

SIGNAL COMPRESSION

9. Lossy image compression: SPIHT and S+P

9.1 SPIHT embedded coder

9.2 The reversible multiresolution transform S+P

9.3 Error resilience in embedded coding

9.1 Embedded Tree-Based Coding - Overview

- Supports desirable features:
 - embedded coding
 - fast compression and decompression
 - very precise distortion and rate control
 - support for lossless compression
 - great scalability
 - no need for training
- Two techniques are used:
 - Tree-based recursive partitioning scheme
 - Bit plane coding, permitting progressive transmission with gradual quality improvement
- History
 - Lewis and Knowles 1991 - A 64 Kb/s video codec using the 2-D wavelet transform DCC'91
They use the first time trees to exploit the statistical properties found in pyramidal decomposition of natural images.

- Shapiro 1993, 1996. Embedded image coding using zero-trees of wavelets coefficients, IEEE Trans. Signal Processing 1996. (Earlier version at ICASSP'93)
Here the first time embedded trees and bit plane coding were used. EZW was the turning point in lossy coding (see Chapter 8 of the lectures, describing EZW coding).
- Said and Pearlman 1993, Image compression using spatial orientation trees, ISCAS'93
Said and Pearlman 1996, A new fast and efficient codec based on set partitioning in hierarchical trees. IEEE Trans. Circuits Syst. Video Technology, June 1996.
SPIHT method was introduced here, as an efficient improvement of EZW. The arithmetic coding (absolutely necessary in EZW) is not necessary here (however, one may gain small improvements by using arithmetic coding).
- Said and Pearlman 1996, An image multiresolution for lossless and lossy compression IEEE Trans. Image Processing, Sept. 1996
Progressive from lossy to lossless compression is possible with the S+P multiresolution transformation.
- Recent developments: Use with different image transformations
SPIHT can be used successfully with 8×8 DCT or 16×16 DCT.
The lifting scheme allows the development of better transforms for lossless compression
- Recent developments: Resilience to transmission error
SPIHT can be changed to cope successfully with bit errors or even packet losses.

SPIHT Progressive Image Transmission. Preliminaries

- The original image is the set of pixels $p_{i,j}$ where (i, j) is the pixel coordinate. Denote \mathbf{p} the original image and \mathbf{c} the coefficients

$$\mathbf{c} = \Omega(\mathbf{p})$$

obtained with a unitary hierarchical subband transformation $\Omega(\cdot)$. The array \mathbf{c} has the same dimensions as the array \mathbf{p} , and $c_{i,j}$ is the transform coefficient of coordinates (i, j) (Note: the coefficient $c_{i,j}$ has a significance very different of that of $p_{i,j}$).

- After receiving an approximate description of the coefficients, $\hat{\mathbf{c}}$, the decoder can obtain the reconstructed image as $\hat{\mathbf{p}} = \Omega^{-1}(\hat{\mathbf{c}})$.
- The distortion of the reconstruction is given by

$$D_{MSE} = \frac{1}{N} \|\mathbf{p} - \hat{\mathbf{p}}\|^2 = \frac{1}{N} \|\mathbf{c} - \hat{\mathbf{c}}\|^2 = \frac{1}{N} \sum_i \sum_j (c_{i,j} - \hat{c}_{i,j})^2$$

If the transform coefficient $c_{i,j}$ is transmitted exactly, the MSE decreases by $c_{i,j}^2/N$. This shows that the coefficients with the larger magnitudes should be sent first.

Also the information in the value of $c_{i,j}$ can also be ranked according to its binary representation, the most significant bits should be transmitted first, using the bit-plane method.

Transmission of the coefficient values

- Assume the coefficients are ordered according to the number of bits necessary for the binary representation of their magnitude (See Fig. 1). The bits in the lowest rows are the least significant bits.
- Assume that, besides the ordering information, the decoder receives the number μ_n corresponding to the number of coefficients such that $2^n \leq |c_{i,j}| < 2^{n+1}$. In the example of Fig.1 $\mu_5 = 2, \mu_4 = 2, \mu_3 = 4, \dots$. The bits of the coefficients should be sent as indicated by the arrows in Figure 1.
- Because the coefficients are in decreasing order of magnitude, the leading "0" bits and the first "1" of any column do not need to be transmitted.
- The progressive transmission can be implemented by Algorithm I (see the figure labeled Algorithm I).
- Normally, good quality images can be recovered after a relatively small fraction of the pixel coordinates are transmitted.
- A large fraction of bit budget is used in the sorting pass, there the sophisticated coding is necessary.

BIT ROW

		sign	s	s	s	s	s	s	s	s	s	s	s	s	s
msb	5		1	1	0	0	0	0	0	0	0	0	0	0	0
	4	→		1	1	0	0	0	0	0	0	0	0	0	0
	3	→	→		1	1	1	1	0	0	0	0	0	0	0
	2	→	→	→	→	→		1	1	1	1	1	1	1	1
	1	→	→	→	→	→	→	→	→	→	→	→	→	→	→
lsb	0	→	→	→	→	→	→	→	→	→	→	→	→	→	→

Fig. 1. Binary representation of the magnitude-ordered coefficients.

Algorithm 1:

- 1) output $n = \lfloor \log_2(\max_{(i,j)} \{|c_{i,j}|\}) \rfloor$ to the decoder;
- 2) output μ_n , followed by the pixel coordinates $\eta(k)$ and sign of each of the μ_n coefficients such that $2^n \leq |c_{\eta(k)}| < 2^{n+1}$ (**sorting pass**);
- 3) output the n th most significant bit of all the coefficients with $|c_{i,j}| \geq 2^{n+1}$ (i.e., those that had their coordinates transmitted in previous sorting passes), in the same order used to send the coordinates (**refinement pass**);
- 4) decrement n by one, and go to Step 2).

Set partitioning sorting algorithm

- The ordering data is not explicitly transmitted. The encoder and decoder use the same sorting algorithm, and the results of comparisons in the branching of the encoder is transmitted to the decoder, who can duplicate encoder execution path.
- We do not *sort* all coefficients. We need only to select coefficients such that $2^n \leq |c_{i,j}| < 2^{n+1}$, with n decremented after each pass. Given n , a coefficient is significant if $2^n \leq |c_{i,j}|$, otherwise it is insignificant.
- The sorting algorithm divides the set of pixels into partitioning subsets \mathcal{T}_m and performs the magnitude test

$$\max_{(i,j) \in \mathcal{T}_m} \{|c_{i,j}|\} \geq 2^{n??} \quad (25)$$

The subset is insignificant if the answer is "no" (all coefficients in \mathcal{T}_m are insignificant). If the answer is "yes", the subset is significant, and a certain rule is used to partition \mathcal{T}_m into new subsets, which will be in their turn tested for significance. The rule is chosen such that the number of magnitude comparisons (bits to be transmitted) is minimized.

- Each node of the tree corresponds to a pixel and is identified by the pixel coordinate. Its direct descendants are called offsprings, corresponds to the pixels of the same spatial orientation in the next finer level of the pyramid.
- Each node, has either no offspring (the leaves) or four offsprings, which always form a group of adjacent pixels. In Fig. 2 the arrows are oriented from the parent to the offsprings.
- The pixels in the highest level of the pyramid are the tree roots and are also grouped in 2×2 adjacent pixels and in each group of four, one of them (the star) does not have any offsprings.
- The following sets of coordinates are needed in the explanation:
 - $\mathcal{O}(i, j)$: set of coordinates of all offsprings of node (i, j) .
 - $\mathcal{D}(i, j)$: set of coordinates of all descendants of node (i, j) .
 - \mathcal{H} : set of coordinates of all spatial orientation tree roots (nodes in the highest pyramid level).
 - $\mathcal{L}(i, j) = \mathcal{D}(i, j) - \mathcal{O}(i, j)$.
- The set are easily found, e.g. except at highest and lowest pyramid levels
 $\mathcal{O}(i, j) = \{(2i, 2j), (2i, 2j + 1), (2i + 1, 2j), (2i + 1, 2j + 1)\}$.
- The rules for splitting the set (e.g. when found significant):
 - The initial partition is formed with the sets (i, j) and $\mathcal{D}(i, j)$, for all $(i, j) \in \mathcal{H}$.

- If $\mathcal{D}(i, j)$ is significant, then it is partitioned into $\mathcal{L}(i, j)$ plus the four single-element sets with $(k, l) \in \mathcal{O}(i, j)$.
- If $\mathcal{L}(i, j)$ is significant, then it is partitioned into the four sets $\mathcal{D}(k, l)$ with $(k, l) \in \mathcal{O}(i, j)$.

The coding algorithm

- The significance information is stored in three ordered lists:
 - LIS - list of insignificant sets
 - LIP - list of insignificant pixels
 - LSP - list of significant pixels

The entries in LIS are sets of the type $\mathcal{D}(i, j)$ (type A) or type $\mathcal{L}(i, j)$ (type B).

- During the sorting pass the pixels in the LIP - which were insignificant in the previous pass - are tested, and those that became significant are moved to the LSP.

The sets are sequentially evaluated following the LIS order, and when a set is found significant it is removed from the list and partitioned. The new sets with more than one element are added back to LIS, while the one element sets are added to the end of LIP or LSP, according to their being significant.

- The algorithm is presented in Algorithm II , being essentially the same as Algorithm I, but using set partitioning approach in sorting pass.
- The significance function is defined as follows:

$$S_n(\mathcal{T}_m) = \begin{cases} 1 & \text{if } \max_{(i,j) \in \mathcal{T}_m} \{|c_{i,j}|\} \geq 2^n \\ 0 & \text{otherwise} \end{cases}$$

- The decoding algorithm is identical, except all occurrences of the "output" function must be replaced by the "input" function.
- An additional task of the decoder is to update the reconstructed image. The decoder will use the information that $2^n \leq |c_{i,j}| < 2^{n+1}$, and the sign of $c_{i,j}$ to reconstruct $\hat{c}_{i,j} = \pm 1.5 \times 2^n$. During the refinement pass, the decoder adds or subtracts 2^{n-1} to $\hat{c}_{i,j}$.
- Arithmetic coding is not strictly necessary, the bitstream is very efficient.
- If used, arithmetic coding may improve several percents the compression ration. The mapping of bits to symbols for the arithmetic coding tries to make use of similar features of the 2×2 blocks of pixels.

Algorithm II:

- 1) **Initialization:** output $n = \lceil \log_2(\max_{(i,j)} \{|c_{i,j}|\}) \rceil$;
set the LSP as an empty list, and add the coordinates $(i, j) \in \mathcal{H}$ to the LIP, and only those with descendants also to the LIS, as type *A* entries.
- 2) **Sorting Pass:**
 - 2.1) for each entry (i, j) in the LIP do:
 - 2.1.1) output $S_n(i, j)$;
 - 2.1.2) if $S_n(i, j) = 1$ then 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 $S_n(k, l) = 1$ then add (k, l) to the LSP and output the sign of $c_{k,l}$;
 - if $S_n(k, l) = 0$ then add (k, l) to the end of the LIP;
 - * if $\mathcal{L}(i, j) \neq \emptyset$ then move (i, j) to the end of the LIS, as an entry of type *B*, and go to Step 2.2.2); otherwise, 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
 - * add each $(k, l) \in \mathcal{O}(i, j)$ to the end of the LIS as an entry of type *A*;
 - * remove (i, j) from the LIS.
- 3) **Refinement Pass:** for each entry (i, j) in the LSP, except those included in the last sorting pass (i.e., with same n), output the n th most significant bit of $|c_{i,j}|$;
- 4) **Quantization-Step Update:** decrement n by 1 and go to Step 2.

Example

- The same example as for EZW encoding in the previous lecture.

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

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

- First pass

The value of n is 4. The three lists at the encoder are:

- LIP: $\{(0, 0) \rightarrow 26; (0, 1) \rightarrow 6; (1, 0) \rightarrow -7; (1, 1) \rightarrow 7\}$
- LIS: $\{(0, 1)\mathcal{D}; (1, 0)\mathcal{D}; (1, 1)\mathcal{D}; \}$
- LSP: $\{\}$

We examine the contents of LIP. The coefficient at location $(0,0)$ is greater than 16, therefore it is significant and we transmit a 1, then a 0 to indicate the coefficient is positive and move the coordinate $(0,0)$ as the first entry in LSP.

The next three coefficients in the list LIP are all insignificant (in absolute value below the threshold 16). We transmit a 0 for each coefficient and leave them in LIP.

The next step we 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 (1,0) and (1,1) we see that none is significant at this value of the threshold, therefore we transmit a 0 for each set.

In the refinement pass we do not do anything, since there are no elements from previous pass in LSP.

We transmitted 8 bits at the end of this pass

10000000

and the three lists are now

- LIP: $\{(0, 1) \rightarrow 6; (1, 0) \rightarrow -7; (1, 1) \rightarrow 7\}$
 - LIS: $\{(0, 1)\mathcal{D}; (1, 0)\mathcal{D}; (1, 1)\mathcal{D}; \}$
 - LSP: $\{(0, 0) \rightarrow 26\}$
- Second pass

We decrement n to 3, the threshold is now $2^3 = 8$.

We first examine the contents of LIP. Each is insignificant at this threshold, so we transmit three zeros.

We next examine the contents of LIS. The descendants of the coefficient at location $(0, 1)$ are 13, 10, 6, 4, the first two being significant. The set $\mathcal{D}(0, 1)$ is significant. We transmit a 1 for this and examine the offsprings of the coefficient at location $(0, 1)$. The first offspring is significant positive, we transmit a 1 followed by a 0. The same happens with the second offspring, so we send another 1 followed by 0. We also move the coordinates of these two coefficients to LSP. The next two offsprings are both insignificant, therefore we transmit a 0 for each and move them to LIP. As $\mathcal{L}(0, 1) = \{\}$, we remove $(0, 1)\mathcal{D}$ from LIS.

Looking at the other elements from LIS, both of these are insignificant, therefore we send a 0 for each.

In the refinement pass we examine the content of LSP from the previous pass. There is only one element, with value 26. The third most significant bit of 26 is 1, so we transmit a 1 ($26_{10} = 11010_2$ has the bits: $b_4 = 1, b_3 = 1, b_2 = 0, b_1 = 1, b_0 = 0$).

In this second pass we transmitted 13 bits:

$$0001101000001$$

and the three lists are now

- LIP: $\{(0, 1) \rightarrow 6; (1, 0) \rightarrow -7; (1, 1) \rightarrow 7; (1, 2) \rightarrow 6; (1, 3) \rightarrow 4\}$
- LIS: $\{(1, 0)\mathcal{D}; (1, 1)\mathcal{D}; \}$
- LSP: $\{(0, 0) \rightarrow 26; (0, 2) \rightarrow 13; (0, 3) \rightarrow 10\}$

- Third pass

We decrement n to 2, the threshold is now $2^2 = 4$. The bits sent during this pass are

$$1011101010110110011000010$$

and the list at the end of the pass are:

- LIP: $\{(3, 0) \rightarrow 2; (3, 1) \rightarrow -2; (2, 3) \rightarrow -3; (3, 2) \rightarrow -2; (3, 3) \rightarrow 0\}$

- LIS: $\{\}$

- LSP: $\{(0, 0) \rightarrow 26; (0, 2) \rightarrow 13; (0, 3) \rightarrow 10; (0, 1) \rightarrow 6; (1, 0) \rightarrow -7; (1, 1) \rightarrow 7; (1, 2) \rightarrow 6; (1, 3) \rightarrow 4; (2, 0) \rightarrow 4; (2, 1) \rightarrow -4; (2, 2) \rightarrow 4; \}$

- **Decoding**

The decoder initializes every list as the encoder:

- LIP: $\{(0, 0); (0, 1); (1, 0); (1, 1)\}$

- LIS: $\{(0, 1)\mathcal{D}; (1, 0)\mathcal{D}; (1, 1)\mathcal{D}; \}$

- LSP: $\{\}$

- **Decoding, first pass** After receiving the bit string 10000000 the decoder can change the lists to

- LIP: $\{(0, 1); (1, 0); (1, 1)\}$

- LIS: $\{(0, 1)\mathcal{D}; (1, 0)\mathcal{D}; (1, 1)\mathcal{D}; \}$
- LSP: $\{(0, 0)\}$

The reconstruction of the image at this point is

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

- **Decoding, second pass** After receiving the bit string 0001101000001 the decoder can change the lists to
 - LIP: $\{(0, 1); (1, 0); (1, 1); (1, 2); (1, 3)\}$
 - LIS: $\{(1, 0)\mathcal{D}; (1, 1)\mathcal{D}; \}$
 - LSP: $\{(0, 0); (0, 2); (0, 3)\}$

The reconstruction of the image at this point is

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

• **Decoding, third pass** After receiving the bit string 1011101010110110011000010 the decoder can change the lists to

– LIP: $\{(3, 0); (3, 1); (2, 3); (3, 2); (3, 3)\}$

– LIS: $\{\}$

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

The reconstruction of the image at this point is

26	6	14	10
-6	6	6	6
6	-6	6	0
0	0	0	0

SPIHT - practical issues

- Comparison with JPEG

Next page presents side-by-side some images compressed with JPEG (using xv) and with SPIHT to exactly the same file size. Question: which is which?

Lena 512 x 512 @ 0.31 bpp: SPIHT PSNR = 35.12 dB

JPEG PSNR = 31.8 dB (quality factor 15%).

- Subjective Evaluation

This is a serious subjective evaluation test for reconstructed images. One image was compressed with SPIHT, the other is the original. There are two pages, each showing one case of the following:

Lena 512 x 512 @ 0.5 bpp: PSNR = 37.12 dB.

Lena 512 x 512 @ 1.0 bpp: PSNR = 40.41 dB.

- SPIHT versus CREW - a fun historical view from Said and Pearlman

“In our paper ”Reversible Image compression via multiresolution representation and predictive coding”, presented at the SPIE VCIP Symposium, Cambridge, MA, Nov. 1993, we proposed a transformation for lossless compression called S+P transform (Sequential + Prediction), and presented some embedded coding results with an earlier version of SPIHT.

In March of 1995 Ricoh researchers presented at the IEEE DCC, Snowbird, UT, a paper called "CREW: Compression with reversible embedded wavelets", proposing "reversible wavelets" which are equal to a special case of our S+P transform (predictor "A" in the SPIE paper), and propose a version of EZW for progressive transmission. So, the two works are indeed very similar, but the least we can say is that our work was published more than a year earlier."

Comparison of Rate-Distortion curves with and without arithmetic coding

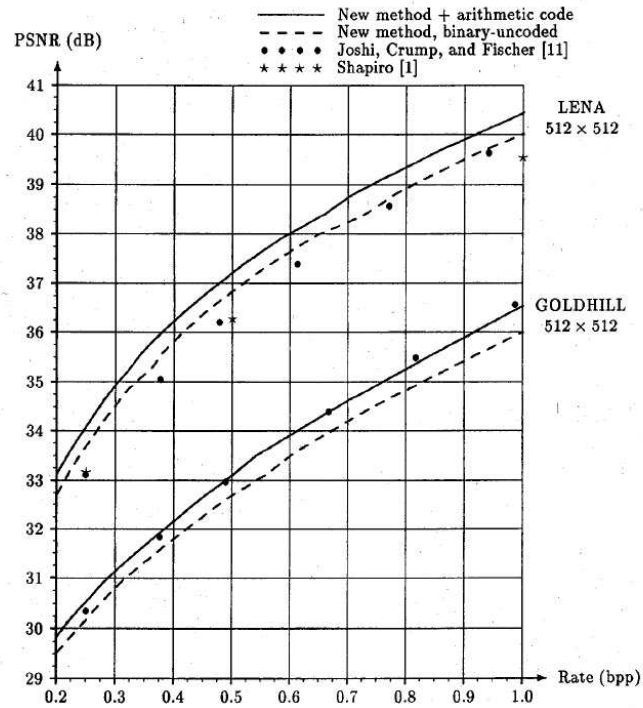


Fig. 3. Comparative evaluation of the new coding method.

Comparison of encoding and decoding times with and without arithmetic coding

TABLE I
EFFECT OF ENTROPY-CODING THE SIGNIFICANCE INFORMATION
ON THE CPU TIMES (s) TO CODE AND DECODE THE
IMAGE LENA 512×512 (IBM RS/6000 WORKSTATION)

rate (bpp)	binary uncoded		arithmetic coded	
	code	decode	code	decode
0.25	0.07	0.04	0.18	0.14
0.50	0.14	0.09	0.33	0.29
1.00	0.27	0.17	0.64	0.57

9.2 The S+P transform

An Image Multiresolution representation for Lossless and Lossy Compression

- it is similar to subband decomposition
- the transformation can be computed with only integer additions and shift operations
- A sequence of integers $c_n, n = 0, \dots, N-1$ with N even, can be represented by two sequences:

$$\begin{aligned}l_n &= \lfloor (c_{2n} + c_{2n+1})/2 \rfloor, \quad n = 0, \dots, N/2 - 1 \\h_n &= c_{2n} - c_{2n+1}, \quad n = 0, \dots, N/2 - 1\end{aligned}\tag{26}$$

The sequences l_n and h_n form the S transform of c_n . Since the sum and difference of two integers correspond to either two odd or two even integers, the truncation is used to remove the redundancy in the least significant bit.

- The division and downward truncation are realised by a single bit-shift.
- The inverse transformation is

$$\begin{aligned}c_{2n} &= l_n + \lfloor (h_n + 1)/2 \rfloor \\c_{2n+1} &= c_{2n} - h_n\end{aligned}$$

S+P cont.

- The twodimensional transformation is done by applying S+P sequentially to the rows and columns of the image, to obtain the LL,HL,LH, and HH images. Applying again S+P to the LL image we get the usual hierarchical pyramid.
- The maximum number of bits required to represent each pixel in the LL images does not change with each transformation. On the other hand, the other bands (LH,HL,HH) require a signed representation with a larger number of bits.
- Except for the truncation, the transformation can be considered a subband decomposition.
- To use S+P transformation in SPIHT, the transformation needs to be (close to) unitary (in embedded lossy compression a unitary transformation allows to set correctly the transmission priorities by using bit planes). A good approximation is

$$l_n = (c_{2n} + c_{2n+1})/\sqrt{(2)}$$
$$h_n = (c_{2n} - c_{2n+1})/\sqrt{(2)}$$

Combining the corrective factors in the two-dimensional transformation we get the scaling factors as powers of two.

9.3 Robust wavelet zerotree image compression with fixed length packetization

- the output of a wavelet zerotree coder is manipulated into fixed-length segments
- the segments are independently decodable, and errors occurring in one segment do not propagate into any other.
- the method provides both excellent compression and graceful degradation
- the scheme can perform region-based compression, in which specified portions of the image are coded to higher quality

Coping with bit errors

- retransmission protocols (ARQ) allows the decoder to request that the encoder sends a packet again. The cost is the introduced delay.
- Forward error control allows a certain number of errors to be corrected, with the cost of extra bits added.
- The target bitrates: 0.1 to 0.4 bpp. Packet length is 53 bytes (48 bytes are free to be used)
- The encoder begins with the basic SPIHT algorithm (with no arithmetic coder) and four levels of wavelet decomposition.

- For a 512×512 image there are 1024 coefficients in the low-low band; each has 255 descendants in its tree.
- The encoder encodes the image out to a target bit rate (e.g. 0.2 bpp) and stores the bits.
- Each stored bit is associated exactly with one of 1024 trees. SPIHT puts out bit streams in which the bits corresponding to different trees are interleaved, to yield progressivity.
- To achieve noise robustness, the progressivity is sacrificed, the output stream is de-interleaved and organized into 1024 variable length sub-streams, where each sub-stream contains information pertaining to only one tree of coefficients.

Region of interest (ROI)

- The user specifies the ROI and its desired quality level.
- All coefficients in ROI trees are multiplied by some factor > 1 , increasing their apparent significance. Therefore they will be coded using more bits. The side information required by the decoder includes the pre-multiplication factor and the parameters necessary to specify the region.