Foundations and Trends® in
Signal Processing
Vol. 2, No. 3 (2008) 181–246

DOI: 10.1561/2000000014

now
the essence of knowledge

Image Wavelet Coding Systems: Part II 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

This monograph describes current-day wavelet transform image coding systems. As in the first part, steps of the algorithms are explained thoroughly and set apart. An image coding system consists of several stages: transformation, quantization, set partition or adaptive entropy coding or both, decoding including rate control, inverse transformation, de-quantization, and optional processing (see Figure 1.6). 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 subblocks of subbands, other methods, such as SBHP (Subband Block Hierarchical Partitioning) [3] and EZBC (Embedded Zero Block Coder) [8], 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 (Set Partitioning In Hierarchical Trees) [15], whose output bitstream can be decoded partially corresponding to a designated region of interest and is simultaneously quality and resolution scalable.

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

Subband/Wavelet Coding Systems

1.1 Introduction

This monograph describes coding systems, primarily for images, that use the principles and algorithms explained in the first part. A complete coding system uses a conjunction of compression algorithms, entropy coding methods, source transformations, statistical estimation, and ingenuity to achieve the best result for the stated objective. The obvious objective is compression efficiency, stated as the smallest rate, in bits per sample, for a given distortion in lossy coding or smallest rate or compressed file size in lossless coding. However, other attributes may be even more important for a particular scenario. For example, in medical diagnosis, decoding time may be the primary concern. For mobile devices, small memory and low power consumption are essential. For broadcasting over packet networks, scalabilty in bit rate and/or resolution may take precedence. Usually to obtain other attributes, some compression efficiency may need to be sacrificed. Of course, one tries to obtain as much efficiency as possible for the given set of attributes wanted for the system. Therefore, in our description of systems, we shall also explain how to achieve other attributes besides compression efficiency.

1.2 Wavelet Transform Coding Systems

The wavelet transform consists of coefficients grouped into subbands belonging to different resolutions or scales with octave frequency separation. As such, it is a natural platform for producing streams of code bits (hereinafter called *codestreams*) that can be decoded at multiple resolutions. Furthermore, since the coefficients are the result of short finite impulse response (FIR) filters acting upon the input data, they retain local characteristics of the data.[1] Most natural images show wavelet transforms with magnitudes of their coefficients generally decreasing as the subband scale or resolution increases. In other words, most of the energy of these images is packed into the lower frequency subbands. Furthermore, there are intra-subband and inter-scale statistical dependencies that can be exploited for compression. A typical wavelet transform with three scales, that of the 512×512 Lena image, is displayed in Figure 1.1. The values of the coefficients may be negative, except in the lowest frequency subband in the top left corner, and require precision exceeding the eight bits of most displays. Therefore, for display purposes, the coefficients in all subbands were scaled to the range of 256 grey levels. In all but the lowest frequency subband, the zero value corresponds to the middle grey level of 128. One can clearly see the edge detail propagating across scales to the same relative locations in the higher frequency subbands as one moves from the top left to the top right, bottom left, or bottom right of the transform. However, this detail becomes quite faint in the highest frequency subbands, where the coefficients are predominantly middle grey with actual values of zero. Notice also that within subbands, the close neighbors of a coefficient do not change much in value, except across edges. All these characteristics make the wavelet transform an effective platform for efficient compression with the attributes of resolution scalability and random access decoding. The arrangement of the coefficients into subbands belonging to different scales or resolutions makes possible encoding or decoding different scales. Random access to selected regions of interest in the image is possible, because

[1] An introduction to wavelet transforms and the explanation of its implementation by two-channel filter banks appear in Appendix A in Part I.

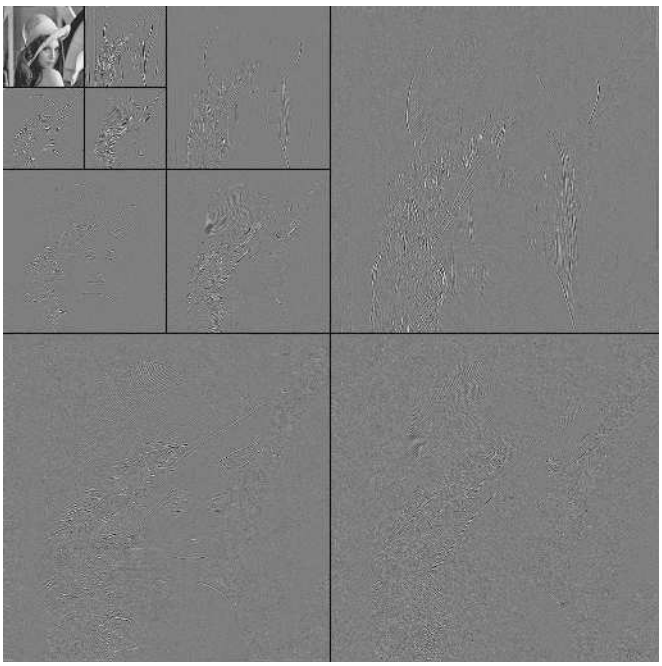

Fig. 1.1 Display of image subbands of a 3-level, dyadic wavelet transform. Middle grey level of 128 corresponds to 0 value of a coefficient in all subbands, excluding the lowest frequency one in the top left corner.

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

The subband labeling diagram for the wavelet transform in Figure 1.1 is depicted in Figure 1.2. This subband arrangement was produced by three stages of alternate low and high pass horizontal and vertical filterings of the resulting low horizontal and low vertical subbands followed by 2:1 downsampling. (The first stage input is just the source image itself.) We show the analysis and synthesis for two stages explicitly in Figures 1.3 and 1.4. The LL_2 (low horizontal, low vertical, 2nd stage) subband is just a coarse, factor of 2^2 reduction (in both dimensions) of the original image. Upsampling and filtering of LL_2, LH_2, HL_2, and HH_2 subbands yields the LL_1 subband, which is two times the scale of LL_2, but still scaled down by a factor of 2^1 from

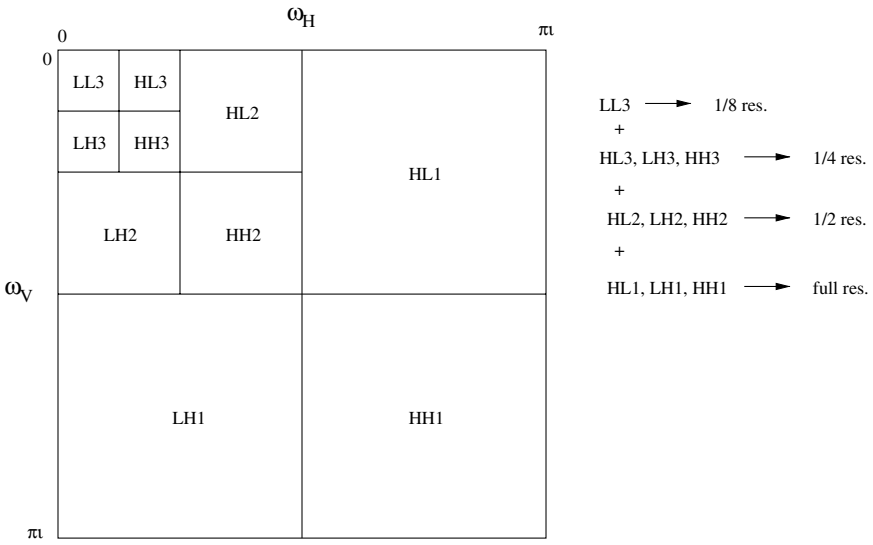

Fig. 1.2 Subbands of a 3-level, dyadic wavelet transform.

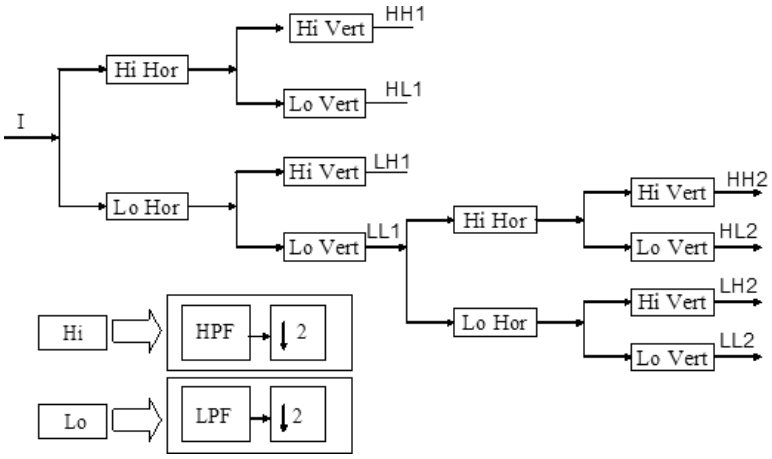

Fig. 1.3 Two-level recursive lowpass filter analysis of image I.

the original. And so it repeats on the LL_1 subband for one more stage to obtain the full scale reconstruction of the original input. Therefore at each synthesis stage, we can obtain a reduced scale version of the original image.

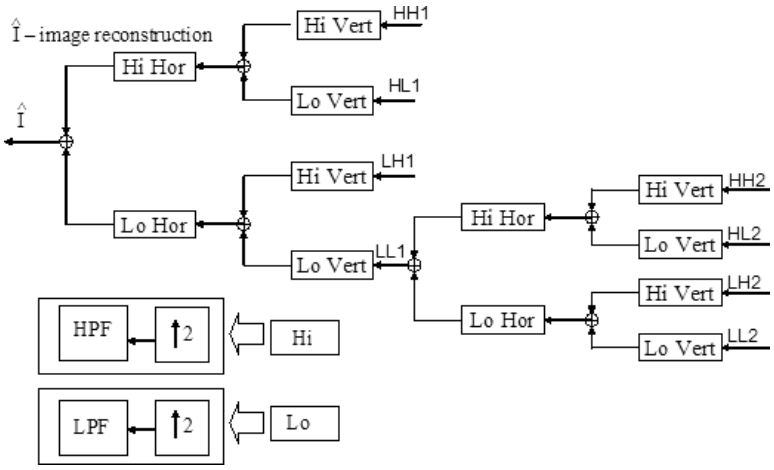

Fig. 1.4 Two-level recursive lowpass filter synthesis of image $\hat{I}$.

In order to synthesize just any given (rectangular) region of an image, one needs only to locate the coefficients in the corresponding regions in the subbands of the wavelet transform and apply them to the same synthesis filter bank. These regions are located in the subbands in the same relative position as in the image, as illustrated in Figure 1.5 for an image and its two-level wavelet transform. The fractional area in these subbands is slightly larger than that in the image, because coefficients outside the designated region result from filtering of image samples inside the region only near the boundaries, due to the finite length of the filters. That is why the rectangles in the subbands that exactly correspond to the image rectangle are shown inside larger rectangles. Again, the region can be reconstructed at different resolutions, if desired.

The best filters to use from the standpoint of achieving the best compression efficiency or highest coding gain have real number tap values, represented as floating point numbers that are precise only within the limits of the computer. These filters produce floating point wavelet coefficients. Hence, the inverse transform, done again with floating point filters, may not produce an exact replica of the source. For example, if a filter tap value contains a factor of $1 + \sqrt{3}$, then the floating point representation cannot be exact and all subsequent

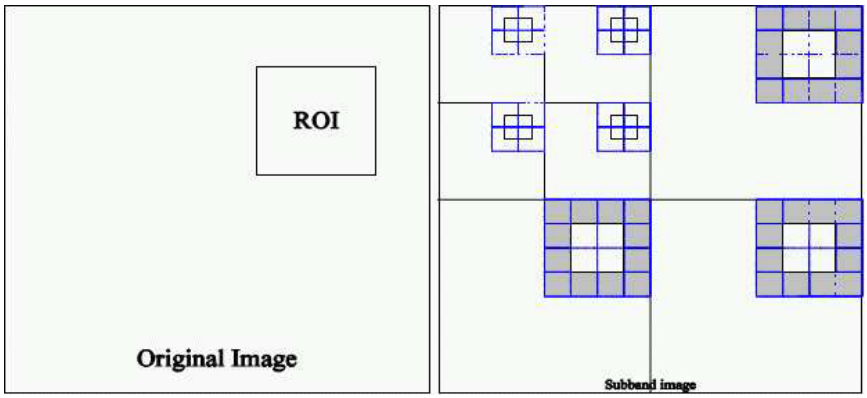

Fig. 1.5 Subband regions in the wavelet transform corresponding to an image region. The inner rectangles correspond to the exact corresponding fractional area of the image region of interest (ROI). The areas between the inner and outer rectangles contain coefficients needed to reconstruct ROI exactly.

mathematical operations will propagate this inexactness. Coding the wavelet coefficients means converting them to a compact integer representation. Therefore, even if the coding is perfectly lossless, the inverse wavelet transform may reconstruct the image with error from the original and one cannot guarantee perfectly lossless image compression in a wavelet transform system using floating point filters. One achieves what is often called *virtually lossless* reconstruction. In most practical applications, that is not a hindrance, but sometimes, often for legal reasons that arise in certain fields such as diagnostic medicine, it is crucial to achieve perfectly lossless compression. Therefore, for perfectly lossless image compression in a wavelet-based coding system, one must use filters that operate with integer arithmetic to produce integer transform coefficients. This also assures that there are no errors due to limited precision in the synthesis stage. There will be a small, usually tolerable, reduction in potential coding gain from the use of these integer-to-integer filters for wavelet transformation.

1.3 Generic Wavelet-based Coding Systems

The generic wavelet transform coding system, regardless of the source, normally starts with subband/wavelet transformation of the source

input. There is often then an optional pre-processing step that consists of statistical estimation, segmentation, weighting, and/or classification of the transform coefficients. Then the coefficients are subjected to quantization and/or set partitioning, usually followed by entropy coding, such as Huffman or arithmetic coding. Along with overhead information generated by the pre-processor, the encoded transform coefficients are written to the output codestream. The block diagram of this generic system is shown in Figure 1.6.

The decoding system, also shown in Figure 1.6, reverses the process by decoding the codestream to reconstruct the wavelet transform, post-processing this transform, according to the pre-processor's actions, and then inverting the wavelet transform to reconstruct the source. One simple example of pre-processing is weighting transform coefficients to affect the distribution of code bits, in order to enhance visual quality of an image or aural quality of audio. The post-processing step must do inverse weighting, or the reconstruction will be distorted. The pairs of the pre- and post-processors and the entropy encoder and decoder are optional for some coding systems, so are depicted in dashed line boxes in Figure 1.6.

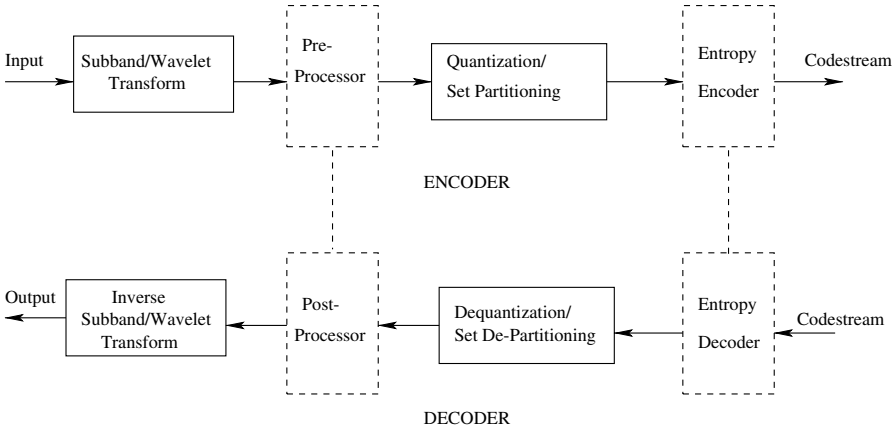

Fig. 1.6 Encoder and decoder of subband/wavelet transform coding system. The boxes with dashed lines denote optional actions.

1.4 Compression Methods in Wavelet-based Systems

We now describe various methods to compress and de-compress coefficients of a wavelet transform of a source, such as the image shown in Figure 1.1. We shall describe methods referring specifically to images, although these methods can almost always be applied to one-dimensional or higher-dimensional wavelet transforms with obvious modifications. We shall start with some of the set-partition coding methods of the previous section.

Within the Quantization/Set Partitioning system block in Figure 1.6, the quantization is necessary for floating point transform coefficients and optional for integer ones. Any quantization will result in reconstruction error, which is unavoidable for floating point coefficients. For embedded coding enabled through coding of bit planes, the quantization may be just a mere truncation of every coefficient to the nearest integer. Deeper quantization is implicit when the coding stops above the least significant $n = 0$ bit plane. Suppose coding stops just after completing bit plane n. Then every coefficient has an indeterminate added value between 0 and $2^n - 1$, corresponding to a quantizer interval. The decoder then assigns a reconstruction point within this interval.

For explicit quantization, the coefficients are quantized with a particular type of uniform quantizer, called a uniform, dead-zone quantizer. For this quantizer, thresholds are uniformly spaced by step size Δ, except for the interval containing zero, called the dead-zone, which extends from $-\Delta$ to $+\Delta$. Figure 1.7 illustrates the input–output characteristic of a uniform dead-zone quantizer with 7 quantizer levels and mid-point reconstruction values. This quantization is enacted by scaling by the step size and truncating to integers to produce the indices of the quantization intervals (often called quantization bins). The mathematical operation upon the input x to produce a bin index q, given a quantizer step size Δ is

$$q = \text{sign}(x)\lfloor |x|/\Delta \rfloor. \tag{1.1}$$

The reconstruction (de-quantization) $\hat{x}$ is given by

$$\hat{x} = \begin{cases} (q + \xi)\Delta, & q > 0 \\ (q - \xi)\Delta, & q < 0 \\ 0, & q = 0 \end{cases} \tag{1.2}$$

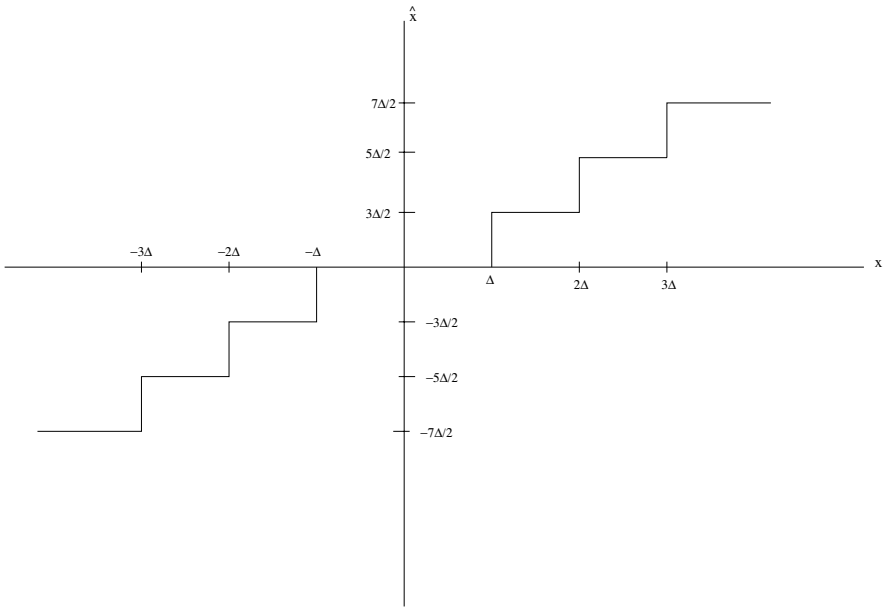

Fig. 1.7 Input–output characteristic of a 7 level, uniform dead-zone quantizer.

where $0 \leq \xi < 1$. Heretofore, the bin index q will be called the *quantizer level*. The parameter ξ is often set to place the reconstruction value at the centroid of the quantization interval. It has been derived through a model and confirmed in practice that $\xi \approx 0.38$ usually works well. In many cases, $\xi = 0.5$ is used, which places the reconstruction at the interval's midpoint. It is important to realize that when $-\Delta < x < \Delta$, the quantizer level and reconstruction value are both 0. For a subband or linear transform, there may be many coefficients, belonging especially to higher frequencies, that are set to 0. The array of quantizer levels (bin indices) q are further encoded losslessly. As we have seen, clusters of zeros can be represented with very few bits.

For optimal coding that minimizes mean squared error for a given number of bits, statistically independent subbands encoded with nonzero rate should exhibit the same mean squared error. Although not statistically independent, the subbands are often modeled as independent and are encoded independently. In such circumstances, use of the same step size for all subbands minimizes the mean-squared

reconstruction error for high rates and orthonormal synthesis filters. In accordance with this usual model, notwithstanding less than optimal coding, all subbands encoded with non zero rates are quantized with the same step size Δ.

Often, the pre-processor function of weighting subbands to enhance perceptual quality or to compensate for scaling of non-orthonormal synthesis filters is combined with quantization by using different step sizes among the subbands. In this case, there is posited a set of step sizes, $\{\Delta_m\}$, $m = 1, 2, \ldots, M$, with Δ_m being the step size for subband m. The quantization follows the same formulas as (1.1) and (1.2).

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

$$\text{PSNR} = \frac{255^2}{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(x[i,j] - \hat{x}[i,j])^2}, \tag{1.3}$$

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

1.5 Block-based Wavelet Transform Set Partition Coding

In this section, we shall describe block-based techniques, presented in detail in Section 2 of Part I, for set partition coding of the quantizer levels. The principles of providing resolution and quality scalable coding are explained first, followed by the descriptions of the specific techniques of SBHP (Subband Block Hierarchical Partitioning) [3],

the JPEG2000 Standard, Part 1 [10], and EZBC (Embedded Zero-Block Coder) [7, 8].

The quantizer levels are represented in sign-magnitude format and tests for significance are enacted on the magnitudes. There are various choices for setting the order of coding of the subbands, coding and/or partitioning within subbands, setting the thresholds for significance, coding the results of significance tests, and coding the coefficient values. We shall describe coding systems that utilize some of these possible combinations of choices.

First, we consider the subbands as the blocks to be coded. The order in which the subbands are coded follows the indicated zigzag path from lowest to highest frequency subband, as illustrated in Figure 1.8. The advantages of following this particular path is that the image is encoded and decoded progressively in resolution or in scale. For example, referring to Figure 1.8, decoding just the LL_3 (denoted LL in the figure) subband produces just a 1/8 scale reconstruction. Decoding LL_3, HL_3, LH_3, and HH_3 produces a 1/4 scale reconstruction, and so forth. One also follows this path in the search related to quality progressive coding,

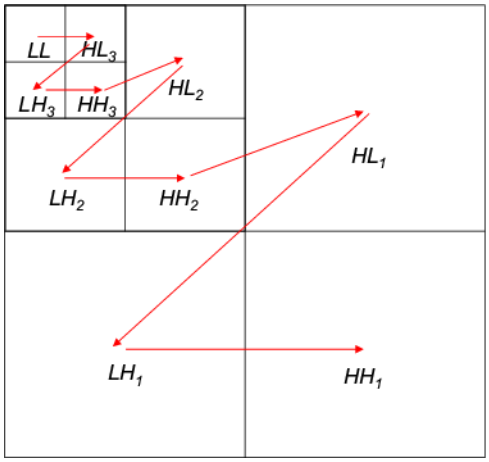

Fig. 1.8 Scanning order of subbands in a 3-level wavelet decomposition. Subbands formed are named for horizontal and vertical low- or high-passband and level of decomposition, e.g., LH_2 is horizontal low passband and vertical high passband at second recursion level.

because the subband energy tends to decrease along this path for most natural images. The general framework is delineated in Algorithm 1.1.

Algorithm 1.1. Framework for progressive-resolution wavelet transform coding.

(1) Start with M subbands $\{SB_k\}$ having quantization levels of their coefficients denoted by $q_{i,j}$.
(2) Calculate top threshold in each subband, i.e., calculate $n_k = \log_2 \lfloor s_k \rfloor$, $s_k = \max_{(i,j) \in SB_k} |q_{i,j}|$. Order subbands SB_k according to the progressive-resolution zigzag scan, as shown in Figure 1.8.
(3) Encode $n_1, n_2, \ldots, n_M$.
(4) For $k = 1, 2, \ldots, M$, if $n_k > 0$, encode SB_k.

Any one of the methods described in the previous part can be used for coding of the subbands in Step 4 of Algorithm 1.1. We shall describe below how some of these methods fit in this framework.

1.5.1 Progressive Resolution Coding

First, we consider the (quantized) subbands to be coded progressively from lower to higher resolution by the fixed-threshold, recursive quadri-section procedure in Algorithm 2.1 in Part I. Recall that this algorithm produces (approximate) value-embedded code, but not bit-embedded code. The thresholds are fixed to be successive integer powers of two and tests for significance are enacted on the magnitudes of the coefficients, as defined in Equation (2.9) in Part I. The maximum threshold, $2^{n_{\max}}$, is calculated for each subband. We deposit these thresholds into the header of the codestream. We scan the subbands in order from lowest to highest as illustrated in Figure 1.8 and partition each one in turn by the recursive quadri-section procedure of Algorithm 2.1, Part I. Briefly, we start with an LIS (List of Insignificant Sets) for each subband, each initialized with the coordinates of its top left corner. Each LIS set of a subband is tested at the current threshold, say 2^n, and if not significant, a "0" is sent to the codestream. But, if significant, the set is split

into four equal quadrants,[2] each of which is tested at threshold 2^n for significance. Significant quadrants are labeled with "1" and insignificant ones are labeled with "0", thereby creating a 4-bit binary mask that is encoded and sent to the codestream. The insignificant quadrants are appended to the bottom of their subband LIS, represented by their top left corner coordinates. Significant sets continue to be split and labeled in the same way until all significant single elements are located and encoded. Then we lower the threshold from 2^n to 2^{n-1} and test the LIS sets in order of increasing size. For each subband, we start with its initial threshold $2^{n_{\max}}$, lower it by a factor of 2 in each succeeding pass, and continue through the last pass at threshold 2^0. This coding method, which follows Algorithm 2.1, Part I, would be used to encode SB_k in Step 4 of Algorithm 1.1, the general framework of progressive (scalable) resolution coding. The following example illustrates its use and displays coding results.

Example 1.1 Resolution Scalable Coding. The source image is the 512×512, 8 bits per pixel, grey Goldhill image, shown in Figure 1.9. This image is then wavelet transformed and its wavelet coefficients are then quantized through scaling by a factor of 0.31 (step size of 1/0.31) and truncating to integers. The resolution scalable coding algorithm described above encodes the wavelet transform's quantization bin indices losslessly to a rate of 1.329 bpp (codestream size of 43,538 bytes). Portions of this codestream corresponding to the desired resolution are then decoded, de-quantized, and inverse wavelet transformed, to produce the reconstructed images. Figure 1.10 shows the reconstructed images for full, one-half, and one-quarter resolutions.

The aggregate of 4-bit binary masks corresponds to a quadtree code, as noted in Section 2, Part I. It is advantageous to entropy code these 4-bit masks instead of sending the raw quadri-section codes to the codestreams (see Table 2.3, Part I). Even a simple fixed Huffman code of 15 symbols (0000 cannot occur) shows improvement. A more

[2] For nonsquare images, the splitting is into quadrants as nearly equal in size as possible. At the last splitting stages, when a single row or column with more than one element is reached, binary splitting is employed.

Fig. 1.9 Original 512 × 512, 8 bits/pixel Goldhill image.

Fig. 1.10 Reconstructions from codestream of Goldhill coded to rate 1.329 bpp, quantizer step size = 1/0.31 at full, 1/2, and 1/4 resolutions.

sophisticated fixed Huffman code can be conditioned on the size of the block and/or the particular subband. Another level of sophistication is to adapt the code by estimating and updating the significance state probabilities as coding proceeds. Certainly, arithmetic coding may be substituted for Huffman coding in the above scenarios for entropy coding.

The decoder has prior knowledge of the wavelet transform decomposition and scanning order of its subbands, and receives the image storage parameters and subbands' maximal thresholds from the header of the incoming codestream. It therefore knows which subbands to skip in every scan corresponding to the current threshold. It initializes the LIS of every subband with its corner coordinates. It reads the significance decisions, so is able to build the same LIS populations. It will then know when it is reading the code of significant coefficients and will decode them.

1.5.2 Quality-Progressive Coding

Encoding every subband completely in the zig–zag order above produces a resolution-progressive codestream. Furthermore, each subband's codestream is approximately progressive in quality or value, because coefficients with larger most significant bits (MSB's) precede those with smaller MSB's. Figure 1.11 illustrates the coding order of the magnitude bits of coefficients in a value-progressive scan. The coefficients are ordered by the level of their MSB's and every coefficient's bits are coded from its MSB to its LSB (least significant bit). Although the subbands individually are progressive in quality, the composite codestream is not. Bits from larger coefficients in a subband scanned later will follow those of smaller coefficients from a subband scanned earlier. One can re-organize this codestream to be progressive in value, if we

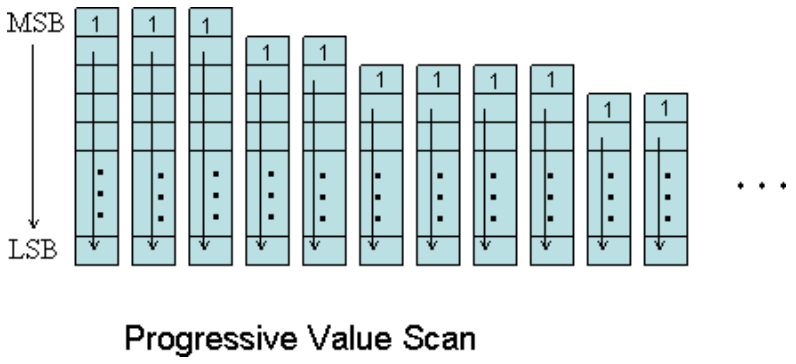

Fig. 1.11 Coding order for a progressive value codestream.

insert indicators[3] to separate the bits from coding passes at different thresholds. The code bits of coefficients significant for the same threshold are kept together in the codestream. With the aid of these indicators, we can gather together from different subbands the code bits of coefficients significant for the same threshold. Therefore, in the decoder, the coefficients with larger significance are decoded before those of lesser significance. This re-organization scheme does not order the values within the same threshold, so is only partially progressive in value. Furthermore, this re-organization may vitiate the progressiveness in resolution. One or more lower resolution subbands may not have significant coefficients at the current threshold, while higher resolution ones do. If we continue to collect the bits from these higher resolution significant coefficients at the current threshold, then we will not be keeping together code bits from the same resolution.

One can re-organize this codestream not only to be partially value-progressive, but also to be bit-embedded. Again note that bits belonging to the same threshold (with same MSB) are kept together in the codestream. The value-progressive scan reads code bits from the most significant bit downward to the least significant before proceeding to the next coefficient with the same most significant bit. We can produce a bit-embedded codestream, if starting with the largest most significant bit plane, we always move to the same bit plane of the next coefficient in the scan, whether in the same or a different subband. Referring to Figure 1.12, this path corresponds to scanning a bit plane from extreme left to extreme right at the same horizontal (bit plane) level, dropping down one level, and returning to the extreme left to repeat the rightward scan holding the same level. The general framework for value-progressive coding is delineated in Algorithm 1.2.

Algorithm 1.2. Framework for value-progressive wavelet transform coding.

(1) Start with M subbands $\{SB_k\}$ having quantization levels of their coefficients denoted by $q_{i,j}$.

---

[3] Indicators are markers or counts of bits inserted into the codestream header.

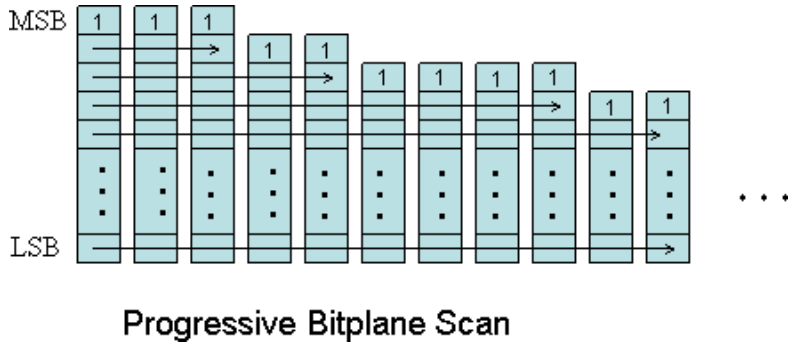

Fig. 1.12 Coding order for a bit embedded codestream.

(2) Calculate top threshold in each subband, i.e., calculate $n_k = \log_2 \lfloor s_k \rfloor$, $s_k = \max_{(i,j)\in SB_k} |q_{i,j}|$. Order subbands SB_k according to the progressive-resolution zig–zag scan, as shown in Figure 1.8.
(3) Encode $n_1, n_2, \ldots, n_M$.
(4) Initialize LIS with top left corner coordinates of subbands in order of $SB_1, SB_2, \ldots, SB_M$.
(5) Let $k = 1$.
  (a) Let $n = n_k$. If $n > 0$,
    i. Do single pass of set partition coding of SB_k at threshold 2^n until all single significant elements are located.
    ii. Encode significant single elements with (no more than) $n + 1$ bits.
  (b) Move to next subband, i.e., let $k = k + 1$.
  (c) If $k \leq M$, go to Step 5a.
  (d) If $k > M$, decrement n, i.e., let $n = n - 1$ and reset $k = 1$.
  (e) if $n > 0$, return to Step 5(a)i; if $n = 0$, stop.

The coding passes in Step 5(a)i may be accomplished either by direct recursive quadrature splitting (Step 2, Algorithm 2.1, Part I)

or octave band partitioning and recursive quadrature splitting of the SPECK (Set Partitioning Embedded bloCK) Algorithm [9] (Algorithm 2.6, Part I). The initial $\mathcal{S}$ sets for the subbands would be the 2×2 blocks in their top left corners. Instead of maintaining an LSP (List of Significant Points) and executing a refinement pass, the single elements are encoded immediately and written to the codestream when found to be significant, as in Step 5(a)ii in Algorithm 1.2.

As before, if there are one or more subbands with empty layers at the given bit plane, the codestream will lose its progressiveness in resolution. Then higher bit plane bits from higher resolution subbands will reside among those from the lower resolution ones. Therefore a truncated codestream cannot contain purely bits from a certain scale.

Another way to obtain a bit-embedded codestream is to keep the same threshold in the zig–zag scan across the subbands and maintain a list of significant elements, called the List of Significant Points (LSP), initially empty. The starting threshold is the largest among all the subbands and is almost always that of the lowest frequency subband. When the current threshold exceeds the maximal threshold of a given subband, that subband is skipped. For the threshold 2^n, we code a subband by the same quadri-section procedure described above until all the single elements are located. The coordinates of the significant single elements are added to the LSP. The next subband in the scan order is visited and encoded in the same way, until all the subbands have been visited at the threshold 2^n. The bits in the same bit plane n from coefficients found significant at higher thresholds are then read from the LSP and written to the codestream. The threshold is then lowered by a factor of 2 to 2^{n-1} and the scan returns to its first subband to test its sets in the LIS left after the just-completed threshold pass. The LIS sets within a subband decrease in size from top to bottom and are visited in reverse order from bottom to top, so that the smallest sets are tested first. In this way, we obtain a bit-embedded codestream, because code bits from the higher bit planes always precede those from the lower bit planes.

Example 1.2 Reconstructions of different quality from a bit-embedded codestream. The wavelet transform of the Goldhill

image is encoded, but this time using a quality scalable algorithm giving a bit-embedded codestream of size 65,536 bytes (rate of 2.00 bits/pixel). (The transform coefficients are not scaled and are truncated to integers, if necessary.) The full codestream and its truncation to sizes of 32,768 bytes (1.00 bpp), 16,384 bytes (0.50 bpp), and 8,192 bytes (0.25 bpp) are then decoded, de-quantized, and inverse wavelet transformed. The reconstructions and their PSNR's are shown in Figure 1.13.

Fig. 1.13 Reconstructions of Goldhill from same codestream by a quality scalable coding method.

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

1.5.3 Octave Band Partitioning

Instead of a predetermined scan order of subbands, we could use octave band partitioning of the wavelet transform, as is done in conjunction with the SPECK algorithm described in the last section. We would need only to calculate and send the maximum threshold for the entire transform and have the potential to locate larger insignificant sets. Recalling briefly this procedure, the initial $\mathcal{S}$ set is the lowest frequency subband and the remainder of the transform comprises the initial $\mathcal{I}$ set. These sets are depicted in Figure 1.14. The set $\mathcal{S}$ is tested for significance at the current threshold, and if significant, is coded as above using recursive quadrisection. If not significant, the $\mathcal{I}$ set is then tested for significance, and, if significant, it is partitioned into three $\mathcal{S}$ sets and an $\mathcal{S}$ set, as shown in Figure 1.14. These three $\mathcal{S}$ sets are exactly the three subbands that complete the next higher resolution scale. If $\mathcal{I}$ is not significant, then the threshold is lowered by a factor of 2, the octave band partitioning is re-initialized, and only the resident LIS entries left from the last threshold are tested when the $\mathcal{S}$ set for a subband becomes significant. As the threshold becomes lower, more sets of three subbands that comprise the next higher resolution are encoded. Therefore, this scheme of octave band partitioning is naturally progressive in resolution.

A binary digit is sent to the codestream after every significance test. An insignificant subband then is indicated by a single "0" at the given threshold. So the maximal threshold of a subband is conveyed by a succession of "0"s and a single "1" when it becomes significant. Sometimes, a "0" indicating insignificance is shared among insignificant subbands, as in an $\mathcal{I}$ set. The previous method sent the

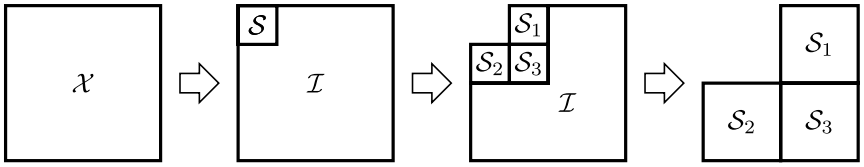

Fig. 1.14 Partitioning of image $\mathcal{X}$ into sets $\mathcal{S}$ and $\mathcal{I}$, and subsequent partitioning of set $\mathcal{I}$.

maximal threshold of every subband to the codestream. Although it did not require another bit to skip insignificant subbands, more bits are needed for conveying the maximal threshold of every subband than in the method of octave band partitioning.

Example 1.3 Effect of Mask Coding. As an example, we performed one experiment to compare an uncoded and coded same step size quantization of the wavelet transform (4 levels, 9/7 biorthogonal filters) of the Lena (512×512) image. The quantization was done according to Equation (1.1) with the same step size (Δ) for all subbands and the coded one used fixed Huffman coding of the masks only. We used octave band partitioning for determining the scanning order of the subbands. No overhead marker bits were in the codestream, as there was no attempt to produce progressiveness in quality. As expected, the identical uncoded and coded reconstructions showed the same PSNR of 37.07 dB. However, coding the masks resulted in an average bit rate of 0.500 bpp, whereas without coding the average bit rate was 0.531 bpp, an increase of 6%. These reconstructions are displayed in Figure 1.15.

Fig. 1.15 Uncoded (left) and coded (right) reconstructions of 512×512 lena image with identical quantizer step sizes. Both have PSNR = 37.07 dB; rate of uncoded (left) = 0.531 bpp and rate of coded (right) is 0.500 bpp.

1.5.4 Direct Bit-embedded Coding Methods

The utilization of SPECK just described achieves complete coding of a lower resolution level before the next higher one. As explained in Part I, SPECK realizes bit-embedded coding when the threshold is held at the same value across subbands and through all resolution levels before it is lowered by a factor of 2 for the next pass through the LIS. Such a procedure violates resolution progressiveness, because bits from different resolution levels are intermingled in the codestream. The benefit is the approximate realization of the optimal rate allocation that prescribes that the most important of the code bits, that is, the ones contributing to the largest reduction in distortion, are sent to the codestream. The coding can then be terminated when the bit budget is reached with the confidence the lowest distortion for this budget is achieved for the corresponding reconstruction. One does not need to do any codestream re-organization or Lagrangian optimization, as required when certain portions of the wavelet transform are independently and completely coded.

The coding procedure of SPECK (or SPIHT (Set Partitioning in Hierarchical Trees [15]) will produce optimal rate allocation. The coding order is from highest to lowest bit plane and significance bits before refinement bits in the same bit plane. Distortion reduction for bits coded in the nth bit plane, being proportional to 2^n, decreases exponentially with this order. Furthermore, the significance bits of coefficients revealed significant first at bit plane n always reduces the distortion more[4] than that of same bit plane refinement bits of previously significant coefficients.

1.5.5 Lossless Coding of Quantizer Levels with Adaptive Thresholds

The coding paradigm remains the same, except now we decide to use adaptive thresholds rather than fixed ones. The coding mechanism using adaptive thresholds was described in the AGP (Alphabet and Group Partitioning) method Part I. We take the subbands as the blocks

[4] The magnitude of the error decreases by a factor of 3 for midpoint reconstruction.

to be coded and test them starting as usual with the lowest frequency subband. Briefly, the alphabet is partitioned by symbol grouping and representing each coefficient by its group index and symbol index. A typical alphabet partitioning scheme appears in Table 2.5 of Part I. The group indices are coded via set partitioning. The symbol indices are represented by their magnitude range and sign in raw binary.

The subbands are now encoded by the adaptive-threshold, quadrature splitting procedure described in Algorithm 2.5 of Part I. The values to be coded are the group indices. The initial thresholds of the subbands are the maxima of their respective group indices. These thresholds are encoded and written to the codestream. Briefly, starting with the lowest frequency subband, the subband to be encoded is split into four nearly equal size sets. The maxima of these sets are compared to the subband maximum. Those with lower maxima are labeled with "0" and those equal are labeled with "1", thereby creating a four bit mask that is sent to the codestream. The "1"-labeled (significant) sets are again split into four quadrants which are again labeled with "0" or "1," depending whether the quadrant maximum is lower or equal to the subband maximum. This recursion of splitting just the "1"-labeled sets continues until the quadrants become single elements. Only the "0"-labeled elements are encoded and written to the codestream, because the "1"-labeled elements are known to be equal to the threshold. The maxima of the "0"-labeled (insignificant) quadrants remaining comprise new lower testing thresholds. The new threshold for the next pass is the maximum among these thresholds, which is the largest group index magnitude among these remaining insignificant sets. These insignificant sets are now recursively split and tested against this new lower threshold in the same way until all coefficients in the subband are encoded. These insignificant sets are always visited in order of smallest to largest, so that significant single elements are located as early as possible. When "0"-labeled sets are 2×2 blocks having small maxima, instead of splitting, they can be coded together using an extension alphabet to obtain coding gains. Once a subband is encoded, the same processing moves to the nonvisited subband with the largest maximum group index magnitude greater than zero. A subband

with such maximum equalling 0 has all its elements 0 in value. The 0 in the four-bit mask sent to the codestream conveys this information.

Algorithm 1.3. Wavelet transform coding by set partitioning with adaptive thresholds.

(1) Start with M subbands $\{SB_k\}$ and represent quantized coefficients $\hat{c}_{i,j}$ with group index, symbol index pairs $(g_{i,j}, r_{i,j})$, where $r_{i,j}$ includes sign.
(2) Calculate maximum group index in each subband and order subbands by decreasing maxima. That is, let $s_k = \max_{(i,j)\in SB_k} g_{i,j}$. Order subbands SB_k either by progressive resolution zig–zag scan or by decreasing maxima, $s_1 \geq s_2 \geq \cdots \geq s_M$.
(3) Encode $s_1, s_2, \ldots, s_M$.
(4) For $k = 1, 2, \ldots, M$, if $s_k > 0$, do set partitioning coding of SB_k using Algorithm 2.5 in Part I.

The steps of this coding procedure are delineated in Algorithm 1.3. There, the subbands are ordered by decreasing maximum value (group index). For images, SB_1 is almost always the lowest frequency subband and the ordering by maximum decreasing value is often progressive in resolution. But for other sources, such as audio, the largest maximum may belong to another subband. This prior ordering, whether by maximum value or by increasing resolution, simplifies the step of moving to the next subband to be coded. Moreover, for ordering by decreasing maximum, once we come to a subband with maximum value of 0, we can stop the coding. For some applications, it might be advantageous to produce a value-progressive codestream. Achieving such property requires crossing subbands to code an LIS with the next lower threshold, instead of executing the next pass at the threshold that is the largest maximum of the LIS sets of the current subband. This means that a separate LIS must be maintained for every subband. Algorithm 1.4 presents the steps of this value-progressive coding method.

Algorithm 1.4. Value-progressive, wavelet transform coding by set partitioning with adaptive thresholds.

(1) Start with M subbands $\{SB_k\}$ and represent quantized coefficients $\hat{c}_{i,j}$ with group index, symbol index pairs $(g_{i,j}, r_{i,j})$, where $r_{i,j}$ includes sign.
(2) Calculate maximum group index in each subband and order subbands by decreasing maxima. That is, let $s_k = \max_{(i,j)\in SB_k} g_{i,j}$ and order subbands SB_k so that $s_1 \geq s_2 \geq \cdots \geq s_M$.
(3) For every $k = 1, 2, \ldots, M$,
    (a) Encode s_k.
    (b) Set initial thresholds $t_k = s_k$.
    (c) Initialize a list LIS_k with top left corner coordinates of subband $\{SB_k\}$.
(4) Let $k = 1$.
    (a) If $t_k > 0$, do single pass of set partition coding of SB_k in Step 2 of Algorithm 2.5, Part I.
    (b) If $t_k > 0$ and there are multiple element sets on LIS_k,
        i. Reset t_k equal to largest maximum among the sets of LIS_k;
        ii. Encode t_k.
        iii. Move to subband with largest current threshold, i.e., SB_{k^*}, where $k^* = \arg\max_{\ell=1,2,\ldots,k,k+1} t_\ell$.
        iv. Set $k = k^*$ and return to Step 4a.
    (c) If $t_k > 0$ and all the LIS_k sets comprise single elements, encode them with no more than $\lfloor \log_2 t_k \rfloor + 1$ bits, write to codestream buffer, and stop.
    (d) If $t_k = 0$, stop.

The new lower thresholds in either mode, value-progressive or not, are encoded efficiently using differential coding. The thresholds are

always decreasing and the successive differences are small. Recall that these thresholds are group indices, the largest of which is under 32. A unary code for these differences is a good choice.

Modest gains in coding efficiency may be obtained through entropy coding of significant coefficients either singly or in 2×2 blocks when the threshold is small. Recall that a small threshold allows extension alphabets of small size. In the quadri-section steps (see Step 4a in Algorithm 1.4), four-bit binary masks are generated to convey whether the quadrant maxima are below or equal to the testing threshold. These masks are encoded with a fixed 15-symbol Huffman code. We carried out the experiment of encoding the same quantized wavelet transform of the Lena image, but now used Algorithm 1.3 with entropy coding of masks and small-threshold 2×2 blocks. The resulting code rate was 0.479 bits/pixel for the same 37.07 dB PSNR. Therefore, even with the extra overhead of coding the thresholds, the combination of adaptive thresholds and simple entropy coding of these blocks saved about 4% in rate or file size.

1.5.6 Tree-Block Coding

In Section 2, Part I we introduced a partition of the wavelet transform into spatial orientation trees (SOT's) and formed these trees into blocks, called *tree-blocks*, as depicted in Figure 1.16. We illustrated the order of coding these tree-blocks (Algorithm 2.3, Part I) by the fixed-threshold, quadrature splitting method (Algorithm 2.1, Part I). However, any block coding method can be chosen to code the tree-blocks. In fact, the adaptive-threshold set partition coding algorithms can be used for coding the tree-blocks, instead of the subbands taken as blocks. All it requires is the substitution of tree-blocks for subbands in the input to these algorithms. More specifically, suppose we denote the tree-blocks as $S_1, S_2, \ldots, S_T$, where T is the number of these blocks, which is usually equal to number of coefficients in the lowest frequency subband. Then in Algorithms 1.3 and 1.4, we replace subband SB_k with S_k and subband count M with tree-block count T. Again, we performed the analogous experiment of coding the same quantized wavelet transform of the Lena image, but now coded tree-blocks instead of subbands.

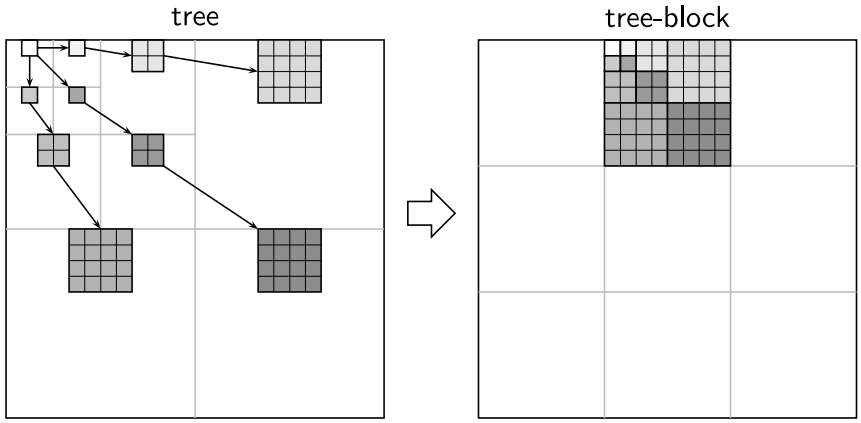

Fig. 1.16 Rearranging a spatial orientation tree into a tree-block placed in the position of the image region it represents.

The code rate turned out to be 0.485 bpp, which is slightly higher than the 0.479 bpp resulting from coding subbands as blocks. There are two factors contributing to the superiority of the subband block coding. First, the interdependence of coefficients in subbands may be stronger than the interdependence in the SOT's. Second, the subbands constitute mainly bigger blocks, so that larger insignificant sets and zero blocks are located. In a 5-level wavelet decomposition of a 512×512 image, there are 256 tree-blocks each with 1024 coefficients, but only 16 subbands having from 256 to 65,536 coefficients. However, coding in tree-blocks provides natural random access to regions directly in the codestream, since each tree-block corresponds to a different image region.

1.5.7 Coding of Subband Subblocks

Coding blocks that are contained within subbands, unlike direct coding of blocks of source samples, will not produce blocking artifacts at low code rates, because the filtering in the inverse transform performs a weighted average of the reconstructed coefficients that crosses block boundaries.[5] This realization led to systems that divide the subbands

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

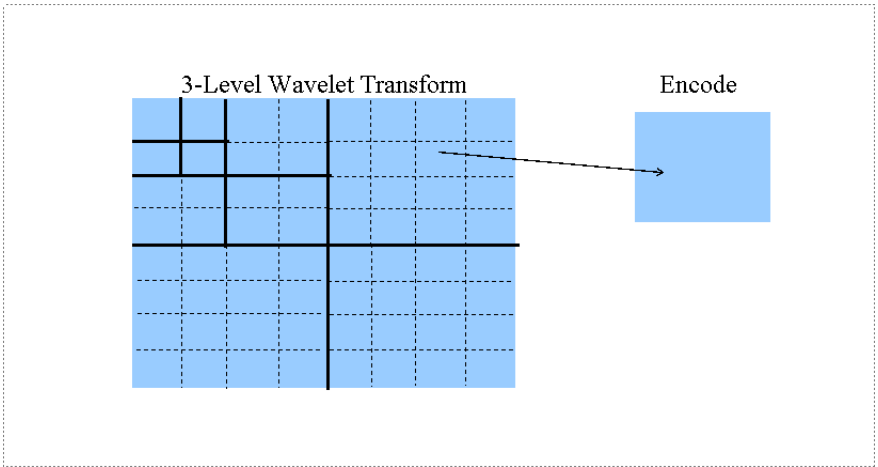

Fig. 1.17 Division of wavelet subbands into subblocks. Note subbands of coarsest level are too small to be divided.

into square blocks, typically 32×32 or 64×64 in their dimensions, and that code these blocks. The subbands of a 3-level wavelet transform divided into these so-called *subblocks* are depicted in Figure 1.17. The smallest subbands are not divided if their sizes are comparable to that of the subblock. The figure therefore shows no division of the subbands at the coarsest level.

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

The coding of subblocks of subbands entails extra codestream overhead in the form of the initial (largest) threshold for every subblock and

markers to delineate boundaries between subblocks.[6] The thresholds are either group indices or bit-plane maxima and can be compressed, if necessary. For example, consider again the 5-level wavelet decomposition of a 512×512 image, for a subblock size of 32×32. There are 64 subblocks in each of the three level-1 subbands, 16 subblocks in each of the three level-2 subbands, 4 subblocks in each of the level-3 subbands, 1 subblock in each of the three level-4 subbands, and 1 subblock making up remaining subbands for a total of 256 blocks. Assuming that the thresholds are maximal bit-plane levels, no threshold can exceed 13 for an original 8-bit image. Therefore, four bits for each threshold gives an overhead of 1024 bits or 0.004 bits per pixel. At very low bit rates, such overhead may be problematical. These thresholds, especially those associated with subblocks within a subband, are statistically dependent and can be encoded together to save overhead rate.

1.5.8 Coding the Initial Thresholds

The initial thresholds that are bit-plane maxima of subblocks contained within a subband are often not too different, especially for the lower level, high frequency subbands. Therefore, it is often advantageous to encode the initial thresholds in each level-1 subband separately and group the initial thresholds in the remaining higher level subbands together. For the 5-level wavelet decomposition of the 512×512 image, there would be coded 4 8×8 arrays of initial thresholds, 3 belonging to the level-1 subbands and 1 belonging to all the remaining subbands of the higher levels.

Let us assume that these initial subblock thresholds are a square array of integers $n[i,j]$, $i,j = 1,2,\ldots,K$, representing the bit plane maxima, the $n_{\max}$'s of these subblocks. Let $n_M = \max_{i,j=1,2,\ldots,K} n[i,j]$ be the maximum of these integers. Because the thresholds within a block are dependent, they are likely to be close in value to n_M. Because small values require fewer code bits than large ones, it will be more efficient to encode the differences from the maximum, that is, $\bar{n}[i,j] = n_M - n[i,j]$. In the array of numbers $\{\bar{n}[i,j]\}$, the smaller

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

numbers are the most probable, so they fit the favorable scenario of coding by recursive quadri-section procedure with adaptive thresholds, which is Algorithm 2.5 in Part I. We give examples below to illustrate this approach.

Example 1.4. Example of coding a 4×4 array of thresholds. Suppose the initial bit-plane thresholds are held in the following array:

$$\begin{array}{cccc} 2 & 2 & 3 & 3 \\ 3 & 2 & 2 & 3 \\ 3 & 4 & 4 & 3 \\ 3 & 3 & 2 & 3 \end{array}$$

Subtracting each element from $n_M = 4$, the array to be coded is

$$\begin{array}{cccc} 2 & 2 & 1 & 1 \\ 1 & 2 & 2 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 2 & 1 \end{array}$$

The bits to represent the block maximum of $n_M = 4$ are not counted here, because they are already sent as overhead anyway to assist entropy coding of the coefficients in the block. The overall difference array maximum $\bar{n}_{\max} = 2$. Splitting the array into four quadrants, the quadrant maxima are 2, 2, 1, and 2 in raster order. The 4-bit significance mask to be transmitted is therefore 1,1,0,1. Splitting the first "1"-labeled quadrant gives 4 single elements with significance bits 1,1,0,1. Likewise, splitting of the next two "1"-labeled quadrants gives significance values of 0,0,1,0 and 0,0,1,0, respectively. A "1" bit conveys that the element value is $\bar{n}_{\max} = 2$. The "0"-bits attached to the seven single elements mean that their values are 0 or 1, so that they can each be encoded naturally with a single bit.

Returning to the original split, the insignificant third quadrant has its $\bar{n}_{\max} = 1$. The quad-split yields the significance pattern 1,0,1,1, whose bits are exactly its values, which is always the case when the group maximum is 1.

The full array maximum $\bar{n}_{\max} = 2$ must also be encoded. Because it cannot exceed $n_M = 4$, 3 bits suffice to represent this value of 2. When

split into four quadrants, only the maximum of the one "0"-labeled quadrant is unknown to the decoder. Therefore, its maximum of 1 is encoded with 1 bit, because it is known to be less than 2.

Counting the total number of code bits, we have 5 4-bit masks, 7 value bits, and 4 group maxima bits for a total of 31 bits. Compared to raw coding of the array of 16 thresholds, which consumes 3 bits for each, since we know the maximum is 4, we have saved 48−31=17 bits or about 35%.

Another way to represent the original threshold array is by successive differences in a linear or zig–zag scan. For example, a zig–zag scan from the top left corner produces the zig–zag and successive difference sequences below:

$$\begin{array}{cccccccccccccccc} 2 & 2 & 3 & 3 & 2 & 3 & 3 & 2 & 4 & 3 & 3 & 4 & 3 & 3 & 2 & 3 \\ & 0 & -1 & 0 & 1 & -1 & 0 & 1 & -2 & 1 & 0 & -1 & 1 & 0 & 1 & -1 \end{array}$$

Probably the simplest and most effective way to code the successive difference sequence is by a unary (comma) code.[7] But now there are negative integers, so we must map these numbers to positive ones. For an integer k,

$$\begin{array}{ll} k \to 2k & \text{if } k \geq 0 \\ k \to 2|k| - 1 & \text{if } k < 0. \end{array}$$

The successive difference sequence maps to

$$0 \quad 1 \quad 0 \quad 2 \quad 1 \quad 0 \quad 2 \quad 3 \quad 2 \quad 0 \quad 1 \quad 2 \quad 0 \quad 2 \quad 1$$

The unary code for this sequence consumes only 32 bits. We also need 3 more bits to represent the leading 2 for a total of 35 bits, four more than the recursive quadrature splitting method.

The recursive quadri-section method is especially efficient when all the thresholds in an array are equal. For example, suppose now all the elements in the initial 4×4 initial threshold array equal 4. The difference from maximum array has all elements of 0. So the difference array maximum of 0 represents the entire array. It requires only 3 bits to

[7] The unary code for a non-negative integer is the same number of 0s followed by a 1, which acts as a comma. For example, $0 \to 1$, $1 \to 01$, $2 \to 001$, and so forth.

represent this 0 maximum and hence the original array of 16 thresholds of 4.

The above recursive quadri-section procedure for coding the thresholds usually consumes fewer bits than methods using a linear scan, because it exploits better the two-dimensional statistical dependencies. It also is quite simple, in that it requires just comparisons and set splitting and does not use entropy coding. The masks when uncoded account for more than one-half of the code bits. Simple entropy coding, e.g., fixed Huffman coding with 15 symbols, might be employed beneficially to reduce the mask bit rate.

1.5.9 The SBHP Method

Now that we have described dividing subbands into subblocks and the means to code their initial thresholds, we now explain two particular methods to code the subblocks: SBHP and JPEG2000. First, we consider the simpler of the two methods, called SBHP for Subband Block Hierarchical Partitioning [3]. The coding method of SBHP is SPECK, initialized by an $\mathcal{S}$ set consisting of the 2×2 array of coefficients in the top left corner of the subblock and with the $\mathcal{I}$ as the remainder of the subblock, as illustrated in Figure 1.14. When the threshold is lowered such that the $\mathcal{I}$ set becomes significant, the $\mathcal{I}$ set is split into three $\mathcal{S}$ sets adjoining the existing $\mathcal{S}$ and another $\mathcal{I}$ set for the remainder. Recall that the $\mathcal{S}$ sets are encoded by recursive quadrature splitting according to significance tests with fixed decreasing power of 2 thresholds. For each subblock, the method uses an LIS list and an LSP list to store coordinates of insignificant sets (including singleton sets) and significant coefficients, respectively. However, the subblock is fully encoded from the top threshold down to the threshold associated either with some large bit rate for lossy coding or through the 2^0 bit plane for lossless coding. The lists are then cleared and re-initialized for the next subblock to be coded. All the subblocks in every subband are independently coded to the same lossy bit rate or to its lossless bit rate in the same way. We pass through the subbands in the progressive resolution order and through subblocks within a subband in raster order. We obtain a sequence of codestreams, one for each subblock, where each

codestream is bit-embedded, but when assembled together into one composite codestream is not embedded in any sense. We may reorganize this composite codestream by interleaving the subcodestreams at the same bit-plane level to obtain an embedded codestream. The effect is the same as holding the threshold in the current pass across the subblocks and subbands, before lowering the threshold for the next pass. We can exercise rate control by cutting the reorganized codestream at the desired file size.

Another way to control the rate is the method to be described later in more detail in Section 1.7. By this method, a series of increasing slope parameters $\lambda_1, \lambda_2, \ldots, \lambda_J$ are specified and every subcodestream is cut, so that the magnitude of its distortion-rate slope matches each parameter in turn until the desired total number of bits among the codestreams is reached. Because SPECK codes subblocks in progressive bit plane order, approximate, but fairly accurate calculations of distortion changes with rate can be made very fast and in-line by counting the number of newly significant bits at each threshold. Then the codestream can be reorganized for quality and/or resolution scalability.

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

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

For each context, a simple, fixed Huffman code is used for coding the 15 possible mask patterns. Recall that all-zero cannot occur. The label of a pattern is the decimal number corresponding to the four binary symbols read in raster order, left-to-right and top-to-bottom. Table 1.1 contains Huffman codes for each context.

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

Table 1.1 Huffman codes for 4-bit masks in SBHP.

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

1.5.10 JPEG2000 Coding

The framework for the entropy coding engine of the JPEG2000 coding system is the same as that of SBHP. JPEG2000 encodes subblocks of subbands independently and visits the subbands and subblocks in the same order as previously described. The entropy coding engine is called EBCOT for Embedded Block Coding with Optimized Truncation and is the brainchild of David S. Taubman [18]. In the parlance of JPEG2000, the subblocks are called *code-blocks*. The method of coding is context-based, binary arithmetic coding of bit planes from top (most significant) to bottom (least significant), as will be described.

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

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

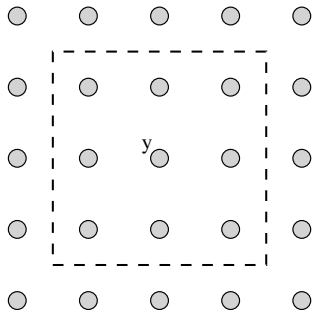

Fig. 1.18 Eight pixel neighborhood of pixel y for context formation.

Each of these passes encodes a different set of bits and together these passes encode the bits of all pixels in the bit plane. That is why these passes are called *fractional bit-plane coding*. The specific order above is chosen to code first those bits that are likely to give the greatest distortion reductions. The first pass, the SPP, visits only (previously) insignificant pixels with a "preferred" neighborhood, where at least one of its eight nearest neighbors is (previously) significant. An 8-pixel neighborhood of the current pixel y is depicted in Figure 1.18. If a "1" is encountered, changing the state from insignificant to significant, then its associated sign is encoded. The SPP pass is skipped for the first ($n_{\max}$) bit plane, because nothing is yet significant, meaning that there is no preferred neighborhood.

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

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

We explain next how we encode the bits encountered in these three passes. The short explanation is that each pass has associated with it sets of contexts that are functions of significance patterns for the 8-pixel neighborhood of the bit to be encoded. Probability estimates of the bit

values (0 or 1) given each context are accumulated as encoding proceeds and the bit is arithmetically encoded using these probability estimates. Of course, the devil is in the details, on which we shall elaborate.

Significance Coding — Normal Mode. We now describe the normal mode of significance coding that is used exclusively in the SPP and partially in the CUP passes. As mentioned, the coding method is context-based, adaptive arithmetic coding of the binary symbols within a bit plane. The context template is the 8-pixel neighborhood of Figure 1.18, where the number of possible (binary) significance patterns is $2^8 = 256$. So-called context quantization or reduction is necessary for reasons of lowering computational complexity and collection of sufficient statistical data (to avoid so-called *context dilution*) for each pattern. A reduction to nine contexts is achieved by counting numbers of significant pixels in the horizontal, vertical, and diagonal neighbors. Table 1.2 lists the nine contexts for code-blocks in LL and LH subbands.

The context label descends in value from strongest to weakest horizontal neighbor significance. The HL subbands show strong dependence in the vertical direction, so their contexts, shown in Table 1.3, are numbered with the reversal of the Sum H and Sum V columns in Table 1.2.

The HH subbands show diagonal dependencies, so the contexts for code-blocks in these subbands are numbered by descending diagonal significance, accordingly in Table 1.4.

Table 1.2 Contexts for Code-Blocks in LL and LH subbands.

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

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

Table 1.3 Contexts for code-blocks in HL subbands.

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

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

Table 1.4 Contexts for code-blocks in HH subbands.

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

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

Sign Coding. In the SPP or CUP of a bit plane, whenever a "1" is encountered, changing the state from insignificant (0) to significant (1), the pixel value's sign is also encoded. The context template for sign coding consists just of the two horizontal and two vertical neighbors in Figure 1.18. An intermediate value characterizes the significance and sign pattern either of the horizontal or vertical neighbors. This intermediate value, denoted as $\bar{\chi}$, takes values as follows:

$\bar{\chi} = 1$ when one neighbor is insignificant and the other significant and positive, or both neighbors significant and positive;
$\bar{\chi} = -1$ when one neighbor is insignificant and the other significant and negative, or both neighbors significant and negative;
$\bar{\chi} = 0$ when both neighbors are significant and opposite in sign, or both neighbors insignificant.

Table 1.5 Contexts for code-blocks for sign coding.

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

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

Since the horizontal and vertical neighbor pairs have three intermediate values each, there are $3^2 = 9$ possible patterns or context values. We expect that the conditional distribution of the sign to be coded given a particular pattern to be identical to the negative of that sign given the sign complementary pattern. Therefore, we group a pattern and its sign complementary pattern into a single context, thereby reducing to 5 distinct sign contexts. A flipping factor χ^{flip} =1 or −1 is used to change the sign of the bit to be coded according to the pattern or its complement, respectively. The contexts, their labels, and flipping factors are exhibited in Table 1.5. Denoting the sign to be coded as χ, the coded binary symbol κ_{sign} for the given context is

$$\begin{aligned} \kappa_{\text{sign}} &= 0 \quad \text{if} \quad \chi \cdot \chi^{\text{flip}} = 1 \\ \kappa_{\text{sign}} &= 1 \quad \text{if} \quad \chi \cdot \chi^{\text{flip}} = -1. \end{aligned} \tag{1.4}$$

Magnitude Refinement Coding. Only previously significant pixels are visited in the MRP (Magnitude Refinement Pass) through a given bit plane. The associated bits refine these pixel values. The coded bit equals the actual bit of the pixel in the bit plane. We define three contexts in the context template for magnitude refinement coding. The value of each context is determined by whether or not the bit to be coded is the first refinement bit and the sum of the significance states of the 8 pixels in the template. We distinguish three contexts as follows:

— Not the first refinement bit.
— First refinement bit and sum of 8-pixel significance states is 0

Table 1.6 Contexts for magnitude refinement coding

Context label	Pixel's first refinement bit?	Sum H + Sum V + Sum D
17	No	
16	Yes	> 0
15	Yes	0

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

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

We summarize the context definitions and labels in Table 1.6.

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

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

run mode. However, when we encounter pixels with 1s in the scan of the bit plane, they become significant and make the neighborhoods of succeeding pixels "preferred," thereby triggering normal mode coding.

The preceding coding modes were not dependent on the scan pattern through the code-block. However, in order to describe the details of run mode coding, we need to specify this scan pattern. The code-block is divided into horizontal stripes of four rows per stripe. Within each stripe, the scan proceeds down the four-pixel columns from the leftmost to the rightmost column. After the last pixel of the rightmost column is visited, the scan moves to the leftmost column of the next stripe below.[8] This scan pattern is illustrated in Figure 1.19.

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

(1) Four consecutive pixels in the scan shown in Figure 1.19 must currently be insignificant.

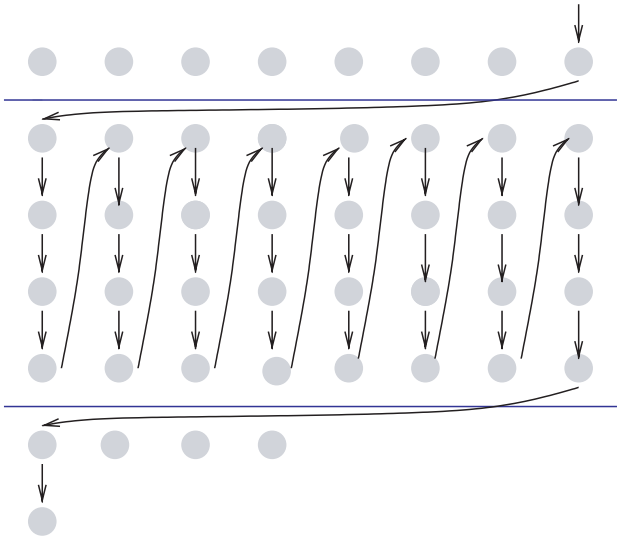

Fig. 1.19 Column-wise scan pattern of stripes within a code-block.

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

(2) All four of these pixels must have insignificant neighborhoods.
(3) The four consecutive pixels must be aligned with a column of a stripe.

In run mode, a binary "run interruption" symbol is coded to indicate whether (0) or not (1) all four pixels remain insignificant in the current bit plane. This symbol is coded using its natural context of four consecutive pixels remaining insignificant or not. JPEG2000 labels this context of four consecutive pixels as number 9.[9] A pixel becomes significant when a "1" is read in the current bit plane. We count the number r of "0"s until a "1" is reached and encode this run number followed by the sign of the pixel associated with the "1" terminating the run. The run-length r ranges from 0 to 3, because it has already been signified that there is at least one "1". The run-length r is close to uniform in distribution, so its values are encoded with 2 bits, the most significant one sent first to the codestream.[10] The significances of the remaining samples of the four, if any, are encoded using the normal mode. Coding continues in normal mode until conditions to enter run mode are again encountered.

Example 1.5 (Run Mode Coding Example). Suppose we are in run mode and the run interruption symbol "1" signifies that one or more of the four pixels in a column has become significant. Let the bit-plane bits of the four pixels be 0010. Since $r = 2$, 1 and then 0 are sent to the codestream. Furthermore, the sign associated with the 1 following 00 is coded in the usual way and sent to the codestream. The next bit of 0 is then coded using the normal mode.

The operation of JPEG2000's arithmetic coder, which is the MQ-Coder, has purposely not been described here. We refer the reader to JPEG2000 textbooks [1, 19] for such material. Clearly, JPEG2000

[9] Some textbooks count this context as two states, significant and insignificant four consecutive pixels.
[10] In the standard, the MQ-coder is set to a nonadaptive mode with a fixed uniform probability distribution to code the run lengths.

coding is heavily dependent on arithmetic coding, much more so than set partitioning coders, and for that reason is considered to be computationally complex. The payoff is excellent compression efficiency. Due to its bit-plane coding from most to least significant, the resulting codestream of every code-block is bit-embedded. Also, the coding of subblocks of subbands independently allows resolution scalability and random access to spatial regions directly within the codestream. Despite its excellent performance and range of scalability options, some applications, especially those involving hardware implementation constrained in size and power consumption, cannot tolerate the high complexity. This complexity is one of the reasons that JPEG2000 has not seen such wide adoption thus far.

Scalable Lossy Coding in JPEG2000. The JPEG2000 method of coding of subblocks produces bit-embedded subcodestreams. Each subcodestream has a distortion versus rate characteristic, because it is associated with a different part of the wavelet transform. In order to achieve the best lossy reconstruction at a given rate, each subcodestream must be cut to its optimal size, so that the total of the sizes equals the size corresponding to the given rate as closely as possible. Sets of subcodestream sizes or cutting points corresponding to a number of rates can be calculated during encoding. The codestream is then considered to be built up in *layers*. A method to determine the optimal cutting points for a number of layers will be described later in Section 1.7. Briefly, according to this method, a series of increasing slope parameters $\lambda_1, \lambda_2, \ldots, \lambda_J$ are specified and every subcodestream is cut, so that the magnitude of its distortion-rate slope matches each parameter in turn until the desired total number of bits among the codestreams is reached.

1.5.11 The Embedded Zero-Block Coder (EZBC)

The arithmetic coding that is mandatory in JPEG2000 can be applied optionally in the embedded block coders, such as SPECK and SBHP, to encode sign and refinement bits. Furthermore, both SPECK and SBHP used simple, fixed Huffman coding of the 4-bit masks that signify the significance state of the four quadrants obtained in splitting

a significant set. Adaptive versions of Huffman or arithmetic coding, either unconditioned or conditioned on context, can also be used for coding these masks. Another method called EZBC (Embedded Zero Block Coder) [7, 8] that we describe now does use these more complex forms of entropy coding.

The EZBC coder utilizes SPECK's quadri-section coding of full wavelet subbands and visits them in the zig–zag order as shown in Figure 1.8. The transition from one subband to the next can be either at the same threshold or after passing through all thresholds from top to bottom. The former produces a bit-embedded composite codestream and the latter a resolution-progressive one with separate bit-embedded subband codestreams. What distinguishes EZBC from normal SPECK is the entropy coding of the significance map, sign bits, and refinement bits. The original SPECK used arithmetic coding of the 4-bit masks, but did no entropy coding of sign and refinement bits. EZBC entropy encodes the sign and refinement bits in the manner of JPEG2000. It also performs entropy coding of the significance decision bits in the masks adaptively using a context template consisting of significance bits in neighboring masks and parent subbands. We shall describe next this coding in more detail.

The coding of bits in the masks is best illustrated by an example. Consider the case shown in Figure 1.20 of two levels of splitting of a 4×4 block of wavelet coefficients. (In Part I is a figure of three levels of splitting of an 8×8 block of wavelet coefficients.) Shown are the successively smaller quadrants created by the splittings and the associated quadtree and its code. We have created virtual splits, marked by dashed lines, of the insignificant quadrants 2 and 3 resulting from the first level split. Likewise, in the quadtree, we have created corresponding virtual nodes. The four "0"s in these quadrants are associated with these virtual nodes and are not part of the significance map code, because they are redundant. Once a set is deemed insignificant, all its subsets are known to be insignificant. However, the inclusion of these virtual splits allow us to define a context template of the 8 nearest neighbor quadtree nodes. The context is the significance bit pattern of these eight nodes. We show in Figure 1.20 a node or significance bit to be coded in solid shading of its set and its 8-neighbor context in

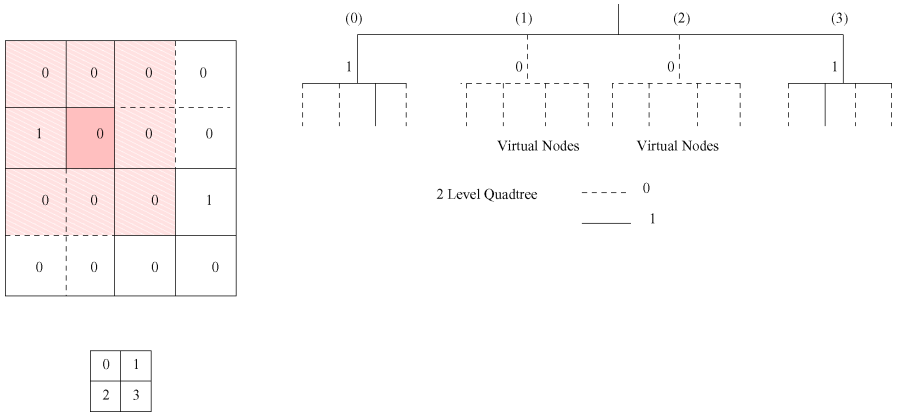

Fig. 1.20 Two levels of quadrisection of 4×4 block and associated quadtree, including virtual nodes.

lighter pattern shading. Every nonvirtual node at this second level of the quadtree is coded in this manner. In fact, the (nonvirtual) nodes in each level of the quadtree, corresponding to the significance decision bits in each level of quadrisection, are coded in this manner. Nodes associated with sets near edges of the block will have some nearest neighbors outside the subband. Such external neighbor nodes are considered always to be "0". Subband sizes are usually much larger than 4×4, so nodes associated with edges will be relatively few in most levels of the quadtree.

Actually, the context just described applies only to the bottom or last level of the quadtree, where the nodes correspond to individual pixels. It was chosen to account for strong dependence among neighboring quadtree nodes (sets) within a subband. For the other levels of the quadtree, we add to this context the parent in the spatial orientation tree (SOT), in order to exploit dependence across scales of a wavelet transform. It is mainly beneficial at low bit rates where relatively few coefficients and hence nodes have become significant. However, this parent is associated with a quadtree node at the next higher splitting level. The reason is that the current node to be coded is a fusion of the four nodes in its subband that are the children of this particular parent node in the SOT. These associations are depicted in Figure 1.21.

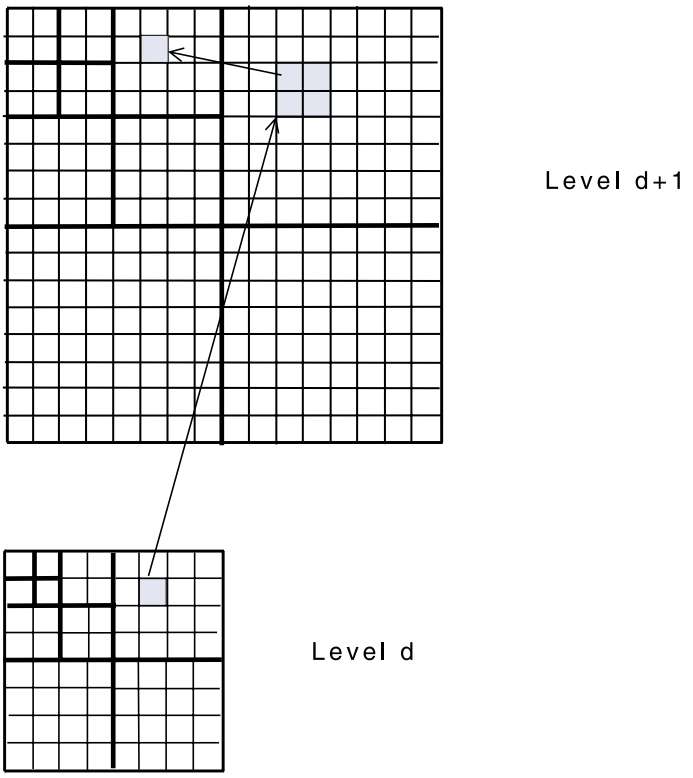

Fig. 1.21 Two successive levels of quadtree with nodes located in wavelet transform and dependency relationships.

Tables similar to those for JPEG2000 define the context labels. For those readers interested, these tables may be found in Hsiang's doctoral thesis [7]. The number of labels is about the same as that for JPEG2000. Probability estimates are gathered for each context as coding proceeds in order for the arithmetic coder to compute the length and bits of the codeword that specifies a corresponding location in the $[0,1)$ unit interval.

Conditional De-Quantization. EZBC also utilizes conditional de-quantization to reduce the reconstruction error. The probabilistic centroid of the quantization interval minimizes the mean squared error of a quantizer. When you decode one of these wavelet transform coders, you obtain a sequence of quantization bin indices that

identify the quantization intervals. The so-called de-quantizer sets the reconstruction value within the decoded quantization interval. The midpoint or the 38% point is often used arbitrarily, because the actual probability distribution is unknown. The 38% point is based on a probability model assumption that skews more of the probability toward the lower end of the interval. An estimate of the true reconstructed value can be obtained by realizing that the refinement bit from the next lower bit plane tells whether or not the value is in the lower or upper half of the current quantization interval $[\tau, 2\tau)$, where τ is the current threshold. Specifically, a refinement "0" from the next lower bit plane indicates that the value is in the first half of the interval. Therefore, except for decoding the bottom bit plane, we can count the number of next lower "0" refinement bits from previously decoded coefficients to estimate the probability $\hat{p}_0$ that the true value is in the interval $[\tau, 1.5\tau)$. The probability density function is assumed to be step-wise uniform, with step heights $\hat{p}_o/(\tau/2)$ for the first half and $(1-\hat{p}_o)/(\tau/2)$ for the second half of the interval. Using this estimated probability density function in the centroid definition, a formula for the centroid can be easily derived in terms of $\hat{p}_o$ and τ. These quantities are substituted into this formula to determine the reconstruction value. The centroid y_c of a probability density function $p_o(y)$ in the interval is given by

$$y_c = \int_{\tau}^{2\tau} y p_o(y) dy. \tag{1.5}$$

Substituting the stepwise uniform density function

$$p_o(y) = \begin{cases} \frac{2\hat{p}_o}{\tau}, & \tau \le y < 1.5\tau \\ \frac{2(1-\hat{p}_o)}{\tau}, & 1.5\tau \le y < 2\tau \\ 0, & \text{elsewhere} \end{cases} \tag{1.6}$$

produces the reconstruction value

$$y_c = \left(\frac{7}{4} - \frac{1}{2}\hat{p}_o\right)\tau. \tag{1.7}$$

The performance of EZBC surpasses that of JPEG2000, usually by 0.3 to 0.6 dB in PSNR, depending on the image and the bit rate. Presumably, the use of conditional de-quantization and the parent context and the overhead in JPEG2000 to enable random access and other codestream attributes account for this advantage. Neither the conditional de-quantization nor the parent context adds much computation. The JPEG2000 Working Group deliberately chose not to use parent context, so that each subblock would be coded independently. Such a choice preserves the features of random access, region-of-interest enhancement coding, and transmission error recovery. The conditional de-quantization utilizes the counts of "0" bits already accumulated for arithmetic coding of the refinement bits, and substitutes into the reconstruction formula 1.7, so involves very little in extra computation. EZBC is less complex computationally than JPEG2000, because it codes fewer symbols. JPEG2000 must pass through the bit planes of every coefficient, while EZBC passes only through bit planes of coefficients that have become significant. Furthermore, EZBC passes through each bit plane only once versus three times for JPEG2000. Also, in its bit-embedded mode, there is no post-compression Langrangian optimization for rate control.

Any bit plane coder can utilize this method of conditional de-quantization. For the images tested in trials of EZBC, PSNR gains of 0 to 0.4 dB are claimed. Presumably these gains are relative to mid-point de-quantization. Most likely, there will be smaller gains with respect to SPIHT and JPEG2000, because they use the usually more accurate 38% (or 3/8) point for reconstruction.[11]

1.6 Tree-Based Wavelet Transform Coding Systems

We have described several block-based wavelet transform coding systems, so now we turn to those which are tree-based. Already in Part I, we have presented the SPIHT and EZW (Embedded Zerotree Wavelet [16]) algorithms operating on trees in the transform domain, because these algorithms require the energy compaction properties

[11] JPEG2000 allows the option of midpoint or 3/8 point reconstruction.

of transforms to be efficient. These algorithms were described in their breadth-first search or bit-embedded coding modes and without entropy coding of their output bits. Here we shall explain how to realize the attributes of resolution scalability and random access of the block-based systems. Furthermore, we shall describe the entropy coding normally utilized in these tree-based systems. To do so without being redundant, SPIHT will be the test case for the methods that follow. The same methods work for EZW with obvious modification.

1.6.1 Fully Scalable SPIHT

The SPIHT algorithm is described in detail in Part I. Here we shall review the basic idea of locating individual coefficients and sets of coefficients whose magnitudes fall below a given threshold. The transform space is partitioned into trees descending from three of the four pixels in every 2×2 group plus the remaining upper left corner pixels in the lowest frequency subband, denoted by $\mathcal{H}$. A partition generated from a 2×2 group is called a tree-block and is depicted in Figure 1.22. The members of this partition or tree-block are three spatial orientation trees (SOT's), branching vertically, diagonally, and horizontally, and the corner pixel. The constituent sets of an SOT are all descendants from a root node, called a $\mathcal{D}$ set, the set of four offspring from a node, called an $\mathcal{O}$ set, and the grand-descendant set, denoted by $\mathcal{L}$, consisting of all descendants of members of the offspring set, $\mathcal{O}$. These sets are illustrated in Figure 1.22 for the particular case where the root node is in the lowest frequency subband.

The Sorting Pass. First, the largest integer power of 2 threshold, $\tau_{max} = 2^{n_{\max}}$, that does not exceed the largest magnitude coefficient in the entire transform is found. The algorithm maintains three control lists: the LIP (list of insignificant pixels), LIS (list of insignificant sets), and LSP (list of significant pixels). The LIP is initialized with all coordinates in $\mathcal{H}$, the LIS with all coordinates in $\mathcal{H}$ with descendants, and LSP as empty. The initial entries in the LIS are marked as Type A to indicate that the set includes all descendants of their roots. The algorithm then proceeds to test members of the LIP for significance at the current threshold. A binary symbol of 0 for "insignificant" or 1 for

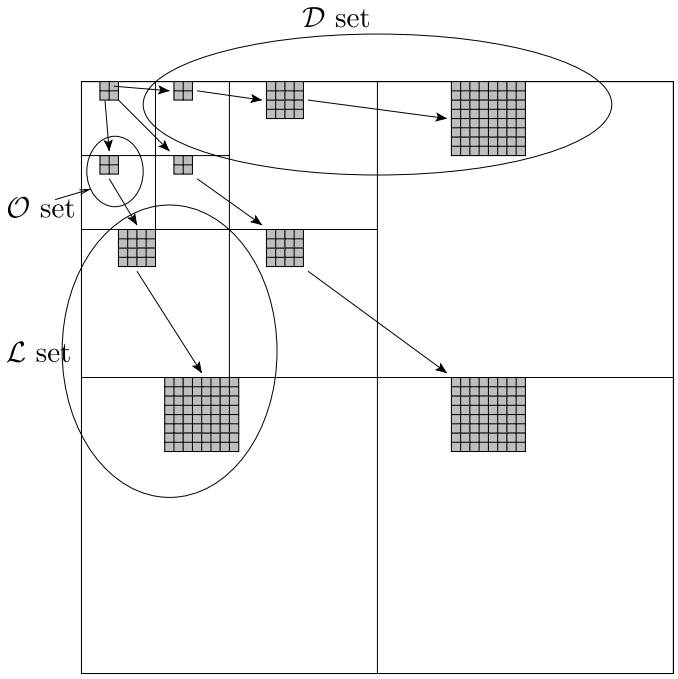

Fig. 1.22 Illustration of set types in a tree-block with its three constituent SOT's descending from a 2×2 group in the lowest frequency subband in a three level, 2D wavelet transform. A full descendant $\mathcal{D}$ set, an offspring $\mathcal{O}$ set, and a grand-descendant $\mathcal{L}$ set are encircled in the diagram. All pixels (coefficients) in grey, including the upper left corner pixel in the 2×2 group, belong to the tree-block.

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

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

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

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

The Refinement Pass. The refinement pass is executed immediately after every sorting pass, except for the first one at threshold $2^{n_{\max}}$, before lowering the threshold. The refinement pass at a threshold 2^n

consists of outputting the nth bit of the magnitude of pixels in the LSP that became significant at a higher threshold. This corresponds to collecting and writing the bits through bit plane n of all coefficents previously found to be significant.

The Decoder. The conveyance of the outcomes of the significance tests allows the decoder to recreate the actions of the encoder. The decoder initializes the same lists and populates them according to the significance test bits received from the codestream. As the codestream is being read, the coefficients belonging to the LSP coordinate entries can be progressively decoded and reconstructed, and all other coefficients are set to 0. The inverse wavelet transform then produces the reconstruction of the source image.

1.6.2 Resolution Scalable SPIHT

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

Referring to Figure 1.8, the correspondence of subbands to resolution level r is described in Table 1.7. The missing LL_r subbands in this table are the union of all the subbands at resolutions lower than r. To reconstruct an image at a given resolution $r > 0$, all subbands at the current and lower resolution levels are needed. For example, we need subbands $HL_2, LH_2, HH_2, HL_3, LH_3, HH_3, LL_3$ in order to reconstruct the image at resolution $r = 2$.

Let us assume that there are m levels of resolution numbered $r = 0, 1, \ldots, m-1$ ($m-1$ is the number of wavelet decompositions).

Table 1.7 Correspondences of resolution levels to subbands.

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

The way to distinguish resolution levels and avoid their intermingling on the lists is to maintain separate LIP, LIS, and LSP lists for each resolution. Let $\text{LIP}_r, \text{LIS}_r$, and LSP_r be the control lists for resolution r, $r = 0, 1, \ldots, m-1$. That means that coordinates located in subbands belonging only to resolution r reside in $\text{LIP}_r, \text{LIS}_r$, and LSP_r. As usual, whenever a pixel or set is tested, the binary result of 0 or 1 is immediately written to the codestream. For each resolution, starting with $r = 0$, we proceed with the usual SPIHT search for significance of pixels and sets at all thresholds from highest to lowest. This process creates entries to the lists at the current and next higher resolution. To see how this works, suppose we enter the process for some resolution r. Consider testing LIP_r at some threshold $\tau = 2^n$ and if significant, we move its coordinates to LSP_r. If not significant, its coordinates stay on LIP_r. After exiting LIP_r, LIS_r is tested at the same threshold. If an entry in LIS_r is insignificant, it stays on this list. However, if significant, it is removed from this list and the tree descending from the associated location is split into its four offspring and its grand-descendant set. The four offspring belong to the next higher resolution $r + 1$, so are placed on LIP_{r+1}. The grand-descendant set is designated by the coordinates of its grand-parent, so is then appended to LIS_r as Type B. The algorithm tests this LIS_r Type B entry immediately. If not significant, it stays as is. If significant, its four subtrees descending from offspring are placed on LIS_{r+1} as Type A entries designated by their associated offspring coordinates.

There are complications that arise with this procedure. Since we pass through all thresholds at resolution r before entering the lists of resolution $r + 1$, we must indicate the threshold at which single pixels are placed on LIP_{r+1}. These pixels are the offspring created from the splitting of a significant grand-descendant set. Instead of just putting these offspring immediately on LIP_{r+1}, they are first tested

for significance and placed on LIP_{r+1} if insignificant or on LSP_{r+1} if significant, along with tags indicating the threshold at which they were tested. When testing reaches resolution $r+1$, new entries to LIP_{r+1} and LSP_{r+1} should be inserted into the lists in positions that preserve the order of decreasing thresholds. These threshold tags, just like Type A or B, do not have to be sent, because the decoder can re-create them when it populates its lists. The procedure is stated in its entirety in Algorithm 1.5. It is essentially a modification to two dimensions of the three-dimensional RARS-SPIHT algorithm [2].

Algorithm 1.5. Resolution Scalable SPIHT

Notation

- $c_{i,j}$ is value of wavelet coefficient at coordinates (i,j).
- Significance function for bit plane n (threshold 2^n) $\Gamma_n(\mathcal{S}) = 1$, if entity (coefficient or set) $\mathcal{S}$ significant; otherwise 0.

Initialization step

- Initialize n to the number of bit planes $n_{\max}$.
- $\mathrm{LSP}_0 = \emptyset$
- LIP_0 : all the coefficients without any parents (the 4 coefficients of the 2×2 root block)
- LIS_0 : all coefficients from the LIP_0 with descendants (3 coefficients as only one has no descendant)
- For $r \neq 0$, $\mathrm{LSP}_r = \mathrm{LIP}_r = \mathrm{LIS}_r = \emptyset$.

For each r from 0 to maximum resolution $m-1$
For each n from $n_{\max}$ to 0 (bit planes)

Sorting pass

For each entry (i,j) of the LIP_r which had been added at a threshold strictly greater to the current n

- Output $\Gamma_n(i,j)$
- If $\Gamma_n(i,j) = 1$, move (i,j) to LSP_r and output the sign of $c_{i,j}$ (1)

For each entry (i,j) of the LIS_r which had been added at a bit plane greater than or equal to the current n

- If the entry is type A
  - Output $\Gamma_n(\mathcal{D}(i,j))$
  - If $\Gamma_n(\mathcal{D}(i,j)) = 1$ then
    - For all $(k,\ell) \in \mathcal{O}(i,j)$: output $\Gamma_n(k,\ell)$; If $\Gamma_n(k,\ell) = 1$, add (k,ℓ) to the LSP_{r+1} and output the sign of $c_{k,\ell}$ else, add (k,ℓ) to the end of the LIP_{r+1} **(2)**
    - If $\mathcal{L}(i,j) \neq \emptyset$, move (i,j) to the LIS_r as a type B entry
    - Else, remove (i,j) from the LIS_r
- If the entry is type B
  - Output $\Gamma_n(\mathcal{L}(i,j))$
  - If $\Gamma_n(\mathcal{L}(i,j)) = 1$
    - Add all the $(k,\ell) \in \mathcal{O}(i,j)$ to the LIS_{r+1} as a type A entry
    - Remove (i,j) from the LIS_r

Refinement pass

- For all entries (i,j) of the LSP_r which had been added at a bit plane strictly greater than the current n: Output the nth most significant bit of $c_{i,j}$.

This resolution scalable algorithm puts out the same bits as the original quality scalable algorithm, but in different order. In order to distinguish the resolutions, counts of bits coded at each resolution are written into a codestream header. Then we can decode up to a resolution level less than the full resolution, with confidence that we will get the same reconstruction as if we had encoded the source image reduced to the same resolution. However, we have lost control of the rate in this algorithm. The steps in Algorithm 1.5 describe coding through all the

bit planes, where the final file size or bit rate is undetermined. In order to enable rate control, we require knowledge of the bit plane boundaries in every resolution level. It is simple enough when encoding to count the number of bits coded at every threshold in every resolution level and put these counts into a file header. We depict the resolution scalable codestream in Figure 1.23 with resolution and threshold boundaries indicated by these bit counts. Then to reconstruct a desired resolution level r_d, we discard the bits coded from higher resolution levels and re-organize the codestream to the quality scalable structure, such as that shown in Figure 1.24. This structure is obtained by interleaving the code bits belonging to the remaining resolutions $r = 0, 1, 2, \ldots, r_d$ at the same thresholds. One follows the zig–zag scan of the subbands through resolution level r_d to extract first the bits coded at maximum threshold $2^{n_{\max}}$, then re-scans to extract bits coded at threshold $2^{n_{\max}-1}$, and so forth. Then the codestream can be truncated at the point corresponding to the target bitrate. To decode, one must use the quality scalable version of SPIHT or reverse the re-organization and use the resolution scalable version.

Actually, the algorithm for quality scalable SPIHT can be described just by interchanging the order of the two "For" loops in Algorithm 1.5. That accomplishes passing through all the resolution levels at every

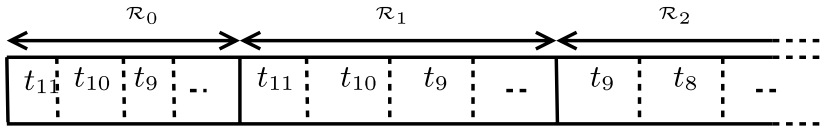

Fig. 1.23 Resolution scalable bitstream structure. $\mathcal{R}_0, \mathcal{R}_1, \ldots$ denote the segments with different resolutions, and $t_{11}, t_{10}, \ldots$ the different thresholds ($t_n = 2^n$). Note that in $\mathcal{R}_2$, t_{11} and t_{10} are empty.

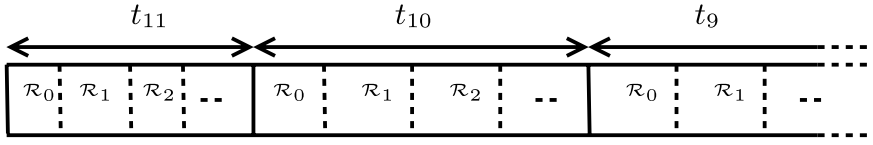

Fig. 1.24 Quality scalable bitstream structure. $\mathcal{R}_0, \mathcal{R}_1, \ldots$ denote the segments of different resolutions, and $t_{11}, t_{10}, \ldots$ the different thresholds ($t_n = 2^n$). At higher thresholds, some of the finer resolutions may be empty.

threshold, instead of vice-versa. Clearly, this interchange is exactly what we need to achieve quality scalability.

1.6.3 Block-oriented SPIHT Coding

The resolution scalable SPIHT algorithm can be further modified to provide *random access* decoding. Random access decoding means that a designated portion of the image can be reconstructed by extracting and decoding only an associated segment of the codestream. Recall that a block-tree of a wavelet transform, illustrated in Figure 1.22, corresponds to a region of the image geometrically similar in size and position of its 2×2 root group within the LL subband. Using the example of this figure, the dimensions of the LL subband and image are 8×8 and 64×64, respectively. Therefore, the second 16×16 block at the top of the image is similar to the 2×2 group root in the LL subband. In reality, this 2×2 wavelet coefficient group does not reconstruct the corresponding 16×16 image block exactly, because surrounding coefficients within the filter span contribute to values of pixels near the borders on the image block. However, the correct image block is reproduced, although not perfectly. The tree-blocks are mutually exclusive and their union forms the full wavelet transform.

Algorithm 1.6. Block-Oriented SPIHT Coding

- Form tree-blocks $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_K$ that partition the wavelet transform.
- For $k = 1, 2, \ldots, K$, code $\mathcal{B}_k$ either with Algorithm 2.4, Part I (neither quality nor resolution scalable) or Algorithm 1.5 (quality and resolution scalable).

Random access decoding can be realized therefore by coding all the tree-blocks separately. In fact, we can use either the quality or resolution scalable SPIHT to encode each tree-block in turn. We just initialize the lists of the SPIHT algorithm in the usual way, but only with the coordinates of the 2×2 group in the LL subband that is the root of the tree-block to be coded. The following steps of the quality

or resolution scalable SPIHT algorithm are exactly as before. When the algorithm finishes a tree-block, it moves to the next one until all the tree-blocks are coded. These steps are delineated in Algorithm 1.6. Within every tree-block, the codestream is either quality or resolution scalable, depending on which SPIHT algorithm is used.

Example 1.6 (Using Block-oriented SPIHT to Decode a Region of Interest). We demonstrate an example of using block-oriented SPIHT coding (Algorithm 1.6) to reconstruct a region interest from a portion of the codestream. Again, as in Example 1.1, the Goldhill source image's wavelet transform is quantized with step size 1/0.31, and the bin indices are losslessly coded to a codestream size corresponding to 1.329 bpp. The lossless coding method follows Algorithm 2.4, Part I, which is not quality scalable. The coding algorithm is block-oriented, so a segment of the codestream containing the coded blocks corresponding to the desired region of interest are extracted and decoded. Figure 1.25 shows the reconstruction of the region of interest beside that of the fully coded image.

Fig. 1.25 Reconstructions from same codestream of 512×512 Goldhill, quantizer step size $= 1/0.31$, coded to rate 1.329 bpp, and of 70×128 region at coordinates (343, 239).

1.7 Rate Control for Embedded Block Coders

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

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

$$\begin{aligned} D(B_T) &= \sum_{k=1}^{K} d_{\mathcal{B}_k}(b_k) \quad \text{subject to} \\ B_T &= \sum_{k=1}^{K} b_k, \end{aligned} \tag{1.8}$$

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

$$J(\lambda) = \sum_{k=1}^{K} (d_{\mathcal{B}_k}(b_k) + \lambda b_k), \tag{1.9}$$

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

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

For a given λ, we start calculating distortion-rate slope magnitudes starting from high rates to lower rates. These magnitudes increase with decreasing rate. We stop when we first reach

$$\frac{d^i_{\mathcal{B}_k} - d^{i+1}_{\mathcal{B}_k}}{b^{i+1}_k - b^i_k} \geq \lambda. \tag{1.10}$$

and declare the smaller of the rates b^i_k as the rate solution for block $\mathcal{B}_k$. We repeat for all $k = 1, 2, \ldots, K$. Algorithm 1.7 details this method.

Algorithm 1.7 Optimal bit allocation to independent blocks

Notation

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

(1) *Initialization*

(a) Obtain sequence of (rate, distortion) points $\{(b^i_k, d^i_{\mathcal{B}_k})\}$ for every block $\mathcal{B}_k$, $k = 1, 2, \ldots, K$.

(b) Set a bit budget target B_T to allocate.

(c) Choose $\lambda > 0$.

(2) *Main*

For $k = 1, 2, \ldots, K$ (for all blocks)

(a) Set initial i large for large rate.

(b) For steadily decreasing i calculate

$$\frac{\delta d^i_k}{\delta b^i_k} = \frac{d^i_{\mathcal{B}_k} - d^{i+1}_{\mathcal{B}_k}}{b^{i+1}_k - b^i_k}$$

only until $\geq \lambda_j$ and stop.[12] Let $i = i^*_k$ be the stopping index at smaller of the two rates.

---

[12] In practice the convexity assumption may fail and $\frac{\delta d^i_k}{\delta b^i_k} < \frac{\delta d^i_k}{\delta b^i_k}$. In that case, the point $(b^i_k, d^i_{\mathcal{B}_k})$ and both previous slope calculations involving this point must be discarded and replaced with $\frac{\delta d^i_k}{\delta b^i_k} = \frac{d^{i-1}_{\mathcal{B}_k} - d^{i+1}_{\mathcal{B}_k}}{b^{i+1}_k - b^{i-1}_k}$.

(c) Denote rate solution for kth block $b_k(\lambda) = b_k^{i_k^*}$.

(3) Total number of bits is $B(\lambda) = \sum_{k=1}^{K} b_k^{i_k *}$.

When we choose λ, the total rate, $B(\lambda) = \sum_k b_k$, is revealed only after the optimization procedure. We do know that λ is the magnitude of the common slope at the rate solution points. Therefore, we start with small λ corresponding to a high rate and steadily increase λ to test slopes at smaller rates. In this way, we can store a table of correspondences of λ to $B(\lambda)$ and then know the value of λ needed for the target rate B_T.

One of the problems with this procedure is that a rather odd set of total rates usually results. More often, one would like to find the cutting points for a given set of rates, such as 0.25, 0.50, 0.75, 1.0, and 2.0 bpp perhaps. Therefore, one needs to determine the unique slope parameter λ that produces each one of these rates. The bisection method starts with an interval of slope magnitudes that contains the solution for the prescribed rate and determines in which half of this interval lies the solution. Then the solution point is narrowed to one of the halves of this half-interval and so forth until the interval containing the solution is sufficiently small. The bisection procedure for a single rate is detailed in Algorithm 1.8. In calculating these rates for the various values of λ, we needed to calculate the distortion-rate slopes as in Step 2b in Algorithm 1.7. When you need to find the optimal parameters λ^* for a series of rates, it is particularly easy to set initial intervals once the first λ^* for the largest rate is found. Taking the rate targets in decreasing order, the distortion-rate slopes are increasing, so one can always use the last solution as the lower end point of the next slope interval.

Algorithm 1.8 Bisection procedure for optimal bit allocation

(1) Set target rate $B_T = \sum_{k=1}^{K} b_k(\lambda^*)$. To find λ^*. Set precision parameter $\epsilon > 0$.

(2) Find slope parameters $\lambda_1 > 0$ and $\lambda_2 > \lambda_1$ such that $B(\lambda_2) < B_T < B(\lambda_1)$.

(3) Let $\lambda_3 = (\lambda_2 + \lambda_1)/2$.

(4) Set $i = 3$.
(5) If $B(\lambda_i) < B_T,\ \lambda_{i+1} = (\lambda_i + \lambda_{j^*})/2, j^* = \arg\min_{j<i} B(\lambda_j) > B_T$.
If $B(\lambda_i) > B_T,\ \lambda_{i+1} = (\lambda_i + \lambda_{j^*})/2, j^* = \arg\max_{j<i} B(\lambda_j) < B_T$.
(Recursively determines half interval containing target rate.)
(6) If $|B(\lambda_{i+1}) - B_T| < \epsilon$, stop. Let $\lambda_{i+1} = \lambda^*$.
Else, set $i = i + 1$ and return to Step 5.

1.8 Conclusion

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

The coding methods featured in this section are the basic ones. The purpose of this section was to explain the fundamental methods, so as to provide the student and professional with the tools to extend or enhance these methods according to their own purposes. Many enhancements and hybrids appear in the literature. We shall mention just a few of them to suggest further readings. For example, only the basic JPEG2000 (Part I) coding algorithm was described along with its scalability features. There is a Part 2 standard with many extensions. Even in Part 1, built-in features of error resilience and region-of-interest encoding (max-shift method) have not been presented. Interested readers may consult the JPEG2000 standardization documents [10][11] and the book by Taubman and Marcellin [19].

There have been published many extensions and enhancements of the SPIHT and SPECK methods. SPIHT especially has inspired thousands of attempts to improve upon the original work [15]. One way was to dispense with the control lists, especially the LSP, because they

grow long, requiring large memory, for higher rates and large images. Articles by Wheeler and Pearlman [20] and Shively *et al.* [17] report the use of fixed memory arrays to store the states of pixels and sets, thereby alleviating the memory problem. A problem of SPIHT that detracts from its efficiency, especially at low bit rates, is the identification of an insignificant pixel at a high threshold. For each such pixel one bit has to be encoded for every lower threshold. Therefore, there are attempts in the literature to cluster these insignificant pixels, so that a single "0" indicates their common insignificance. One such successful attempt is the block-tree approach of Moinuddin and Khan [13]. Another approach by Khan and Ghanbari [12] clustered insignificant offspring together by creating virtual root nodes on top of the real root nodes of the LL subband.

There have been some notable attempts to combine SPIHT or EZW and vector quantization. The idea is that the significance search can sort vectors according to their locations between shells in higher dimensional spaces. Among such efforts are those by da Silva *et al.* [6] and Cosman *et al.* [5][4], who developed vector EZW coders, and Mukherjee *et al.* [14], who developed a vector SPIHT coder.

References

[1] T. Acharya and P.-S. Tsai, *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*. Hoboken, NJ: Wiley-Interscience, John Wiley & Sons, Inc., 2005.

[2] E. Christophe and W. A. Pearlman, "Three-dimensional SPIHT coding of volume images with random access and resolution scalability," *EURASIP Journal on Image and Video Processing*, vol. 2008, p. 13, doi:10.1155/2008/248905, 2008.

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

[4] P. C. Cosman, S. M. Perlmutter, and K. O. Perlmutter, "Tree-structured vector quantization with significance map for wavelet image coding," *Proceedings of 1995 Data Compression Conference (DCC '95)*, pp. 33–41, 28–30 March 1995.

[5] P. C. Cosman, S. M. Perlmutter, and K. O. Perlmutter, "Vector quantization with zerotree significance map for wavelet image coding," *Conference Record of the Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1419–1423, 30 October–2 November 1995.

[6] E. A. B. da Silva, D. G. Sampson, and M. Ghanbari, "A successive approximation vector quantizer for wavelet transform image coding," *IEEE Transactions on Image Processing*, vol. 5, no. 2, pp. 299–310, 1996.

[7] S.-T. Hsiang, "Highly scalable subband/wavelet image and video coding," PhD Thesis, Electrical, Computer and Systems Engineering Dept., Rensselaer Polytechnic Instute, Troy, NY 12180, USA, http://www.cipr.rpi.edu/ hsiang/thesis_dl.htm+, 2002.

[8] 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.

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

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

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

[12] E. Khan and M. Ghanbari, "Very low bit rate video coding using virtual SPIHT," *Electronics Letters*, vol. 37, no. 1, pp. 40–42.

[13] A. A. Moinuddin and E. Khan, "Wavelet based embedded image coding using unified zero-block-zero-tree approach," *Proceedings on IEEE International Conference on Acoustics, Speeech and Signal Processing (ICASSP 2006)*, vol. 2, pp. 453–456, 2006.

[14] D. Mukherjee and S. K. Mitra, "Successive refinement lattice vector quantization," *IEEE Transactions on Image Processing*, vol. 11, no. 12, pp. 1337–1348, December 2002.

[15] A. Said and W. A. Pearlman, "A new, fast and efficient umage 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.

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

[17] R. R. Shively, E. Ammicht, and P. D. Davis, "Generalizing SPIHT: A family of efficient image compression algorithms," *Proceedings on Acoustics, Speech, and Signal Processing 2000 (ICASSP 2000)*, vol. 4, pp. 2059–2062, 2000.

[18] D. S. Taubman, "High performance scalable image compression with EBCOT," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, 2000.

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

[20] F. W. Wheeler and W. A. Pearlman, "SPIHT image compression without lists," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2000)*, vol. 4, pp. 2047–2050, 2000.