IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY

A PUBLICATION OF THE IEEE CIRCUITS AND SYSTEMS SOCIETY

WWW.IEEE-CAS.ORG

NOVEMBER 2006 VOLUME 16 NUMBER 11 ITCTEM (ISSN 1051-8215)

Low-Complexity Multiresolution Image Compression Using Wavelet Lower Trees

Jose Oliver, *Member, IEEE*, and Manuel P. Malumbres, *Member, IEEE*

***Abstract*—In this paper, a new image compression algorithm is proposed based on the efficient construction of wavelet coefficient lower trees. The main contribution of the proposed lower-tree wavelet (LTW) encoder is the utilization of coefficient trees, not only as an efficient method of grouping coefficients, but also as a fast way of coding them. Thus, it presents state-of-the-art compression performance, whereas its complexity is lower than the one presented in other wavelet coders, like SPIHT and JPEG 2000. Fast execution is achieved by means of a simple two-pass coding and one-pass decoding algorithm. Moreover, its computation does not require additional lists or complex data structures, so there is no memory overhead. A formal description of the algorithm is provided, while reference software is also given. Numerical results show that our codec works faster than SPIHT and JPEG 2000 (up to three times faster than SPIHT and fifteen times faster than JPEG 2000), with similar coding efficiency.**

***Index Terms*—Image compression, low complexity, tree-based coding, wavelets.**

I. Introduction

DURING the last decade, several image compression schemes emerged in order to overcome the known limitations of block-based algorithms that use the discrete cosine transform (DCT). Some of these alternative proposals were based on more complex techniques, like Vector Quantization [2] and Fractal image coding [3], whereas others simply proposed the use of a different and more suitable mathematical transform: the discrete wavelet transform (DWT) [4]. At that time, there was a general idea: more efficient image coders could only be achieved by means of sophisticated techniques with high complexity. The embedded zero-tree wavelet coder (EZW) [5] can be considered the first Wavelet image coder that broke that trend. Since then, many wavelet coders were proposed and finally, the DWT was included in the JPEG 2000 standard [6] due to its compression efficiency among other interesting features (scalability, etc.).

All the wavelet-based image coders, and in general all the transform-based coders, consist of two main stages. In the first one, the image is transformed from spatial domain to another one, in the case of the wavelet transform a combined spatial-frequency domain, called wavelet domain. In the second pass, the transform coefficients are quantized and encoded in an efficient way to achieve high compression efficiency and other features.

The wavelet transform can be implemented as a regular filter-bank, however several strategies have been proposed to reduce the running time and memory requirements. This way, a line-based processing is proposed in [7], whereas an alternative wavelet transform method, called lifting scheme, is proposed in [8]. The lifting transform provides in-place calculation of the coefficients by overwriting the input samples, and reduces the number of operations required to compute the DWT.

On the other hand, the coding pass is not usually improved in terms of complexity and memory usage. When designing a new wavelet image encoder, the most important factor to optimize is usually the rate/distortion (R/D) performance, whereas other features like embedded bit-stream, signal-to-noise ratio (SNR) scalability, spatial scalability and error resilience are also considered. In this paper, we propose an algorithm aiming to achieve state-of-the-art coding efficiency, with very low execution time. Moreover, due to in-place processing of the coefficients, there is no memory overhead (it only needs memory to store the source image). In addition, our algorithm is naturally spatial scalable and it is possible to achieve SNR scalability.

The key idea of the proposed algorithm is the use of wavelet coefficient trees as a fast method of efficiently grouping coefficients. Tree-based wavelet coders have been widely used in the literature [5], [9], [10], [14], presenting good R/D performance. However, their excellent opportunities for fast processing of quantized coefficients have not been clearly shown so far. Wavelet trees are a simple way of grouping coefficients, reducing the total number of symbols to be coded, which involves not only good compression performance but also fast processing. Moreover, for a low-complexity implementation, bit-plane coding, present in many wavelet coders [5], [6], [9], [11], [12], must be avoided. This way, multiple scans of the transform coefficients, which involves many memory accesses and causes high cache miss rates, is not performed, at the expense of generating a non-(SNR)-embedded bitstream.

This paper is organized as follows. In Section II, we analyze the complexity and coding efficiency of some important wavelet image coders, focusing on the non-embedded proposals. In Section III, the proposed tree-based algorithm is described. Section IV describes some implementation details and optimization considerations. Finally, in Section V, we compare our proposal with other wavelet image coders using real implementations.

II. Previous Wavelet Image Coders and Their Complexity

One of the first efficient wavelet image coders reported in the literature is EZW [5]. It is based on the construction of coefficient-trees and successive-approximations, which can be imple-

Manuscript received March 16, 2005; revised May 30, 2006; accepted May 30, 2006. This work was supported by the Spanish Ministry of Education and Science under grant TIC2003-00339. A preliminary version of this paper was first presented at the IEEE Data Compression Conference, Snowbird, UT, in March 2003. This paper was recommended by Associate Editor H. Gharavi.

J. Oliver is with the Department of Computer Engineering (DISCA), Polytechnic University of Valencia, 46022 Valencia, Spain (e-mail: joliver@disca.upv.es).

M. P. Malumbres is with the Department of Physics and Computer Engineering, Miguel Hernandez University, 03202 Elche, Spain (e-mail: mels@umh.es).

Digital Object Identifier 10.1109/TCSVT.2006.883505

mented with bit-plane coding. Due to its successive-approximation nature, it is SNR scalable, although at the expense of spatial scalability. SPIHT [9] is an advanced version of this algorithm, where coefficient-trees are processed in a more efficient way. In this coder, coefficient-trees are partitioned depending on the significance of the coefficients belonging to each tree. If a coefficient in a tree is higher than a threshold determined by the current bit-plane, the tree is successively divided following some established partitioning rules. Both EZW and SPIHT need the computation of coefficient-trees to search for significant coefficients and they take various iterations focusing on a different bit-plane, which involves high computational complexity. A block-based version of SPIHT, called SPECK, is presented in [11]. The main difference of this new version is that coefficients are grouped and partitioned using rectangular structures instead of coefficient trees. A low-complexity implementation of SPECK, called SBHP [12], was proposed in the framework of JPEG 2000. In SBHP, an image is first divided into blocks (sized 64×64 or 128×128) and then, each block is processed as in SPECK. To reduce complexity, Huffman coding is used instead of arithmetic coding, which causes a decrease in coding efficiency.

In the final JPEG 2000 standard [6], the proposed algorithm (a modified version of EBCOT [13]) does not use coefficient-trees, but it performs bit-plane coding in code-blocks, with three passes per plane so that the most important information is encoded in first place. In order to overcome the disadvantage of not using coefficient-trees, it uses an iterative optimization algorithm, based on the Lagrange multiplier method, along with a large number of contexts. JPEG 2000 obtains both spatial and SNR scalability by reordering the encoded image. Hence, the final bitstream is very versatile. However, this encoder is complex, mainly due to the use of the time-consuming iterative optimization algorithm, the introduction of a reordering stage, and the use of bit-plane coding with many contexts. In addition, a larger bitstream than the one actually encoded is usually generated.

Many times, wavelet image encoders have features that are not always needed, but which make them both CPU and memory intensive. Fast processing is often preferable. Low complexity may be desirable for high-resolution digital camera shooting, where high delays could be annoying and even unacceptable (even more for modern digital cameras, which tend to increase their resolution). In general, image editing for large images (especially in GIS applications) cannot be easily tackled with the complexity of previous encoders.

Since iterative methods and bit-plane coding must be avoided to reduce complexity, very fast coding can only be achieved through simpler non-SNR-embedded techniques (like in baseline JPEG). In these encoders, an image is encoded at a constant quality after applying a uniform quantization to the coefficients.

A. Non-Embedded Coders

One of the first tree-based non-embedded coders is SFQ [10]. This wavelet encoder uses space quantization by means of a tree-pruning algorithm, which modifies the shape of the trees by pruning their branches, whereas a scalar quantization is applied for frequency quantization. Although it achieves higher compression than SPIHT, the iterative tree pruning stage makes it about five times slower than SPIHT.

Non-embedded coding was first proposed to reduce complexity in tree-based wavelet coding in [14], where a non-embedded version of SPIHT was proposed. In this modified SPIHT, once a coefficient is found to be significant, all the significant bits are encoded, avoiding the refinement passes (see [9] for details). Note that a non-embedded version of SPECK and SBHP is also possible with the same modifications. Although these non-embedded versions are faster than the original ones, neither multiple image scan nor bit-plane processing of the sorting passes (used to find significant coefficients) is avoided, and hence, the complexity problem still remains. Note that these modifications of SPIHT and SPECK are neither SNR nor resolution scalable.

Besides the JPEG standard, other DCT-based non-embedded image coders have been proposed. In particular, the intra mode of the H.264 standard [16]. However, due to the use of time-consuming prediction techniques on the coding side, the coding/decoding processes are very asymmetric, and the resulting encoder is very slow.

III. Multiresolution Image Coding Using Lower Trees

For the most part, digital images are represented with a set of pixels, P. The encoder proposed in this paper is applied to a set of coefficients C resulting from a dyadic decomposition $\Omega(\cdot)$, with $C = \Omega(P)$. The most commonly used decomposition for image compression is the hierarchical wavelet subband transform [4], thus an element $c_{i,j} \in C$ is called transform coefficient. In a wavelet transform, we call LH_1, HL_1, and HH_1 the subbands resulting from the first level of the image decomposition, corresponding to horizontal, vertical and diagonal frequencies. The rest of the transform is computed with a recursive wavelet decomposition on the remaining low frequency subband, until a desired decomposition level (N) is achieved (LL_N is the remaining low frequency subband).

In Section II, we mentioned that one of the main drawbacks in previous wavelet-based image encoders is their high complexity. Many times, that is due to time-consuming iterative methods, and bit-plane coding used to provide a fully embedded bitstream. Although embedding is a nice feature in an image coder, it is not always needed and other alternatives, like spatial scalability, may be more valuable depending on the final application. In this section, we propose a tree-based coding algorithm that is able to encode the wavelet coefficients without performing an image scan per bit plane.

Tree-based wavelet image encoders are proved to efficiently store the transform coefficients, achieving good performance results. However, in the algorithm proposed in this paper, a tree-based structure is introduced, not only to remove redundancy among subbands, but also as a simple and fast way of grouping coefficients.

As in the rest of tree-based encoders, coefficients can be logically arranged as trees, as shown in Fig. 1. In this figure, we observe that the coefficients in the wavelet subbands (except the leaves) have always four direct descendants (i.e., four descendants at a distance of one), while the rest of descendants can be recursively obtained from the direct descendants. On the other

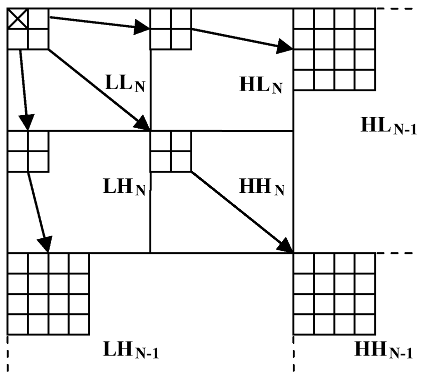

Fig. 1. Coefficient-trees in the proposed algorithm.

hand, in the $\mathrm{LL_N}$ subband, three out of every 2×2 coefficients have four direct descendants, and the remaining coefficient in the 2×2 block has no descendant.

In our proposal, the quantization process is performed with two strategies: one coarser and another finer. The finer one consists in applying a scalar uniform quantization to the coefficients, and it can be applied with the normalization factor in the lifting transform (if the normalization factor in the lifting scheme is K, and the quantization factor is Q, only a multiplication by $K \times Q$ is needed). The coarser one is based on removing bit planes from the least significant part of the coefficients, and it is performed while the algorithm is applied. Related to this bit plane quantization, we define $rplanes$ as the number of least significant bits to be removed.

We say that a coefficient $c_{i,j}$ is significant if it is different from zero after discarding the least significant $rplanes$ bits, in other words, if $|c_{i,j}| \geq 2^{rplanes}$. In our proposal, significant coefficients are encoded by using arithmetic coding, with a symbol indicating the number of bits needed to encode that coefficient. Then, the significant bits and sign are binary encoded (in other words, "raw encoded").

Regarding the insignificant coefficients, a coefficient is called lower-tree root if this coefficient and all its descendants are insignificant (i.e., lower than $2^{rplanes}$). The set formed by all these coefficients is a lower-tree. We use the symbol $LOWER$ to point out that a coefficient is the root of a lower-tree. The rest of coefficients in the lower-tree are labeled as $LOWER\_COMPONENT$, but are not encoded. On the other hand, if a coefficient is lower than $2^{rplanes}$, but it does not form or belong to a lower-tree because it has at least one significant descendant, it is considered $ISOLATED\_LOWER$.

A. Lower-Tree Encoder Algorithm

Once we have defined the basic concepts to understand the algorithm, we are ready to describe the coding process. It is a two-pass algorithm. During the first pass, the wavelet coefficients are properly labeled according to their significance and the lower-trees are formed. In the second image pass, the coefficient values are coded by using the labels computed in the first pass.

The coding algorithm is presented in Algorithm 1. Let us describe this algorithm. At the encoder initialization, the number of bits needed to represent the highest coefficient $(maxplane)$ is calculated. This value and the $rplanes$ parameter are output to the decoder. Afterwards, we initialize an adaptive arithmetic encoder that is used to encode the number of bits required by the significant coefficients, and the $LOWER$ and $ISOLATED\_LOWER$ symbols.

In the first image pass, the lower-tree labeling process is performed in a recursive way, by building the lower-trees from leaves to root. In the first level subbands, coefficients are scanned in 2×2 blocks and, if the four coefficients are insignificant (i.e., lower than $2^{rplanes}$), they are considered part of the same lower-tree, being labeled as $LOWER\_COMPONENT$. Then, when scanning higher level subbands, if a 2×2 block has four insignificant coefficients, and all their direct descendants are $LOWER\_COMPONENT$, the coefficients in that block are also labeled as $LOWER\_COMPONENT$, increasing the size of the lower-tree.

However, when at least one coefficient in the block is significant, the lower-tree cannot continue growing. In that case, an insignificant coefficient in the block is labeled as $LOWER$ if all its descendants are $LOWER\_COMPONENT$, otherwise an insignificant coefficient is labeled as $ISOLATED\_LOWER$.

In the second pass, all the subbands are explored from the Nth level to the first one, and all their coefficients are scanned in medium-sized blocks (to take advantage of data locality). For each coefficient in a subband, if it is a lower-tree root or an isolated lower, the corresponding $LOWER$ or $ISOLATED\_LOWER$ symbol is encoded. On the other hand, if a coefficient has been labeled as $LOWER\_COMPONENT$ no output is needed because this coefficient is already represented by the lower-tree to which it belongs.

A significant coefficient is coded as follows. A symbol indicating the number of bits required to represent that coefficient is arithmetically coded, and the significant bits and sign are "raw coded". However, two types of numeric symbols are used depending on the direct descendants of that coefficient. (a) A regular numeric symbol $(nbits_{i,j})$, which simply shows the number of bits needed to encode a coefficient, (b) and a special "$LOWER$ numeric symbol" $(nbits_{i,j}^{LOWER})$, which not only indicates the number of bits of the coefficient but also the fact that its descendants are labeled as $LOWER\_COMPONENT$, and thus they belong to a lower-tree not yet codified. This type of symbol is able to represent efficiently some special lower-trees, in which the root coefficient is significant and the rest of coefficients are insignificant. Note that the number of symbols needed to represent both sets of numeric symbols is $2 \times (2^{maxplane} - 2^{rplanes})$, therefore the arithmetic encoder must be initialized to handle at least this amount of symbols, along with two additional symbols: the $LOWER$ and $ISOLATED\_LOWER$ symbols. Observe that the first rplanes bits and the most significant non-zero bit are not encoded (the decoder can deduce the most significant non-zero bit through the arithmetic symbol that indicates the number of bits required to encode this coefficient).

An important difference between our tree-based wavelet algorithm and others like [5] and [9] is how the coefficient tree building process is detailed. Our algorithm includes a simple

Algorithm 1: Lower Tree Wavelet (LTW) Encoder.

```
(E1) INITIALIZATION
  output rplanes
  output $maxplane = \max_{\forall c_{i,j} \in C} \left\{ \left\lceil \log_2 \left( \left| c_{i,j} \right| \right) \right\rceil \right\}$
(E2) CALCULATE SYMBOLS (first image pass)
  Scan the level subbands (from level 1 to N) in 2×2 blocks.
  For each block $B_n$
    if $|c_{i,j}| < 2^{rplanes} \wedge$ (descendant($c_{i,j}$)=LOWER_COMPONENT $\vee \neg\exists$ descendant($c_{i,j}$) ) $\forall c_{i,j} \in B_n$
      set $c_{i,j}$=LOWER_COMPONENT $\forall c_{i,j} \in B_n$
    else for each $c_{i,j} \in B_n$
      if $|c_{i,j}| < 2^{rplanes}$
        if (descendant($c_{i,j}$)=LOWER_COMPONENT $\vee \neg\exists$ descendant($c_{i,j}$) ) set $c_{i,j}$=LOWER
                                                  else set $c_{i,j}$=ISOLATED_LOWER
(E3) OUTPUT THE COEFFICIENTS (second image pass)
  Scan the subbands (from level N to 1) in 2×2 blocks
  For each $c_{i,j}$ in a subband
    if $c_{i,j} \neq$ LOWER_COMPONENT
      if $c_{i,j}$=LOWER                      arithmetic_output LOWER
      else if $c_{i,j}$=ISOLATED_LOWER        arithmetic_output ISOLATED_LOWER
      else
        $nbits_{i,j} = \left\lceil \log_2 \left( \left| c_{i,j} \right| \right) \right\rceil$
        if (descendant($c_{i,j}$)$\neq$LOWER_COMPONENT$\vee \neg\exists$ descendant($c_{i,j}$)) arithmetic_output $nbits_{i,j}$
                                                  else arithmetic_output $nbits_{i,j}^{LOWER}$
        output $\text{bit}_{nbits_{i,j}-1}\left(\left|c_{i,j}\right|\right) \ldots \text{bit}_{rplane+1}\left(\left|c_{i,j}\right|\right)$
        output sign($c_{i,j}$)
```

Note: $\text{bit}_n(c)$ is a function that returns the nth bit of c.

and efficient recursive method (within the E2 stage) to determine if a coefficient has significant descendants in order to form coefficient trees. However, EZW and SPIHT leave it as an open and implementation dependent aspect, which increases drastically the algorithm complexity if it is not carefully solved (for example, if searches for significant descendants are independently calculated for every coefficient whenever they are needed). Anyway, an efficient implementation for EZW and SPIHT would increase their memory consumption due to the need to store the maximum descendant for every coefficient obtained during a pre-search stage.

B. Lower-Tree Decoder Algorithm

The decoding algorithm performs the reverse process in only one pass, since symbols are directly decoded from the incoming bitstream. All the subbands must be scanned in the order used in the encoder. The granularity of this scan is 2×2 coefficient blocks, thus coefficients that share the same parent (i.e., sibling coefficients) are handled together. Note that all the coefficient blocks have ascendant except those in the $\mathrm{LL_N}$ subband.

When an insignificant coefficient is decoded, we set its value to 0, since its order of magnitude is unknown (we only know that it is lower than $2^{rplanes}$). Later, insignificant coefficients in a lower-tree are automatically propagated, since when the parent of four sibling coefficients has been set to 0, all the descendant coefficients are also assigned a value of 0, so that lower-trees are recursively generated. However, if an isolated lower coefficient has been decoded, it must not be propagated as a lower-tree. Hence, a different value must be assigned. For this case, we keep this coefficient as $ISOLATED\_LOWER$ until its 2×2 direct descendants are scanned. At that moment, we can safely update its value to 0 without risk of unwanted propagations, because no more direct descendants of this coefficient will be scanned.

C. Encoder Features

The proposed algorithm is resolution scalable due to the selected scanning order and the nature of the wavelet transform. This way, the first subband that the decoder attains is the $\mathrm{LL_N}$, which is a low-resolution scaled version of the original image. Then, the decoder progressively receives the remaining subbands, from lower frequency subbands to higher ones, which are used as a complement to the low-resolution image to recursively double its size, which is known as Mallat decomposition [17]. Spatial and SNR scalability are closely related features. Spatial resolution allows us to have different resolution images of the same image. Through interpolation techniques, all these

51	42	-9	2	4	4	0	-1
25	17	10	11	3	1	0	2
12	3	3	-2	2	-2	-5	3
-9	-3	3	-3	0	3	-1	2
-4	1	1	-2	0	2	1	3
2	-3	0	2	1	-1	-1	-2
1	3	2	1	1	2	-3	1
-2	-3	3	-12	2	0	2	1

Fig. 2. A 2-level wavelet transform of an 8 × 8 example image.

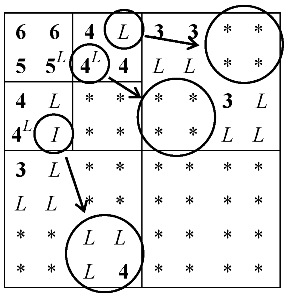
6 **6** **4** *L* **3** **3** * *
5 **5**^{L} **4**^{L} **4** *L* *L* * *
4 *L* * * * * **3** *L*
4^L *I* * * * * *L* *L*
3 *L* * * * * * *
L *L* * * * * * *
* * *L* *L* * * * *
* * *L* **4** * * * *

Fig. 3. Symbols resulting from applying our algorithm to the example image.

images could be resized to the original size, so that the larger an image is, the closer to the original it gets. Therefore, this algorithm could also be used for SNR scalability purposes. Other features, like the definition of Regions Of Interest (ROI), can be implemented in this coder by using a specific $rplanes$ parameter for those coefficients in the trees representing that special area.

As in most non-embedded encoders (like baseline JPEG, H.264 intra mode, non-embedded SPIHT/SPECK, etc.), precise rate control is not feasible in the proposed algorithm because it is usually achieved with bit-plane coding (like in SPIHT) or iterative methods (like in JPEG 2000 or SFQ), which significantly increase the complexity of the encoder. Nevertheless, rate control is still possible based on a fast analysis of the transform image features by extracting one or more numerical parameters that are employed to determine the quantization parameters needed to approximate a bit rate (see [15] for details).

D. A Simple Example

Fig. 2 shows a small 8 × 8 image that has been transformed using a 2-level DWT. In this section, we show the result of applying our tree-based encoder using this sample image. These wavelet coefficients have been scalar quantized, and the selected $rplanes$ parameter is 2. Regarding the maxplane parameter, it can be easily computed as $maxplane = \lceil \log_2(51) \rceil = 6$.

When the coarser quantization is applied using $rplanes = 2$, the values within the interval $]-(2^2)\ldots 2^2[$ are absolutely quantized. Thus, all the significant coefficients can be represented by using from 3 to 6 bits, and hence, the symbol set needed to represent the significance map is $\{\mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{3}^L, \mathbf{4}^L, \mathbf{5}^L, \mathbf{6}^L, L, I\}$. In this symbol set, special "$LOWER$ numeric symbols" are marked with a superscript L, an I symbol represents an isolated lower coefficient, and an L indicates a regular lower symbol (the root of a lower-tree). Coefficients belonging to a previously encoded lower-tree (those labeled as $LOWER\_COMPONENT$) are not encoded and, in our example, are represented with a star (*). Fig. 3 shows the symbols resulting from applying our algorithm to the example image, and if we scan the subbands from the highest level (N) to the lowest one (1), in 2 × 2 blocks, from left to right and top to bottom, the resulting bitstream is the one illustrated in Fig. 4. Note that the sign is not necessary for the LL subband since its coefficients are always positive.

IV. Implementation Considerations

Implementation details and further adjustments may improve the performance of a compression algorithm. In this section, we give a guide for a successful implementation of the LTW algorithm. All the improvements introduced in this section should preserve fast processing.

Context coding has been widely used to improve the R/D performance in image compression. Although high-order context modeling presents high complexity, simpler context coding can be efficiently employed without noticeable increase in execution time. We propose the use of two contexts based on the significance of the left and the upper coefficients, thus if they both are insignificant or close to insignificant, a different model is used for coding. Adding a few more models to establish more significance levels (and thus more number of contexts) would improve compression efficiency. However, it would slow down the algorithm, mainly due to the context formation evaluation and the higher memory usage.

Recall that $maxplane$ indicates the number of bits needed to represent the highest coefficient in the wavelet decomposition. This value (along with $rplanes$) determines the number of symbols needed for the arithmetic encoder. However, coefficients in different subbands tend to be different in magnitude order, so this parameter can be specifically set for every subband level. In this manner, the arithmetic encoder is initialized exactly with the number of symbols needed in every subband, which increases the coding efficiency.

Related to the coarser quantization, consider the case in which three coefficients in a block are insignificant, and the fourth value is very close to insignificant. In this case, we can consider that the entire block is insignificant and all its coefficients can be labeled as $LOWER\_COMPONENT$. The slight error introduced is compensated by the saving in bit budget.

In general, each of these R/D improvements causes a slight increase in PSNR (from 0.05 to 0.1 dB, depending on the source image and the target bitrate).

V. Comparison With Other Wavelet Coders Using Real Implementations

We have implemented the lower-tree wavelet (LTW) coding and decoding algorithms in order to test their performance. They have been implemented using standard C++ language (without using assembly language or platform-dependant features), and the simulation tests have been performed on a regular Personal Computer (with a 500 MHz Pentium Celeron

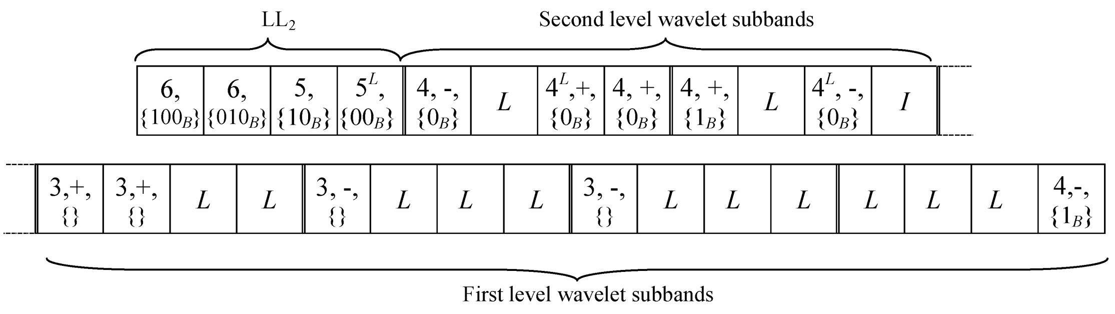

Fig. 4. Example image encoded using the proposed tree-based algorithm.

TABLE I
PSNR (dB) WITH DIFFERENT BIT RATES AND CODERS USING LENA (512 × 512)

codec\ rate	EZW	SPECK	Jasper/ JPEG2000	H.264 intra mode	SPIHT	LTW
1	39.55	40.25	40.31	40.36	40.41	40.50
0.5	36.28	37.10	37.22	37.30	37.21	37.35
0.25	33.17	34.03	34.04	34.42	34.11	34.31
0.125	30.23	n/a	30.84	31.50	31.10	31.27

TABLE II
PSNR (dB) WITH DIFFERENT BIT RATES AND CODERS FOR CAFÉ, WOMAN, BARBARA, AND VISTEX DATABASE

	Café (2560x2048)			*Woman* (2560x2048)		
codec\ rate	SPIHT	Jasper/ JPEG2000	LTW	SPIHT	Jasper/ JPEG2000	LTW
1	31.74	32.04	32.03	38.28	38.43	38.53
0.5	26.49	26.80	26.85	33.59	33.63	33.82
0.25	23.03	23.12	23.24	29.95	29.98	30.16
0.125	20.67	20.74	20.76	27.33	27.33	27.52
	VisTex texture database (512x512)			*Barbara* (512x512)		
codec\ rate	SPIHT	Jasper/ JPEG2000	LTW	SPIHT	Jasper/ JPEG2000	LTW
1	35.73	35.90	36.07	36.41	37.11	36.68
0.5	31.68	31.78	31.99	31.39	32.14	31.76
0.25	28.41	28.36	28.67	27.58	28.34	28.07
0.125	25.88	25.71	26.07	24.86	25.25	25.22

processor with 256 KB L2 cache), generating image files that contain the compressed images, including the file headers required for self-containing decompression. The reader can easily perform new tests by using the LTW implementation available at http://www.disca.upv.es/joliver/LTW.

In order to compare our algorithm with other wavelet encoders, we have selected the classical Lena and Barbara images (monochrome, 8 bpp, 512 × 512), the VisTex texture database (monochrome, 8 bpp, 512 × 512) from the MIT media lab, and the Café and Woman images (monochrome, 8 bpp, 2560 × 2048) from the JPEG 2000 test bed. The widely used Lena image (from the USC) allows us to compare it with practically all the published algorithms, since results are commonly expressed using this image. The VisTex set allows testing the behavior of the coders when coding textures. Finally, the Café and Woman images are less blurred and more complex than Lena and represent pictures taken with a 5-Mpixel-high definition digital camera.

Table I provides R/D performance when using the Lena image. In general, we see that LTW achieves similar or better results than the other coders, including the JPEG 2000 standard, whose results have been obtained using the reference software Jasper [18], an official implementation included in the ISO/IEC 15444-5 standard. For SPIHT, we have used the implementation provided by the authors in the original paper. For H.264 intra mode coding, results have been obtained with the JM 9.6 reference software. For this coder, the PSNR result for each bitrate has been interpolated from the nearest bitrates, since the quantization granularity is very low, and an exact bitrate cannot be achieved.

PSNR results for the rest of images are shown in Table II. In this only SPIHT and Jasper have been compared to LTW, since the compiled versions of the rest of coders have not been released, or results are not published for these images. We can observe that R/D performance for Café is still higher using our algorithm, although Jasper performs similar to LTW. For Woman, we see that our algorithm exceeds in performance both SPIHT and Jasper. In the case of the textures from the VisTex database, we present the average PSNR in this Table, which shows that, in general, LTW works better than JPEG 2000 and SPIHT for coding of textures. Only a few images from the set (and only at high-bit rates) exhibit better coding results for JPEG 2000 (details of the results for each image in the set are available at http://www.disca.upv.es/joliver/csvt/textures.pdf). However, at high bit rates, Jasper encodes high-frequency images, like Barbara, better than LTW. Two reasons may explain it. First, when high-frequency images are encoded at high bit rates, many coefficients in high-frequency subbands are significant, and hence our algorithm is not able to build large lower-trees. Second, recall that JPEG 2000 includes many contexts, and it causes higher performance for high-frequency images. In our experiments, we have observed that if more than two contexts are used in LTW, the R/D performance for Barbara is close to the results shown with Jasper, but at the cost of higher execution time.

On the other hand, the reader can perform a subjective evaluation of the images Lena and Barbara encoded at 0.125 bpp with SPIHT, JPEG 2000 and LTW by using Fig. 5, which shows a detail of the face of Lena and another detail of Barbara's checked trousers. In the first group of pictures, if we look carefully at Lena's lips, eyes and nose, we can observe that LTW offers

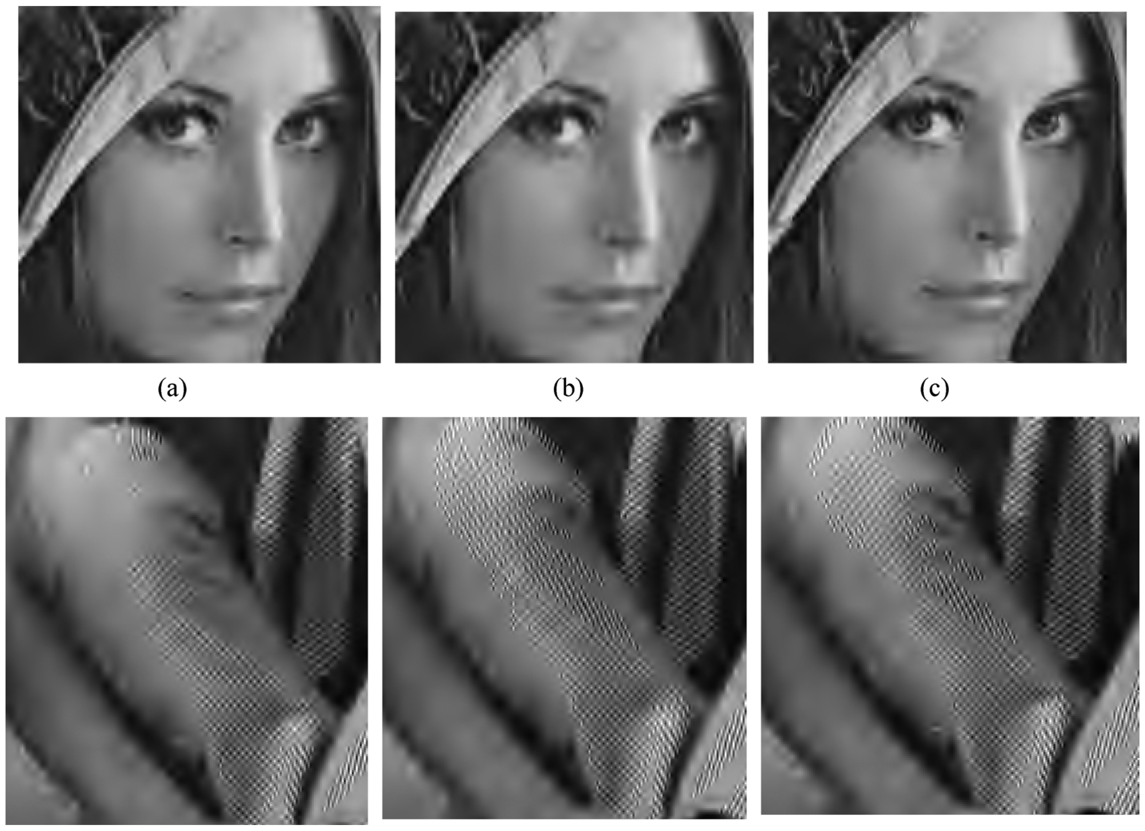

Fig. 5. Details of the Lena and Barbara images (at 0.125 bpp) for a subjective evaluation, using (a) SPIHT, (b) JPEG 2000, and (c) LTW.

TABLE III
EXECUTION TIME COMPARISON FOR LENA AND CAFÉ (TIME IN MILLION OF CPU CYCLES)

	Lena coding (512x512)			*Lena decoding* (512x512)		
codec\ rate	SPIHT	Jasper / JPEG 2000	LTW	SPIHT	Jasper / JPEG 2000	LTW
1	119.4	256.1	51.3	92.8	72.3	47.1
0.5	72.3	238.2	32.9	48.7	51.4	27.1
0.25	48.7	223.4	24.0	29.5	38.1	17.4
0.125	36.8	211.3	19.1	20.5	31.1	12.3
	CafÈ coding (2560x2048)			*CafÈ decoding* (2560x2048)		
codec\ rate	SPIHT	Jasper / JPEG 2000	LTW	SPIHT	Jasper / JPEG 2000	LTW
1	2400.1	6907.6	963.4	1935.3	1475.2	911.6
0.5	1399.3	6543.9	654.3	1024.1	991.8	569.5
0.25	889.8	6246.2	476.7	586.9	763.6	366.6
0.125	624.5	6058.2	378.8	372.1	635.3	253.4

the best results, achieving better definition and sharpness than SPIHT and especially than JPEG 2000. However, if we analyze a highly detailed image such as Barbara, and we focus on a high-frequency area (e.g., the trousers in the Barbara image) we can clearly see that JPEG 2000 gives the best results, and SPIHT, which decompresses an image full of blurred areas, produces the worst results. Both results are consistent with the PSNR results, and confirm that tree-based algorithms work better with moderate to low detailed images, while JPEG 2000 is a better option in detailed images, due to the reasons aforementioned.

The main advantage of the LTW algorithm is its lower complexity. Table III shows that our algorithm greatly outperforms SPIHT and Jasper in terms of execution time.[1] For medium sized images (512×512), our encoder is from 3.25 to 11 times faster than Jasper, whereas LTW decoder executes from 1.5 to 2.5 times faster than Jasper decoder, depending on the rate. In the case of SPIHT, our encoder is from 2 to 2.5 times faster, and the decoding process is from 1.7 to 2.1 times faster. With larger images, like Café (2560×2048), the advantage is greater. LTW encoder is from 5 to 16 times faster than Jasper and the decoder is from 1.5 to 2.5 times faster. With respect to SPIHT, our algorithm encodes Café from 1.7 to 3 times faster, and decodes it from 1.5 to 2.5 times faster.

Note that in these tables we have only evaluated the coding and decoding processes, and not the transform stage, since the wavelet transform used is the same in all the cases; the popular Daubechies 9/7 biorthogonal wavelet filter. Other wavelet transforms, like Daubechies 23/25, have shown better compression performance. However, this improvement is achieved with more filter taps, and thus increasing the execution time of the DWT.

[1]Measuring the complexity of algorithms is a hard issue. Execution time of their implementation is largely dependant of the optimization level. This way, there are commercial implementations of JPEG 2000 not included in the ISO/IEC 15444-5 that are faster than Jasper, however they are usually implemented by using platform dependant pieces of code (in assembly language) and multimedia SIMD instructions. In our tests, SPIHT, JPEG 2000 and LTW implementations are, as far as possible, written and compiled under the same conditions, using plain C/C++ language and MS Visual C++6.0 for all them.

Other non-embedded encoders, like SFQ and H.264 intra mode, are much slower than LTW. In particular, SFQ coding is more than 10 times slower than LTW, while the H.264 encoder (JM 9.6 implementation) is more than 50 times slower than our proposal, due to the time-consuming predictive algorithm used in this encoder.

Besides compression performance and complexity, a third major issue usually considered in an image coder is the memory usage. Our wavelet encoder and decoder are able to perform in-place processing of the wavelet coefficients, and thus they do not need to handle additional lists or memory-consuming structure. This way, only 21 Mbytes are needed to encode the Café image using LTW (note that 20 Mbytes are needed to store the image in memory using *integer* type), whereas SPIHT and Jasper require 42 and 64 Mbytes, respectively.[2]

VI. Conclusion

In this paper, we have presented a new wavelet image encoder based on the construction and efficient coding of wavelet lower-trees (LTW). Its compression performance is within the state-of-the-art, achieving similar results as other popular algorithms (SPIHT is improved in 0.2–0.4 dB, and JPEG 2000 with Lena in 0.35 dB as mean value).

However, the main contribution of this algorithm is its lower complexity. Depending on the image size and bitrate, it is able to encode an image up to 15 times faster than Jasper and three times faster than SPIHT. Thus, it can be stated that the LTW coder is one of the fastest efficient image coders that can be reported in the literature.

Therefore, due to its lower complexity, its high symmetry, its simply design, and the lack of memory overhead, we think that the LTW is a good candidate for real-time interactive multimedia communications, allowing implementations both in hardware and in software.

[2]Results obtained with the Windows XP task manager, "peak memory usage" column.

References

[1] J. Oliver and M. P. Malumbres, "Fast and efficient spatial scalable image compression using wavelet lower trees," in *Proc. IEEE Data Compression Conf.*, Snowbird, UT, Mar. 2003, pp. 133–142.

[2] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Norwell, MA: Kluwer, 1991.

[3] A. E. Jacquin, "Image coding based on a fractal theory of iterated contractive image transformation," *IEEE Trans. Image Process.*, vol. 1, pp. 18–30, Jan. 1992.

[4] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. Image Processing*, vol. 1, no. 2, pp. 205–220, Feb. 1992.

[5] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3445–3462, Dec. 1993.

[6] *JPEG2000 Image Coding System*, ISO/IEC 15444-1, 2000.

[7] C. Chrysafis and A. Ortega, "Line-based, reduced memory, wavelet image compression," *IEEE Trans. Image Process.*, vol. 9, pp. 378–389, Mar. 2000.

[8] W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets," *App. Comp. Harmon. Anal.*, vol. 3, pp. 186–200, 1996.

[9] A. Said and A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 3, pp. 243–250, Jun. 1996.

[10] Z. Xiong, K. Ramchandran, and M. Orchard, "Space-frequency quantization for wavelet image coding," *IEEE Trans. Image Process.*, vol. 46, no. 5, pp. 677–693, May 1997.

[11] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said, "Efficient, low-complexity image coding with a set-partitioning embedded block coder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 11, pp. 1219–1235, Nov. 2004.

[12] C. Chrysafis, A. Said, A. Drukarev, A. Islam, and W. A. Pearlman, "SBHP—a low complexity wavelet coder," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2000, pp. 2035–2038.

[13] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Image Process.*, vol. 9, no. 7, pp. 1158–1170, Jul. 2000.

[14] W. A. Pearlman, "Trends of tree-based, set partitioning compression techniques in still and moving image systems," in *Proc. Picture Coding Symp.*, Apr. 2001, pp. 1–8.

[15] O. Lopez, M. Martinez-Rach, J. Oliver, and M. P. Malumbres, "A heuristic bit rate control for non-embedded wavelet image coders," in *Proc. Int. Symp. ELMAR Focused Multimedia Signal Process.*, Jun. 2006, pp. 13–16.

[16] A. Al, B. P. Rao, S. S. Kudva, S. Babu, D. Sumam, and A. V. Rao, "Quality and complexity comparison of H.264 intra mode with JPEG2000 and JPEG," in *Proc. IEEE ICIP*, 2004, pp. 525–528.

[17] S. Mallat, "A theory for multiresolution signal decomposition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 7, pp. 669–718, Jul. 1989.

[18] M. Adams, Jasper Software Reference Manual (v1.6). Oct. 2002, ISO/IEC JTC 1/SC 29/WG 1 N 2415.