IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, June 1996

A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees*

Amir Said
Faculty of Electrical Engineering, P.O. Box 6101
State University of Campinas (UNICAMP), Campinas, SP, 13081, Brazil

William A. Pearlman
Department of Electrical, Computer, and Systems Engineering
Rensselaer Polytechnic Institute, Troy, NY, 12180, U.S.A.

Abstract

Embedded zerotree wavelet (EZW) coding, introduced by J. M. Shapiro, is a very effective and computationally simple technique for image compression. Here we offer an alternative explanation of the principles of its operation, so that the reasons for its excellent performance can be better understood. These principles are partial ordering by magnitude with a set partitioning sorting algorithm, ordered bit plane transmission, and exploitation of self-similarity across different scales of an image wavelet transform. Moreover, we present a new and different implementation, based on set partitioning in hierarchical trees (SPIHT), which provides even better performance than our previosly reported extension of the EZW that surpassed the performance of the original EZW. The image coding results, calculated from actual file sizes and images reconstructed by the decoding algorithm, are either comparable to or surpass previous results obtained through much more sophisticated and computationally complex methods. In addition, the new coding and decoding procedures are extremely fast, and they can be made even faster, with only small loss in performance, by omitting entropy coding of the bit stream by arithmetic code.

I. Introduction

Image compression techniques, especially non-reversible or lossy ones, have been known to grow computationally more complex as they grow more efficient, confirming the tenets of source coding theorems in information theory that a code for a (stationary) source approaches optimality

*This paper was presented in part at the IEEE Int. Symp. on Circuits and Systems, Chicago, IL, May 1993.

in the limit of infinite computation (source length). Notwithstanding, the image coding technique called embedded zerotree wavelet (EZW), introduced by Shapiro [1], interrupted the simultaneous progression of efficiency and complexity. This technique not only was competitive in performance with the most complex techniques, but was extremely fast in execution and produced an embedded bit stream. With an embedded bit stream, the reception of code bits can be stopped at any point and the image can be decompressed and reconstructed. Following that significant work, we developed an alternative exposition of the underlying principles of the EZW technique and presented an extension that achieved even better results [6].

In this article, we again explain that the EZW technique is based on three concepts: (1) partial ordering of the transformed image elements by magnitude, with transmission of order by a subset partitioning algorithm that is duplicated at the decoder, (2) ordered bit plane transmission of refinement bits, and (3) exploitation of the self-similarity of the image wavelet transform across different scales. As to be explained, the partial ordering is a result of comparison of transform element (coefficient) magnitudes to a set of octavely decreasing thresholds. We say that an element is significant or insignificant with respect to a given threshold, depending on whether or not it exceeds that threshold.

In this work, crucial parts of the coding process—the way subsets of coefficients are partitioned and how the significance information is conveyed—are fundamentally different from the aforementioned works. In the previous works, arithmetic coding of the bit streams was essential to compress the ordering information as conveyed by the results of the significance tests. Here the subset partitioning is so effective and the significance information so compact that even binary uncoded transmission achieves about the same or better performance than in these previous works. Moreover, the utilization of arithmetic coding can reduce the mean squared error or increase the peak signal to noise ratio (PSNR) by 0.3 to 0.6 dB for the same rate or compressed file size and achieve results which are equal to or superior to any previously reported, regardless of complexity. Execution times are also reported to indicate the rapid speed of the encoding and decoding algorithms. The transmitted code or compressed image file is completely embedded, so that a single file for an image at a given code rate can be truncated at various points and decoded to give a series of reconstructed images at lower rates. Previous versions [1, 6] could not give their best performance with a single embedded file, and required, for each rate, the optimization of a certain parameter. The new method solves this problem by changing the transmission priority, and yields, with one embedded file, its top performance for all rates.

The encoding algorithms can be stopped at any compressed file size or let run until the compressed file is a representation of a *nearly* lossless image. We say *nearly* lossless because the compression *may* not be reversible, as the wavelet transform filters, chosen for lossy coding, have non-integer tap weights and produce non-integer transform coefficients, which are truncated to finite precision. For perfectly reversible compression, one must use an integer multiresolution transform, such as the S+P transform introduced in [14], which yields excellent reversible compression results when used with the new extended EZW techniques.

This paper is organized as follows. The next section, Section II, describes an embedded coding or progressive transmission scheme that prioritizes the code bits according to their reduction in distortion. In Section III are explained the principles of partial ordering by coefficient magnitude and ordered bit plane transmission and which suggest a basis for an efficient coding method. The set partitioning sorting procedure and spatial orientation trees (called zerotrees previously) are detailed in Sections IV and V, respectively. Using the principles set forth in the previous sections, the coding and decoding algorithms are fully described in Section VI. In Section VII, rate, distortion, and execution time results are reported on the operation of the coding algorithm on test images and the decoding algorithm on the resultant compressed files. The figures on rate are calculated from actual compressed file sizes and on mean squared error or PSNR from the reconstructed images given by the decoding algorithm. Some reconstructed images are also displayed. These results are put into perspective by comparison to previous work. The conclusion of the paper is in Section VIII.

II. Progressive Image Transmission

We assume that the original image is defined by a set of pixel values $p_{i,j}$, where (i,j) is the pixel coordinate. To simplify the notation we represent two-dimensional arrays with bold letters. The coding is actually done to the array

$$\mathbf{c} = \Omega(\mathbf{p}), \tag{1}$$

where $\Omega(\cdot)$ represents a unitary hierarchical subband transformation (e.g., [4]). The two-dimensional array $\mathbf{c}$ has the same dimensions of $\mathbf{p}$, and each element $c_{i,j}$ is called *transform coefficient* at coordinate (i,j). For the purpose of coding we assume that each $c_{i,j}$ is represented with a fixed-point binary format, with a small number of bits—typically 16 or less—and can be treated as an integer.

In a progressive transmission scheme, the decoder initially sets the reconstruction vector $\hat{\mathbf{c}}$

to zero and updates its components according to the coded message. After receiving the value (approximate or exact) of some coefficients, the decoder can obtain a reconstructed image

$$\hat{\mathbf{p}} = \Omega^{-1}(\hat{\mathbf{c}}). \tag{2}$$

A major objective in a progressive transmission scheme is to select the most important information—which yields the largest distortion reduction—to be transmitted first. For this selection we use the mean squared-error (MSE) distortion measure,

$$D_{\text{mse}}(\mathbf{p} - \hat{\mathbf{p}}) = \frac{||\mathbf{p} - \hat{\mathbf{p}}||^2}{N} = \frac{1}{N} \sum_i \sum_j (p_{i,j} - \hat{p}_{i,j})^2, \tag{3}$$

where N is the number of image pixels. Furthermore, we use the fact that the Euclidean norm is invariant to the unitary transformation Ω, i.e.,

$$D_{\text{mse}}(\mathbf{p} - \hat{\mathbf{p}}) = D_{\text{mse}}(\mathbf{c} - \hat{\mathbf{c}}) = \frac{1}{N} \sum_i \sum_j (c_{i,j} - \hat{c}_{i,j})^2. \tag{4}$$

From (4) it is clear that if the exact value of the transform coefficient $c_{i,j}$ is sent to the decoder, then the MSE decreases by $|c_{i,j}|^2/N$. This means that the coefficients with larger magnitude should be transmitted first because they have a larger content of information.[1] This corresponds to the progressive transmission method proposed by DeVore *et al.* [3]. Extending their approach, we can see that the information in the value of $|c_{i,j}|$ can also be ranked according to its binary representation, and the most significant bits should be transmitted first. This idea is used, for example, in the bit-plane method for progressive transmission [2].

Following, we present a progressive transmission scheme that incorporates these two concepts: ordering the coefficients by magnitude and transmitting the most significant bits first. To simplify the exposition we first assume that the ordering information is explicitly transmitted to the decoder. Later we show a much more efficient method to code the ordering information.

III. Transmission of the Coefficient Values

Let us assume that the coefficients are ordered according to the minimum number of bits required for its magnitude binary representation, that is, ordered according to a one-to-one mapping $\eta : I \mapsto I^2$, such that

$$\left\lfloor \log_2 |c_{\eta(k)}| \right\rfloor \geq \left\lfloor \log_2 |c_{\eta(k+1)}| \right\rfloor, \quad k = 1, \ldots, N. \tag{5}$$

[1] Here the term *information* is used to indicate how much the distortion can decrease after receiving that part of the coded message.

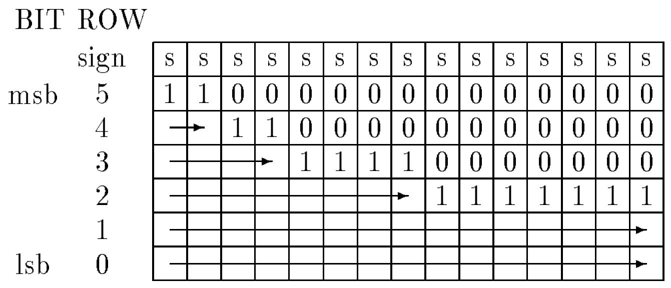


Figure 1: Binary representation of the magnitude-ordered coefficients.

Fig. 1 shows the schematic binary representation of a list of magnitude-ordered coefficients. Each column k in Fig. 1 contains the bits of $c_{\eta(k)}$. The bits in the top row indicate the sign of the coefficient. The rows are numbered from the bottom up, and the bits in the lowest row are the least significant.

Now, let us assume that, besides the ordering information, the decoder also receives the numbers 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 we have $\mu_5 = 2,\ \mu_4 = 2,\ \mu_3 = 4$, etc. Since the transformation Ω is unitary, all bits in a row have the same content of information, and the most effective order for progressive transmission is to sequentially send the bits in each row, as indicated by the arrows in Fig. 1. Note that, 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, since they can be inferred from μ_n and the ordering.

The progressive transmission method outlined above can be implemented with the following algorithm to be used by the encoder.

ALGORITHM I

1. output $n = \left\lfloor \log_2 \left(\max_{(i,j)}\{|c_{i,j}|\} \right) \right\rfloor$ 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 1, and go to Step **2**.

The algorithm stops at the desired the rate or distortion. Normally, good quality images can be recovered after a relatively small fraction of the pixel coordinates are transmitted.

The fact that this coding algorithm uses uniform scalar quantization may give the impression that it must be much inferior to other methods that use non-uniform and/or vector quantization. However, this is not the case: the ordering information makes this simple quantization method very efficient. On the other hand, a large fraction of the "bit-budget" is spent in the sorting pass, and it is there that the sophisticated coding methods are needed.

IV. Set Partitioning Sorting Algorithm

One of the main features of the proposed coding method is that the ordering data is not explicitly transmitted. Instead, it is based on the fact that the execution path of any algorithm is defined by the results of the comparisons on its branching points. So, if the encoder and decoder have the same sorting algorithm, then the decoder can duplicate the encoder's execution path if it receives the results of the magnitude comparisons, and the ordering information can be recovered from the execution path.

One important fact used in the design of the sorting algorithm is that we do not need to sort all coefficients. Actually, we need an algorithm that simply selects the coefficients such that $2^n \leq |c_{i,j}| < 2^{n+1}$, with n decremented in each pass. Given n, if $|c_{i,j}| \geq 2^n$ then we say that a coefficient is *significant*; otherwise it is called *insignificant*.

The sorting algorithm divides the set of pixels into partitioning subsets $\mathcal{T}_m$ and performs the magnitude test

$$\max_{(i,j)\in\mathcal{T}_m} \{|c_{i,j}|\} \geq 2^n \ ? \tag{6}$$

If the decoder receives a "no" to that answer (the *subset* is insignificant), then it knows that all coefficients in $\mathcal{T}_m$ are insignificant. If the answer is "yes" (the subset is significant), then a certain rule shared by the encoder and the decoder is used to partition $\mathcal{T}_m$ into new subsets $\mathcal{T}_{m,l}$, and the significance test is then applied to the new subsets. This set division process continues until the magnitude test is done to all single coordinate significant subsets in order to identify each significant coefficient.

To reduce the number of magnitude comparisons (message bits) we define a set partitioning rule that uses an expected ordering in the hierarchy defined by the subband pyramid. The objective is to create new partitions such that subsets expected to be insignificant contain a large number of elements, and subsets expected to be significant contain only one element.

To make clear the relationship between magnitude comparisons and message bits, we use the function

$$S_n(\mathcal{T}) = \begin{cases} 1, & \max\limits_{(i,j)\in\mathcal{T}} \{|c_{i,j}|\} \geq 2^n, \\ 0, & \text{otherwise}, \end{cases} \tag{7}$$

to indicate the significance of a set of coordinates $\mathcal{T}$. To simplify the notation of single pixel sets, we write $S_n(\{(i,j)\})$ as $S_n(i,j)$.

V. Spatial Orientation Trees

Normally, most of an image's energy is concentrated in the low frequency components. Consequently, the variance decreases as we move from the highest to the lowest levels of the subband pyramid. Furthermore, it has been observed that there is a spatial self-similarity between subbands, and the coefficients are expected to be better magnitude-ordered if we move downward in the pyramid following the same spatial orientation. (Note the mild requirements for ordering in (5).) For instance, large low-activity areas are expected to be identified in the highest levels of the pyramid, and they are replicated in the lower levels at the same spatial locations.

A tree structure, called *spatial orientation tree,* naturally defines the spatial relationship on the hierarchical pyramid. Fig. 2 shows how our spatial orientation tree is defined in a pyramid constructed with recursive four-subband splitting. Each node of the tree corresponds to a pixel, and is identified by the pixel coordinate. Its direct descendants (offspring) correspond to the pixels of the same spatial orientation in the next finer level of the pyramid. The tree is defined in such a way that each node has either no offspring (the leaves) or four offspring, which always form a group of 2×2 adjacent pixels. In Fig. 2 the arrows are oriented from the parent node to its four offspring. The pixels in the highest level of the pyramid are the tree roots and are also grouped in 2×2 adjacent pixels. However, their offspring branching rule is different, and in each group one of them (indicated by the star in Fig. 2) has no descendants.

The following sets of coordinates are used to present the new coding method:

- $\mathcal{O}(i,j)$: set of coordinates of all offspring of node (i,j);
- $\mathcal{D}(i,j)$: set of coordinates of all descendants of the node (i,j);
- $\mathcal{H}$: set of coordinates of all spatial orientation tree roots (nodes in the highest pyramid level);

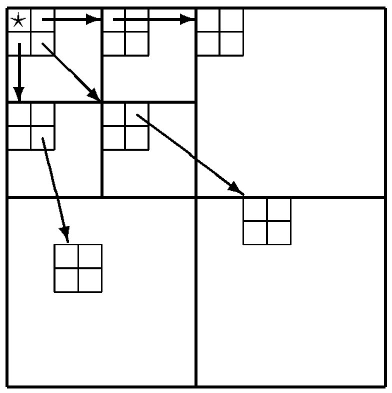


Figure 2: Examples of parent-offspring dependencies in the spatial-orientation tree.

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

For instance, except at the highest and lowest pyramid levels, we have

$$\mathcal{O}(i,j) = \{(2i,2j),(2i,2j+1),(2i+1,2j),(2i+1,2j+1)\}\,. \tag{8}$$

We use parts of the spatial orientation trees as the partitioning subsets in the sorting algorithm. The set partitioning rules are simply:

1. the initial partition is formed with the sets $\{(i,j)\}$ and $\mathcal{D}(i,j)$, for all $(i,j) \in \mathcal{H}$;
2. if $\mathcal{D}(i,j)$ is significant then it is partitioned into $\mathcal{L}(i,j)$ plus the four single-element sets with $(k,l) \in \mathcal{O}(i,j)$.
3. if $\mathcal{L}(i,j)$ is significant then it is partitioned into the four sets $\mathcal{D}(k,l)$, with $(k,l) \in \mathcal{O}(i,j)$.

VI. Coding Algorithm

Since the order in which the subsets are tested for significance is important, in a practical implementation the significance information is stored in three ordered lists, called *list of insignificant sets* (LIS), *list of insignificant pixels* (LIP), and *list of significant pixels* (LSP). In all lists each entry is identified by a coordinate (i,j), which in the LIP and LSP represents individual pixels, and in the LIS represents either the set $\mathcal{D}(i,j)$ or $\mathcal{L}(i,j)$. To differentiate between them we say that a LIS entry is of type A if it represents $\mathcal{D}(i,j)$, and of type B if it represents $\mathcal{L}(i,j)$.

During the sorting pass (see Algorithm I) the pixels in the LIP—which were insignificant in the previous pass—are tested, and those that become significant are moved to the LSP.

Similarly, sets are sequentially evaluated following the LIS order, and when a set is found to be significant it is removed from the list and partitioned. The new subsets with more than one element are added back to the LIS, while the single-coordinate sets are added to the end of the LIP or the LSP, depending whether they are insignificant or significant, respectively. The LSP contains the coordinates of the pixels that are visited in the refinement pass.

Below we present the new encoding algorithm in its entirety. It is essentially equal to Algorithm I, but uses the set-partitioning approach in its sorting pass.

ALGORITHM II

```
1. Initialization: output $n = \left\lfloor \log_2 \left( \max_{(i,j)} \{ |c_{i,j}| \} \right) \right\rfloor$; 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; 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
              * 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**.

One important characteristic of the algorithm is that the entries added to the end of the LIS in Step 2.2 are evaluated before that same sorting pass ends. So, when we say "for each entry in the LIS" we also mean those that are being added to its end. With Algorithm II the rate can be precisely controlled because the transmitted information is formed of single bits. The encoder can also use property in equation (4) to estimate the progressive distortion reduction and stop at a desired distortion value.

Note that in Algorithm II all branching conditions based on the significance data S_n—which can only be calculated with the knowledge of $c_{i,j}$—are output by the encoder. Thus, to obtain the desired decoder's algorithm, which duplicates the encoder's execution path as it sorts the significant coefficients, we simply have to replace the words *output* by *input* in Algorithm II. Comparing the algorithm above to Algorithm I, we can see that the ordering information $\eta(k)$ is recovered when the coordinates of the significant coefficients are added to the end of the LSP, that is, the coefficients pointed by the coordinates in the LSP are sorted as in (5). But note that whenever the decoder inputs data, its three control lists (LIS, LIP, and LSP) are identical to the ones used by the encoder at the moment it outputs that data, which means that the decoder indeed recovers the ordering from the execution path. It is easy to see that with this scheme coding and decoding have the same computational complexity.

An additional task done by decoder is to update the reconstructed image. For the value of n when a coordinate is moved to the LSP, it is known that $2^n \leq |c_{i,j}| < 2^{n+1}$. So, the decoder uses that information, plus the sign bit that is input just after the insertion in the LSP, to set $\hat{c}_{i,j} = \pm 1.5 \times 2^n$. Similarly, during the refinement pass the decoder adds or subtracts 2^{n-1} to $\hat{c}_{i,j}$ when it inputs the bits of the binary representation of $|c_{i,j}|$. In this manner the distortion gradually decreases during both the sorting and refinement passes.

As with any other coding method, the efficiency of Algorithm II can be improved by entropy-coding its output, but at the expense of a larger coding/decoding time. Practical experiments have shown that normally there is little to be gained by entropy-coding the coefficient signs or the bits put out during the refinement pass. On the other hand, the significance values are not equally probable, and there is a statistical dependence between $S_n(i,j)$ and $S_n(\mathcal{D}(i,j))$, and also between the significance of adjacent pixels.

We exploited this dependence using the adaptive arithmetic coding algorithm of Witten *et al.* [7]. To increase the coding efficiency, groups of 2×2 coordinates were kept together in the lists, and their significance values were coded as a single symbol by the arithmetic coding algorithm. Since the decoder only needs to know the transition from insignificant to significant (the inverse is impossible), the amount of information that needs to be coded changes according to the number m of insignificant pixels in that group, and in each case it can be conveyed by an entropy-coding alphabet with 2^m symbols. With arithmetic coding it is straightforward to use several adaptive models [7], each with 2^m symbols, $m \in \{1, 2, 3, 4\}$, to code the information in a group of 4 pixels.

By coding the significance information together the average bit rate corresponds to a m-th order entropy. At the same time, by using different models for the different number of insignificant pixels, each adaptive model contains probabilities *conditioned* to the fact that a certain number of adjacent pixels are significant or insignificant. This way the dependence between magnitudes of adjacent pixels is fully exploited. The scheme above was also used to code the significance of trees rooted in groups of 2×2 pixels.

With arithmetic entropy-coding it is still possible to produce a coded file with the exact code rate, and possibly a few unused bits to pad the file to the desired size.

VII. Numerical Results

The following results were obtained with monochrome, 8 bpp, 512×512 images. Practical tests have shown that the pyramid transformation does not have to be exactly unitary, so we used 5-level pyramids constructed with the 9/7-tap filters of [5], and using a "reflection" extension at the image edges. It is important to observe that the bit rates are not entropy estimates—they were calculated from the actual size of the compressed files. Furthermore, by using the progressive transmission ability, the sets of distortions are obtained from *the same file,* that is, the decoder read the first bytes of the file (up to the desired rate), calculated the inverse subband transformation, and then compared the recovered image with the original. The distortion is measured by the peak signal to noise ratio

$$\text{PSNR} = 10 \ \log_{10}\left(\frac{255^2}{\text{MSE}}\right) \quad \text{dB}, \tag{9}$$

where MSE denotes the mean squared-error between the original and reconstructed images.

Results are obtained both with and without entropy-coding the bits put out with Algorithm II. We call the version without entropy coding *binary-uncoded.* In Fig. 3 are plotted the PSNR versus rate obtained for the luminance (Y) component of LENA both for binary uncoded and entropy-coded using arithmetic code. Also in Fig. 3, the same is plotted for the luminance image GOLDHILL. The numerical results with arithmetic coding surpass in almost all respects the best efforts previously reported, despite their sophisticated and computationally complex algorithms (e.g., [5, 8, 9, 10, 13, 15]). Even the numbers obtained with the binary uncoded versions are superior to those in all these schemes, except possibly the arithmetic and entropy constrained trellis quantization (ACTCQ) method in [11]. PSNR versus rate points for competitive schemes, including the latter one, are also plotted in Fig. 3. The new results also surpass those in the original EZW [1], and are comparable to those for extended EZW in [6], which along with ACTCQ rely on arithmetic coding. The binary uncoded figures are only 0.3 to 0.6 dB lower in PSNR than the corresponding ones of the arithmetic coded versions, showing the efficiency of the partial ordering and set partitioning procedures. If one does not have access to the best CPU's and wishes to achieve the fastest execution, one could opt to omit arithmetic coding and suffer little consequence in PSNR degradation. Intermediary results can be obtained with, for example, Huffman entropy-coding. A recent work [12], which reports similar performance to our arithmetic coded ones at higher rates, uses arithmetic and trellis coded quantization (ACTCQ) with classification in wavelet subbands. However, at rates below about 0.5 bpp, ACTCQ is not as efficient and classification overhead is not insignificant.

Note in Fig. 3 that in both PSNR curves for the image LENA there is an almost imperceptible "dip" near 0.7 bpp. It occurs when a sorting pass begins, or equivalently, a new bit-plane begins to be coded, and is due to a discontinuity in the slope of the rate $\times$ distortion curve. In previous EZW versions [1, 6] this "dip" is much more pronounced, of up to 1 dB PSNR, meaning that their embedded files did not yield their best results for all rates. Fig. 3 shows that the new version does not present the same problem.

In Fig. 4, the original images are shown along with their corresponding reconstructions by our method (arithmetic coded only) at 0.5, 0.25, and 0.15 bpp. There are no objectionable artifacts, such as the blocking prevalent in JPEG-coded images, and even the lowest rate images show good visual quality. Table I shows the corresponding CPU times, excluding the time spent in the image transformation, for coding and decoding LENA. The pyramid transformation time was 0.2 s in an IBM RS/6000 workstation (model 590, which is particularly efficient for floating-point operations). The programs were not optimized to a commercial application level, and these times are shown just to give an indication of the method's speed. The ratio between the

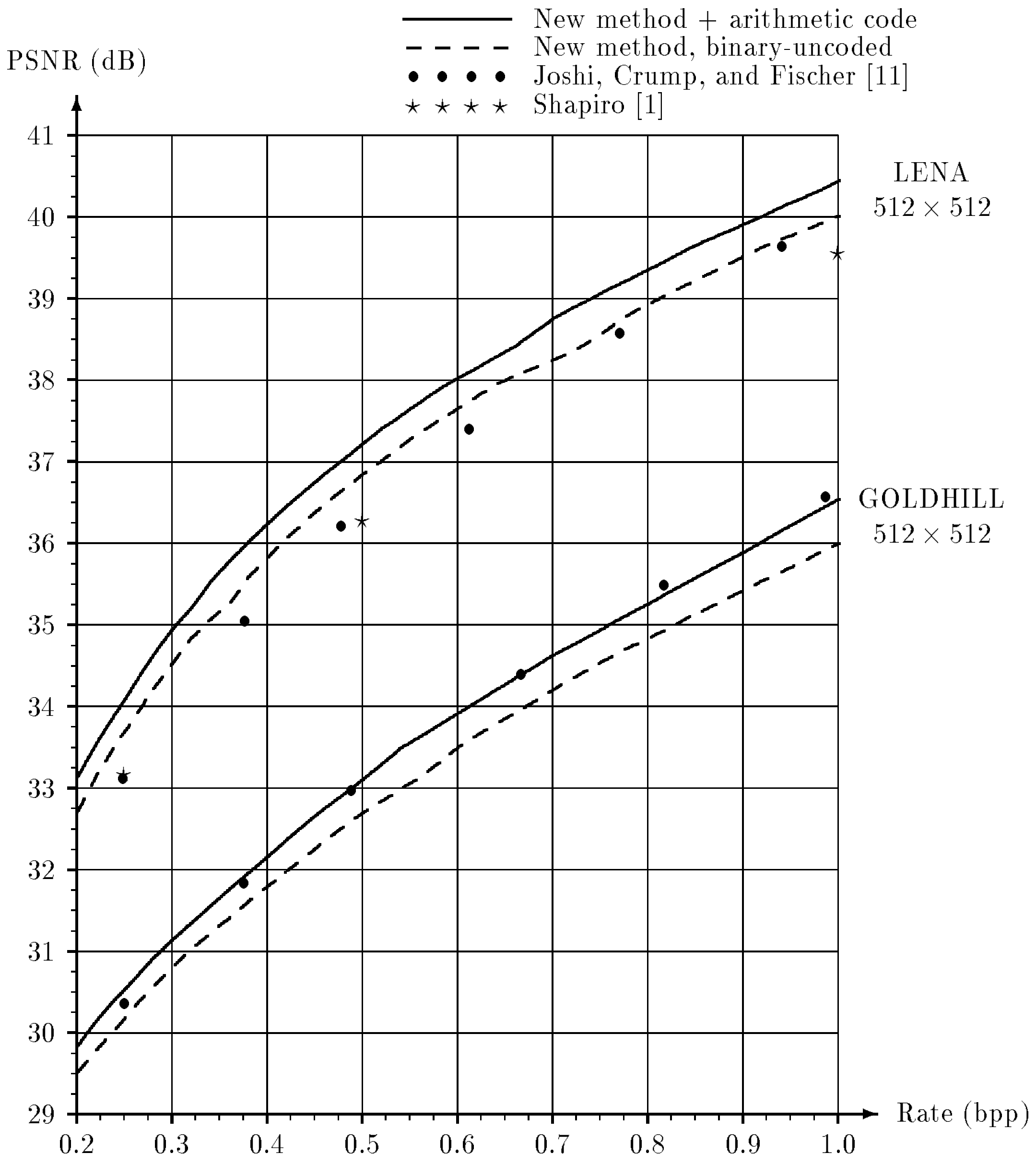


Figure 3: Comparative evaluation of the new coding method.

coding/decoding times of the different versions can change for other CPUs, with a larger speed advantage for the binary-uncoded version.

VIII. Summary and Conclusions

We have presented an algorithm that operates through set partitioning in hierarchical trees (SPIHT) and accomplishes completely embedded coding. This SPIHT algorithm uses the principles of partial ordering by magnitude, set partitioning by significance of magnitudes with

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

Table I: Effect of entropy-coding the significance data on the CPU times (s) to code and decode the image LENA 512 $\times$ 512 (IBM RS/6000 workstation).

respect to a sequence of octavely decreasing thresholds, ordered bit plane transmission, and self-similarity across scale in an image wavelet transform. The realization of these principles in matched coding and decoding algorithms is a new one and is shown to be more effective than in previous implementations of EZW coding. The image coding results in most cases surpass those reported previously on the same images, which use much more complex algorithms and do not possess the embedded coding property and precise rate control. The software and documentation, which are copyrighted and under patent application, may be accessed in the Internet site with URL of `http://ipl.rpi.edu/SPIHT` or by anonymous ftp to `ipl.rpi.edu` with the path `pub/EW_Code` in the compressed archive file `codetree.tar.gz`. (The file must be decompressed with the command `gunzip` and exploded with the command '`tar xvf`'; the instructions are in the file `codetree.doc`.) We feel that the results of this coding algorithm with its embedded code and fast execution are so impressive that it is a serious candidate for standardization in future image compression systems.

References

[1] J.M. Shapiro, "Embedded image coding using zerotrees of wavelets coefficients," *IEEE Trans. Signal Processing,* vol. 41, pp. 3445–3462, Dec. 1993.

[2] M. Rabbani, and P.W. Jones, *Digital Image Compression Techniques,* SPIE Opt. Eng. Press, Bellingham, Washington, 1991.

[3] R.A. DeVore, B. Jawerth, and B.J. Lucier, "Image compression through wavelet transform coding," *IEEE Trans. Inform. Theory,* vol. 38, pp. 719–746, March 1992.

[4] E.H. Adelson, E. Simoncelli, and R. Hingorani, "Orthogonal pyramid transforms for image coding," *Proc. SPIE*, vol. 845 – Visual Comm. and Image Proc. II, Cambridge, MA, pp. 50–58, Oct. 1987.

[5] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. Image Processing,* vol. 1, pp. 205–220, April 1992.

[6] A. Said and W.A. Pearlman, "Image compression using the spatial-orientation tree," *IEEE Int. Symp. on Circuits and Systems,* Chicago, IL, pp. 279–282, May, 1993.

[7] I.H. Witten, R.M. Neal, and J.G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, pp. 520–540, June 1987.

[8] P. Sriram and M.W. Marcellin, "Wavelet coding of images using trellis coded quantization," *SPIE Conf. on Visual Inform. Process.*, Orlando, FL, pp. 238–247, April 1992; also "Image coding using wavelet transforms and entropy-constrained trellis quantization," *IEEE Trans. Image Processing*, vol. 4, pp. 725–733, June 1995.

[9] Y.H. Kim and J.W. Modestino, "Adaptive entropy coded subband coding of images," *IEEE Trans. Image Processing*, vol. IP-1, pp. 31–48, Jan. 1992.

[10] N. Tanabe and N. Farvardin, "Subband image coding using entropy-constrained quantization over noisy channels," *IEEE J. Select. Areas in Commun.*, vol. 10, pp. 926–943, June 1992.

[11] R.L. Joshi, V.J. Crump, and T.R. Fischer, "Image subband coding using arithmetic and trellis coded quantization," *IEEE Trans. Circ. & Syst. Video Tech.*, vol. 5, pp. 515–523, Dec. 1995.

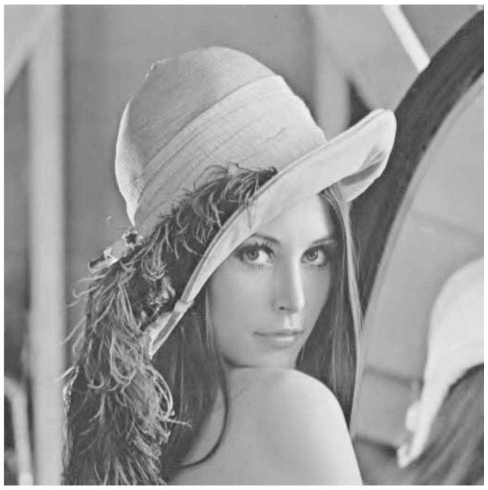
[12] R.L. Joshi, T.R. Fischer, R.H. Bamberger, "Optimum classification in subband coding of images," *Proc. 1994 IEEE Int. Conf. on Image Processing*, vol. II, pp. 883–887, Austin, TX, Nov. 1994.

[13] J.H. Kasner and M.W. Marcellin, "Adaptive wavelet coding of images," *Proc. 1994 IEEE Conf. on Image Processing*, vol. 3, pp. 358–362, Austin, TX, Nov. 1994.

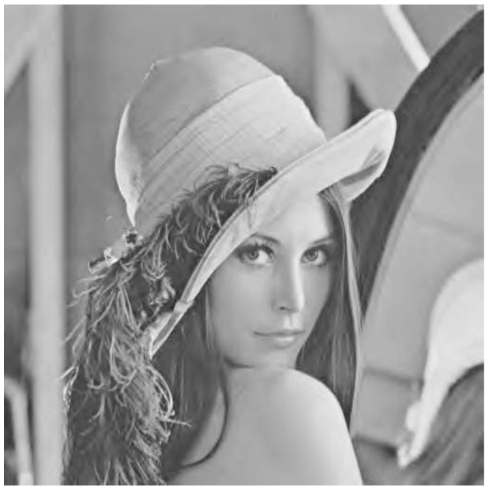
[14] A. Said and W.A. Pearlman, "Reversible Image compression via multiresolution representation and predictive coding," *Proc. SPIE Conf. Visual Communications and Image Processing '93*, Proc. SPIE 2094, pp. 664–674, Cambridge, MA, Nov. 1993.

[15] D.P. de Garrido, W.A. Pearlman and W.A. Finamore, "A clustering algorithm for entropy-constrained vector quantizer design with applications in coding image pyramids," *IEEE Trans. Circ. and Syst. Video Tech.*, vol. 5, pp. 83–95, April 1995.

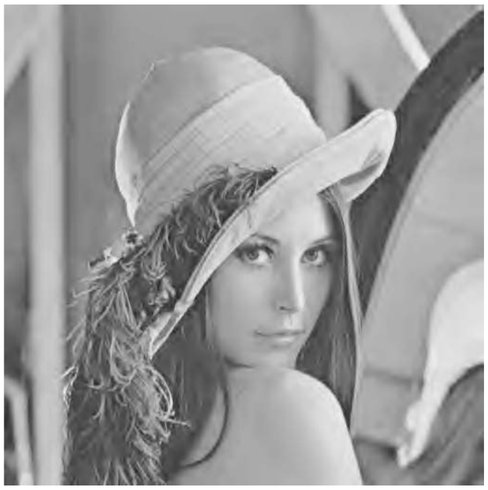
(a) Original LENA



(b) Rate = 0.5 bpp, PSNR = 37.2 dB



(c) Rate = 0.25 bpp, PSNR = 34.1 dB



(d) Rate = 0.15 bpp, PSNR = 31.9 dB

Figure 4: Images obtained with the arithmetic code version of the new coding method.