true not just for sorting but for any algorithm based on comparisons or on any type of branching.

The actual algorithm used by SPIHT is based on the realization that there is really no need to sort *all* the coefficients. The main task of the sorting pass in each iteration is to select those coefficients that satisfy $2^n \le |c_{i,j}| < 2^{n+1}$. This task is divided into two parts. For a given value of n, if a coefficient $c_{i,j}$ satisfies $|c_{i,j}| \ge 2^n$, then we say that it is *significant*; otherwise, it is called *insignificant*. In the first iteration, relatively few coefficients will be significant, but their number increases from iteration to iteration, since n keeps getting decremented. The sorting pass has to determine which of the significant coefficients satisfies $|c_{i,j}| < 2^{n+1}$ and transmit their coordinates to the decoder. This is an important part of the algorithm used by SPIHT.

The encoder partitions all the coefficients into a number of sets T_k and performs the significance test

$$\max_{(i,j)\in T_k} |c_{i,j}| \ge 2^n \ ?$$

on each set T_k. The result may be either "no" (all the coefficients in T_k are insignificant, so T_k itself is considered insignificant) or "yes" (some coefficients in T_k are significant, so T_k itself is significant). This result is transmitted to the decoder. If the result is "yes," then T_k is partitioned by both encoder and decoder, using the same rule, into subsets and the same significance test is performed on all the subsets. This partitioning is repeated until all the significant sets are reduced to size 1 (i.e., they contain one coefficient each, and that coefficient is significant). This is how the significant coefficients are identified by the sorting pass in each iteration.

The significant test performed on a set T can be summarized by

$$S_n(T) = \begin{cases} 1, & \max_{(i,j)\in T} |c_{i,j}| \ge 2^n, \\ 0, & \text{otherwise.} \end{cases} \tag{5.28}$$

The result, $S_n(T)$, is a single bit that is transmitted to the decoder. Since the result of each significance test becomes a single bit written on the compressed stream, the number of tests should be minimized. To achieve this goal, the sets should be created and partitioned such that sets expected to be significant will be large and sets expected to be insignificant will contain just one element.

5.14.2 Spatial Orientation Trees

The sets T_k are created and partitioned using a special data structure called a *spatial orientation tree*. This structure is defined in a way that exploits the spatial relationships between the wavelet coefficients in the different levels of the subband pyramid. Experience has shown that the subbands in each level of the pyramid exhibit spatial similarity (Figure 5.17b). Any special features, such as a straight edge or a uniform region, are visible in all the levels at the same location.

The spatial orientation trees are illustrated in Figure 5.57a,b for a 16×16 image. The figure shows two levels, level 1 (the highpass) and level 2 (the lowpass). Each level is divided into four subbands. Subband LL2 (the lowpass subband) is divided into four groups of 2×2 coefficients each. Figure 5.57a shows the top-left group,

while Figure 5.57b shows the bottom-right one. In each group, each of the four coefficients (except the top-left one, marked in gray) becomes the root of a spatial orientation tree. The arrows show examples of how the various levels of these trees are related. The thick arrows indicate how each group of 4×4 coefficients in level 2 is the parent of four such groups in level 1. In general, a coefficient at location (i,j) in the image is the parent of the four coefficients at locations $(2i,2j)$, $(2i+1,2j)$, $(2i,2j+1)$, and $(2i+1,2j+1)$.

The roots of the spatial orientation trees of our example are located in subband LL2 (in general, they are located in the top-left LL subband, which can be of any size), but any wavelet coefficient, except the gray ones on level 1 (also except the leaves), can be considered the root of some spatial orientation subtree. The leaves of all those trees are located on level 1 of the subband pyramid.

In our example, subband LL2 is of size 4×4, so it is divided into four 2×2 groups, and three of the four coefficients of a group become roots of trees. Thus, the number of trees in our example is 12. In general, the number of trees is 3/4 the size of the highest LL subband.

Each of the 12 roots in subband LL2 in our example is the parent of four children located on the same level. However, the children of these children are located on level 1. This is true in general. The roots of the trees are located on the highest level and their children are on the same level, but from then on, the four children of a coefficient on level k are themselves located on level $k-1$.

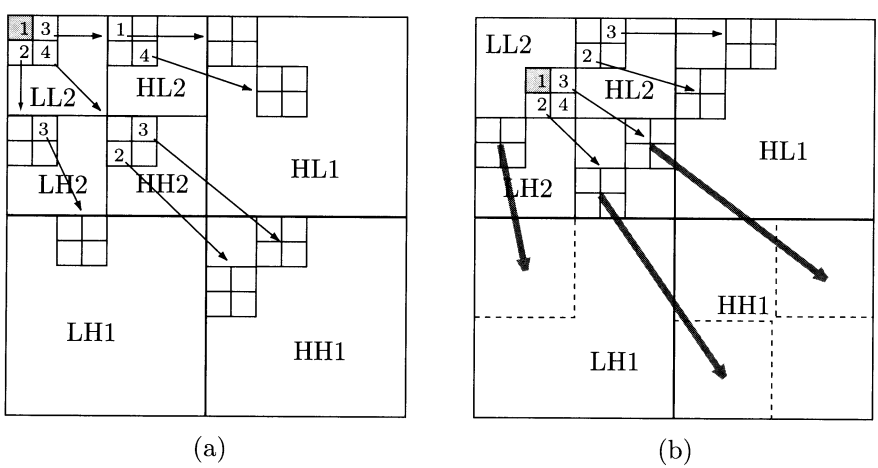

Figure 5.57: Spatial Orientation Trees in SPIHT.

We use the terms *offspring* for the four children of a node, and *descendants* for the children, grandchildren, and all their descendants. The set partitioning sorting algorithm uses the following four sets of coordinates:

1. $\mathcal{O}(i,j)$: the set of coordinates of the four offspring of node (i,j). If node (i,j) is a leaf of a spatial orientation tree, then $\mathcal{O}(i,j)$ is empty.
2. $\mathcal{D}(i,j)$: the set of coordinates of the descendants of node (i,j).

3. $\mathcal{H}(i,j)$: the set of coordinates of the roots of all the spatial orientation trees (3/4 of the wavelet coefficients in the highest LL subband).
4. $\mathcal{L}(i,j)$: The difference set $\mathcal{D}(i,j) - \mathcal{O}(i,j)$. This set contains all the descendants of tree node (i,j), except its four offspring.

The spatial orientation trees are used to create and partition the sets T_k. The set partitioning rules are as follows:

1. The initial sets are $\{(i,j)\}$ and $\mathcal{D}(i,j)$, for all $(i,j) \in \mathcal{H}$ (i.e., for all roots of the spatial orientation trees). In our example there are 12 roots, so there will initially be 24 sets: 12 sets, each containing the coordinates of one root, and 12 more sets, each with the coordinates of all the descendants of one root.
2. If set $\mathcal{D}(i,j)$ is significant, then it is partitioned into $\mathcal{L}(i,j)$ plus the four single-element sets with the four offspring of (i,j). In other words, if any of the descendants of node (i,j) is significant, then its four offspring become four new sets and all its other descendants become another set (to be significance tested in rule 3).
3. If $\mathcal{L}(i,j)$ is significant, then it is partitioned into the four sets $\mathcal{D}(k,l)$, where (k,l) are the offspring of (i,j).

Once the spatial orientation trees and the set partitioning rules are understood, the coding algorithm can be described.

5.14.3 SPIHT Coding

It is important to have the encoder and decoder test sets for significance in the same way, so the coding algorithm uses three lists called *list of significant pixels* (LSP), *list of insignificant pixels* (LIP), and *list of insignificant sets* (LIS). These are lists of coordinates (i,j) that in the LIP and LSP represent individual coefficients, and in the LIS represent either the set $\mathcal{D}(i,j)$ (a type A entry) or the set $\mathcal{L}(i,j)$ (a type B entry).

The LIP contains coordinates of coefficients that were insignificant in the previous sorting pass. In the current pass they are tested, and those that test significant are moved to the LSP. In a similar way, sets in the LIS are tested in sequential order, and when a set is found to be significant, it is removed from the LIS and is partitioned. The new subsets with more than one coefficient are placed back in the LIS, to be tested later, and the subsets with one element are tested and appended to the LIP or the LSP, depending on the results of the test. The refinement pass transmits the nth most significant bit of the entries in the LSP.

Figure 5.58 shows this algorithm in detail. Figure 5.59 is a simplified version, for readers who are intimidated by too many details.

The decoder executes the detailed algorithm of Figure 5.58. It always works in *lockstep* with the encoder, but the following notes shed more light on its operation:

1. Step 2.2 of the algorithm evaluates all the entries in the LIS. However, step 2.2.1 appends certain entries to the LIS (as type-B) and step 2.2.2 appends other entries to the LIS (as type-A). It is important to realize that all these entries are also evaluated by step 2.2 in the same iteration.
2. The value of n is decremented in each iteration, but there is no need to bring it all the way to zero. The loop can stop after any iteration, resulting in lossy compression. Normally, the user specifies the number of iterations, but it is also

1. Initialization: Set n to $\lfloor \log_2 \max_{i,j}(c_{i,j}) \rfloor$ and transmit n. Set the LSP to empty. Set the LIP to the coordinates of all the roots $(i,j) \in \mathcal{H}$. Set the LIS to the coordinates of all the roots $(i,j) \in \mathcal{H}$ that have descendants.
2. Sorting pass:
   2.1 for each entry (i,j) in the LIP do:
      2.1.1 output $S_n(i,j)$;
      2.1.2 if $S_n(i,j) = 1$, move (i,j) to the LSP and output the sign of $c_{i,j}$;
   2.2 for each entry (i,j) in the LIS do:
      2.2.1 if the entry is of type A, then
         - output $S_n(\mathcal{D}(i,j))$;
         - if $S_n(\mathcal{D}(i,j)) = 1$, then
            * for each $(k,l) \in \mathcal{O}(i,j)$ do:
               · output $S_n(k,l)$;
               · if $S_n(k,l) = 1$, add (k,l) to the LSP, output the sign of $c_{k,l}$;
               · if $S_n(k,l) = 0$, append (k,l) to the LIP;
            * if $\mathcal{L}(i,j) \neq 0$, move (i,j) to the end of the LIS, as a type-B entry, and go to step 2.2.2; else, remove entry (i,j) from the LIS;
      2.2.2 if the entry is of type B, then
         - output $S_n(\mathcal{L}(i,j))$;
         - if $S_n(\mathcal{L}(i,j)) = 1$, then
            * append each $(k,l) \in \mathcal{O}(i,j)$ to the LIS as a type-A entry:
            * remove (i,j) from the LIS:
3. Refinement pass: for each entry (i,j) in the LSP, except those included in the last sorting pass (the one with the same n), output the nth most significant bit of $|c_{i,j}|$;
4. Loop: decrement n by 1 and go to step 2 if needed.

Figure 5.58: The SPIHT Coding Algorithm.

1. Set the threshold. Set LIP to all root nodes coefficients. Set LIS to all trees (assign type D to them). Set LSP to an empty set.
2. Sorting pass: Check the significance of all coefficients in LIP:
   2.1 If significant, output 1, output a sign bit, and move the coefficient to the LSP.
   2.2 If not significant, output 0.
3. Check the significance of all trees in the LIS according to the type of tree:
   3.1 For a tree of type D:
      3.1.1 If it is significant, output 1, and code its children:
         3.1.1.1 If a child is significant, output 1, then a sign bit, add it to the LSP
         3.1.1.2 If a child is insignificant, output 0 and add the child to the end of LIP.
         3.1.1.3 If the children have descendants, move the tree to the end of LIS as type L, otherwise remove it from LIS.
      3.1.2 If it is insignificant, output 0.
   3.2 For a tree of type L:
      3.2.1 If it is significant, output 1, add each of the children to the end of LIS as an entry of type D and remove the parent tree from the LIS.
      3.2.2 If it is insignificant, output 0.
4. Loop: Decrement the threshold and go to step 2 if needed.

Figure 5.59: A Simplified SPIHT Coding Algorithm.

possible to have the user specify the acceptable amount of distortion (in units of MSE), and the encoder can use Equation (5.27) to decide when to stop the loop.

3. The encoder knows the values of the wavelet coefficients $c_{i,j}$ and uses them to calculate the bits S_n (Equation (5.28)), which it transmits (i.e., writes on the compressed stream). These bits are input by the decoder, which uses them to calculate the values of $c_{i,j}$. The algorithm executed by the decoder is that of Figure 5.58 but with the word "output" changed to "input."

4. The sorting information, previously denoted by $m(k)$, is recovered when the coordinates of the significant coefficients are appended to the LSP in steps 2.1.2 and 2.2.1. This implies that the coefficients indicated by the coordinates in the LSP are sorted according to

$$\lfloor \log_2 |c_{m(k)}| \rfloor \geq \lfloor \log_2 |c_{m(k+1)}| \rfloor,$$

for all values of k. The decoder recovers the ordering because its three lists (LIS, LIP, and LSP) are updated in the same way as those of the encoder (remember that the decoder works in lockstep with the encoder). When the decoder inputs data, its three lists are identical to those of the encoder at the moment it (the encoder) output that data.

5. The encoder starts with the wavelet coefficients $c_{i,j}$; it never gets to "see" the actual image. The decoder, however, has to display the image and update the display in each iteration. In each iteration, when the coordinates (i, j) of a coefficient $c_{i,j}$ are moved to the LSP as an entry, it is known (to both encoder and decoder) that $2^n \leq |c_{i,j}| < 2^{n+1}$. As a result, the best value that the decoder can give the coefficient $\hat{c}_{i,j}$ that is being reconstructed is midway between 2^n and $2^{n+1} = 2 \times 2^n$. Thus, the decoder sets $\hat{c}_{i,j} = \pm 1.5 \times 2^n$ (the sign of $\hat{c}_{i,j}$ is input by the decoder just after the insertion). During the refinement pass, when the decoder inputs the actual value of the nth bit of $c_{i,j}$, it improves the value 1.5×2^n by adding 2^{n-1} to it (if the input bit was a 1) or subtracting 2^{n-1} from it (if the input bit was a 0). This way, the decoder can improve the appearance of the image (or, equivalently, reduce the distortion) during *both* the sorting and refinement passes.

It is possible to improve the performance of SPIHT by entropy coding the encoder's output, but experience shows that the added compression gained this way is minimal and does not justify the additional expense of both encoding and decoding time. It turns out that the signs and the individual bits of coefficients output in each iteration are uniformly distributed, so entropy coding them does not produce any compression. The bits $S_n(i, j)$ and $S_n(\mathcal{D}(i, j))$, on the other hand, are distributed nonuniformly and may gain from such coding.

5.14.4 Example

We assume that a 4×4 image has already been transformed, and the 16 coefficients are stored in memory as 6-bit signed-magnitude numbers (one sign bit followed by five magnitude bits). They are shown in Figure 5.60, together with the single spatial orientation tree. The coding algorithm initializes LIP to the one-element set $\{(1, 1)\}$, the LIS to the set $\{\mathcal{D}(1, 1)\}$, and the LSP to the empty set. The largest coefficient is 18, so n is set to $\lfloor \log_2 18 \rfloor = 4$. The first two iterations are shown.

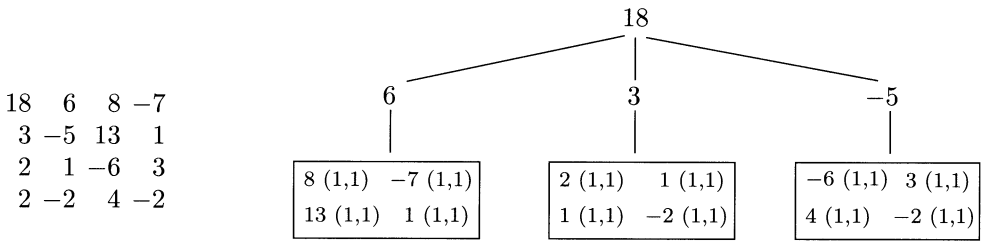

Figure 5.60: Sixteen Coefficients and One Spatial Orientation Tree.

Sorting Pass 1:
$2^n = 16$.
Is (1,1) significant? yes: output a 1.
LSP = $\{(1,1)\}$, output the sign bit: 0.
Is $\mathcal{D}(1,1)$ significant? no: output a 0.
LSP = $\{(1,1)\}$, LIP = $\{\}$, LIS = $\{\mathcal{D}(1,1)\}$.
Three bits output.
Refinement pass 1: no bits are output (this pass deals with coefficients from sorting pass $n-1$).
Decrement n to 3.
Sorting Pass 2:
$2^n = 8$.
Is $\mathcal{D}(1,1)$ significant? yes: output a 1.
Is $(1,2)$ significant? no: output a 0.
Is $(2,1)$ significant? no: output a 0.
Is $(2,2)$ significant? no: output a 0.
LIP = $\{(1,2),(2,1),(2,2)\}$, LIS = $\{\mathcal{L}(1,1)\}$.
Is $\mathcal{L}(1,1)$ significant? yes: output a 1.
LIS = $\{\mathcal{D}(1,2),\mathcal{D}(2,1),\mathcal{D}(2,2)\}$.
Is $\mathcal{D}(1,2)$ significant? yes: output a 1.
Is $(1,3)$ significant? yes: output a 1.
LSP = $\{(1,1),(1,3)\}$, output sign bit: 1.
Is $(2,3)$ significant? yes: output a 1.
LSP = $\{(1,1),(1,3),(2,3)\}$, output sign bit: 1.
Is $(1,4)$ significant? no: output a 0.
Is $(2,4)$ significant? no: output a 0.
LIP = $\{(1,2),(2,1),(2,2),(1,4),(2,4)\}$,
LIS = $\{\mathcal{D}(2,1),\mathcal{D}(2,2)\}$.
Is $\mathcal{D}(2,1)$ significant? no: output a 0.
Is $\mathcal{D}(2,2)$ significant? no: output a 0.
LIP = $\{(1,2),(2,1),(2,2),(1,4),(2,4)\}$,
LIS = $\{\mathcal{D}(2,1),\mathcal{D}(2,2)\}$,
LSP = $\{(1,1),(1,3),(2,3)\}$.
Fourteen bits output.

Refinement pass 2: After iteration 1, the LSP included entry $(1,1)$, whose value is $18 = 10010_2$.
One bit is output.
Sorting Pass 3:
$2^n = 4$.
Is $(1,2)$ significant? yes: output a 1.
LSP $= \{(1,1),(1,3),(2,3),(1,2)\}$, output a sign bit: 1.
Is $(2,1)$ significant? no: output a 0.
Is $(2,2)$ significant? yes: output a 1.
LSP $= \{(1,1),(1,3),(2,3),(1,2),(2,2)\}$, output a sign bit: 0.
Is $(1,4)$ significant? yes: output a 1.
LSP $= \{(1,1),(1,3),(2,3),(1,2),(2,2),(1,4)\}$, output a sign bit: 1.
Is $(2,4)$ significant? no: output a 0.
LIP $= \{(2,1),(2,4)\}$.
Is $D(2,1)$ significant? no: output a 0.
Is $D(2,2)$ significant? yes: output a 1.
Is $(3,3)$ significant? yes: output a 1.
LSP $= \{(1,1),(1,3),(2,3),(1,2),(2,2),(1,4),(3,3)\}$, output a sign bit: 0.
Is $(4,3)$ significant? yes: output a 1.
LSP $= \{(1,1),(1,3),(2,3),(1,2),(2,2),(1,4),(3,3),(4,3)\}$, output a sign bit: 1.
Is $(3,4)$ significant? no: output a 0.
LIP $= \{(2,1),(2,4),(3,4)\}$.
Is $(4,4)$ significant? no: output a 0.
LIP $= \{(2,1),(2,4),(3,4),(4,4)\}$.
LIP $= \{(2,1),(3,4),(3,4),(4,4)\}$, LIS $= \{\mathcal{D}(2,1)\}$,
LSP $= \{(1,1),(1,3),(2,3),(1,2),(2,2),(1,4),(3,3),(4,3)\}$.
Sixteen bits output.
Refinement Pass 3:
After iteration 2, the LSP included entries $(1,1)$, $(1,3)$, and $(2,3)$, whose values are $18 = 10010_2$, $8 = 1000_2$, and $13 = 1101_2$. Three bits are output
After two iterations, a total of 37 bits has been output.

Bibliography

Said, A., and W. A. Pearlman (1996), "A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees," *IEEE Transactions on Circuits and Systems for Video Technology*, **6**(6):243–250, June.

5.14.5 QTCQ

Closely related to SPIHT, the QTCQ (quadtree classification and trellis coding quantization) method [Banister and Fischer 99] uses fewer lists than SPIHT and explicitly forms classes of the wavelet coefficients for later quantization by means of the ACTCQ and TCQ (arithmetic and trellis coded quantization) algorithms of [Joshi, Crump, and Fischer 93].

The method uses the spatial orientation trees originally developed by SPIHT. This type of tree is a special case of a quadtree. The encoding algorithm is iterative. In the nth iteration, if any element of this quadtree is found to be significant, then