using zeroblocks of subband/wavelet coefficients and context modeling," *IEEE International Conference on Circuits and Systems* (*ISCAS2000*), vol. 3, pp. 662–665, 2000.

[10] A. Islam and W. A. Pearlman, "An embedded and efficient low-complexity hierarchical image coder," *Visual Communications and Image Processing '99, Proceedings of SPIE*, vol. 3653, pp. 294–305, January 1999.

[11] ISO/IEC 15444-1, *Information Technology-JPEG2000 Image Coding System, Part 1: Core Coding System*, (2000).

[12] ISO/IEC 15444-2, *Information Technology-JPEG2000 Extensions, Part 2: Core Coding System*, (2001).

[13] N. S. Jayant and P. Noll, *Digital Coding of Waveforms*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[14] M. J. Narasimha and A. M. Peterson, "On the computation of the discrete cosine transform," *IEEE Transactions on Communications*, vol. 26, pp. 934–936, June 1978.

[15] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said, "Efficient, low-complexity image coding with a set-partitioning embedded block coder," *IEEE Transactions as Circuits and Systems for Video Technology*, vol. 14, pp. 1219–1235, November 2004.

[16] R. M. Rao and A. S. Bopartikar, *Wavelet Transforms: Introduction to Theory and Applications*. Massachusetts, Reading, NJ: Addison Wesley Longman, Inc, 1998.

[17] A. Said and W. A. Pearlman, "An image multiresolution representation for lossless and lossy compression," *IEEE Transactions on Image Processing*, vol. 5, pp. 1303–1310, September 1996.

[18] A. Said and W. A. Pearlman, "A new, fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, June 1996.

[19] A. Said and W. A. Pearlman, "Low-complexity waveform coding via alphabet and sample-set partitioning," *Visual Communications and Image Processing '97, Proceedings SPIE*, vol. 3024, pp. 25–37, 1997.

[20] J. M. Shapiro, "Embedded image coding using zerotress of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3445–3462, 1993.

[21] G. Strang and T. Nguyen, *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1997.

[22] D. S. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards, and Practice*. Boston/Dordrecht/London: Kluwer Academic Publishers, 2002.

[23] M. Vetterli and J. Kovačević, *Wavelets and Subband Coding*. Prentice Hall PTR, NJ: Upper Saddle River, 1995.

[24] Z. Xiong, O. Gulyeruz, and M. T. Orchard, "A DCT-based embedded image coder," *IEEE Signal Processing Letters*, vol. 3, pp. 289–290, November 1996.

[25] Y. Yemini and J. Pearl, "Asymptotic properties of discrete unitary transforms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, pp. 366–371, October 1979.