ELSEVIER

Signal Processing: *Image Communication* 15 (1999) 231–254

SIGNAL PROCESSING: IMAGE COMMUNICATION

www.elsevier.nl/locate/image

# Speed versus quality in low bit-rate still image compression

Amir Averbuch[a,*], Moshe Israeli[b], Francois Meyer[c]

[a] *Department of Computer Science, School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel*
[b] *Faculty of Computer Science, Technion, Haifa 32000, Israel*
[c] *Yale University School of Medicine, Department of Diagnostic Radiology, 333 Cedar Street, P.O. Box 208042, New Haven, CT 06250-8042, USA*

Received 13 March 1998

**Abstract**

This paper presents a fast and modified version of the embedded zero-tree wavelet (EZW) coding algorithm that is based on [4,5,15,20] for low bit-rate still image compression applications. The paper presents the trade-off between the image compression algorithm speed and the reconstructed image quality measured in terms of PSNR. The measurements show a performance speedup by a factor of 6 in average over the EZW [20] and the algorithm presented in [5]. This speedup causes a PSNR degradation of the reconstructed image by 0.8–1.8 dB in average. Nevertheless, the reconstructed image looks 'fine' with no particular visible artifacts even if we have an average degradation of 1 dB in PSNR. The fast algorithm with its achieved speedup is based on consecutive application of three different techniques: (1) Geometric multiresolution wavelet decomposition [15]. It contributes a speedup factor of 4 in comparison to the symmetric wavelet decomposition in [5,20]. It has a prefect reconstruction property which does not degrade the quality. (2) Modified and reduced version of the EZW algorithm for zero-tree coefficient classification. It contributes a speedup factor of 6 in comparison to the full tree processing in [5,20]. It degrades the quality of the reconstructed image. This includes efficient multiresolution data representation which enables fast and efficient traversing of the zero-tree. (3) Exact model coding of the zero-tree coefficients. It contributes a speedup factor of 15 in comparison to the adaptive modeling in [5]. This is a lossless step which reduces the compression rate because its entropy coder is sub-optimal. The paper presents a detailed description of the above algorithms accompanied by extensive performance analysis. It shows how each component contributes to the overall speedup. The fast compression has an option of manual allocation of compression bits which enables a better reconstruction quality at a pre-selected 'zoom' area. 

*Keywords:* Wavelet; Geometric decomposition; Compression; EZW

## 1. Introduction

The embedded zero-tree wavelet (EZW) coding algorithm which was first introduced by Shapiro [20] is a very effective technique for low bit-rate still image compression. Other related algorithms were presented in [4,5,12,15,18,19]. Nevertheless, it is common for all these methods that the computational requirements of a still image compression present a challenge to the hardware architecture which exist in today's market. This paper presents

* Corresponding author.

0923-5965/99/$ – see front matter 
PII: S0923-5965(98)00057-5

the trade-off between the speed of still image compression algorithm and the reconstructed image quality measured in PSNR. As we shall see, significant performance speedup of the algorithm may be achieved. This speedup, though, is accompanied by degradation of the reconstructed image quality. In order to achieve faster compression, three main techniques were used consecutively:

1. *Geometric wavelet decomposition* [15]. This technique speeds-up the multi-resolution wavelet decomposition significantly without any effect on the reconstructed image quality. In fact, it provides a perfect reconstruction when using it with a lossless coding algorithm. It enables to perform direct two-dimensional convolution while reducing substantially the number of operations and with no need to have a transpose of the image (matrix).
2. *Modified and reduced version of the EZW algorithm for zero-tree coefficient classification.* We describe a very efficient data structure in which the zero-tree is scanned. In addition, a shorter scan in each quantization iteration is performed. Only parent and its immediate children are considered. This technique degrades the reconstructed image quality since it affects coding efficiency of how the wavelet coefficients are scanned and quantized. This technique comprises the depth (how many levels in the tree) we scan in order to have a better correlated sequences for speed.
3. *A combination of exact model arithmetic coding and binary coding.* This technique affects the obtained image quality because it is sub-optimal in comparison to the adaptive model arithmetic coding used in [5]. This means that the lossless coding process of the compression algorithm is less efficient in terms of the compression ratio (or PSNR).

Each component of the algorithm contributes to the overall speedup. In general, we see that a total speedup by a factor of 6 is achieved with degradation of 0.8–1.8 dB in the reconstructed image PSNR in comparison to [5,20].

In addition to the above techniques two complementary algorithms, which also play part in the time versus quality trade-off, are presented:

1. A 'skip' option in which the image is sub-sampled one level during the wavelet decomposition and quantization. This technique reduces the amount of data in the image by a factor of 4. However, the reconstructed image quality is affected since only 25% of the original data is reproducible and the other 75% is reconstructed by interpolation.
2. A 'zoom' technique in which a pre-selected area in the image is compressed using more bits than the rest of the image. This enables the reconstruction of this area with a better quality than the rest of the image. It used the tree-structure of the reconstructed image in a very efficient way. This technique has a negligible additional computation time cost.

Comparative measurements were taken on a Silicon Graphics/Indigo workstation with a 100 MHz MIPS R4000 CPU processor. Each section in the paper contains a table of comparative performance of the algorithms in terms of the computation time and the PSNR of the reconstructed image.

We have chosen five gray-scale images for performance comparison. The conclusions of this paper, however, are applicable to color images as well. The images we used are described below. They are shown in Section 9.

1. Lenna512 – a $512 \times 512$ image with medium density of detail.
2. Barbara512 – a $512 \times 512$ image with high density of detail.
3. Barbara256 – a $256 \times 256$ image with high density of detail. This image was cut from the Barbara512 image.
4. Fabric512 – a $512 \times 512$ image which have texture and fine features.
5. Carl128 – a $128 \times 128$ image with medium density of detail. This image was taken from a video-phone sequence.

The paper is organized as follows. In Section 2 we review various compression techniques. Section 3 describes the multiresolution wavelet decomposition algorithms and the techniques which are used to improve their speed. In Section 4 the data structures which are used in the quantization part of the compression algorithm are described. Section 5

details the zero-tree classification and coding scheme. Section 6 describes the compression algorithms and Section 7 discusses the 'skip' option. Section 8 presents the 'zoom' technique and Section 9 demonstrates some performance comparisons.

## 2. Overview of compression techniques

There are many types of coding algorithms such as statistical coding, pulse code modulation (PCM), predictive coding, transform coding and interpolative and extrapolative coding [3,11,16,21,23]. Besides these categories, there are other schemes which are tailored for specific types of images. Each of these categories can be further divided based on whether the parameters of the coder are fixed or whether they change as a function of the type of data that is being coded (adaptive). In practice, a compression algorithm may include a mixture of these approaches in a compatible manner in order to achieve proper cost/performance trade-offs.

Statistical coding, which is also called entropy coding, is generally information-preserving (lossless). Statistical coding uses a probability model which estimates the entropy of the data which is being coded. They are used in wavelet-based algorithms such as [4,5,12,14,18,19], in JPEG [17,22] and in the proposed algorithm.

The proposed algorithm is a transform coding [2]. The discrete cosine transform (DCT) [2] is part of JPEG compression standard [17,22]. Despite of all their advantages, window-based transform coding such as JPEG/DCT generates blocks at low bit-rates. The blocking effect is a natural consequence of the independent processing of each block. It is percieved in images as visible discontinuities across block boundaries. The authors used the local cosine transform (LCT) as basis functions that overlap adjacent blocks, thus reducing the blocking effect [1].

Quantization of the transformed images can be accomplished via scalar quantization, vector quantization [4,8,9,13,16], or more tailored methods that utilize the internal structure of the transform such as [4,5,7,14,15,18–20] which use the properties of the multiresolution decomposition.

Fractal compression [6,10] uses a set of contractive transformations (fractal functions) that map from a defined rectangle of the real plane to smaller portions of that rectangle. These transformations are applied iteratively to the image.

## 3. Performance of the multiresolution wavelet decomposition

The multiresolution wavelet decomposition process involves recursive four-subband splitting of the image. The two-dimensional decomposition is achieved by convolution and decimation of the image, dimension by dimension (tensor product), with the filter. The level (or depth) of decomposition is a parameter to the algorithm. We used a pair of symmetric, biorthogonal decomposition and reconstruction filters of length 9 and 7, respectively. In [4,5,15,20], this filter produced the best results in terms of the reconstructed image PSNR. We implemented the wavelet decomposition in two different methods: (1) application of side by side of symmetric two-step 1-D convolution and (2) geometric decomposition of the filter which reduces the number of operations and performs a very efficient two-dimensional convolution [15]. Both methods are reversible in the sense that they enable a perfect reconstruction of the decomposed image. The difference between them lies in the computation speed: in method 1, we utilize the biorthogonal filter symmetry and pre-computation of the convolution indices which enable almost free mirroring of the image at the borders, while in method 2 the filters are factored utilizing their internal structure. The 2-D convolution and decimation is done without the need to do transpose the matrix (image). Although significant speedup is achieved by method 2, the reconstructed image quality is not affected at all.

Table 1 summarizes the time performance for different decomposition algorithms as measured on a Silicon Graphics/Indigo workstation with a 100 MHz MIPS R4000 CPU processor.

In Table 1 we can see that there is a speedup factor of 4 in favor of geometric decomposition in comparison to symmetric decomposition.

Table 1
Comparison of the performance time in seconds of the geometric decomposition and symmetric decomposition algorithms. The images were decomposed into 6 levels using a symmetric biorthogonal filter of length 9

| Image | Symmetric decomp. (in s) | Geometric decomp. (in s) |
|---|---|---|
| Lenna512 | 0.93 | 0.21 |
| Barbara512 | 0.93 | 0.21 |
| Fabric512 | 0.93 | 0.21 |
| Barbara256 | 0.23 | 0.055 |
| Carl128 | 0.04 | 0.02 |

## 4. Multiresolution data structures

The fast modified EZW algorithm uses a special data representation and data structures which enable efficient data access. In [5,20], a data representation model which is called zero-tree was used. In the improved algorithm, a transformation of the zero-tree structure is performed. This transformation reorders the zero-tree coefficients and creates a data structure which is more suitable for the procedures which follow the wavelet decomposition, specifically it fits the proposed scanning of the tree and the quantization technique.

### 4.1. The zero-tree representation

The wavelet decomposition generates a series of coefficients which represents the image after the application of the multi-level geometric two-dimensional convolution with the biorthogonal filter of length 9. The decomposition is performed with predetermined number of levels ($N$). As depicted in Fig. 1, this series of coefficients is spatially mapped as either a tree or a forest, where the coarsest level LL block (LL[$N$]) is the tree root or the forest roots (in case LL($N$) includes more than one coefficient). The emerging order of the wavelet coefficients from the geometric multiresolution decomposition does not fit efficient physical scanning of the zero-tree (as we see in Fig. 1) which is a critical component in the quantization process. Therefore, they have to go through major physical reorganization. In the following description of the algorithm, we shall refer to the LL[$N$] block as the tree root. The forest roots are a generalization of this notation. The tree root sons are the LH[$N$], HL[$N$] and HH[$N$] blocks coefficients. The coefficients in blocks LH[$i$], HL[$i$] and HH[$i$], where $1 \leqslant i \leqslant N-1$, represent the offspring of these blocks, respectively. This spatial representation is called the image zero-tree. In the zero-tree, the direct offspring of each coefficient corresponds to the pixels of the same spatial orientation in the next finer level of the decomposition. In the zero-tree, each node (except for the leaves which have none and the roots which have only three) has four direct offspring (sons).

### 4.2. Zero-tree reordering

One of the key factors in the computation speedup is to have a data representation which enables to traverse the zero-tree efficiently. The zero-tree coefficients are reordered to enable fast access to the coefficients during the classification and coding phases. In [5,20], an algorithm for coding the zero-tree coefficients was introduced. This algorithm requires an extensive traversing of the zero-tree structure, up and down the whole tree, in order to identify the significant coefficients which are subsequently quantized. The depth of the search (how many levels we go) determines the quality and the compression rate. In order to speedup this process, the direct offspring of a coefficient in the zero-tree (its sons) are stored in consecutive memory locations. This is performed for all the levels of the zero-tree (levels of the wavelet decomposition). Thus, during the decomposition of level $n$ and the creation of blocks LH[$n+1$], HL[$n+1$] and HH[$n+1$], the coefficients in these blocks are reordered. The LL[$n+1$] block is not reordered since it is consequently decomposed into blocks of the next level.

Since this improved algorithm requires extensive breadth-first-search (BFS) traversal of the zero-tree, a reordering speeds up the required travel and search. During the BFS traversal, the decomposition levels are scanned one after the other, from the coarsest level (zero-tree root) to the finest level (first

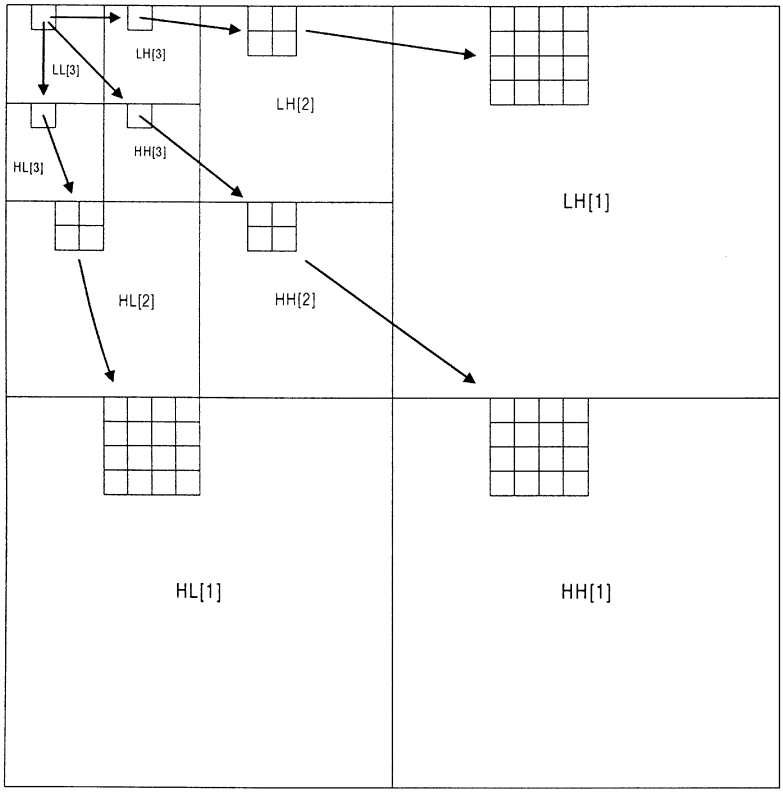



Fig. 1. Parent-offspring dependency in zero-tree for three levels.

decomposition level or zero-tree leaves). The zero-tree reordering allows manipulation and access to each father and its sons in the zero-tree in the most efficient way.

*4.2.1. Reordering from planar to hypercube blocks*

The reordering of the zero-tree coefficients is demonstrated logically in Fig. 2 and the actual physical layout of the memory is given in Fig. 3. We assume we have an array of $N$ (the depth of the decomposition) pointers where each level $j$ is accessed via the pointer $j$ in the array of pointers. A parent in row $j$ sees his four children in row $j+1$ in the following way: if he occupies location $k+$ offset 3 in level $j$ then the beginning of his first child starts in contiguous location in memory at the next level $j+1$ with offset $4k+3$. Thus, if we have parent in the $k$th location in row $j$ his four children are located contiguously in row $j+1$ starting in location $4k$. The fast transformation that takes the wavelet coefficients from their locations after the convolution and decimation to the one which is described by Fig. 3 is based on [24] and is described in detail hereafter.

In planar ordering as used in the C programming language, the index of a sample $I$ in an $N$-dimensional matrix located at index $i_k$ along dimension $k$ of length $I_k$ is

$$\text{index}\,(I_{(i_0,i_1,\ldots,i_{N-1})}) = \sum_{j=0}^{N-1} i_j \prod_{k=0}^{j-1} l_k$$

$$= i_0 + l_0(i_1 + l_1(\ldots(i_{N-2} + l_{N-2}\cdot l_{N-1})\ldots)).$$

The length of each dimension can be splitted into a multiple of a power of 2, $l_k = h_k 2^{b_k}$. Hence, the matrix can be splitted into $B = \prod_k h_k$

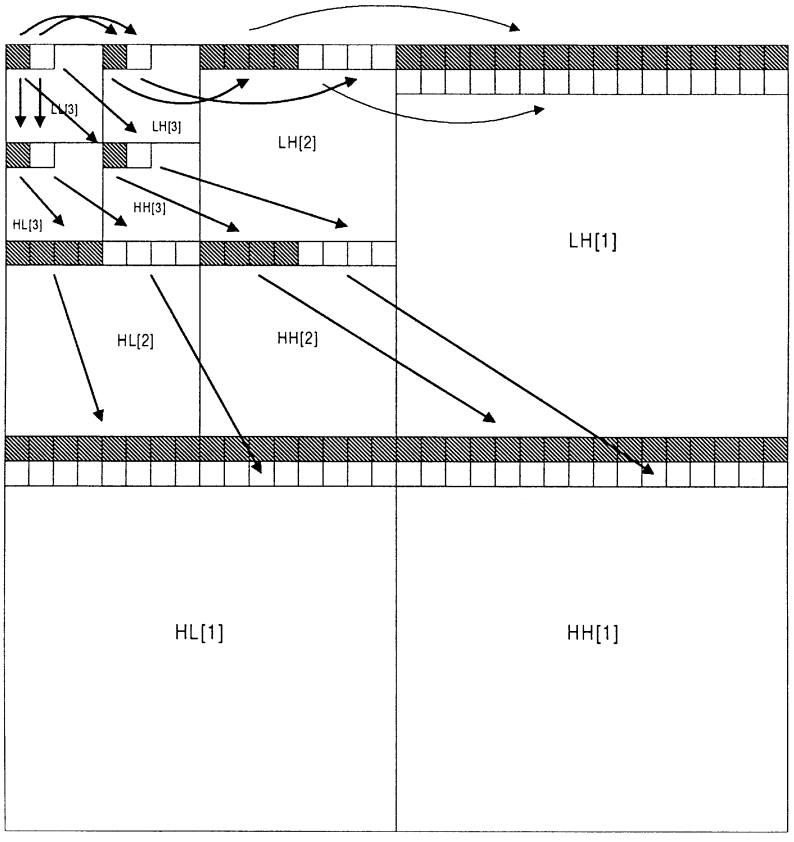



Fig. 2. Parent-offspring physical dependency in the reordered zero-tree.

sub-matrices whose size along each dimension is a power of 2.

We can now reindex each sample of the matrix within the sub-matrix it belongs to. $I_{(i_0,i_1,\dots,i_{N-1})} = I_{(q_0,q_1,\dots,q_{N-1}),(r_0,r_1,\dots,r_{N-1})}$ where $q_k = i_k \operatorname{div} h_k$ is the sub-matrix index along dimension $k$ and $r_k = i_k \bmod h_k$ is the index of the sample along dimension $k$ within the sub-matrix.

We wish to reorder the original matrix into these sub-matrices in such way that the sub-matrices are contiguous in memory and that the sub-matrix $M_{(q_0,q_1,\dots,q_{N-1})}$ holding the sample $I_{(q_0,q_1,\dots,q_{N-1}),(r_0,r_1,\dots,r_{N-1})}$ starts in memory at index

$$I_M = 2^{\sum_k b_k} \sum_{j=0}^{N-1} q_j \prod_{k=0}^{j-1} h_k.$$

The index of sample $I_{(q_0,q_1,\dots,q_{N-1}),(r_0,r_1,\dots,r_{N-1})}$ within that sub-matrix is

$$\text{index}\,(I_{(q_0,q_1,\dots,q_{N-1}),(r_0,r_1,\dots,r_{N-1})}) = \sum_{j=0}^{N-1} [r_j\, 2^{\sum_{k=1}^{j-1} b_k}].$$

Now that the original matrix has been ordered into sub-matrices whose sizes along each dimension is a power of 2. Each sub-matrix can now be reordered from a planar ordering into a hypercube like structure.

The index $r_k$ along each dimension $k$ of the sub-matrix can be seen in its binary representation $r_k = \sum_{j=0}^{b_k-1} s_{k,j} \cdot 2^j$ where $s_{k,j}$ is the value of bit $j$.

Let $T_{k,j}$ be the number of dimension $i \leqslant k$ in the sub-matrix whose size is $\geqslant 2^j$:

$$T_{k,j} = |\{b_{i \leqslant k} \geqslant j\}|.$$

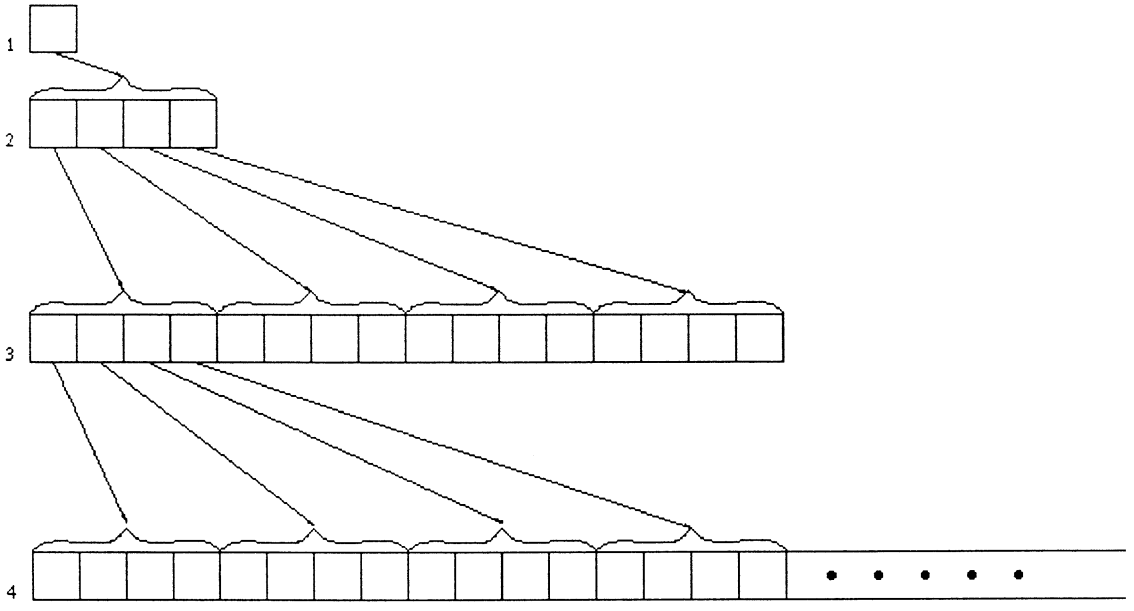



Fig. 3. The physical layout of parents and children in the zero tree.

We can now compute the new index of $I_{(q_0,q_1,\dots,q_{N-1}),(r_0,r_1,\dots,r_{N-1})}$ within the sub-matrix $M_{(q_0,q_1,\dots,q_{N-1})}$ as

$$\text{new index}\,(I_{(q_0,q_1,\dots,q_{N-1}),(r_0,r_1,\dots,r_{N-1})}) = \sum_{k,j} 2^{T_{k,l}+\sum_{i<j} T_{N-1,l}} S_{k,j}.$$

For example, if we have an image of size $3\cdot 2^i \times 5\cdot 2^j$ than it is mapped by the above transformation to 15 blocks where each block is of size $2^i \times 2^j$. Each block get reordered in the following way: $t_0y_2y_1y_0x_1x_0 \to y_2y_1x_1t_0y_0x_0$.

## 5. Zero-tree coding

As we shall see in the following sections, the zero-tree coefficients are coded and compressed during consecutive iterations of the coding algorithm. In each loop iteration, the coefficients are first coded using a classification algorithm and then entropy coded using binary and arithmetic coding methods. During the zero-tree reordering process, the coefficients are truncated to integer values. Since the values are eventually quantized and coded using entropy coding, and since the quantization process produces values in integer resolution, this truncation has no impact on the reconstructed image quality. As part of the truncation, the zero-tree coefficients are stored in special data constructs. In these constructs, a separate field is maintained for each of the following:

1. Value – the coefficient absolute value.
2. Sign – the coefficient sign.
3. Type – the coefficient type (set during the classification process).

This memory construct enables usage of a positive integer threshold during the quantization and classification phases for all the wavelet coefficients.

The coding is accomplished via a special algorithm for coefficient classification. This classification algorithm takes advantage of the correlation between the parent coefficient and its offspring in the zero-tree. This close correlation evolves from the same spatial orientation of these coefficients in respect to the original image. The present algorithm is an improvement of [5,20]. The improvement in computation time lies mostly in the

bottom-up classification which avoids the need for traversing up and down the whole zero-tree in search for all the descendants of every coefficient. In the proposed algorithm all the classification decisions are made between parent and its immediate children only.

### 5.1. Coefficient classification

The classification and coding are performed iteratively until a stopping condition is met (usually a predefined bit budget). In each iteration of the classification a threshold is used. This threshold becomes smaller with each iteration, thus refining the quantization of the coefficients which are coded. A speedup of the classification algorithm is achieved by selecting thresholds which are exponents of 2. This permits testing of a coefficient against the threshold by performing a bitwise 'and' operation. Also, if a threshold which is an exponent of 2 is initially selected, then all the subsequent thresholds would also hold this requirement since for every classification iteration, the threshold is decreased by a factor of 2 (or right-shifted).

Fig. 4 depicts a flow-chart of the classification and the coding process.

During the classification, each wavelet coefficient is tagged with one of the following types:

1. Positive (POS). A coefficient which is equal or larger than the threshold, and which has not been coded in the previous iterations (it is smaller than *threshold* · 2).
   As we shall see in the classification example given below, the positive coefficients are later refined into negative (NEG) coefficients in case the original coefficient was negative (i.e. the sign field is set).
2. Zero (Z). A coefficient which is smaller than the threshold.
3. Zero-tree root (ZTR). A coefficient which is smaller than the threshold and which fulfills the following requirements:
   3.1. All its immediate descendants (first degree only) in the zero-tree are smaller than the threshold.
   3.2. Its immediate father in the zero-tree is not a ZTR.

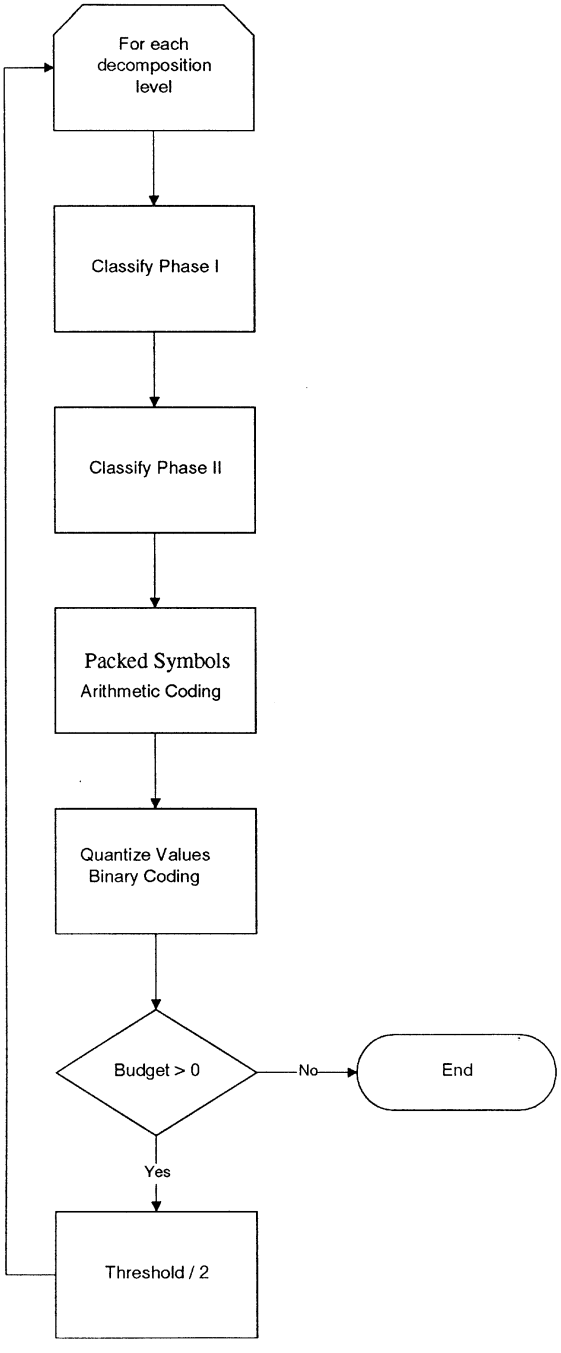



Fig. 4. General compression flow-chart.

4. Positive ZTR (PZT). A coefficient which is equal to or larger than the threshold which fulfills the requirements 3.1 and 3.2 of the ZTR coefficient above.

In analogy to the positive coefficients above, the positive ZTR coefficients are later refined into negative ZTR (NZT) coefficients in case the original coefficient was negative (i.e. the sign field is set).

5. Isolated zero (IZ). A coefficient which is smaller than the threshold but has at least one positive immediate descendant.

The coefficients which are significant to the zero-tree reconstruction are the positive, negative, zero-tree root, positive ZTR, negative ZTR, and isolated zero (all except the zero coefficients). When a coefficient is classified as significant, a pointer to that coefficient is set. For this purpose, an array of pointers to the significant coefficients is built during each classification iteration. This array enables manipulation of only the significant coefficients in the following steps of the compression process, without having to traverse the zero-tree again in order to locate them.

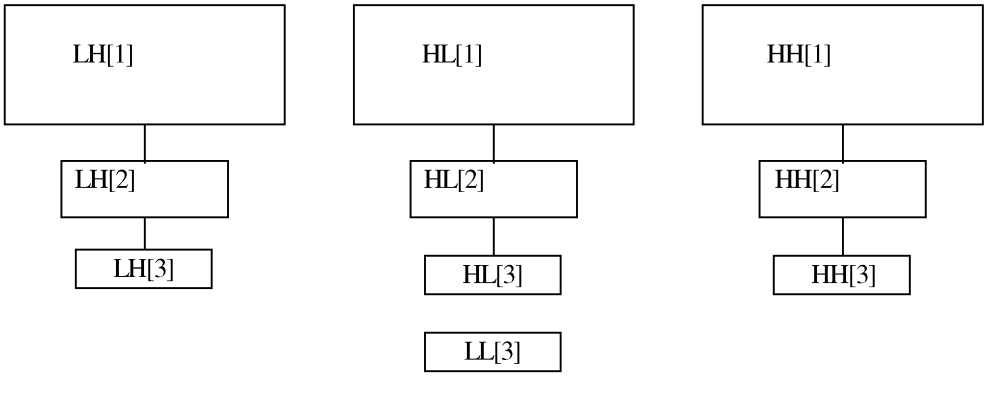
Figs. 5 and 6 depict the flow-charts of the classification algorithm phases. These flow-charts refer to as the classification of one decomposition level during one classification iteration. Thus, the 'sons' refer to the group of 4 sons (or 3 sons in case of the zero-tree root) in the zero-tree structure. A 'father' refers to the mutual father of these sons in the next coarser level of the decomposition in the zero-tree.

The algorithm above presents the following improvements over the classification algorithm presented in [5,20]:

1. The bottom-up classification algorithm avoids the need to traverse the zero-tree and to search for all the descendants of every coefficient. In each decomposition level, only the coefficients of the current level and their mutual father in the coarser decomposition level are processed. The sons of a mutual father are ordered in consecutive memory locations (as in Fig. 3). This step substantially reduces the complexity of the algorithm.
2. The introduction of the PZT and NZT types enables to have more compression. It has been observed that in many cases, and especially in the finer decomposition levels that have larger blocks, there exist many cases where a POS or a NEG coefficient can be also a ZTR. Without the PZT and NZT types, such a situation would require the coding of the types of the coefficient and of its four sons in the zero-tree (POS, ZTR, ZTR, ZTR, ZTR). With the improved algorithm, only one type (PZT or NZT) is coded.
3. Integer arithmetic and the choice of thresholds which are powers of 2.

### 5.2. Classification example

In order to demonstrate the improvement of the above classification algorithm we give an example of a $8 \times 8$ image. The zero-tree representation of this image after decomposition into three levels is given below. For this example we chose to plot the zero-tree as a tree rather than as a matrix for the convenience of the representation. The zero-tree is depicted as follows:



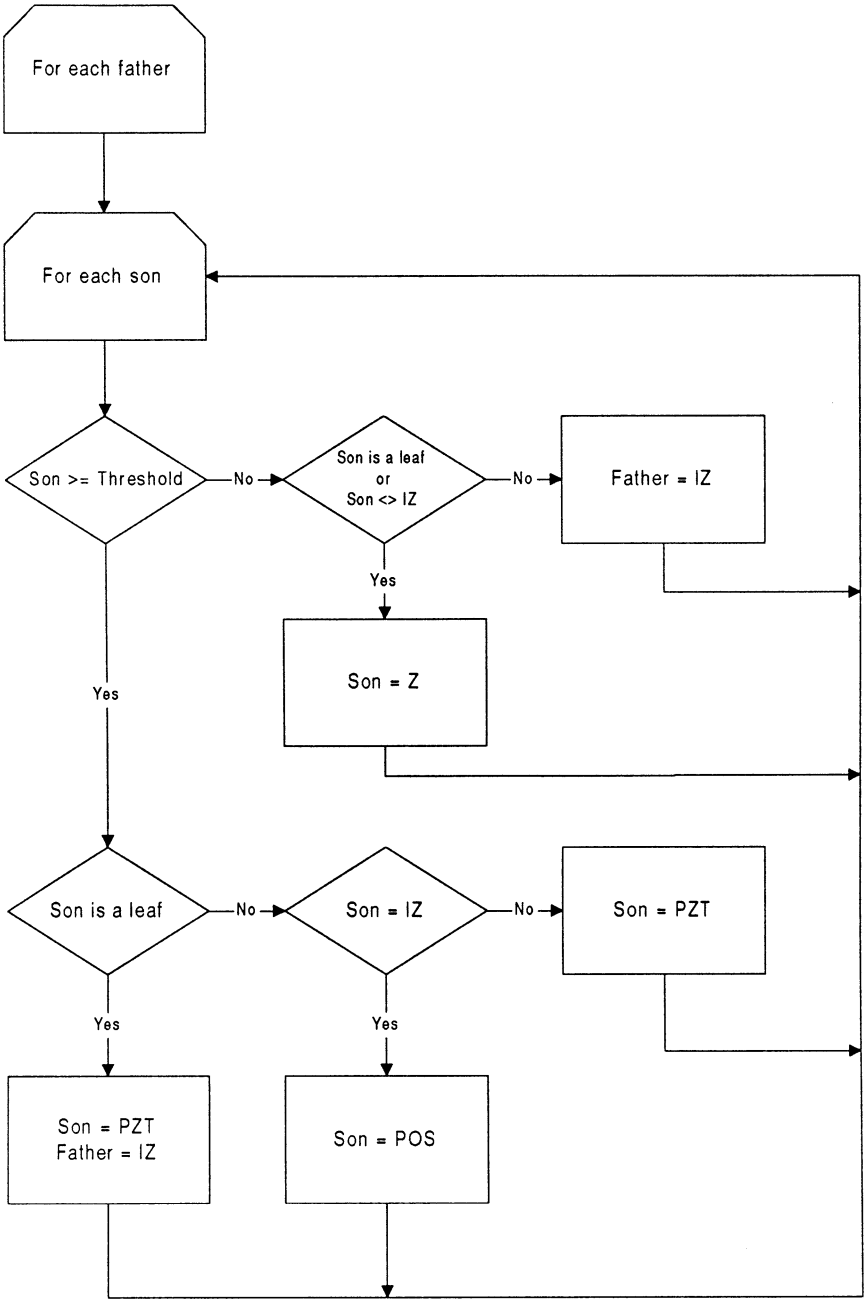



Fig. 5. Flow-chart of the classification phase I. It is done between two levels on each parent and its immediate children.

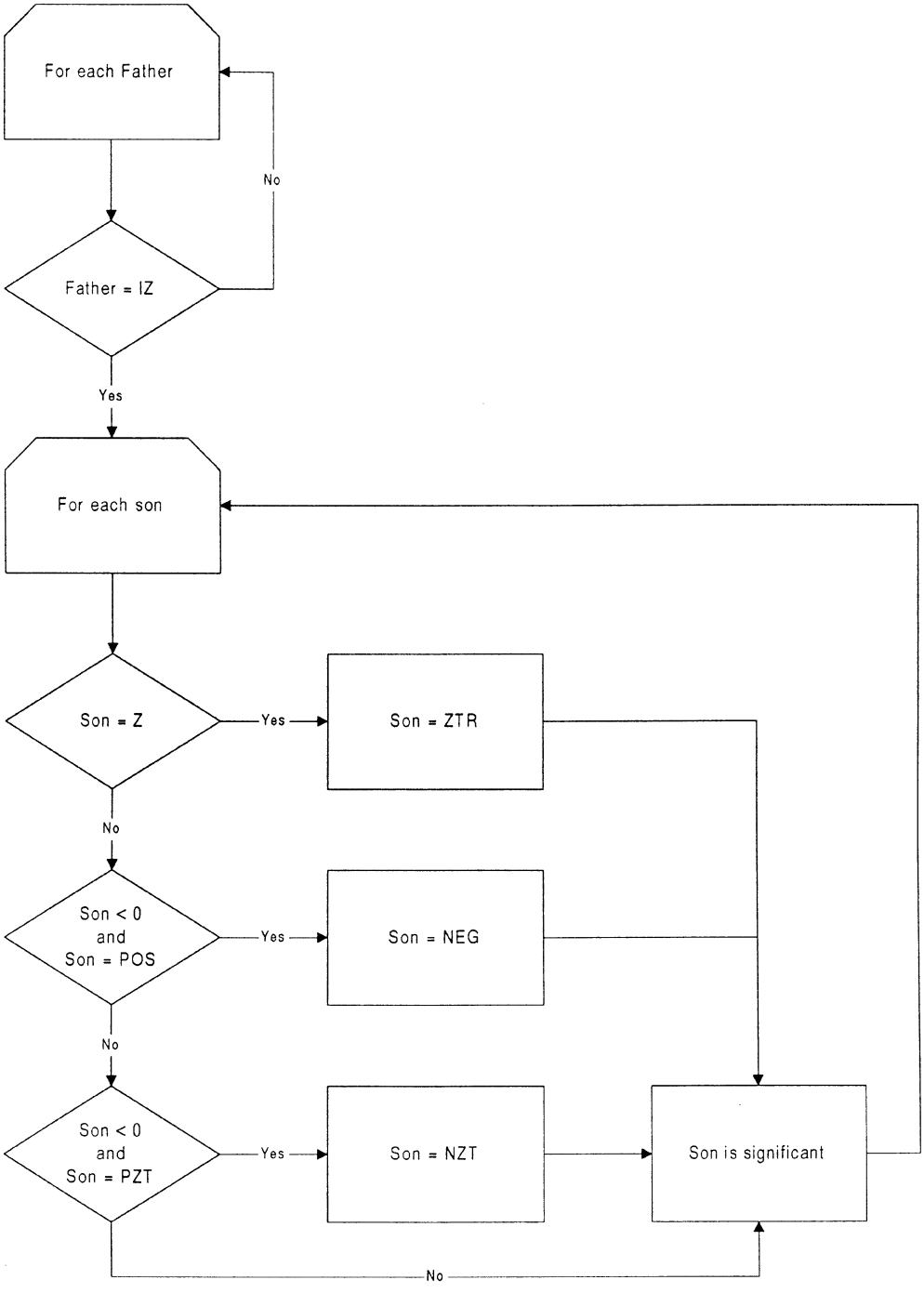
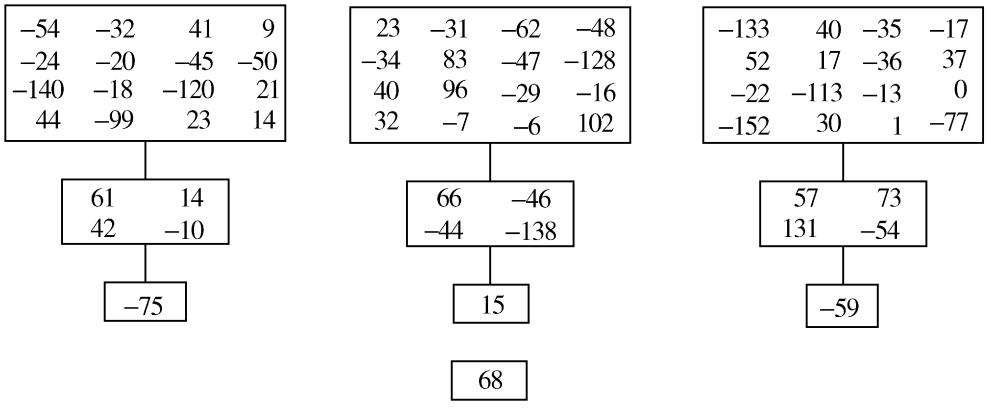
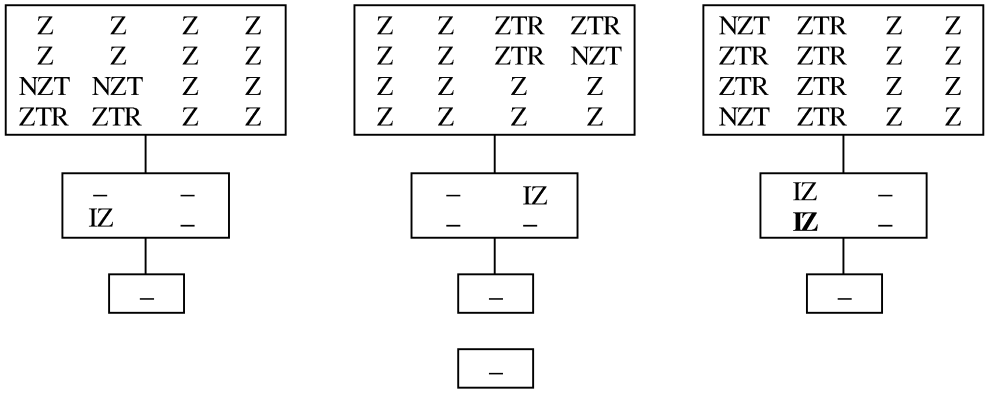



Fig. 6. Flow-chart of the classification phase II.

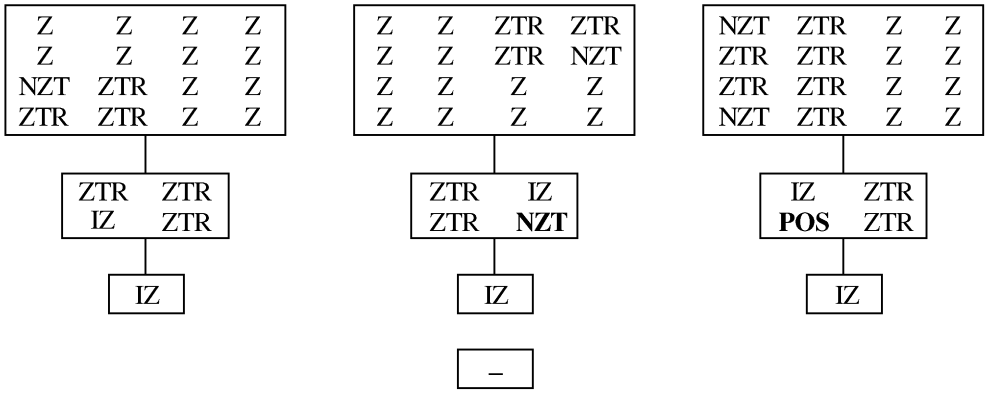
The decomposed image zero-tree is




The following is obtained after classification of the first level of the zero-tree (LH[1], HL[1] and HH[1]). The threshold is selected to be 128 since the largest coefficient is 152 (in absolute value).




After classification of the second decomposition levle (LH[2], HL[2] and HH[2]) with the same threshold (128), the following is obtained:



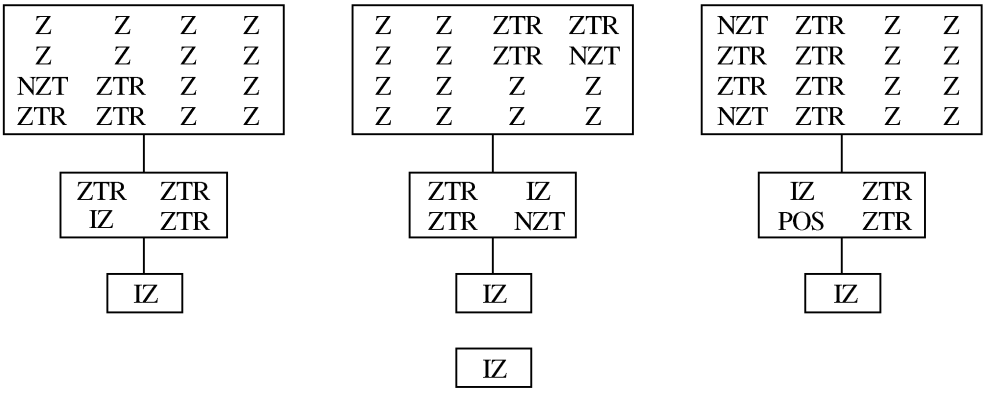

We should note two phenomena at this stage:

1. A coefficient which was temporarily classified as isolated zero (IZ) during the classification of its sons, was eventually classified as positive (POS). This coefficient is marked in the above example in bold. The temporary classification set during

the classification of the sons is only an indication that at least one offspring of this coefficient is significant, and thus it cannot be a zero-tree root (or negative ZTR or positive ZTR either). This indication propagates up the zero-tree during the classification of the upper levels until the root. This mechanism of propagation of the sub-tree status eliminates the need to traverse up and down the zero-tree in search for significant coefficients.

2. A coefficient was classified as negative zero-tree root (NZT). With the classification in [5], we obtain a negative classified coefficient and four zero-tree root sons, thus requiring compression of five types (NEG and 4 ZTR-s). With the improved classification, only one type (NZT) is compressed.

Finally, moving one more level, we obtain the classification of the zero-tree root. The complete classification of the zero-tree with threshold 128 is




Since each classification is based on the analysis between two neighboring levels we get 'weaker' classification and therefore we get less compression because we do not utilize the fact that the coefficient classification symbols (from Section 5.1) can describe the same 'trend' even beyond two neighboring levels. Thus, this 'trend' can produce better compact description of the trends in the classification tree but the computation complexity is substantially increased as we see in the next section.

### 5.3. Packing the symbols of the tree and bit-plane quantization of the wavelet values

After classification phases I and II are completed (flow-chart in Fig. 4), as it was described in Sections 5.1 and 5.2, we packed the symbols that describe the internal structure of the tree by arithmetic coding and quantize and entropy coding the values of the wavelet coefficients via bit-plane coding as described in Section 6.

### 5.4. Classification performance comparison

Table 2 summarizes the performance (in s) of the full classification algorithm which is described in [5,20] and the improved classification algorithm as measured on a Silicon Graphics/Indigo workstation with a 100 MHz MIPS R4000 CPU processor.

In Table 2 we can see that we get a speedup factor between 5 and 8 when using the improved classification algorithm. We can see that for Barbara512, which is relatively more dense in detail, the classification speedup is 5.5, while for Lenna512 with the same image size, we get a speedup of 7.3.

## 6. Entropy coding

The classification algorithm produces a stream of symbols with classification types (POS, NEG, ZTR, PZT, NZT, IZ and Z). These types represent the structure of the zero-tree at the current threshold

Table 2
Comparison of the time performance in seconds between the full classification algorithm and the improved (new) classification algorithm. The classification in both cases was performed on 6 decomposition levels with the same 9-7 filters

| Image | Comp. ratio | Full class. (in s) | New class. (in s) |
|---|---|---|---|
| Lenna512 | 40 | 3.43 | 0.47 |
| Barbara512 | 40 | 2.95 | 0.53 |
| Fabric512 | 40 | 3.02 | 0.50 |
| Barbara256 | 10 | 0.87 | 0.12 |
| Carl128 | 20 | 0.2 | 0.04 |

quantization level. We can see though that the Z types are redundant, and the zero-tree structure can be obtained using only the non-Z or significant types [5,20]. Thus, only the significant types are passed to the coding algorithm. The coding algorithm uses lossless entropy coding of these types and packed the data efficiently. In addition to the significant types, the values of the POS, NEG, PZT and NZT coefficients are also passed to the coder (the values of the ZTR, IZ and Z coefficients are smaller than the threshold). These values are coded using binary coding.

The significant types are coded using arithmetic coding. There are three main variations of the arithmetic coding technique which can be used for the final phase of the compression process:

1. *Fixed model.* A fixed distribution model of the coded symbols is a priori determined. The model is uncorrelated with the coded stream. This technique is the most time efficient since it requires no adaptive model computation and no model updates.
2. *Exact model.* As in the fixed model, a constant model is pre-determined, but here it is a priori computed using the complete input stream. Exact model coding is generally more efficient than fixed model coding in terms of the compression ratio because the model is correlated with the coded stream.
3. *Adaptive model.* Makes no a priori assumptions on the coded stream and adapts on-line the model to the variations in the stream. Adaptive model coding is usually the most efficient in terms of the achieved compression ratio, but is also the least time efficient. It is used in [5].

In order to speed up the coding algorithm, without sacrificing the compression ratio efficiency too much, exact model for the arithmetic coder of the zero-tree coefficients types is used. In the full EZW algorithm in [5,20] both the types and the values of the coefficients are coded using adaptive arithmetic coding modeling.

The first step of the coding part of the algorithm is to compute the distribution (or the model) of the significant types of the zero-tree. We use array of pointers which was set for the significant types during Phase II of the classification algorithm. While computing this model, two additional modifications are introduced:

1. The values of the POS, NEG, PZT and NZT coefficients are stored in a separate array. In the next coding iterations with the corresponding smaller thresholds, these values will also be binary coded.
2. The values of the POS, NEG, PZT and NZT coefficients in the zero-tree are zeroed (they are stored in step 1 above). These coefficients will be classified as non-significant (ZTR, IZ or Z) in the coming classification iterations and will not require allocation of bits for their type.

The array of POS, NEG, PZT and NZT values is not initialized between the coding iterations. Rather, it is augmented in every iteration with the newly identified coefficients of that iteration. In the last iteration, all the coded values are stored in this array and are coded at the resolution of the last threshold.

### 6.1. Coding performance comparison

Table 3 summarizes the performance time in seconds between the adaptive model coding algorithm which is used in [5] and the exact model/binary coding algorithm as measured on a Silicon Graphics/Indigo workstation with a 100 MHz MIPS R4000 CPU processor.

From this table we see that we get a speedup factor between 7 and 20 when using the exact

Table 3
Comparison between the performance time in seconds between the adaptive model and the exact model/binary coding algorithms

| Image | Comp. ratio | Adaptive arithmetic coding (in s) | Exact arithmetic/binary coding (in s) |
|---|---|---|---|
| Lenna512 | 40 | 1.04 | 0.05 |
| Barbara512 | 40 | 1.02 | 0.06 |
| Fabric512 | 40 | 1.02 | 0.06 |
| Barbara256 | 10 | 0.39 | 0.05 |
| Carl128 | 20 | 0.08 | 0.005 |

model/binary coder. The coder computation time is linear to the compression bit budget. This is not the case with the adaptive model coder.

## 7. Skipping the finest decomposition level

The finest decomposition level of the image which consists of the LH[1], HL[1] and HH[1] blocks represents the most detail fragments of the image. These blocks are the largest in the zero-tree structure and include 75% of the coefficients in the zero-tree. Usually, the coefficients in these blocks are relatively small in comparison to the coarser levels of the decomposition. This is due to the fact that the decomposition process is energy-compacting and as the decomposition becomes coarser, the image energy is contained in less coefficients, making their values higher in average. We experimented with sub-sampling the image one decomposition level. The following summarize the algorithmic implications of this option:

1. *Decomposition*: keep only LL[1] in the first level of the decomposition and ignore the rest.
2. *Classification and compression*: the leaves of the zero-tree are the LH[2], HL[2] and HH[2] blocks coefficients instead of LH[1], HL[1] and HH[1]. Only 25% of the coefficients in the full zero-tree are processed.
3. *Decompression*: the LH[1], HL[1] and HH[1] blocks are assumed to be zero. The LH[1], HL[1] and HH[1] are interpolated resulting in a smoothing which has a de-noising and blurring effects on the reconstructed image.

Tables 4–6 summarize the performance time in seconds of the different compression algorithms with and without the 'skip' option. The algorithms performed six levels of decomposition.

Table 4
Comparison of the performance time in seconds between the symmetric decomposition and geometric decomposition algorithms with and without the 'skip' option. The algorithms performed six levels of decomposition

| Image | Symmetric decomp. (in s) | | Geometric decomp. (in s) | |
|---|---|---|---|---|
| | No skip | Skip | No skip | Skip |
| Lenna512 | 0.93 | 0.48 | 0.21 | 0.18 |
| Barbara512 | 0.93 | 0.48 | 0.21 | 0.18 |
| Barbara256 | 0.23 | 0.12 | 0.055 | 0.048 |

Table 5
Comparison of the performance time in seconds between the full classification and the improved classification algorithms with and without the 'skip' option. The algorithms performed six levels of decomposition

| Image | Comp. ratio | Full class. (in s) | | New class. (in s) | |
|---|---|---|---|---|---|
| | | No skip | Skip | No skip | Skip |
| Lenna512 | 40 | 3.43 | 0.89 | 0.47 | 0.13 |
| Barbara512 | 40 | 2.95 | 0.76 | 0.53 | 0.14 |
| Barbara256 | 10 | 0.87 | 0.25 | 0.12 | 0.05 |

We can see that in symmetric decomposition the 'skip' option provides a speedup by a factor of 2. With the geometric decomposition, the achieved speedup using the 'skip' option is only 1.2. The 'skip' option is impractical for small images (i.e., Carl128), since the loss of resolution due to the sub-sampling would result in a blurry image.

Table 6
Comparison of the performance time in seconds between the adaptive model coder and the exact model/binary coder with and without the 'skip' option

| Image | Comp. ratio | Adaptive arithmetic coding (in s) | | Exact arithmetic/binary coding (in s) | |
|---|---|---|---|---|---|
| | | No skip | Skip | No skip | Skip |
| Lenna512 | 40 | 1.04 | 0.5 | 0.05 | 0.05 |
| Barbara512 | 40 | 1.02 | 0.44 | 0.06 | 0.06 |
| Barbara256 | 10 | 0.39 | 0.25 | 0.05 | 0.05 |

In Table 6 we can see that with both classification algorithms, the 'skip' option provides a speedup factor of 3.

From Table 6 we can see that with the exact arithmetic/binary coder, the 'skip' option provides no speedup. This supports the observation that the computation time of the new coder is linear with the compression bit budget. With the adaptive arithmetic coder, a speedup by a factor of 2 is achieved.

## 8. Zooming

### 8.1. The zoom concept

In many compression applications, we would like to allocate different bit budgets to different areas of the image. In other words, we want to force the compression algorithm to allocate more bits to a pre-selected area (for example a person's face). This motivation led us to define a parameterized "zoom area". The "zoom area" parameters are:

1. Width of the selected area.
2. Height of the selected area.
3. Location $(x, y)$ of the beginning of the selected area which is an offset from the upper left corner of the image,
4. Zoom budget: percent of the total compression budget which is allocated exclusively to the 'zoom area'.

### 8.2. Zoom area representation

The structure of the zero-tree is suitable to perform selective compression in a pre-selected area. As with the entire image, the zoom area is also represented as a zero-tree (referred to as zoom zero-tree). This zero-tree is a segment of the full zero-tree. For the construction of the zoom zero-tree, we exploit the spatial orientation of the zero-tree blocks which defines the zoom area in the same relative locations in all the levels of the zero-tree (Fig. 7). In order to obtain the zoom zero-tree, we first take the coefficients in the finest decomposition level which lie in the pre-defined zoom area. Then, we traverse up the zero-tree from these coefficients to their ancestors in the zero-tree. We stop

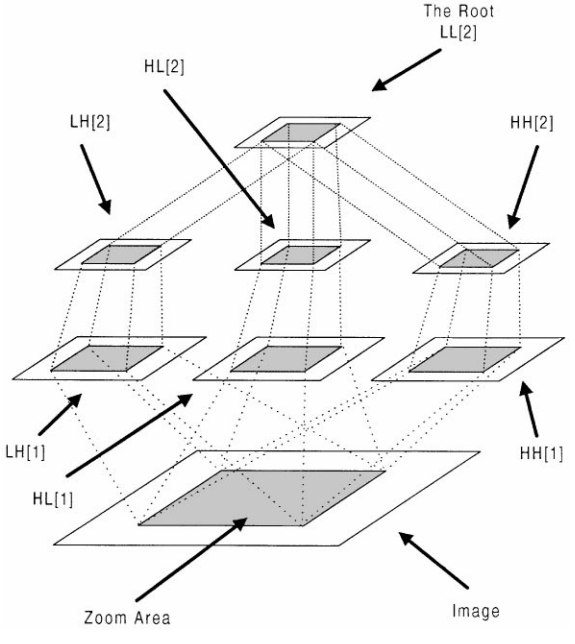



Fig. 7. Zoom zero-tree construction.

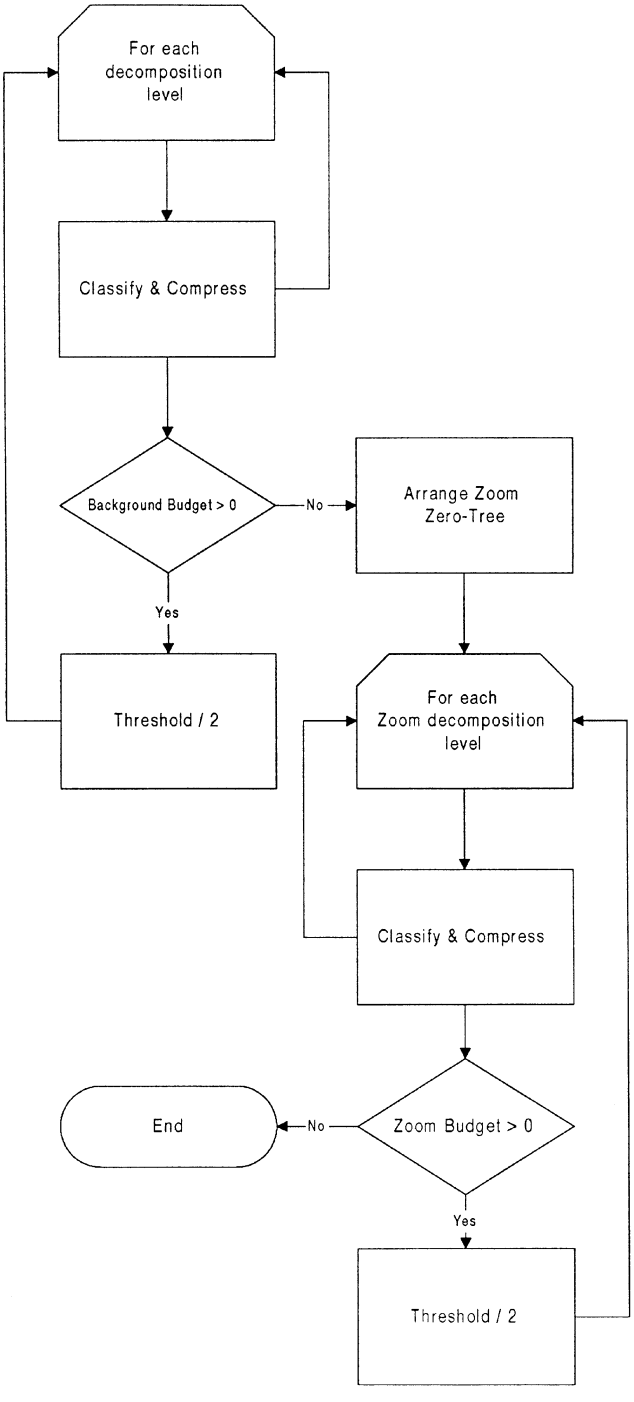



Fig. 8. Zoom compression flow-chart.

traversing the zero-tree when we either reach the zero-tree root (or LL[$N$] block) or if we are left with only one coefficient on either dimensions of the zoom area. While traversing up the zero-tree, the zoom area which is defined by the sub-tree which we have traversed may become larger than the initially defined zoom area. For instance, if we have three coefficients in the zoom zero-tree in a certain decomposition level, and in the next coarser decomposition level we have these coefficient's father in the zoom zero-tree. This actually means that the fourth son of this coefficient will also be included in the zoom zero-tree. This feature enables to use the exact same algorithms for classification and coding for the zoom area as were used for the entire image because the zero-tree structure is preserved.

Fig. 7 depicts the zoom zero-tree construction in the full zero-tree with two decomposition levels.

### 8.3. Zoom coding and compression

The zoom area is given with pre-defined bit budget which allows us to get better image quality in this zoom area. This is achieved in the following way: first, we apply the classification processes I and II on the entire image first (including the zoom area). Then the entropy coder is initially applied on the zoom area with the predefined budget for this area, and then it is applied on the image without the zoom area.

Prior to compression of the zoom area, the zoom zero-tree coefficients are copied into a separate data structure. In this structure, as with the entire image zero-tree, the sons of each coefficient are stored in consecutive memory locations. Thus a speedup of the classification and coding of the zoom zero-tree is achieved as in the full zero-tree.

The zoom zero-tree coefficients are classified and coded using the exact same algorithms as are used with the full image. Fig. 8 depicts the flow-chart of the compression algorithm using the zoom technique.

### 8.4. Zoom performance comparison

Tables 7 and 8 summarize the performance time in seconds between the different fast compression algorithms. The measurements were taken with and without the 'zoom' option.

In the tables we can see that with the improved classification algorithm, the 'zoom' option has almost no effect on the computation speed. This is also true when both the 'zoom' and 'skip' options are applied.

We can see that with the exact arithmetic and binary coder, the 'zoom' option has almost no effect on the computation speed. This is also true when both the 'zoom' and 'skip' options are applied.

## 9. Performance comparison

We have studied some images which are typical for compression implementations. The performance of the algorithms was measured both in

Table 7
The performance time in seconds between the improved classification algorithm with and without the 'zoom' option. The algorithm performed six levels of decomposition. The zoom area occupies 25% of the full image at the image center. The bit budget which is allocated to the zoom area is 50% of the total compression budget

| Image | Comp. ratio | No skip | | Skip | |
|---|---|---|---|---|---|
| | | No zoom | Zoom | No zoom | Zoom |
| Lenna512 | 40 | 0.47 | 0.5 | 0.13 | 0.13 |
| Barbara512 | 40 | 0.53 | 0.63 | 0.14 | 0.16 |
| Barbara256 | 10 | 0.12 | 0.17 | 0.05 | 0.05 |
| Carl128 | 20 | 0.04 | 0.02 | – | – |

Table 8
The performance time in seconds of the exact model/binary coder with and without the 'zoom' option. The zoom area occupies 25% of the full image at the image center. The bit budget which is allocated to the zoom area is 50% of the total compression budget

| Image | Comp. ratio | No skip | | Skip | |
|---|---|---|---|---|---|
| | | No zoom | Zoom | No zoom | Zoom |
| Lenna512 | 40 | 0.05 | 0.05 | 0.05 | 0.04 |
| Barbara512 | 40 | 0.06 | 0.05 | 0.06 | 0.05 |
| Barbara256 | 10 | 0.05 | 0.05 | 0.05 | 0.04 |
| Carl128 | 20 | 0.005 | 0.006 | – | – |

terms of the computation time and the PSNR of the reconstructed image where applicable.

The peak signal-to-noise ratio (PSNR) in dB of the reconstructed image is defined as

Mean square error

$$= MSE = \frac{\sum_{i=1}^{SizeX} \sum_{j=1}^{SizeY} (New_{i,j} - Original_{i,j})^2}{SizeX \cdot SizeY}$$

Peak signal-to-noise ratio = PSNR

$$= 20 \log_{10}\left(\frac{255}{\sqrt{MSE}}\right),$$

where $New$ is the reconstructed image, $Original$ is the original image and $SizeX$, $SizeY$ are the height and width of the image.

Table 9 summarizes the overall image quality of the fast EZW algorithm in comparison to the full compression algorithm which was described in [5] in terms of the reconstructed image PSNR in dB.

Table 9
Comparison of the overall image quality between the fast EZW algorithm, the full compression algorithm [5,20] and the SPIHT [19] algorithm in terms of the reconstructed image PSNR in dB

| Image | Comp. ratio | Fast EZW (PSNR) | Full EZW (PSNR) | SPIHT (PSNR) |
|---|---|---|---|---|
| Lenna512 | 40 | 29.6 | 31 | 30.8 |
| Barbara512 | 42 | 25.6 | 26 | 26.4 |
| Fabric512 | 40 | 30.5 | 32 | 32 |
| Barbara256 | 10 | 29.0 | 30.5 | 30.1 |
| Carl128 | 22 | 24.7 | 26.5 | 26.2 |

Table 10
Comparison of the performance time in seconds of the overall compression algorithms. The fast EZW includes the geometric decomposition, reduced classification, and exact arithmetic/binary coding. The algorithms performed image decomposition into six levels using a biorthogonal symmetric filter of length 9. Measurements were taken with and without the 'skip' option and 'zoom' option. The 'zoom area' size is 25% of the full image and at the image center. The bit budget which was allocated to the 'zoom area' is 50% of the total compression budget

| Image | Comp. ratio | Fast EZW (in s) | | | | Full EZW (in s) | |
|---|---|---|---|---|---|---|---|
| | | No skip | | Skip | | No skip | Skip |
| | | No zoom | Zoom | No zoom | Zoom | | |
| Lenna512 | 40 | 0.73 | 0.76 | 0.36 | 0.35 | 5.4 | 1.87 |
| Barbara512 | 40 | 0.8 | 0.89 | 0.38 | 0.39 | 4.9 | 1.68 |
| Fabric512 | 40 | 0.8 | | | | 5.0 | |
| Barbara256 | 10 | 0.225 | 0.275 | 0.148 | 0.138 | 1.49 | 0.62 |
| Carl128 | 20 | 0.065 | 0.046 | – | – | 0.32 | – |

(a) Original lenna 512×512

(b) Compression factor 40

(c) Compression factor of 40 with skip 1

(d) Compression ratio 40.
Zoom offset=(128,192)
Zoom size=(256,256)
Zoom budget=88%

Fig. 9. Compression of Lenna512 × 512.

The PSNR metric is not applicable when using the 'skip' or the 'zoom' options because these methods impose non-uniform bit allocation and thus the obtained image quality cannot be measured by the PSNR.

From the table we can see that for most images, the fast EZW degrades the reconstructed image PSNR by 0.8–1.8 dB. In images with high density of detail where the full compression algorithm performs 'relatively poor' (Barbara512), the fast EZW degrades the image PSNR by only 1 dB.

Table 10 summarizes the performance time in seconds of the overall compression algorithms as measured on a Silicon Graphics/Indigo workstation with a 100 MHz MIPS processor.

We can see that without the 'skip' option, in larger images, the fast EZW algorithm achieves a speedup factor of 7. In smaller images, the

(a) Original barbara 512×512

(b) Compression factor of 40

(c) Compression factor 40 with skip 1

(d) Compression ratio 40.
Zoom offset=(192,64)
Zoom size=(256,256)
Zoom budget=74%

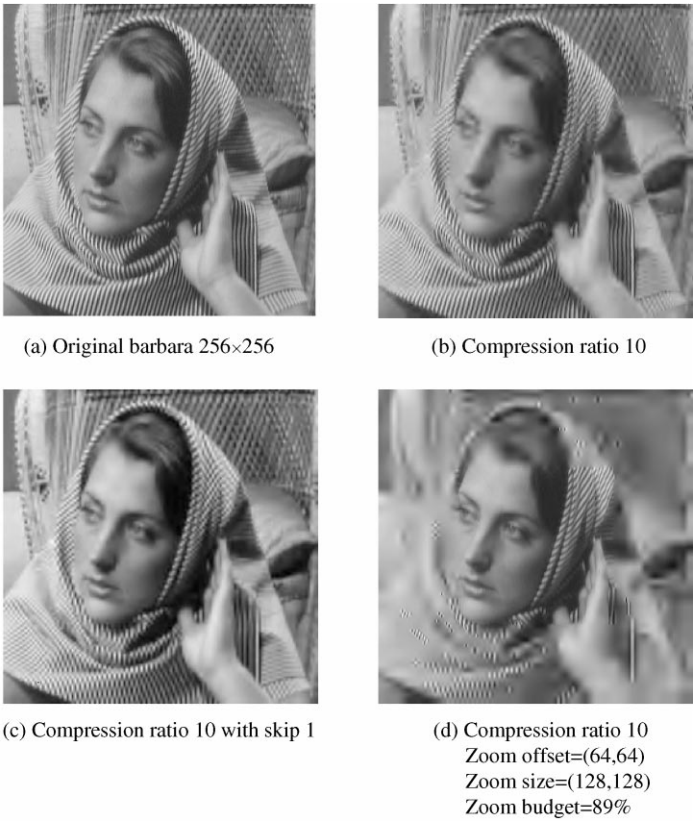
Fig. 10. Compression of Barbara512 × 512.

speedup is 6. With the 'skip' option, the speedup is decreased to 5 in larger images and 4 in smaller ones. The 'zoom' option has a negligible effect on the computation time.

Figs. 9–14 contain examples of the images which were tested with the fast compression algorithm. We present images which were generated using the 'skip' and 'zoom' options where applicable.

## 10. Conclusions

The proposed algorithm was proved to be a robust still image compression method which produces a fully embedded bit stream. It generates high quality gray-scale images in very low bit-rates. The computation speed of the algorithm is improved by using several techniques: (1) geometric



(a) Original barbara 256×256

(b) Compression ratio 10

(c) Compression ratio 10 with skip 1

(d) Compression ratio 10
Zoom offset=(64,64)
Zoom size=(128,128)
Zoom budget=89%

Fig. 11. Compression of Barbara256 × 256.

(a) Original fabric 512×512

(b) Compression ratio 40

Fig. 12. Compression of Fabric512 × 512.

Compression by JPEG.
Compression factor 40
(quality factor 5)

Compression by JPEG.
Compression factor 40
(quality factor 5)

Fig. 13. Compression with JPEG.

(a) Original carl 128×128

(b) Compression ratio 20

(c) Compression ratio 20
Zoom offset=(32,32)
Zoom size=(64,64)
Zoom budget=89%

Fig. 14. Compression of Carl128 × 128.

decomposition, (2) reduced and improved zero-tree coding, (3) efficient data representation, and (4) a combination of exact arithmetic and binary coding algorithms. With these techniques, a computation speedup of 6 is obtained. The pitfall of this speedup is the degradation of the reconstructed image PSNR by 0.8–1.8 dB.

## References

[1] G. Aharoni, A. Averbuch, R. Coifman, M. Israeli, Local cosine transform – a method for the reduction of the blocking effect in JPEG, J. Math. Imaging Vision (Special issue on Wavelets) 3 (1993) 7–38.

[2] N. Ahmed, K.R. Rao, Orthogonal Transform for Digital Signal Processing, Springer, New York, 1975.

[3] R.B. Arps, in: Binary image compression, in: Image Transmission Techniques, Academic Press, New York, 1979, pp. 219–276.
[4] A. Averbuch, D. Lazar, M. Israeli, Image compression using wavelet decomposition, IEEE Trans. Image Process. 5 (1) (January 1996) 4–15.
[5] A. Averbuch, R. Nir, Still image compression using coded multiresolution tree, Research Report, Tel Aviv University, 1995.
[6] M.F. Barnsley, L.P. Hurd, Fractal Image Compression, AK Peters, 1993.
[7] P.J. Burt, E.H. Adelson, The Laplacian pyramid as a compact image code, IEEE Trans. Commun. COM-31 (April 1983) 532–540.
[8] W.H. Equitz, Fast algorithms for vector quantization picture coding, M.S. Thesis, MIT, June 1984.
[9] A. Gersho, R. Gray, Vector Quantization and Signal Compression, Kluwer Academic Publishers, Dordrecht, 1991.
[10] A. Jacquin, Fractal image coding based on a theory of iterated contractive image transformations, Visual Comm. Image Process. PIE-1360 (1990).
[11] N.S. Jayant, P. Noll, Digital Coding of Waveforms, Prentice-Hall, Englewood Cliffs, NJ, 1984, pp. 465–482.
[12] A.S. Lewis, G. Knowles, Image compression using 2-D wavelet transform, IEEE Trans. Image Process. 1 (2) (1992) 244–250.
[13] Y. Linde, A. Buzo, R. Gray, An algorithm for vector quantizer designs, IEEE Trans. Commun. COM-28 (January 1980) 84–95.
[14] S.G. Mallat, A theory for multiresolution signal decomposition: the wavelet representation, IEEE Trans. PAMI 11 (7) (July 1989) 674–693.
[15] F. Meyer, A.R. Averbuch, J-O. Stromberg, Fast adaptive wavelet packet image compression, IEEE Trans. Image Process. to appear; appeared also in Data Compression Conf. – DCC98, Snowbird, Utah, 29 March–1 April 1998.
[16] A.N. Netravali, B.G. Haskell, Digital Pictures, Representation and Compression, Plenum Press, New York, 1991.
[17] W.B. Pennebaker, J.L. Mitchell, JPEG Still Image Data Compression Standard, Van Nostrand Reinhold, New York, 1993.
[18] A. Said, W.A. Pearlman, Image compression using the spatial orientation tree, IEEE Internat. Symp. on Circuits and Systems, Chicago, IL, May 1993, pp. 279–282.
[19] A. Said, W.A. Pearlman, A new fast and efficient image code based on set partitioning in hierarchical trees, IEEE Trans. Circuits Systems Video Technol. 6 (June 1996).
[20] J.M. Shapiro, Embedded image coding using zerotrees of wavelets coefficients, IEEE Trans. Signal Process. 41 (December 1993) 3445–3462.
[21] J.A. Storer, Data Compression: Methods and Theory, Computer Science Press, Rockville, MD, 1988.
[22] G.K. Wallace, The JPEG still picture compression standard, Comm. ACM 34 (4) (April 1991) 30–44.
[23] R. Williams, Adaptive Data Compression, Kluwer, Dordrecht, 1990.
[24] L. Woog, Private communication, 1997.