In this example we used a simple scalar quantizer for quantization of the coefficients. However, if we use strategies that are motivated by the properties of the coefficients themselves, we can obtain significant performance improvements. In the next sections we examine two popular quantization strategies developed specifically for wavelets.

14.7 Embedded Zerotree Coder

The embedded zerotree wavelet (EZW) coder was introduced by Shapiro [212]. It is a quantization and coding strategy that incorporates some characteristics of the wavelet decomposition. Just as the quantization and coding approach used in the JPEG standard, which were motivated by the characteristics of the coefficients, were superior to the generic zonal coding algorithms, the EZW approach and its descendants significantly outperform some of the generic approaches. The particular characteristic used by the EZW algorithm is that there are wavelet coefficients in different subbands that represent the same spatial location in the image. If the decomposition is such that the size of the different subbands is different (the first two decompositions in Figure 14.12), then a single coefficient in the smaller subband may represent the same spatial location as multiple coefficients in the other subbands.

In order to put our discussion on more solid ground, consider the 10-band decomposition shown in Figure 14.15. The coefficient a in the upper-left corner of band I represents the same spatial location as coefficients a_1 in band II, a_2 in band III, and a_3 in band IV. In turn, the coefficient a_1 represents the same spatial location as coefficients a_{11}, a_{12}, a_{13}, and a_{14} in band V. Each of these pixels represents the same spatial location as four pixels in band VIII, and so on. In fact, we can visualize the relationships of these coefficients in the form of a tree: The coefficient a forms the root of the tree with three descendants a_1, a_2, and a_3. The coefficient a_1 has descendants a_{11}, a_{12}, a_{13}, and a_{14}. The coefficient a_2 has descendants a_{21}, a_{22}, a_{23}, and a_{24}, and the coefficient a_3 has descendants a_{31}, a_{32}, a_{33}, and a_{34}. Each of these coefficients in turn has four descendants, making a total of 64 coefficients in this tree. A pictorial representation of the tree is shown in Figure 14.16.

Recall that when natural images are decomposed in this manner most of the energy is compacted into the lower bands. Thus, in many cases the coefficients closer to the root of the tree have higher magnitudes than coefficients further away from the root. This means that often if a coefficient has a magnitude less than a given threshold, all its descendants will have magnitudes less than that threshold. In a scalar quantizer, the outer levels of the quantizer correspond to larger magnitudes. Consider the 3-bit quantizer shown in Figure 14.17. If we determine that all coefficients arising from a particular root have magnitudes smaller than T_0 and we inform the decoder of this situation, then for all coefficients in that tree we need only use 2 bits per sample, while getting the same performance as we would have obtained using the 3-bit quantizer. If the binary coding scheme used in Figure 14.17 is used, in which the first bit is the sign bit and the next bit is the most significant bit of the magnitude, then the information that a set of coefficients has value less than T_0 is the same as saying that the most significant bit of the magnitude is 0. If there are N coefficients in the tree, this is a savings of N bits minus however many bits are needed to inform the decoder of this situation.

Before we describe the EZW algorithm, we need to introduce some terminology. Given a threshold T, if a given coefficient has a magnitude greater than T, it is called a *significant*

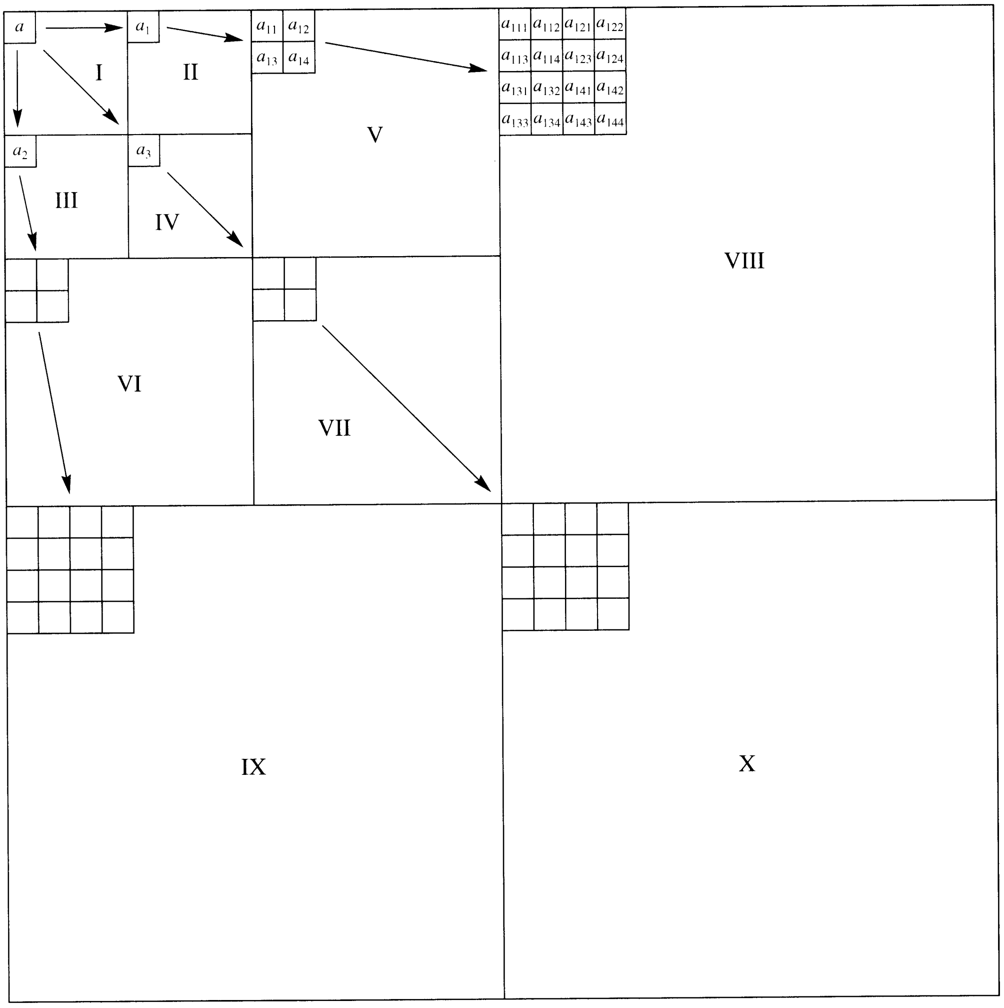
a
a1
a11 a12 a13 a14
a111 a112 a121 a122 a113 a114 a123 a124 a131 a132 a141 a142 a133 a134 a143 a144
a2
a3
I
II
III
IV
V
VI
VII
VIII
IX
X

FIGURE 14.15 A 10-band wavelet decomposition.

coefficient at level T. If the magnitude of the coefficient is less than T (it is insignificant), and all its descendants have magnitudes less than T, then the coefficient is called a *zerotree root*. Finally, it might happen that the coefficient itself is less than T but some of its descendants have a value greater than T. Such a coefficient is called an *isolated zero*.

The EZW algorithm is a multiple-pass algorithm, with each pass consisting of two steps: *significance map encoding* or the *dominant pass*, and *refinement* or the *subordinate pass*. If $c_{\max}$ is the value of the largest coefficient, the initial value of the threshold T_0 is given by

$$T_0 = 2^{\lfloor \log_2 c_{\max} \rfloor}. \tag{14.91}$$

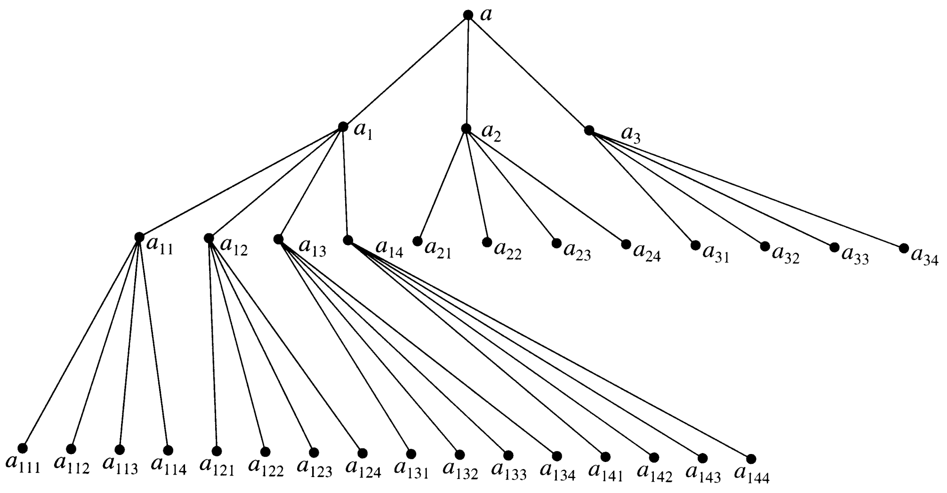
a
a_1
a_2
a_3
a_{11}
a_{12}
a_{13}
a_{14}
a_{21}
a_{22}
a_{23}
a_{24}
a_{31}
a_{32}
a_{33}
a_{34}
a_{111} a_{112} a_{113} a_{114} a_{121} a_{122} a_{123} a_{124} a_{131} a_{132} a_{133} a_{134} a_{141} a_{142} a_{143} a_{144}

FIGURE 14.16 **Data structure used in the EZW coder.**

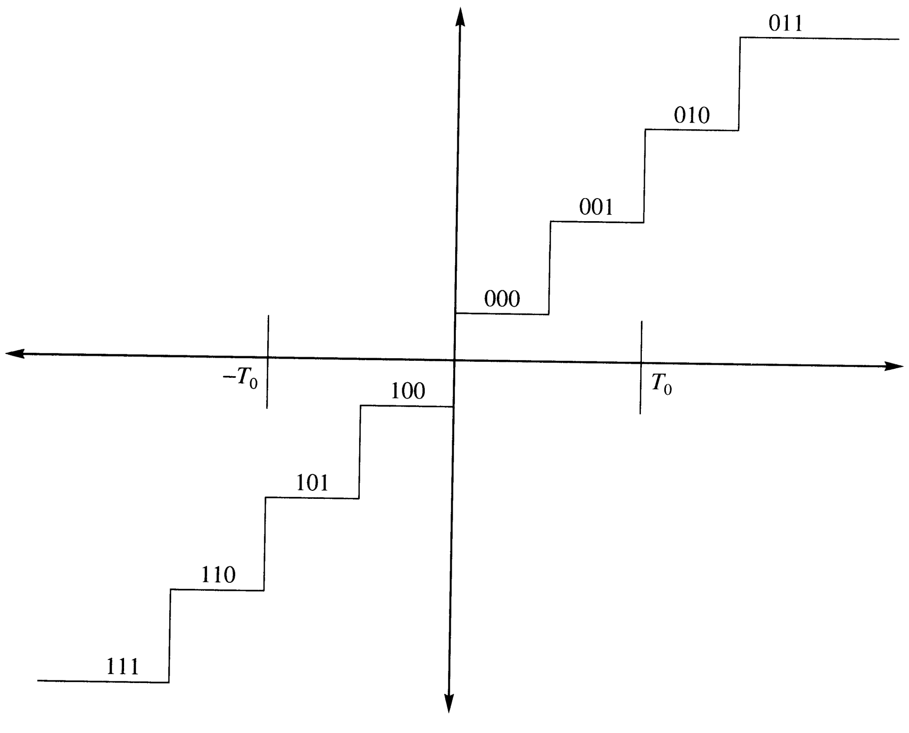

FIGURE 14.17 **A 3-bit midrise quantizer.**

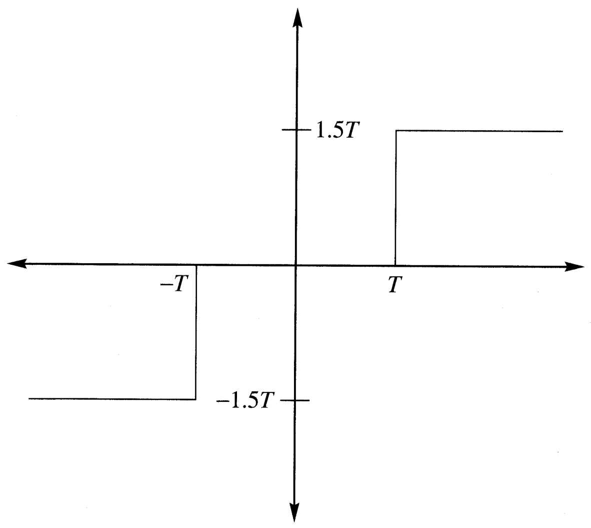

FIGURE 14.18 A three-level midtread quantizer.

This selection guarantees that the largest coefficient will lie in the interval $[T_0, 2T_0)$. In each pass, the threshold T_i is reduced to half the value it had in the previous pass:

$$T_i = \frac{1}{2}T_{i-1}. \tag{14.92}$$

For a given value of T_i, we assign one of four possible labels to the coefficients: *significant positive (sp), significant negative (sn), zerotree root (zr),* and *isolated zero (iz).* If we used a fixed-length code, we would need 2 bits to represent each of the labels. Note that when a coefficient has been labeled a zerotree root, we do not need to label its descendants. This assignment is referred to as *significance map coding.*

We can view the significance map coding in part as quantization using a three-level midtread quantizer. This situation is shown in Figure 14.18. The coefficients labeled *significant* are simply those that fall in the outer levels of the quantizer and are assigned an initial reconstructed value of $1.5T_i$ or $-1.5T_i$, depending on whether the coefficient is positive or negative. Note that selecting T_i according to (14.91) and (14.92) guarantees the significant coefficients will lie in the interval $[T, 2T)$. Once a determination of significance has been made, the significant coefficients are included in a list for further refinement in the refinement or subordinate passes. In the refinement pass, we determine whether the coefficient lies in the upper or lower half of the interval $[T, 2T)$. In successive refinement passes, as the value of T is reduced, the interval containing the significant coefficient is narrowed still further and the reconstruction is updated accordingly. An easy way to perform the refinement is to take the difference between the coefficient value and its reconstruction and quantize it using a two-level quantizer with reconstruction values $\pm T/4$. This quantized value is then added on to the current reconstruction value as a correction term.

The wavelet coefficients that have not been previously determined significant are scanned in the manner depicted in Figure 14.19, with each parent node in a tree scanned before its

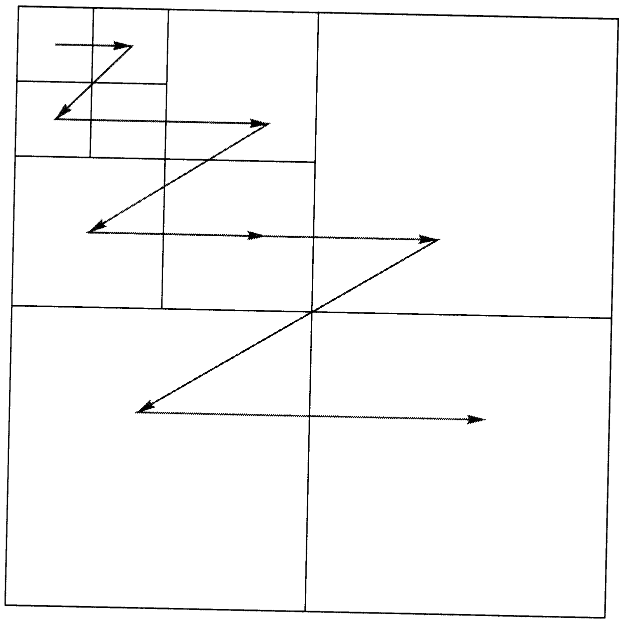

FIGURE 14.19 Scanning of wavelet coefficients for encoding using the EZW algorithm.

offspring. This makes sense because if the parent is determined to be a zerotree root, we would not need to encode the offspring.

Although this may sound confusing, in order to see how simple the encoding procedure actually is, let's use an example.

Example 14.7.1:

Let's use the seven-level decomposition shown below to demonstrate the various steps of EZW:

26	6	13	10
−7	7	6	4
4	−4	4	−3
2	−2	−2	0

To obtain the initial threshold value T_0, we find the maximum magnitude coefficient, which in this case is 26. Then

$$T_0 = 2^{\lfloor \log_2 26 \rfloor} = 16.$$

Comparing the coefficients against 16, we find 26 is greater than 16 so we send *sp*. The next coefficient in the scan is 6, which is less than 16. Furthermore, its descendants (13, 10, 6, and 4) are all less than 16. Therefore, 6 is a zerotree root, and we encode this entire set with the label *zr*. The next coefficient in the scan is −7, which is also a zerotree root, as is 7, the final element in the scan. We do not need to encode the rest of the coefficients separately because they have already been encoded as part of the various zerotrees. The sequence of labels to be transmitted at this point is

$$sp\ zr\ zr\ zr$$

Since each label requires 2 bits (for fixed-length encoding), we have used up 8 bits from our bit budget. The only significant coefficient in this pass is the coefficient with a value of 26. We include this coefficient in our list to be refined in the subordinate pass. Calling the subordinate list L_S, we have

$$L_S = \{26\}.$$

The reconstructed value of this coefficient is $1.5T_0 = 24$, and the reconstructed bands look like this:

24	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

The next step is the subordinate pass, in which we obtain a correction term for the reconstruction value of the significant coefficients. In this case, the list L_S contains only one element. The difference between this element and its reconstructed value is $26 - 24 = 2$. Quantizing this with a two-level quantizer with reconstruction levels $\pm T_0/4$, we obtain a correction term of 4. Thus, the reconstruction becomes $24 + 4 = 28$. Transmitting the correction term costs a single bit, therefore at the end of the first pass we have used up 9 bits. Using only these 9 bits, we would obtain the following reconstruction:

28	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

We now reduce the value of the threshold by a factor of two and repeat the process. The value of T_1 is 8. We rescan the coefficients that have not yet been deemed significant. To emphasize the fact that we do not consider the coefficients that have been deemed significant in the previous pass, we replace them with $\star$:

$\star$	6	13	10
−7	7	6	4
4	−4	4	−3
2	−2	−2	0

The first coefficient we encounter has a value of 6. This is less than the threshold value of 8; however, the descendants of this coefficient include coefficients with values of 13 and 10. Therefore, this coefficient cannot be classified as a zerotree root. This is an example of what we defined as an isolated zero. The next two coefficients in the scan are −7 and 7. Both of these coefficients have magnitudes less than the threshold value of 8. Furthermore, all their descendants also have magnitudes less than 8. Therefore, these two coefficients are coded as *zr*. The next two elements in the scan are 13 and 10, which are both coded as *sp*. The final two elements in the scan are 6 and 4. These are both less than the threshold, but they do not have any descendants. We code these coefficients as *iz*. Thus, this dominant pass is coded as

iz zr zr sp sp iz iz

which requires 14 bits, bringing the total number of bits used to 23. The significant coefficients are reconstructed with values $1.5T_1 = 12$. Thus, the reconstruction at this point is

28	0	12	12
0	0	0	0
0	0	0	0
0	0	0	0

We add the new significant coefficients to the subordinate list:

$$L_S = \{26, 13, 10\}.$$

In the subordinate pass, we take the difference between the coefficients and their reconstructions and quantize these to obtain the correction or refinement values for these coefficients. The possible values for the correction terms are $\pm T_1/4 = \pm 2$:

$$\begin{aligned} 26 - 28 = -2 &\Rightarrow \text{Correction term} = -2 \\ 13 - 12 = 1 &\Rightarrow \text{Correction term} = 2 \\ 10 - 12 = -2 &\Rightarrow \text{Correction term} = -2 \end{aligned} \tag{14.93}$$

Each correction requires a single bit, bringing the total number of bits used to 26. With these corrections, the reconstruction at this stage is

26	0	14	10
0	0	0	0
0	0	0	0
0	0	0	0

If we go through one more pass, we reduce the threshold value to 4. The coefficients to be scanned are

⋆	6	⋆	⋆
−7	7	6	4
4	−4	4	−3
2	−2	−2	0

The dominant pass results in the following coded sequence:

sp sn sp sp sp sp sn iz iz sp iz iz iz

This pass cost 26 bits, equal to the total number of bits used previous to this pass. The reconstruction upon decoding of the dominant pass is

26	6	14	10
−6	6	6	6
6	−6	6	0
0	0	0	0

The subordinate list is

$$L_S = \{26, 13, 10, 6 - 7, 7, 6, 4, 4, -4, 4\}$$

By now it should be reasonably clear how the algorithm works. We continue encoding until we have exhausted our bit budget or until some other criterion is satisfied. ◆

There are several observations we can make from this example. Notice that the encoding process is geared to provide the most bang for the bit at each step. At each step the bits are used to provide the maximum reduction in the reconstruction error. If at any time the encoding is interrupted, the reconstruction using this (interrupted) encoding is the best that the algorithm could have provided using this many bits. The encoding improves as more bits are transmitted. This form of coding is called *embedded coding*. In order to enhance this aspect of the algorithm, we can also sort the subordinate list at the end of each pass using information available to both encoder and decoder. This would increase the likelihood of larger coefficients being encoded first, thus providing for a greater reduction in the reconstruction error.

Finally, in the example we determined the number of bits used by assuming fixed-length encoding. In practice, arithmetic coding is used, providing a further reduction in rate.

14.8 Set Partitioning in Hierarchical Trees

The SPIHT (Set Partitioning in Hierarchical Trees) algorithm is a generalization of the EZW algorithm and was proposed by Amir Said and William Pearlman [199]. Recall that in EZW we transmit a lot of information for little cost when we declare an entire subtree to be insignificant and represent all the coefficients in it with a zerotree root label *zr*. The SPIHT algorithm uses a partitioning of the trees (which in SPIHT are called *spatial orientation trees*) in a manner that tends to keep insignificant coefficients together in larger subsets. The partitioning decisions are binary decisions that are transmitted to the decoder, providing a significance map encoding that is more efficient than EZW. In fact, the efficiency of the significance map encoding in SPIHT is such that arithmetic coding of the binary decisions provides very little gain. The thresholds used for checking significance are powers of two, so in essence the SPIHT algorithm sends the binary representation of the integer value of the wavelet coefficients. As in EZW, the significance map encoding, or set partitioning and ordering step, is followed by a refinement step in which the representations of the significant coefficients are refined.

Let's briefly describe the algorithm and then look at some examples of its operation. However, before we do that we need to get familiar with some notation. The data structure used by the SPIHT algorithm is similar to that used by the EZW algorithm—although not the same. The wavelet coefficients are again divided into trees originating from the lowest resolution

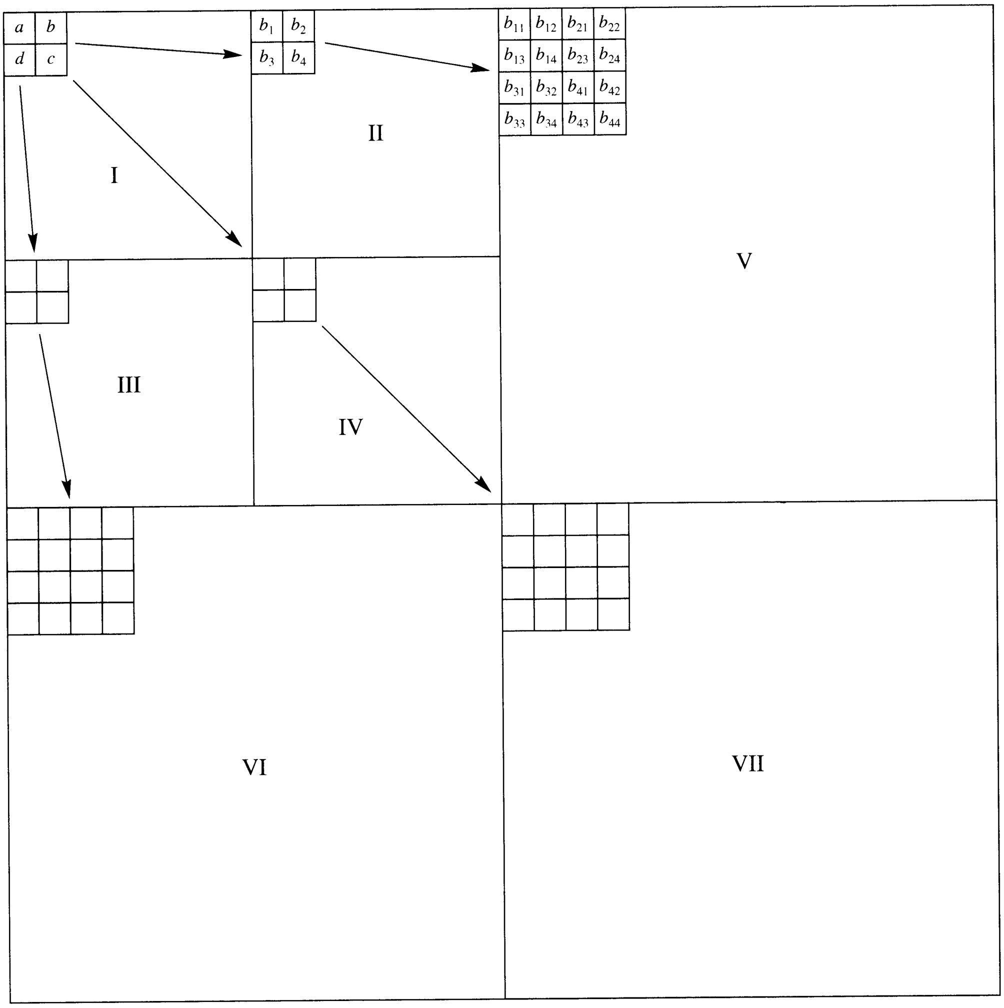

FIGURE 14.20 **Data structure used in the SPIHT algorithm.**

band (band I in our case). The coefficients are grouped into 2×2 arrays that, except for the coefficients in band I, are offsprings of a coefficient of a lower resolution band. The coefficients in the lowest resolution band are also divided into 2×2 arrays. However, unlike the EZW case, all but one of them are root nodes. The coefficient in the top-left corner of the array does not have any offsprings. The data structure is shown pictorially in Figure 14.20 for a seven-band decomposition.

The trees are further partitioned into four types of sets, which are sets of coordinates of the coefficients:

- $\mathcal{O}(i,j)$ This is the set of coordinates of the offsprings of the wavelet coefficient at location (i,j). As each node can either have four offsprings or none, the size of $\mathcal{O}(i,j)$ is either zero or four. For example, in Figure 14.20 the set $\mathcal{O}(0,1)$ consists of the coordinates of the coefficients b_1, b_2, b_3, and b_4.

- $\mathcal{D}(i,j)$ This is the set of all descendants of the coefficient at location (i,j). Descendants include the offsprings, the offsprings of the offsprings, and so on. For example, in Figure 14.20 the set $\mathcal{D}(0,1)$ consists of the coordinates of the coefficients $b_1, \ldots, b_4, b_{11}, \ldots, b_{14}, \ldots, b_{44}$. Because the number of offsprings can either be zero or four, the size of $\mathcal{D}(i,j)$ is either zero or a sum of powers of four.
- $\mathcal{H}$ This is the set of all root nodes—essentially band I in the case of Figure 14.20.
- $\mathcal{L}(i,j)$ This is the set of coordinates of all the descendants of the coefficient at location (i,j) except for the immediate offsprings of the coefficient at location (i,j). In other words,

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

In Figure 14.20 the set $\mathcal{L}(0,1)$ consists of the coordinates of the coefficients $b_{11}, \ldots, b_{14}, \ldots, b_{44}$.

A set $\mathcal{D}(i,j)$ or $\mathcal{L}(i,j)$ is said to be significant if any coefficient in the set has a magnitude greater than the threshold. Finally, thresholds used for checking significance are powers of two, so in essence the SPIHT algorithm sends the binary representation of the integer value of the wavelet coefficients. The bits are numbered with the least significant bit being the zeroth bit, the next bit being the first significant bit, and the kth bit being referred to as the $k-1$ most significant bit.

With these definitions under our belt, let us now describe the algorithm. The algorithm makes use of three lists: the *list of insignificant pixels* (LIP), the *list of significant pixels* (LSP), and the *list of insignificant sets* (LIS). The LSP and LIS lists will contain the coordinates of coefficients, while the LIS will contain the coordinates of the roots of sets of type $\mathcal{D}$ or $\mathcal{L}$. We start by determining the initial value of the threshold. We do this by calculating

$$n = \lfloor \log_2 c_{\max} \rfloor$$

where $c_{\max}$ is the maximum magnitude of the coefficients to be encoded. The LIP list is initialized with the set $\mathcal{H}$. Those elements of $\mathcal{H}$ that have descendants are also placed in LIS as type $\mathcal{D}$ entries. The LSP list is initially empty.

In each pass, we will first process the members of LIP, then the members of LIS. This is essentially the significance map encoding step. We then process the elements of LSP in the refinement step.

We begin by examining each coordinate contained in LIP. If the coefficient at that coordinate is significant (that is, it is greater than 2^n), we transmit a 1 followed by a bit representing the sign of the coefficient (we will assume 1 for positive, 0 for negative). We then move that coefficient to the LSP list. If the coefficient at that coordinate is not significant, we transmit a 0.

After examining each coordinate in LIP, we begin examining the sets in LIS. If the set at coordinate (i,j) is not significant, we transmit a 0. If the set is significant, we transmit a 1. What we do after that depends on whether the set is of type $\mathcal{D}$ or $\mathcal{L}$.

If the set is of type $\mathcal{D}$, we check each of the offsprings of the coefficient at that coordinate. In other words, we check the four coefficients whose coordinates are in $\mathcal{O}(i,j)$. For each coefficient that is significant, we transmit a 1, the sign of the coefficient, and then move the coefficient to the LSP. For the rest we transmit a 0 and add their coordinates to the LIP. Now that we have removed the coordinates of $\mathcal{O}(i,j)$ from the set, what is left is simply the set $\mathcal{L}(i,j)$. If this set is not empty, we move it to the end of the LIS and mark it to be of type $\mathcal{L}$. Note that this new entry into the LIS has to be examined during *this* pass. If the set is empty, we remove the coordinate (i,j) from the list.

If the set is of type $\mathcal{L}$, we add each coordinate in $\mathcal{O}(i,j)$ to the end of the LIS as the root of a set of type $\mathcal{D}$. Again, note that these new entries in the LIS have to be examined during this pass. We then remove (i,j) from the LIS.

Once we have processed each of the sets in the LIS (including the newly formed ones), we proceed to the refinement step. In the refinement step we examine each coefficient that was in the LSP *prior to the current pass* and output the nth most significant bit of $|c_{i,j}|$. We ignore the coefficients that have been added to the list in this pass because, by declaring them significant at this particular level, we have already informed the decoder of the value of the nth most significant bit.

This completes one pass. Depending on the availability of more bits or external factors, if we decide to continue with the coding process, we decrement n by one and continue. Let's see the functioning of this algorithm on an example.

Example 14.8.1:

Let's use the same example we used for demonstrating the EZW algorithm:

26	6	13	10
−7	7	6	4
4	−4	4	−3
2	−2	−2	0

We will go through three passes at the encoder and generate the transmitted bitstream, then decode this bitstream.

First Pass: The value for n in this case is 4. The three lists at the encoder are

$$\begin{aligned}
&\text{LIP: } \{(0,0) \rightarrow 26, (0,1) \rightarrow 6, (1,0) \rightarrow -7, (1,1) \rightarrow 7\}\\
&\text{LIS: } \{(0,1)\mathcal{D}, (1,0)\mathcal{D}, (1,1)\mathcal{D}\}\\
&\text{LSP: } \{\}
\end{aligned}$$

In the listing for LIP, we have included the $\rightarrow \#$ to make it easier to follow the example. Beginning our algorithm, we examine the contents of LIP. The coefficient at location (0,1) is greater than 16. In other words, it is significant; therefore, we transmit a 1, then a 0 to indicate the coefficient is positive and move the coordinate to LSP. The next three coefficients are all insignificant at this value of the threshold; therefore, we transmit a 0 for each coefficient and leave them in LIP. The next step is to examine the contents of LIS. Looking at the descendants of the coefficient at location (0, 1) (13, 10, 6, and 4), we see that none of them are significant at this value of the threshold so we transmit a 0. Looking at the descendants of c_{10} and c_{11}, we can see that none of these are significant at this value of the threshold. Therefore, we transmit a 0 for each set. As this is the first pass, there are no elements from the previous pass in LSP; therefore, we do not do anything in the refinement pass. We have transmitted a total of 8 bits at the end of this pass (10000000), and the situation of the three lists is as follows:

$$\begin{aligned}\text{LIP: } &\{(0,1) \to 6, (1,0) \to -7, (1,1) \to 7\}\\ \text{LIS: } &\{(0,1)\mathcal{D}, (1,0)\mathcal{D}, (1,1)\mathcal{D}\}\\ \text{LSP: } &\{(0,0) \to 26\}\end{aligned}$$

Second Pass: For the second pass we decrement n by 1 to 3, which corresponds to a threshold value of 8. Again, we begin our pass by examining the contents of LIP. There are three elements in LIP. Each is insignificant at this threshold so we transmit three 0s. The next step is to examine the contents of LIS. The first element of LIS is the set containing the descendants of the coefficient at location (0, 1). Of this set, both 13 and 10 are significant at this value of the threshold; in other words, the set $\mathcal{D}(0, 1)$ is significant. We signal this by sending a 1 and examine the offsprings of c_{01}. The first offspring has a value of 13, which is significant and positive, so we send a 1 followed by a 0. The same is true for the second offspring, which has a value of 10. So we send another 1 followed by a 0. We move the coordinates of these two to the LSP. The next two offsprings are both insignificant at this level; therefore, we move these to LIP and transmit a 0 for each. As $\mathcal{L}(0, 1) = \{\}$, we remove $(0, 1)\mathcal{D}$ from LIS. Looking at the other elements of LIS, we can clearly see that both of these are insignificant at this level; therefore, we send a 0 for each. In the refinement pass we examine the contents of LSP from the previous pass. There is only one element in there that is not from the current sorting pass, and it has a value of 26. The third MSB of 26 is 1; therefore, we transmit a 1 and complete this pass. In the second pass we have transmitted 13 bits: 0001101000001. The condition of the lists at the end of the second pass is as follows:

$$\begin{aligned}\text{LIP: } &\{(0,1) \to 6, (1,0) \to -7, (1,1) \to 7(1,2) \to 6, (1,3) \to 4\}\\ \text{LIS: } &\{(1,0)\mathcal{D}, (1,1)\mathcal{D}\}\\ \text{LSP: } &\{(0,0) \to 26, (0,2) \to 13, (0,3) \to 10\}\end{aligned}$$

Third Pass: The third pass proceeds with $n = 2$. As the threshold is now smaller, there are significantly more coefficients that are deemed significant, and we end up sending 26 bits. You can easily verify for yourself that the transmitted bitstream for the third pass is 10111010101101100110000010. The condition of the lists at the end of the third pass is as follows:

$$\begin{aligned}\text{LIP: } &\{(3,0) \to 2, (3,1) \to -2, (2,3) \to -3, (3,2) \to -2, (3,3) \to 0\}\\ \text{LIS: } &\{\}\\ \text{LSP: } &\{(0,0) \to 26, (0,2) \to 13, (0,3) \to 10, (0,1) \to 6, (1,0) \to -7, (1,1) \to 7,\\ &(1,2) \to 6, (1,3) \to 4(2,0) \to 4, (2,1) \to -4, (2,2) \to 4\}\end{aligned}$$

Now for decoding this sequence. At the decoder we also start out with the same lists as the encoder:

$$\begin{aligned} \text{LIP: } & \{(0,0),(0,1),(1,0),(1,1)\} \\ \text{LIS: } & \{(0,1)\mathcal{D},(1,0)\mathcal{D},(1,1)\mathcal{D}\} \\ \text{LSP: } & \{\} \end{aligned}$$

We assume that the initial value of n is transmitted to the decoder. This allows us to set the threshold value at 16. Upon receiving the results of the first pass (10000000), we can see that the first element of LIP is significant and positive and no other coefficient is significant at this level. Using the same reconstruction procedure as in EZW, we can reconstruct the coefficients at this stage as

24	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

and, following the same procedure as at the encoder, the lists can be updated as

$$\begin{aligned} \text{LIP: } & \{(0,1),(1,0),(1,1)\} \\ \text{LIS: } & \{(0,1)\mathcal{D},(1,0)\mathcal{D},(1,1)\mathcal{D}\} \\ \text{LSP: } & \{(0,0)\} \end{aligned}$$

For the second pass we decrement n by one and examine the transmitted bitstream: 0001101000001. Since the first 3 bits are 0 and there are only three entries in LIP, all the entries in LIP are still insignificant. The next 9 bits give us information about the sets in LIS. The fourth bit of the received bitstream is 1. This means that the set with root at coordinate $(0, 1)$ is significant. Since this set is of type $\mathcal{D}$, the next bits relate to its offsprings. The 101000 sequence indicates that the first two offsprings are significant at this level and positive and the last two are insignificant. Therefore, we move the first two offsprings to LSP and the last two to LIP. We can also approximate these two significant coefficients in our reconstruction by $1.5 \times 2^3 = 12$. We also remove $(0,1)\mathcal{D}$ from LIS. The next two bits are both 0, indicating that the two remaining sets are still insignificant. The final bit corresponds to the refinement pass. It is a 1, so we update the reconstruction of the $(0,0)$ coefficient to $24 + 8/2 = 28$. The reconstruction at this stage is

28	0	12	12
0	0	0	0
0	0	0	0
0	0	0	0

and the lists are as follows:

$$\begin{aligned} \text{LIP: } & \{(0,1),(1,0),(1,1),(1,2),(1,3)\} \\ \text{LIS: } & \{(1,0)\mathcal{D},(1,1)\mathcal{D}\} \\ \text{LSP: } & \{(0,0),(0,2),(0,3)\} \end{aligned}$$

For the third pass we again decrement n, which is now 2, giving a threshold value of 4. Decoding the bitstream generated during the third pass (10111010101101100110000010), we update our reconstruction to

26	6	14	10
−6	6	6	6
6	−6	6	0
0	0	0	0

and our lists become

LIP: $\{(3,0),(3,1)\}$

LIS: $\{\}$

LSP: $\{(0,0),(0,2),(0,3),(0,1),(1,0),(1,1),(1,2),(2,0),(2,1),(3,2)\}$

At this stage we do not have any sets left in LIS and we simply update the values of the coefficients. ◆

Finally, let's look at an example of an image coded using SPIHT. The image shown in Figure 14.21 is the reconstruction obtained from a compressed representation that used 0.5 bits per pixel. (The programs used to generate this image were obtained from the following website: *http://ipl.rpi.edu/SPIHT*.) Comparing this with Figure 14.14, we can see a definite improvement in the quality.

Wavelet decomposition has been finding its way into various standards. The earliest example was the FBI fingerprint image compression standard. The latest is the new image compression being developed by the JPEG committee, commonly referred to as JPEG 2000. We take a brief look at the current status of JPEG 2000.

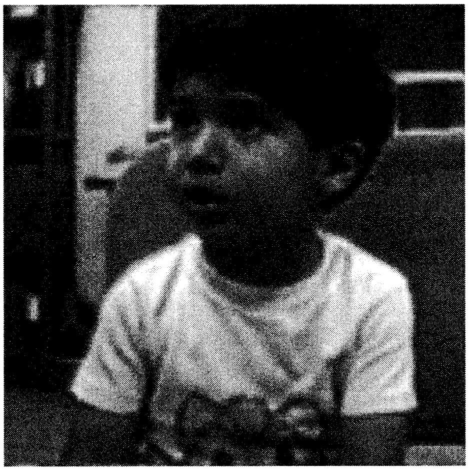

FIGURE 14.21 Reconstruction of Sinan image encoded using SPIHT at 0.5 bits per pixel.