A Low Memory Zerotree Coding for Arbitrarily Shaped Objects

Chorng-Yann Su and Bing-Fei Wu, *Senior Member, IEEE*

***Abstract*—The Set Partitioning In Hierarchical Trees (SPIHT) algorithm is a computationally simple and efficient zerotree coding technique for image compression. However, high working memory requirement is its main drawback for hardware realization. In this study, we present a low memory zerotree coder (LMZC), which requires much less working memory than SPIHT. The LMZC coding algorithm abandons the use of lists, defines a different tree structure, and merges the sorting pass and the refinement pass together. The main techniques of LMZC are the recursive programming and a top-bit scheme (TBS). In TBS, the top bits of transformed coefficients are used to store the coding status of coefficients instead of the lists used in SPIHT. In order to achieve high coding efficiency, shape-adaptive discrete wavelet transforms are used to transformation arbitrarily shaped objects. A compact emplacement of the transformed coefficients is also proposed to further reduce working memory. The LMZC carefully treats "don't care" nodes in the wavelet tree and does not use bits to code such nodes. Comparison of LMZC with SPIHT shows that for coding a 768 × 512 color image, LMZC saves at least 5.3 MBytes[1] of memory but only increases a little execution time and reduces minor peak signal-to noise ratio (PSNR) values, thereby making it highly promising for some memory limited applications.**

***Index Terms*—Arbitrarily shaped image coding, image compression, low memory, recursive programming, shape adaptive zerotree coding.**

I. Introduction

TO SAVE the transmission time or the storage space of an image, nowadays many people widely use the image compression technique to transmit or to store an image. Among various compression techniques, transform coding is a favorite technique. In the past decade, the discrete cosine transform (DCT) has been the most popular transform because it provides an almost optimal performance and can be implemented at a reasonable cost. However, discrete wavelet transform (DWT) has been widely used recently because of its ability to solve the blocking effect introduced by DCT and its suitability in multi-resolution analysis. By taking advantage of DWT, the zerotree coding technique has proven that it is not only computationally simple but also is very effective in compression. In addition, its embedded coding property is beneficial to progressive transmission.

In a related work, the original zerotree algorithm is called embedded zerotree wavelet (EZW), introduced by Shapiro [1]. Said and Pearlman [2] further enhanced the performance of EZW by presenting a more efficient and faster implementation called set partitioning in hierarchical tree (SPIHT). The SPIHT's performance in peak signal-to noise ratio (PSNR) and in execution time is almost the best one among those related works. Because EZW-like coder can provide an efficient coding for still images and visual textures, the MPEG-4 standard also uses it in the visual texture mode [3]. In addition, EZW-like coder can also provide spatial and quality scalabilities, which are the desired functionalities of the MPEG-4 standard [4] and JPEG2000 standard [5]–[7]. Since the excellent performance of EZW-like coders, several other coding algorithms have been developed based on the zerotree theory [8]–[14].

However, the previous zerotree coders generally require some lists to store the states of coefficients and the coordinates of partitioning sets during coding, which leads to high memory requirement and high cost in terms of hardware realization. For example, SPIHT requires three lists, called list of significant pixels (LSP), lost of insignificant pixels (LIP), and list of insignificant sets (LIS). For a 768 × 512 color image, each entry of the lists requires at least $10+9=19$ bits to store the coordinates, where 10 bits are required to represent the column value in the range 0 to 767 and 9 bits are needed for 512 rows. Given that and the total number of list entries is approximately twice of the total number of coefficients, the total memory required is 5.3 MBytes.[2] If integer variables with 4 bytes are used to store the coordinates, it will require about 9.4 MBytes.[3] In addition to this high memory requirement, another drawback of SPIHT is that the number of entries increases as the coded bit rate increases. Therefore, one should prepare enough memory for the application of coding at various bit rates.

To reduce the memory requirement, one method is to use other coding algorithms, for examples, Embedded Block Coding with Optimized Truncation of the embedded bit-streams (EBCOT) [5], [15] and space-frequency decomposition (SFD) [16]. In EBCOT, each subband is partitioned into relatively small blocks of samples, which is called code-blocks. EBCOT generates a separate highly scalable bit-stream for each code-block. Since it is possible to independently compress relatively small code-blocks, EBCOT needs only a small amount of buffer for rate–distortion optimization. However, a high amount of buffer is required for a global optimization

Manuscript received June 26, 2000; revised September 19, 2002. This work was supported by the Program for Promoting Academic Excellence of Universities under Grant 91X101EX-91-E-FA06-4-4. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Touradj Ebrahimi.

C.-Y. Su is with the Department of Industrial Education, National Taiwan Normal University, Taipei 106, Taiwan, R.O.C. (e-mail: scy@cc.ntnu.edu.tw).

B.-F. Wu is with the Department of Electrical and Control Engineering, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: bwu@cc.nctu.edu.tw).

Digital Object Identifier 10.1109/TIP.2002.807359

[1]19 (bits) × 768 × 512 × 3 (colors) × 2/8 bits/1K/1K = 5.3 MB

[2]19 (bits) × 768 × 512 × 3(colors) × 2/8 bits/1K/1K = 5.3 MB

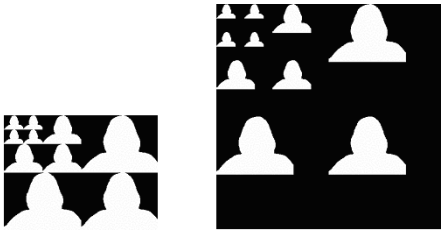
[3]4 (bytes) × 768 × 512 × 3 (colors) × 2/1K/1K = 9.4 MB

because a rate control scheme has to decide which coding passes of a code block should be included into the output bit stream [6]. If not, the rate–distortion information for each truncation point of coding passes for each code block should be at least saved. In SFD, the wavelet tree is trimmed, so that the working memory can be reduced. However, these two algorithms generally are based on a rate–distortion sense, and therefore modest computational complexity is required to find the optimal operating point in the rate–distortion curve.

In this study, we present another method, called a low memory zerotree coder (LMZC). The algorithm of LMZC is similar to SPIHT but different in implementation. The differences between LMZC and SPIHT are on threefold. First, LMZC abandons the use of lists during coding and decoding but preserves the embedded coding property of SPIHT. The main techniques in LMZC are the recursive programming and a top-bit scheme (TBS). Since the wavelet tree structure is of similarity, it is suitable for the use of recursive programming. In TBS, the top bits of transformed coefficients are used to store the status of coefficients instead of the lists. Therefore, no additional memory is needed. Second, a new tree structure like EZW's tree structure [1] but not SPIHT's tree structure is proposed. The new tree structure allows a parent node having at most nine child nodes, and it is suitable for further reducing the working memory for coding an arbitrarily shaped image. Third, LMZC merges the sorting pass and the refinement pass to one single pass to reduce the execution time.

The idea of low memory zerotree coding is originally proposed in our previous work [13]. In that work, we designed the coder for coding rectangular images. Lin and Burgess extended our previous work by proposing a listless zerotree coder (LZC) for coding color images [10]. However, the statistical dependencies between adjacent pixels and between adjacent trees are not exploited in LZC. In this study, we exploit these dependencies by using the multi-models of adaptive arithmetic coding algorithm of Witten *et al.* [17], and extend the idea of low memory zerotree coder to the coding of arbitrarily shaped visual objects. The novelties of the new design are on twofold. First, the shape-adaptive wavelet transform [9], [18] is used to transformation arbitrarily shaped visual objects for getting a better coding efficiency than other transformation schemes. We carefully treat "don't care" nodes in the wavelet tree and use no bits to code such nodes. Second, a compact emplacement [see Fig. 1(a)] is proposed to store the transformed coefficients instead of the regular emplacement [see Fig. 1(b)], so that the working memory can be further reduced.

Fig. 1 shows the corresponding binary alpha planes of an object after a three-scale decomposition. The white regions represent the object regions and black regions are the backgrounds. Fig. 1(a) is the result of the compact emplacement, whereas Fig. 1(b) is the result of the regular emplacement. The size of the bounding box in the compact emplacement is just equal to the size of the smallest box to bind the coded object, whereas the size of the bounding box in the regular emplacement needs to be adjusted to the size of a power of 2 so that the typical one-parent to four-children relation is preserved. By using the compact emplacement, the working memory of coding an arbitrarily shaped image is largely reduced. For example, a 352×261 arbitrarily shaped color image with five-scale decomposition requires a 512×512 bounding box to store the transformed coefficients in the regular emplacement, which needs 3 MBytes[4] (we assume that each transformed coefficient requires four bytes to store its value). However, the size of bounding box is 352×261 in the compact emplacement, which only needs 1.1 MBytes.[5] We can save 1.9 Mbytes or equal to 63% of memory. The penalty of using the compact emplacement is that it requires more execution time to find the number of child nodes of each parent node than the regular emplacement, because the one-parent to four-children relation is broken in the compact emplacement.



(a) (b)

Fig. 1. The corresponding alpha planes of an arbitrarily shaped image after three-scale decomposition (a) for the compact emplacement; (b) for the regular emplacement. The size of the smallest box to bind the image is 352×261. The size of the bounding box in the compact emplacement is also 352×261 but it is 512×512 in the regular emplacement.

The organization of this study is as follows. In the following section, we briefly introduce the algorithm of SPIHT for completeness. Section III describes the method of decomposing an arbitrarily shaped visual object in the compact emplacement. Section IV addresses the coding algorithm of LMZC. In Section V, we analyze the computational complexity of the proposed method. In Section VI, experimental results are shown. Finally, we make the concluding remarks in Section VII.

II. Brief Review of SPIHT

For completeness, we briefly introduce the SPIHT coding algorithm in this section. More details of SPIHT can be referred to [2]. Fig. 2 illustrates the wavelet tree structure of a typical three-scale pyramidal decomposition of an image. The image is generated by three stages of two-dimensional (2-D) DWT [19]. The notations LL_i, HL_i, LH_i, and HH_i denote the output channels from the ith stage. The parent-offspring dependency for tree structures is also demonstrated. Each node has either four offspring or no offspring. The nodes has no offspring are located on $Layer_1$ (i.e., the bands HL_1, LH_1, and HH_1) and some of them are located on the highest layer (one of them indicated by the "$*$" in Fig. 2).

We call a node (i.e., transformed coefficient) $C(i, j)$ at a coarse scale a parent. All nodes at the next finer scale with the same spatial location, and of similar orientation are called children, this set denoted $O(i, j)$. More precisely, $O(i, j) = \{C(2i, 2j), C(2i, 2j+1), C(2i+1, 2j), C(2i+1, 2j+1)\}$ except the nodes at the highest layer ($Layer_4$ in Fig. 2) and the lowest layer ($Layer_1$ in Fig. 2). All nodes at all finer scales with the same spatial location, and of similar orientation are

[4] 4 (bytes) $\times$ 512 $\times$ 512 $\times$ 3 (colors)/1K/1K = 3 MB

[5] 4 (bytes) $\times$ 352 $\times$ 261 $\times$ 3 (colors)/1K/1K = 1.1 MB

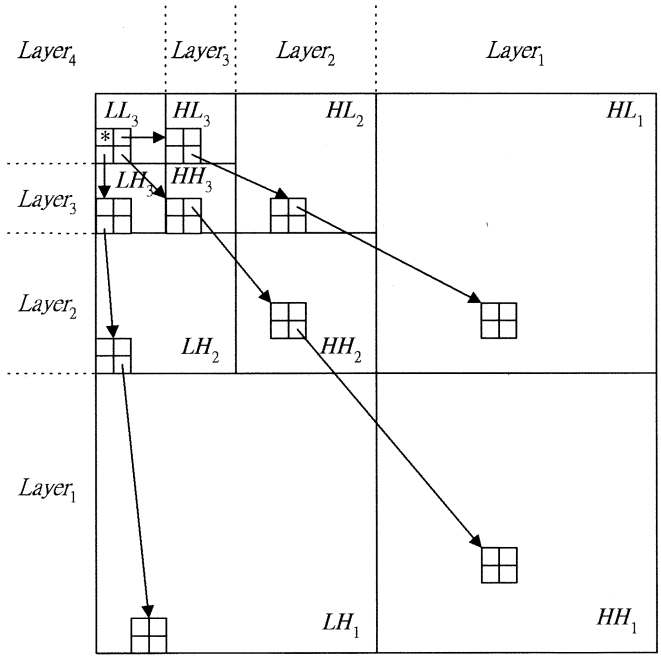


Fig. 2. Examples of the SPIHT tree structure in a typical three-scale pyramidal decomposition of an image. The arrows are oriented from the parent node to its offspring.

called descendant, denoted $D(i, j)$. A set $L(i, j)$ is defined as $L(i, j) = D(i, j) - O(i, j)$, and the set $\boldsymbol{H}$ is the group of coordinates of all the tree roots (nodes in the highest layer). We also refer to a node or a set as significant if the result of the significant test (1) is 1

$$S_n(X(i, j)) = \begin{cases} 1, & \text{if } \max\limits_{C(k, l) \in X(i, j)} \{|C(k, l)|\} \geq 2^n, \\ 0, & \text{otherwise,} \end{cases} \quad (1)$$

where $X(i, j)$ represents $C(i, j)$, $D(i, j)$, or $L(i, j)$. Equation (1) indicates that if the coefficient with maximum magnitude in a set is significant, then the significant test results in 1.

If $D(i, j)$ is significant, then it is partitioned into $O(i, j)$, and $L(i, j)$ if $L(i, j)$ exists. If not, it is a zerotree of type A. If $L(i, j)$ is significant, then it is partitioned into $\{D(2i, 2j), D(2i, 2j + 1), D(2i + 1, 2j), D(2i + 1, 2j + 1)\}$ except the coordinates in the highest layer. Otherwise, it is a zerotree of type B. If we encounter a zerotree, we code such tree as a zerotree symbol, and avoid coding all its nodes. The nodes are scanned by the order of importance. It is performed so that no child is scanned before its parent. Therefore, one starts scanning the nodes $C(i, j)$ for $(i, j) \in \boldsymbol{H}$ and the sets $D(i, j)$ for $(i, j) \in \boldsymbol{H}$. The result of significant test for a node or for a set is coded. In addition, for each node $C(i, j)$, if it is significant, its sign bit is also coded.

The process begins with setting LSP as an empty list, adding the coordinates $(i, j) \in \boldsymbol{H}$ to the LIP, adding those with descendants to the LIS as type A entries, and outputting the maximal value of n. The value of n can be obtained by using

$$n = \left\lfloor \log_2 \max_{(i, j)} \{|C(i, j)|\} \right\rfloor \quad (2)$$

where $\lfloor x \rfloor$ is to truncate x near to zero. Then the following two passes, the sorting pass and refinement pass, are used for every n value. In the sorting pass, we scan each $C(i, j)$ in the LIP and each $D(i, j)$ or $L(i, j)$ in the LIS, extract significant nodes, and put them into the end of the LSP. In the refinement pass, however, another bit of precision is added to the magnitudes of nodes in the LSP. We decrease n by one, i.e., cut the threshold in half, and use these two passes for each n in the order of the sorting pass first until the bit budget is exhausted.

The algorithm addressed above does not consider the statistical dependence between adjacent nodes and between adjacent sets. To increase the coding efficiency, the significance values of 2×2 adjacent nodes (the nodes with the same parent) were grouped and coded as a single symbol by the arithmetic coding algorithm. Since the decoder only needs to know the transition from insignificant to significant, the amount of information that needs to be coded changes according to the number m of insignificant nodes 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 [17], each with 2^m symbols, $m \in \{1, 2, 3, 4\}$ to code the information in a group of at most four pixels. Likely, the significance values of adjacent sets are coded as a single symbol.

III. Transformation

Symmetric extension on both boundaries of each segment is used to reduce edge effects. Among the four symmetric structures, i.e., whole-point symmetry (WS), half-point symmetry (HS), whole-point anti-symmetry (WA), and half-point anti-symmetry (HA) [19]—the WS structure is used for odd length filters.

To reduce computational complexity, we perform the wavelet decomposition separately to the rows and columns of an arbitrarily shaped image. Moreover, the decomposition is only applied to object regions. If an object region contains holes, the decomposition is suspended on the edges. This means, if a row or column of a region is split into M unconnected segments, each segment is filtered and downsampled separately.

In a two-band, perfect reconstruction filter bank, [20] indicated that the filters should satisfy

$$g[n] = (-1)^n \tilde{h}[1 - n] \quad \text{and} \quad \tilde{g}[n] = (-1)^n h[1 - n] \quad (3)$$

where $g[n]$, $h[n]$, $\tilde{g}[n]$, and $\tilde{h}[n]$ represent the analysis high-pass filter, the analysis low-pass filter, the synthesis high-pass filter, and the synthesis low-pass filter, respectively.

Assume that a pair of symmetric odd length filters $(h, \tilde{h})$ is given and defined in the following intervals: $h[n]$ for $-p \leq n \leq p$ and $\tilde{h}[n]$ for $-q \leq n \leq q$. From (3), we can obtain the values: $g[n]$ for $-q+1 \leq n \leq q+1$ and $\tilde{g}[n]$ for $-p+1 \leq n \leq p+1$. According to the definition of discrete convolution, we have the following results:

$$A'[k] = \sum_{n=-p}^{p} h[n]\tilde{a}[k-n] \quad \text{and} \quad B'[k] = \sum_{n=-q+1}^{q+1} g[n]\tilde{a}[k-n] \quad (4)$$

where $A'[k]$ and $B'[k]$ represent the convolution results, and $\tilde{a}[n]$ the extended version of input sequence $a[n]$. Notably, $\tilde{a}[n]$ is of WS structure on its both sides, i.e., WSWS structure herein.

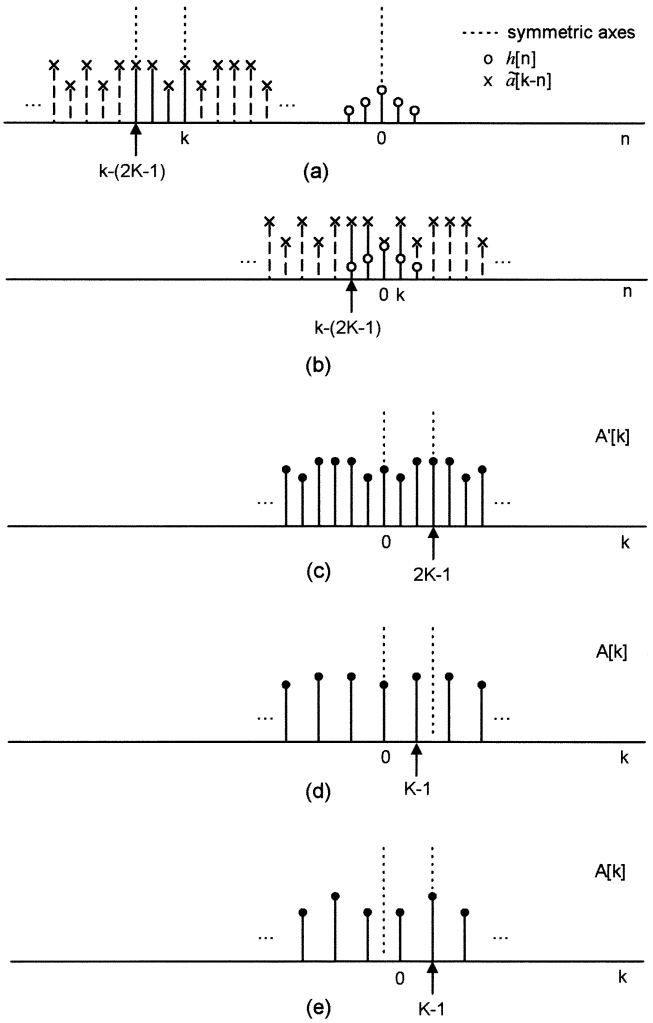


Fig. 3. Sequences involved in computing a discrete convolution. (a) and (b) the sequences $\tilde{a}[k-n]$ and $h[n]$ as a function of n for different values of k. (c) Corresponding output sequence as a function of k. (d) Even downsampling output. (e) Odd downsampling output.

After the convolution, a downsampler, with a downsampling factor of 2, follows in order to remove redundancy. Depending on the retained index terms of a convolution result, even or odd, we have even downsampling (ED) or odd downsampling (OD). For simplicity, we define that

$$A[k] = A'[2k+d] \quad \text{and} \quad B[k] = B'[2k+d] \tag{5}$$

where d represents the type of the downsampler. For ED, $d = 0$. For OD, $d = 1$. Fig. 3 illustrates sequences involved in computing a discrete convolution. The input sequence $a[n]$ is assumed to be of even length $(2K)$. The extended samples of $a[n]$ are denoted with dashed lines. The symmetric axes of sequences are also shown.

Without a loss of generality, each segment can be relocated to a new starting position 0 or 1, depending on its original starting position being even or odd, respectively. According to the starting position (0 or 1) and the length (even or odd) of a segment, there are four cases of processing to address.

Table I lists the symmetric structures of $A[k]$ and $B[k]$ when filtering the four cases. Each sequence ($A[k]$ or $B[k]$) has two symmetric axes, and we only need to retain the samples that fall into the axes. For example, if the symmetric axes of $A[k]$ are (0, $K-1/2$), we retain $A[k]$ for $0 \le k \le K-1$. As can be easily verified, the total length of retained samples for each the pair of $A[k]$ and $B[k]$ is equal to the length of input sequence $a[n]$.

Since ED and OD are available, we have two types of SA-DWT. In this study, the SA-DWT with ED is adopted. Fig. 4 illustrates the successive approaches of such SA-DWT to decompose an arbitrarily shaped object.

The formula corresponding to (4) for reconstruction is

$$a[n] = \sum_{\substack{k=-q \\ k+n-d:\, even}}^{q} \tilde{h}[k] A\left[\frac{k+n-d}{2}\right] + \sum_{\substack{k=-p+1 \\ k+n-d:\, even}}^{p+1} \tilde{g}[k] B\left[\frac{k+n-d}{2}\right]. \tag{6}$$

IV. Coding Algorithm

The LMZC coding algorithm and SPIHT are quite similar, except that LMZC abandons the use of lists, defines a different tree structure, and merges the sorting pass and the refinement pass together. The LMZC coding algorithm is implemented by entropy-coding its output. The statistic dependencies between adjacent nodes and between adjacent trees are exploited in the multi-models of the adaptive arithmetic coding algorithm of Witten *et al.* [17].

Fig. 5 shows the parent-child dependency of the LMZC tree structure and also shows the extent of each band of a 19 $\times$ 14 image in the three-scale SA-DWT decomposition. A parent node may have six child nodes in this figure. If an arbitrarily shaped image is coded, the corresponding binary alpha plane (shape mask) of the image is also decomposed to the same scale. Then, there are two types of nodes in a tree: nodes and out-nodes (with don't care values). The out-nodes are identified from the decomposed alpha plane and do not need to be coded. The LMZC adopts the tree structure like to EZW's tree structure not the SPIHT's tree structure. The node in the highest layer has one child in each of the high frequency bands, (i.e., HL_3, LH_3, and HH_3 in Fig. 5). This is better than SPIHT because SPIHT needs even size of frequency bands in the highest transform level. Therefore, when coding arbitrarily shaped images, the size of bounding box needs to be adjusted to allow even size frequency bands in the highest level. Note that we call, here, the new tree structure is like to but not equal to EZW's tree structure, because the new structure allows that one parent node has more than four and at most nine offspring.

The symbols used in LMZC are like those used in SPIHT. The new notation is $F_c(i, j)$ and $F_d(i, j)$, which represent the significance state of $C(i, j)$ and the significance state of $D(i, j)$, respectively. To identify a significant coefficient or a significant set, the significant test of (1) is revised to

$$S_n(X(i,j)) = \begin{cases} 1, & \text{if } \max\limits_{C(k,l)\in X(i,j)} 2^{n+1} > |C(k,l)| \ge 2^n; \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

TABLE I
OUTPUT SYMMETRIC STRUCTURES WHEN BOTH ANALYSIS FILTERS ARE OF ODD LENGTH AND SYMMETRIC

Input sequence $a[n]$		Convolution results (symmetric axes)		Even downsampling [1] Output (symmetric axes)		Odd downsampling Output (symmetric axes)	
Length	Starting point	$A'[k]$	$B'[k]$	$A[k]$	$B[k]$	$A[k]$	$B[k]$
2K	0	WSWS (0, 2K-1)	WSWS (1, 2K)	WSHS (0, K-1/2)	HSWS (1/2, K)	HSWS (-1/2, K-1)	WSHS (0, K-1/2)
2K	1	WSWS (1, 2K)	WSWS (2, 2K+1)	HSWS (1/2, K)	WSHS (1, K+1/2)	WSHS (0, K-1/2)	HSWS (1/2, K)
2K-1	0	WSWS (0, 2K-2)	WSWS (1, 2K-1)	WSWS (0, K-1)	HSHS (1/2, K-1/2)	HSHS (-1/2, K-3/2)	WSWS (0, K-1)
2K-1	1	WSWS (1, 2K-1)	WSWS (2, 2K)	HSHS (1/2, K-1/2)	WSWS (1, K)	WSWS (0, K-1)	HSHS (1/2, K-1/2)

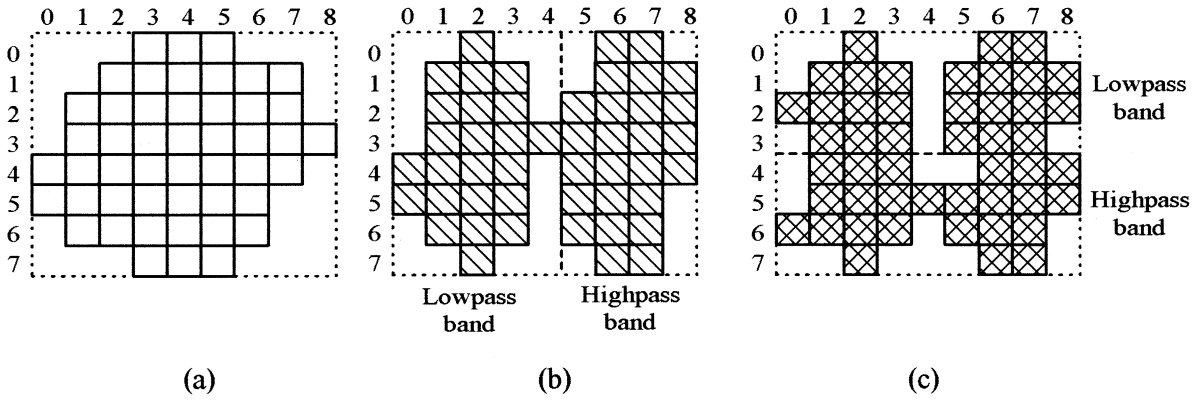


Fig. 4. Successive steps involved in performing an even downsampling SA-DWT forward transformation on an arbitrarily shaped image in the compact emplacement (size of bounding box: 9 × 8). (a) Original image; (b) location of coefficients after horizontal SA-DWT; and (c) location of 2-D SA-DWT coefficients.

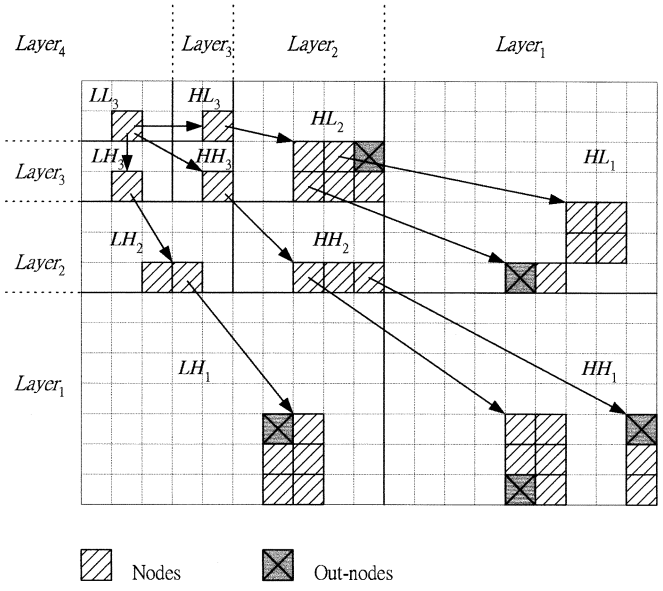


Fig. 5. Examples of parent–children dependencies in the LMZC tree structure. The arrows are oriented from the parent node to its offspring. The size of bounding box is 19 × 14 in this figure. A tree may contain some out-nodes (with don't care values) and they do not need to be coded.

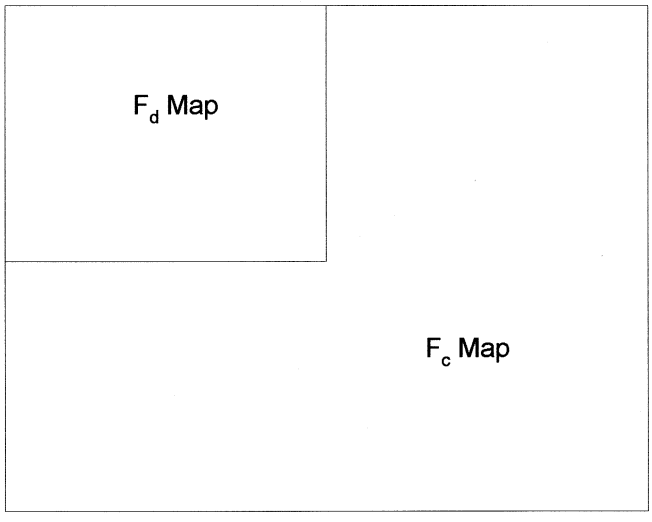


Fig. 6. Sizes of F_d map and F_c map.

where $X(i, j)$ denotes $C(i, j)$ or $D(i, j)$. If integer variables are used to store the values of transformed coefficients, checking whether the magnitude of $C(k, l)$ is fallen into the range 2^n

to $2^{n+1} - 1$ is equivalent to checking whether the nth most significant bit (MSB) of the magnitude of $C(k, l)$ is "1."

Each $C(i, j)$ has an associated binary state variable $F_c(i, j)$. Significance states are initialized to 0 (coefficient is insignificant) and may become 1 (coefficient is significant) during the coding run. Likely, each $D(i, j)$ has an associated binary state variable $F_d(i, j)$ to denote its significance state. Fig. 6 shows the sizes of $F_c(i, j)$ map and $F_d(i, j)$ map. The size of F_c map is the same as the image size, but the size of F_d map is a quarter or near a quarter of the image size because the coefficients in $Layer_1$ do not have descendents. For a color image with size 768×512, these two maps may cost 184 KBytes[6] for all bit rates, which is much less than the 5.3 MBytes memory requirement of SPIHT. In TBS, the top bits of coefficients are used to serve as these two flag maps. Therefore, the memory requirement of these two maps can be further omitted.

In LMZC, the sorting pass and the refinement pass are combined together to reduce the execution time. Before coding, LMZC transforms a RGB color image to a YUV image in order to remove the correlation among R, G, and B components. Also, LMZC initializes the F_c and F_d maps of YUV components to "0." The initial value of n is set to be the index of the MSB of the largest magnitude of Y, U, and V components. For each n, Y, U, and V components are coded in the order of Y, U, and V. In the first coding run, the magnitude, sign, and position information of coefficients in the range 2^n to $2^{n+1} - 1$ are coded. After that, for the next run, the value of n is decreased by one. This process continues until the bit budget is exhausted.

The main coding procedure of LMZC is shown as Fig. 7. For simplicity, we assume, herein, that each coefficient has four offspring. The adjacent coefficients with the same parent are grouped and coded together to remove redundancy. The same scheme is used to code the significance of adjacent sets. LMZC begins coding the coordinates in the highest layer (i.e., the LL_3 in Fig. 5). For each coordinate (i, j), it codes the individual coefficient $C(i, j)$ and the descendant set $D(i, j)$. Whenever coding $C(i, j)$, it checks whether $C(i, j)$ has been significant from the previous coding run according to the flag map $F_c(i, j)$. If $F_c(i, j) = 1$, giving that $C(i, j)$ is significant from the previous coding run, it outputs the refinement bit of $C(i, j)$. Otherwise, it checks the result of $S_n(C(i, j))$. If $S_n(C(i, j)) = 1$, it sets $F_c(i, j)$ to be "1," outputs the most significant bit of $C(i, j)$, and outputs the sign bit of $C(i, j)$. If not, it does nothing.

After coding $C(i, j)$, LMZC checks the significance of $D(i, j)$ according to the flag map $F_d(i, j)$. If $D(i, j)$ is insignificant in the previous coding run [i.e., $F_d(i, j) = 0$], it checks the result of $S_n(D(i, j))$. If $S_n(D(i, j)) = 1$, it sets $F_d(i, j) = 1$. If not, it finishes this branch. When $F_d(i, j) = 1$, the set $D(i, j)$ will be partitioned. Before partitioning $D(i, j)$, LMZC counts both the number of insignificant coefficients, $N_c(i, j)$, and the number of insignificant sets, $N_d(i, j)$, fallen into the $C(i, j)$ offspring set, and performs the significant test (7) to these insignificant coefficients to get a value $V_c(i, j)$, and to the insignificant sets to get a value $V_d(i, j)$. Since the decoder only needs to know the transition from insignificant to significant, the amount of information that needs to be coded changes according to $N_c(i, j)$ [or $N_d(i, j)$] in the offspring set, and in each case the value m of the number of insignificant coefficients (or insignificant sets) is conveyed by an entropy-coding alphabet with 2^m symbols. The value $V_c(i, j)$ [or $V_d(i, j)$] represents one of 2^m symbols. Fig. 8 shows an example to get the values of $V_c(i, j)$ and $V_d(i, j)$. In this example, $N_c(i, j)$ is equal to 1 because only one coefficient is not yet significant from the states of F_c map. By using (7), we can get the values of $V_c(i, j)$ and $V_d(i, j)$. If $N_c(i, j) > 0$, then LMZC outputs $V_c(i, j)$. If not, it does nothing. Likewise, if $N_d(i, j) > 0$, it outputs $V_d(i, j)$. Otherwise, it does nothing. After partitioning $D(i, j)$, LMZC traces into each partitioned branch until each branch is finished.

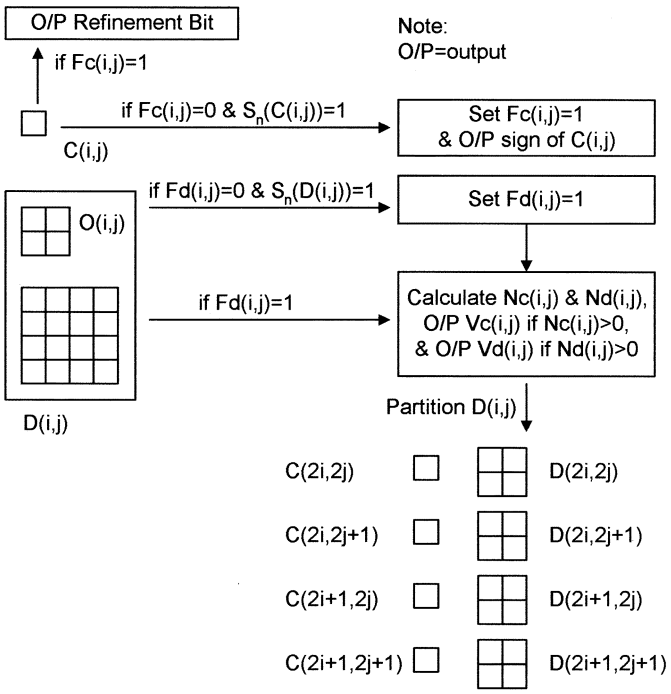


Fig. 7. LMZC coding procedure diagram. A tree contains a node $C(i, j)$ and a descendant set $D(i, j)$. The coder changes the coding branch according to the state of $F_c(i, j)$, the state of $F_d(i, j)$, the result of $S_n(C(i, j))$, and the result of $S_n(D(i, j))$. A significant set $D(i, j)$ is partitioned to four child nodes and four child sets. Before partitioning $D(i, j)$, the transition information of the four child nodes and the four child sets from insignificant to significant are calculated and outputted.

V. Complexity Analysis

In SPIHT, three lists namely LIP, LIS, and LSP are used to store the coordinates of partitioning sets and coefficients. In LMZC, no such lists are used. Instead of the lists, two flag maps should be applied to denote the states of coefficients and sets during coding. Compared with the lists, the memory requirement of these two maps is reduced significantly. If the top bits of coefficients are available to represent the states of the two maps, the memory requirement for these two maps can be further omitted. The penalty of this method is requiring much time to distinguish the magnitude of $C(i, j)$ from the states of $F_c(i, j)$ and $F_d(i, j)$. Therefore, one should trade off the amount of memory requirement against the execution time.

[6]$(768 \times 512 + 384 \times 256) \times 3$ (colors)/8 bits/1K = 184K

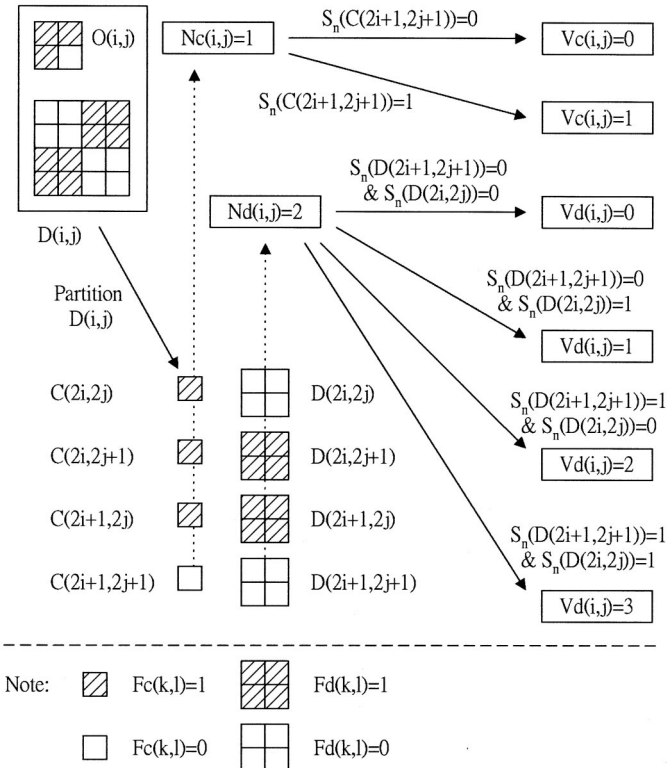


Fig. 8. Example of conveying the information of significance transition to values. In this example, a set $D(i, j)$ is assumed it contains one insignificant child node and two insignificant child sets. After using the significant test to the node and sets, we have two cases for the node and four cases for the sets to be processed. The values $V_c(i, j)$ and $V_d(i, j)$ are used to represent one of the cases for the node and one of the cases for the sets, respectively.

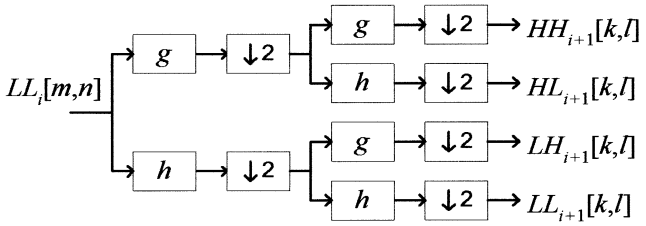


Fig. 9. One stage of an S-scale DWT.

To show that the top bits of coefficients indeed can be used to serve as the two flag maps, we take the upper bound of magnitudes of transformed coefficients into consideration. Fig. 9 shows one stage of an S-scale DWT. In Fig. 9, we use "$\downarrow 2$" to represent the downsampling. When $i = 0$, $LL_0[m, n]$ is equivalent to the input image $a[m, n]$, and the output coefficients are as follows:

$$HH_1[k, l] = \sum g[n] \sum g[m]a[2k - m, 2l - n], \quad (8)$$

$$HL_1[k, l] = \sum g[n] \sum h[m]a[2k - m, 2l - n], \quad (9)$$

$$LH_1[k, l] = \sum h[n] \sum g[m]a[2k - m, 2l - n], \quad (10)$$

$$LL_1[k, l] = \sum h[n] \sum h[m]a[2k - m, 2l - n]. \quad (11)$$

From (8), the magnitude of $HH_1[k, l]$ satisfies that

$$|HH_1[k, l]| \le \sum |g[n]| \sum |g[m]|a[2k - m, 2l - n]$$
$$\le \sum |g[n]| \sum |g[m]|a_{\max} \quad (12)$$

where $a_{\max} = \max_{(n,m)} \{|a[n, m]|\}$. Let $S_h = \sum |h[n]|$ and $S_g = \sum |g[n]|$. Equation (8) can be rewritten to $|HH_1[k, l]| \le S_g^2 a_{\max}$. By a similar analysis as above, we have $|HL_1[k, l]| \le S_g S_h a_{\max}$, $|LH_1[k, l]| \le S_h S_g a_{\max}$, and $|LL_1[k, l]| \le S_h^2 a_{\max}$. For the next scale, the band LL_1 is further decomposed, which results in four bands, i.e., HH_2, HL_2, LH_2, and LL_2. The upper bounds of magnitudes of coefficients in these four bands are separately $|HH_2[k, l]| \le S_g^2 S_h^2 a_{\max}$, $|HL_2[k, l]| \le S_g S_h^3 a_{\max}$, $|LH_2[k, l]| \le S_g S_h^3 a_{\max}$, and $|LL_2[k, l]| \le S_h^4 a_{\max}$.

For an S-scale DWT, by induction, we get

$$|HH_S[k, l]| \le S_g^2 S_h^{2(S-1)} a_{\max} \quad (13)$$

$$|HL_S[k, l]| \le S_g S_h^{2S-1} a_{\max} \quad (14)$$

$$|LH_S[k, l]| \le S_g S_h^{2S-1} a_{\max} \quad (15)$$

$$|LL_S[k, l]| \le S_h^{2S} a_{\max}. \quad (16)$$

Suppose that the 9/7 filter [20] as shown in Table II is used to filter images. Then, $S_h = 1.952\,108\,3$ and $S_g = 1.835\,127\,5$. For a six-scale decomposition, the upper bound of magnitudes of transform coefficients [obtained from (16)] is $3062.294 \times a_{\max}$, which is smaller than 2^{20} for $a_{\max} = 2^8 - 1$. An integer variable occupying four bytes of memory is sufficient to store the value of transformed coefficients and to serve as the two flag maps F_c and F_d.

VI. Experimental Results

The original rectangular images and arbitrarily shaped images are shown in Figs. 10 and 11, respectively. The Lena image, Barbara image, and the Tulips image are of rectangular images. The size of Lena is 512×512, and so is Barbara, and the size of Tulips is 768×512. The Jaguar image, Miss America image, and the Akiyo image are of arbitrarily shaped objects. The size of bounding box is 465×702 to contain the Jaguar image, and it is 352×261 to contain the Miss America image and is 516×421 to contain the Akiyo image. The corresponding binary alpha planes of these three images are shown in Fig. 11(b), (d), and (f). The numbers of pixels contained in the object regions of Jaguar, Miss America, and Akiyo are 187 568, 50 341, and 128 339, respectively. The alpha planes are not coded for simplicity. The 9/7 filter shown in Table II is used, and the symmetric extension is applied to the image edges. The number of decomposition scale is varied from image to image, depending on their sizes. A 26-bit header is saved to each coded file for recording the image width (10 bits), the image height (10 bits), the number of decomposition scale (4 bits) and the number of colors (2 bits). The reconstructed images are compared to their

TABLE II
FILTER COEFFICIENTS OF 9/7 FILTER. $h[-n] = h[n]$ AND $g[-n] = g[n]$

	n=0	n=1	n=2	n=3	n=4
$h[n]$	0.8526987	0.3774027	-0.1106240	-0.0238493	0.0378288
$g[n]$	-0.7884849	0.4180924	0.0406898	-0.0645391	--

(a) (b)

(c)

Fig. 10. Original rectangular images (a) 24-bit color Lena; (b) 8-bit gray Barbara; and (c) 24-bit color Tulips.

original ones in PSNR value. The PSNR value for color images is defined as

$$PSNR = 10 \log \frac{3 \times 255^2}{MSE(R) + MSE(G) + MSE(B)}. \quad (17)$$

We compare the coding results of LMZC with that of LZC [10] and that of SPIHT [2]. The LZC is extended from our previous work, embedded recursive zerotree coding (ERZC) [13]. LZC takes the coding of color image into consideration, and its coding algorithm is much similar to LMZC coding algorithm except that LZC does not exploit the correlation of neighboring pixels or neighboring sets and does not deal with the coding of arbitrarily shaped images. In this study, LMZC not only exploits the correlation of neighboring pixels and neighboring sets completely, but also extends the idea of a low memory implementation from the coding of rectangular and gray image to the coding of arbitrarily shaped and color objects. Since the original SPIHT algorithm [2] and LZC do not deal with the coding of arbitrarily shaped images, we extend them, herein, for a fair comparison. The extended version of SPIHT algorithm uses the regular emplacement of transformed coefficients so that the one-parent to four-children dependency is preserved. In addition, the extended SPIHT uses YUV transform as the intercomponent transform for coding color images. It is different from the Internet version of SPIHT [2], which uses KL transform. As a result, the data shown herein is not the same as the original [2]. Note that when the extended version of SPIHT is used to code a gray rectangular image, the coding results are worse than those of Internet

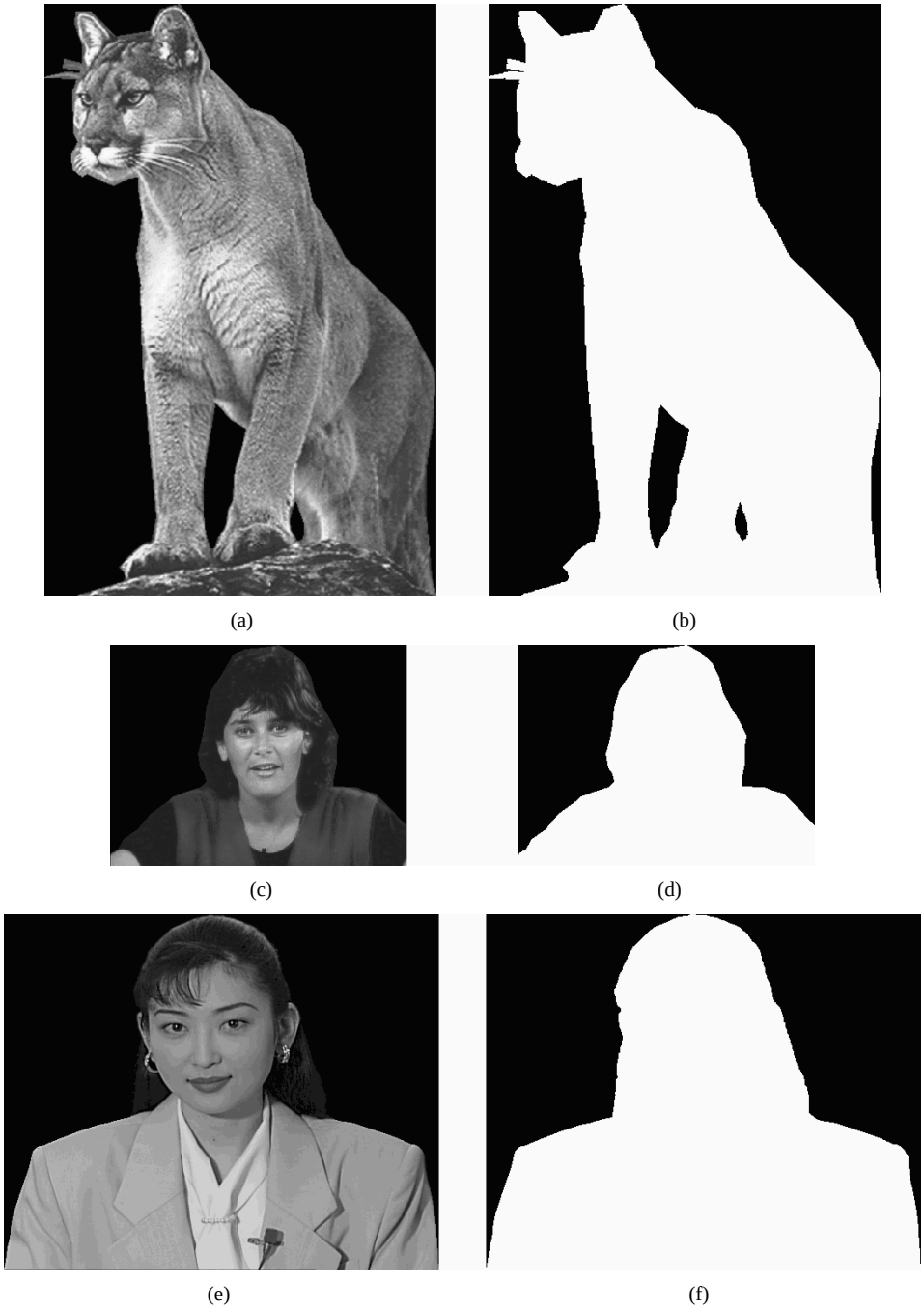


Fig. 11. Original arbitrarily shaped images (a) 24-bit color Jaguar; (c) 24-bit color Miss America; (e) 8-bit gray Akiyo; (b), (d), and (f) are the corresponding alpha planes of (a), (c), and (e), respectively.

SPIHT on average about 0.2 dB. This is caused by the fact that some implementing details are different and that it is difficult to exactly reproduce the same coder without its source codes.

Fig. 12 shows the comparison of performance in PSNR values. The main improvement of LMZC to LZC is on PSNR value. As shown in Fig. 12, LMZC's PSNR values outperform LZC's PSNR values over a wide range of bit rates. On the average, 0.45 dB improvement can be achieved. The PSNR values of LMZC, however, are near to that of SPIHT. In general, LMZC performs better than or near to SPIHT in the coding of arbitrarily shaped images but worse than SPIHT in the coding of rectangular images, especially in the coding of rectangular gray images. For the Barbara image, LMZC's PSNR values are less than SPIHT's PSNR values on the average about 0.4 dB.

Table III shows the comparison on the corresponding CPU times, excluding the time spent in the image transformation, for coding and decoding the Lena image. The programs were not optimized, and these times are shown just to give an indication of the LMZC's speed. Since the LZC does not need to group the states of adjacent pixels and adjacent sets together, it can save little execution time than the LMZC, but the difference is small. The LMZC uses the same strategy as the SPIHT to code the states of

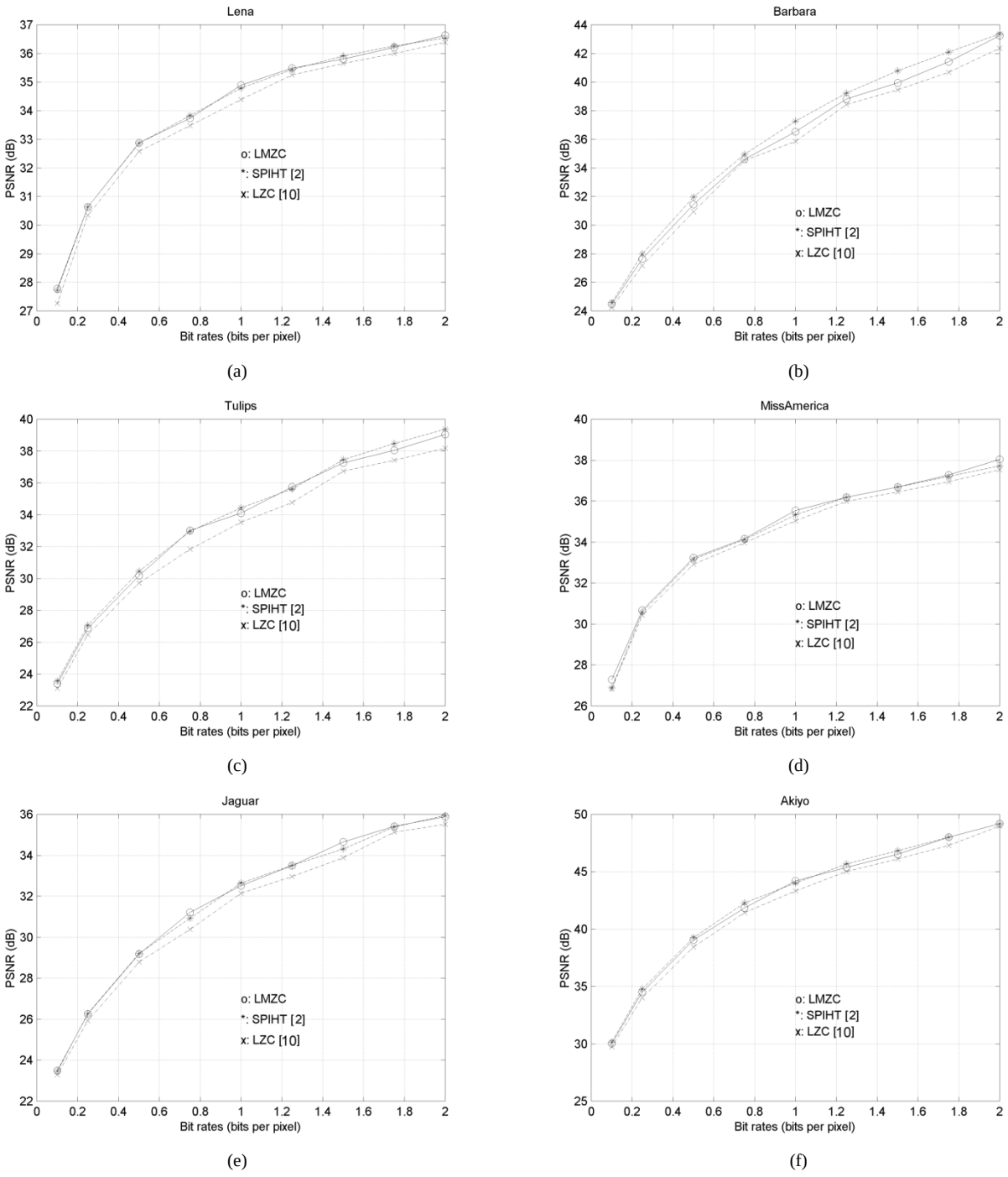
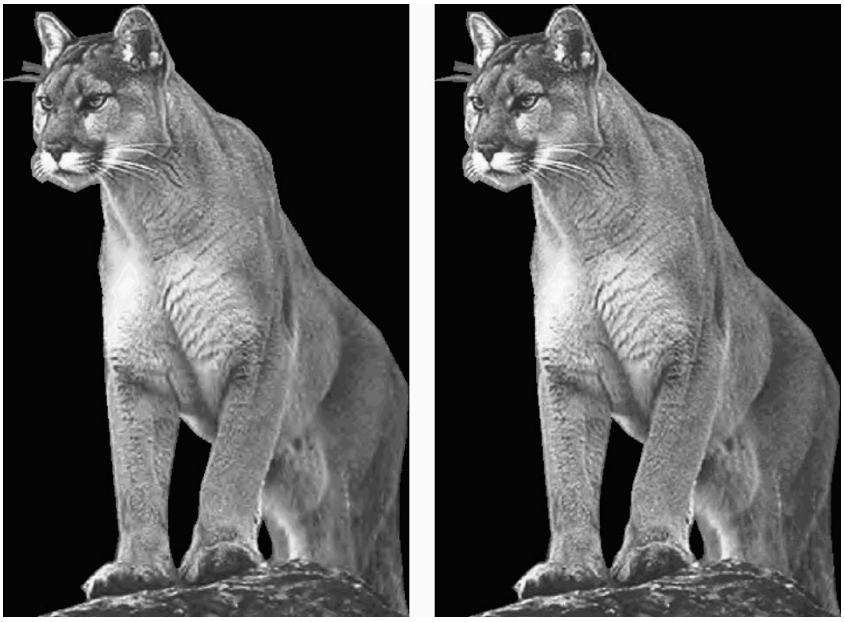


Fig. 12. Comparison of different zerotree entropy coding schemes in PSNR values (a) for Lena; (b) for Barbara; (c) for Tulips; (d) for Miss America; (e) for Jaguar; and (f) for Akiyo.

adjacent pixels and adjacent sets together, so the computational complexity of LMZC and SPIHT is the same in this point of view. However, the LMZC and LZC uses the recursive programming, leading to that CPU must *push* and *pop* the values of some registers, which requires a little execution time. Except this difference, the LMZC records the coding state of $C(i, j)$ [or $D(i, j)$] by setting a top-bit of $C(i, j)$, which is different from the SPIHT that needs to record the coordinate (i, j) into the lists. Setting a top-bit of a coefficient requires one OR operation, but recording the coordinate (i, j) requires two *assign* operations. Thus, LMZC saves a little execution time than SPIHT at recording the coding states.

The main improvement of LMZC to SPIHT is on the amount of working memory. For a 768×512 color image, LMZC can save at least 5.3 MBytes of memory, which leads to a low cost hardware implementation. As well as its advantages for hardware implementation, LMZC algorithm is also more suitable for incorporation into a plug-in program for Internet browsers than SPIHT. Since LMZC requires less memory during encoding and decoding but has a comparable source code complexity, the program footprint is relatively smaller that SPIHT. That is, the working memory used to store program code and to process data is smaller than SPIHT. This is clearly a good feature from the



(a) (b)

Fig. 13. LMZC reconstructed Jaguar images (a) 0.5 bpp (48 : 1), 29.175 dB and (b) 1 bpp (24 : 1), 32.522 dB.

TABLE III
COMPARISON ON THE CUP TIMES (s) TO CODE AND DECODE THE COLOR IMAGE LENA 512 × 512 (PENTIUM III 800 CPU, 128 MB RAM)

Bit rates (bpp)	LZC		LMZC		SPIHT	
	code	decode	code	decode	code	decode
0.50	2.82	0.24	2.83	0.28	2.21	0.55
1.00	3.22	0.37	3.23	0.53	2.46	0.65
2.00	3.88	0.65	3.91	0.94	2.76	0.88

user's point of view. As to LZC, LMZC uses the same size of memory in coding.

Except the above comparison, we also compare the difference between the two pass coding (the sorting pass and the refinement pass) and the one single pass coding (merging the two passes to one) in the PSNR values and in the execution time. The difference is minor in PSNR values but is obvious in the execution time. In our experiments, a two-pass LMZC can elevate about 0.01 dB in PSNR value for coding the Lena image, but it requires an additional flag map to denote those just significant nodes and requires about 0.5 sec for the execution of the second pass.

Fig. 13 shows the LMZC reconstructed images of Jaguar at bit rates 0.5 bpp, and 1 bpp. As shown in Fig. 13, these images show good visual quality and no subject difference from the original one.

VII. CONCLUSION

In this study, we proposed a low memory zerotree coder for coding of arbitrarily shaped objects. The proposed coder takes advantage of the recursive programming and uses the top bits of transformed coefficients to serve as flags, so that no additional memory are needed during coding and decoding. In addition, a compact emplacement of transformed coefficients was proposed to further reduce the working memory for coding arbitrarily shaped objects. Compared with SPIHT, the proposed coder can save at least 5.3 MBytes of memory for coding a 768 × 512 color image. Besides, the proposed coder preserves most of the merits of SPIHT (such as simple computation, effective compression, and embedded coding), thereby making it highly promising for some memory limited applications.

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] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technology*, vol. 6, no. 3, pp. 243–250, June 1996.
[3] R. Koenen, Ed., "Overview of the MPEG-4 Version 1 Standard," in *ISO/IEC JTC1/SC29/WG11 N1909*, Oct. 1997, MPEG97.
[4] T. Sikora, "The MPEG-4 video standard verification model," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 19–31, Feb. 1997.
[5] ISO/IEC, "ISO/IEC 15 444-1, Information technology—JPEG2000 image coding system,", 2000.
[6] M. D. Adams, "The JPEG-2000 still image compression standard," in *ISO/IEC JTC1/SC29/WG1 N2412*, Sept. 2001.
[7] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 still image coding system: An overview," *IEEE Trans. Consumer Electron.*, vol. 46, pp. 1103–1127, Nov. 2000.
[8] J. Li and J. S. Jin, "Structure-related perceptual weighting: A way to improve embedded zerotree wavelet image coding," *Electron. Lett.*, vol. 33, pp. 1305–1306, 1997.
[9] S. Li and W. Li, "Shape-adaptive discrete wavelet transforms for arbitrarily shaped visual object coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, pp. 725–743, Aug. 2000.
[10] W. K. Lin and N. Burgess, "Low memory color image zerotree coding," in *Proc. Information, Decision, and Control 1999*, 1999, pp. 91–95.
[11] S. A. Martucci, I. Sodagar, T. Chiang, and Y.-Q. Zhang, "A zerotree wavelet video coder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 109–118, Feb. 1997.
[12] J. K. Rogers and P. C. Cosman, "Wavelet zerotree image compression with packetization," *IEEE Signal Processing Lett.*, vol. 5, pp. 105–107, 1998.

[13] C. Y. Su and B. F. Wu, "Image coding based on embedded recursive zerotree," in *Proc. ISMIP'97*, Taipei, Taiwan, Dec. 1997, pp. 387–392.
[14] Q. Wang and M. Ghanbari, "Scalable coding of very high resolution video using the virtual zerotree," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 719–727, 1997.
[15] D. Taubman, "High performance image scalable image compression with EBCOT," *IEEE Trans. Image Processing*, vol. 9, pp. 1158–1170, July 2000.
[16] Z. Xiong, K. Ramchandran, and M. T. Orchard, "Space-frequency quantization for wavelet image coding," *IEEE Trans. Image Processing*, vol. 6, pp. 677–693, May 1997.
[17] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, pp. 520–540, June 1987.
[18] S. Li, W. Li, Z. Wu, and H. Sun, "Shape adaptive wavelet coding," in *Proc. ISCAS'98*, Monterey, CA, 1998, pp. 281–284.
[19] A. V. Oppenheim, R. W. Schafer, and J. R. Buch, *Discrete-Time Signal Processing 2/E*. Englewood Cliffs, NJ: Prentice-Hall, 1998, pp. 297–308.
[20] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. Image Processing*, vol. 1, pp. 205–220, Apr. 1992.

Chorng-Yann Su was born in Tainan, Taiwan, R.O.C., in 1967. He received the B.S. and M.S. degrees in industrial education from the National Taiwan Normal University (NTNU), Taipei, Taiwan, in 1991 and 1994, respectively, and the Ph.D. degree in electrical and control engineering from the National Chiao-Tung University, Hsinchu, Taiwan, in 1999.

He was with the Department of Industrial Education of NTNU as a Teacher Assistant in 1992 and is currently an Associate Professor. His research interests include image compression, wavelets, signal processing, and computer vision.

Bing-Fei Wu (S'89–M'92–SM'02) was born in Taipei, Taiwan, R.O.C., in 1959. He received the B.S. and M.S. degrees in control engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1981 and 1983, respectively, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, in 1992.

From 1983 to 1984, he was with the Institute of Control Engineering, National Chiao Tung University, as an Assistant Researcher. From 1985 to 1988, he was with the Department of Communication Engineering at the same university as an Instructor. Since 1992, he has been with the Department of Electrical Engineering and Control Engineering as an Associate Professor and he is currently a Professor. As an active industry consultant, he is also involved in the chip design and applications of the flash memory controller and 3C consumer electronics in multimedia. His areas of research interest include chaotic systems, fractal signal analysis, multimedia coding, wavelet analysis and applications.

Dr. Wu is a member of the Chinese Automatic Control Society, Chinese Institute of Electrical and Electronic Engineers, and Chinese Institute of Engineers. He served as Chairman of Program Committee on The Symposium of Robust Control and Analysis supported by the Ministry of Education of Taiwan and the IEEE Automatic Control System Society, Taipei Chapter, in 1994. He has been the Director of The Research Group of Control Technology of Consumer Electronics in the Automatic Control Section of National Science Council in 1999 to 2000. He also served in the Technical Committee of The 3rd IEEE Pacific-Rim Conference on Multimedia, PCM 2002. He received the Research Awards from National Science Council, Taiwan, in the years of 1992, 1994, and 1996–2000; the Long-Term Dragon Thesis Awards sponsored by The Acer Foundation, in the years of 1994, 1996, and 1998; the Outstanding Teaching Award from National Chiao Tung University, Taiwan, in 2000; the Third Winner Award in the Hewlett-Packard Information Appliance Competition in 2000; the Excellent Award of Officers and Teachers from National Chiao Tung University, in 2001; The First Runner-Up Macronix Golden Silicon Award and The Best Originality Award in the 1st and 2nd Semiconductor and Application Competition sponsored by Macronix International Co., Ltd., Taiwan in 2001–2002, respectively; and the Distinguished Engineering Professor Award from the Chinese Institute of Engineers in 2002.