Fast and efficient image coding for interactive network applications

José Oliver and Manuel Pérez Malumbres

Abstract—**Multimedia network systems require image and video compression schemes in order to efficiently take profit from the available bandwidth.**

In this paper, we present a wavelet still-image coder, called LTW (Lower-Tree Wavelet), based on the construction and codification of coefficient trees as other proposals do. This algorithm is fast and symmetric (except in extremely low bit rates), which makes it adequate for real-time interactive multimedia applications. We have compared our algorithm with several well-known coders in terms of rate/distortion performance using the standard Lena image. Results show that LTW, with lower temporal complexity, achieves better results than EZW (0.9 dB PSNR) and stack-run (0.25 dB). Also, we have tested the temporal complexity of LTW algorithm, resulting 3.5 times faster than an optimized EZW. On the other hand, compared to DCT-based standards, like JPEG, our algorithm outperforms them in 5 dB approx. (at similar bit rates)

Key words—**Still image coding; interactive multimedia network application; wavelet coding; tree oriented encoder.**

I. Introduction

One of the main part within a multimedia network system is the efficient data encoder. Multimedia information naturally takes large amount of data to be represented and thus, a compression system is required in order to avoid wasting bandwidth.

In order to encode image and video data, the most widely used technique is transform domain. During the last decade, some popular standards emerged using the Discrete Cosine Transform (DCT), but currently a new image transform strategy has shown better behavior than the DCT-based ones. This mathematical tool is known as wavelet transform.

A wide variety of wavelet-based image compression schemes have been reported in the literature. The early wavelet image coders [1] were designed to exploit the ability of compacting energy on the wavelet decomposition. They used quantizers and variable-length entropy coders, showing little improvements with respect to the popular DCT-based ones.

However, the properties of wavelet coefficients can be exploited more efficiently. In that sense, Shapiro [3] developed a wavelet-based encoder that considerably improves the previous proposals. The encoder, called Embedded Zero-tree Wavelet encoder (EZW), is mainly based on two questions (a) the similarity between the same kind of sub-band in a wavelet decomposition, and (b) a quantization based on a successive-approximation scheme that can be adjusted in order to get a specific bit rate. The encoder includes an entropy encoder (typically an adaptive arithmetic encoder) as its final stage.

Said and Pearlman [2] proposed a variation of EZW, called SPIHT (Set Partitioning In Hierarchical Trees). It achieves better results than EZW, even without taking into account the final arithmetic encoding stage. The improvements are due to the way it groups the wavelet coefficients and how it stores the significant information.

A different approach to the previous algorithms is the one proposed in [4], known as the stack-run algorithm. This algorithm has a similar structure than JPEG coders. That is, after wavelet decomposition, wavelet coefficients are quantized using a classic quantization scheme. Then, quantized coefficients are entropy coded using a run-length encoder (RLE) and, finally, an arithmetic encoder is used.

In [5], a joint space-frequency quantization scheme was proposed. It uses a spatial quantization, like zero-tree, in combination with a standard scalar quantizer. The idea is based in the fact that natural images are perfectly modeled by a linear combination of compacted energy in both frequency and space domains.

One of the most widely used technique from the above presented ones is tree encoding. However, this kind of coder exhibit an important asymmetry, due to the way that construction of significance coefficient maps and refinement stages are performed in the encoding stage. So this kind of coder, by nature, are not able to work efficiently with interactive multimedia applications.

In this paper, we propose a new wavelet still-image coder that it is simpler and faster than others previously published [3][2]. We have called it LTW (Lower-Tree Wavelet) coder. The main contribution of LTW is the way that it builds the coefficient map. It does not use an iterative loop in order to determine the significant coefficients and to assign them bits. It builds the significant map in only one step using two symbols for pruning tree branches, and then, depending on the required target bit rate, it codes the significant coefficients also in one step. This algorithm significantly reduces the complexity of the encoder stage in such manner that it is similar to the decoder stage. So, another important feature of LTW is its symmetric behavior at certain bit rates.

In section 2 a description of the proposed algorithm is shown. In section 3, we show a performance evaluation

DISCA, UPV, Valencia (Spain). E-mail: {joliver, mperez}@disca.upv.es This work is funded by the Generalitat Valenciana research grant CTIDIB/2002/19

of our proposed scheme in terms of rate/distortion and computation complexity performance metrics. Section 4 presents some design considerations. Finally, in section 5 some conclusions and future work are drawn.

II. The Lower-Tree Wavelet Coder

For the most part, digital images are represented with a set of pixel values. The encoder proposed in this paper can be applied to a set of coefficients *C* resulting from a dyadic decomposition $\Omega(\cdot)$, in order that $C=\Omega(P)$. The most commonly used dyadic decomposition in image compression is the hierarchical wavelet subband transform [1], so an element $c_{i,j} \in C$ is called transform coefficient.

Tree oriented wavelet image encoders are proved to efficiently transmit or store the set *C*, achieving great performance results. In these algorithms, two stages can be established. The first one consists on encoding the significance map, i.e., the location and amount of bits required to represent those coefficients that will be encoded (significant coefficients). In the second stage, significant transform coefficients are encoded, i.e. their sign and magnitude bits, depending on the desired target bit rate.

One of the main drawbacks in previous tree oriented wavelet image encoders is their high temporal complexity. That is mainly due to the bit plane processing at the construction of the significance map, that is performed along different iterations, using a threshold that focuses on a different bit plane in each iteration. Moreover, the bits of the significant coefficients are also bit plane processed.

Our proposed LTW algorithm is able to encode the significance map without performing one loop scan per bit plane. Instead of it, only one scan of the transform coefficients is needed. The LTW also can encode the bits of the significant transform coefficients in only one scan.

Let us define some concepts before the LTW be explained. Like in the rest of tree encoding techniques, coefficients from *C* can be logically arranged as a tree. In our algorithm, every coefficient $c_{a,b}$ in the LL subband (the scaled version of the original image) is the root of a tree. For each root node placed at (*a*, *b*), its offspring will be formed by three coefficients placed at (*a*+width(LL), *b*), (*a*, *b*+height(LL)) and (a+width(LL), b+height(LL)). The offspring of the rest of nodes (*c*, *d*) are the four coefficients placed at (2*c*, 2*d*), (2*c*+1, 2*d*), (2*c*, 2*d*+1), (2*c*+1, 2*d*+1) (except for those nodes in the first level of decomposition subbands, LH0, HL0 and HH0, that represent the leaves of the trees).

We also have to define the order to scan the subbands in the first stage, where the significance map is built. We use a zig-zag order, starting from the LL subband, so that all the subbands at a level *n* are always scanned before the *n*-1 subbands. Finally, coefficients in a subband are scanned in a Morton order. Notice that both the scan order and the trees are defined in a similar way that in Shapiro's EZW algorithm.

Now we are ready to define the algorithm. Let us start with the encoder part. The quantization process is performed by two strategies: one coarser and another finer. The finer one consists on applying a scalar uniform quantization to the coefficients, and it is performed before the LTW algorithm. On the other hand, the coarser one is based on removing bit planes from the least significant part of the coefficients, and it belongs to the LTW encoder. We define *rplanes* as the number of less significant bits that are going to be removed in the LTW.

At the initialization of the encoder, it is calculated the maximum number of bits needed to represent the higher coefficient (*maxplane*) and it is output to the decoder. The *rplanes* parameter is also output. With these data, we initialize an adaptive arithmetic encoder that will be used to transmit the number of bits required to encode any coefficient. We will only transmit those coefficients that require more than *rplanes* bits to be coded, so only *maxplane-rplanes* symbols are needed to represent this information. We also use two extra symbols to efficiently represent the significance map.

In the next stage the significance map is encoded as following. All the subbands are scanned in zig-zag order and for each subband all the coefficients are scanned in Morton order, as explained previously. Then, for each coefficient, if it is significant (i.e., it is different to zero if we discard the first *rplanes* bits) the number of bits required to represent that coefficient is encoded with an adaptive arithmetic encoder. As coefficients in the same subband have similar magnitude, and due to the order we have established to scan the coefficients, the adaptive arithmetic encoder is able to encode very efficiently the number of bits of the transform coefficients. On the other hand, if a coefficient is not significant and all its descendents are not significant (they form a lower-tree), the symbol *LOWER* is encoded and this coefficient and its descendents are marked as not active (initially all them are active). A not active coefficient is not processed any more, neither in the first stage nor in the second one. Finally, if the coefficient is insignificant but it has at least one significant descendent, the symbol *ISOLATED_LOWER* is encoded and only this coefficient is marked as not active.

The second stage consists on encoding the significant coefficients discarding the first *rplanes* bits and their most significant bit (it can be inferred by the decoder). In order to speed up the execution time of the algorithm, we may not use an arithmetic encoder, what results in a very small lost in performance. The sign is transmitted in a similar way.

The LTW encoder and decoder algorithms are defined as follows.

```
Encoder Algorithm:

(E1) INITIALIZATION

    output rplanes
    output $\max plane = \max_{\forall c_{i,j} \in C} \{ \lfloor \log_2(|c_{i,j}|) \rfloor \}$
    mark all $c_{i,j} \in C$ as active
```

(E2) OUTPUT THE SIGNIFICANCE MAP. Scan the subbands (zig-zag order). For each $c_{i,j}$ in a subband

```
    if active( $c_{i,j}$ )
        $nbits_{i,j} = \lceil \log_2(|c_{i,j}|) \rceil$
        if $nbits_{i,j} > rplanes$
            arithmetic_output $nbits_{i,j}$
        else
            mark $c_{i,j}$ as not active
            $D_{i,j} = \{\text{descendant}(c_{i,j})\}$
            $nmaxdesc = \max_{\forall c_{x,y} \in D_{i,j}} \left\{ \left\lceil \log_2\left( |c_{x,y}| \right) \right\rceil \right\}$
            if $nmaxdesc > rplanes$
                arithmetic_output ISOLATED_LOWER
            else
                mark all $c_{x,y} \in D_{i,j}$ as not active
                arithmetic_output LOWER
```

E3) OUTPUT THE SIGNIFICANT TRANSFORM COEFFICIENTS. Scan *C* in an established order. For each $c_{i,j} \in C$

```
    if active( $c_{i,j}$ )
        output
            $\text{bit}_{nbits_{(i,j)}-1}(|c_{i,j}|) \ldots \text{bit}_{rplane+1}(|c_{i,j}|)$
        output sign( $c_{i,j}$ )
```

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

Decoder Algorithm:

D1) INITIALIZATION

```
    input rplanes, maxplane
    mark all $c_{i,j} \in C$ as active
```

D2) INPUT THE SIGNIFICANCE MAP. Scan the subbands in the same order as in E2). For each $c_{i,j}$ in a subband

```
    if active( $c_{i,j}$ )
        arithmetic_input $nbits_{i,j}$
        if $nbits_{i,j}$ =ISOLATED_LOWER
            mark $c_{i,j}$ as not active
        if $nbits_{i,j}$ = LOWER
            $D_{i,j} = \{\text{descendant}(c_{i,j})\}$
            mark $c_{i,j}$ and all $c_{x,y} \in D_{i,j}$ as not active
```

D3) INPUT THE SIGNIFICANT TRANSFORM COEFFICIENTS. Scan *C* in the same order as in E3). For each $c_{i,j} \in C$

```
    if active( $c_{i,j}$ )
        $\text{setbit}_{nbits_{(i,j)}}(|c_{i,j}|)$
        input $\text{bit}_{nbits_{(i,j)}-1}(|c_{i,j}|) \ldots \text{bit}_{rplane+1}(|c_{i,j}|)$
        $\text{setbit}_{\text{rplane}}(|c_{i,j}|)$
        input sign( $c_{i,j}$ )
```

Note: $\text{bit}_n(c)$ is a function that writes the nth bit of *c*, and $\text{setbit}_n(c)$ set one the nth bit of *c*.

Notice that, in the decoder at D3), the *rplane*th bit of each significant coefficient is set to one in order to reduce the error interval of the recovered coefficients.

A. Comparison with other tree-based wavelet encoders

Like in other tree-based wavelet encoders, in the LTW algorithm there are two stages, in the first one the significance map is encoded (it is called dominant pass in EZW and sorting pass in SPIHT) and in the second one the significant coefficients are encoded (called subordinate pass in EZW and refinement pass in SPIHT). Unlike them, in the LTW the significance map and the significant coefficients are encoded in only one iteration, without the need of an iterative loop scanning the same trees once per bit plane. Moreover, several lists must be handled in both the EZW and the SPIHT algorithms, while the LTW does not need the construction of lists. In fact, implementing this algorithm is simpler and it has lower temporal complexity (as shown in section 3).

One disadvantage of the LTW algorithm is that it is not naturally embedded (unlike EZW and SPIHT). Instead of it the bit rate is adjusted using two quantization parameters in the same way as in the widely used JPEG and MPEG standard.

III. SIMULATION RESULTS

We have implemented the LTW encoder and decoder algorithm in order to test its performance. It has been implemented using standard C language, and all the simulation tests have been performed on a regular Personal Computer, with an AMD K7 Processor. The selected image has been the standard Lena (monochrome, 8 bpp, 512x512). This allows us to compare the LTW performance with other codecs.

A six-level dyadic wavelet transform has been used, with biorthogonal 10/18 filter [4], although other filters like 9/7 [1] have shown similar behaviour, as we will see in the next section.

Table 1 presents a performance comparison, in terms of image quality (PSNR) at different bit rates (bpp). It shows that the proposed codec outperforms the EZW in approx. 0.8 dB at low bit rates, and others not tree-oriented codecs, like the stack-run, are also improved. SPIHT uses a more complex algorithm to group the coefficients and therefore achieves slightly higher performance (0.2 dB).

On the other hand, the DCT-based standard JPEG is widely outperformed by all the wavelet-based algorithms, what shows the better performance of wavelet transform compared to discrete cosine transform used in image compression. In particular, at all bit rates, LTW is approximately 5 dB higher than JPEG, in terms of PSNR. This comparison is shown in a subjective way

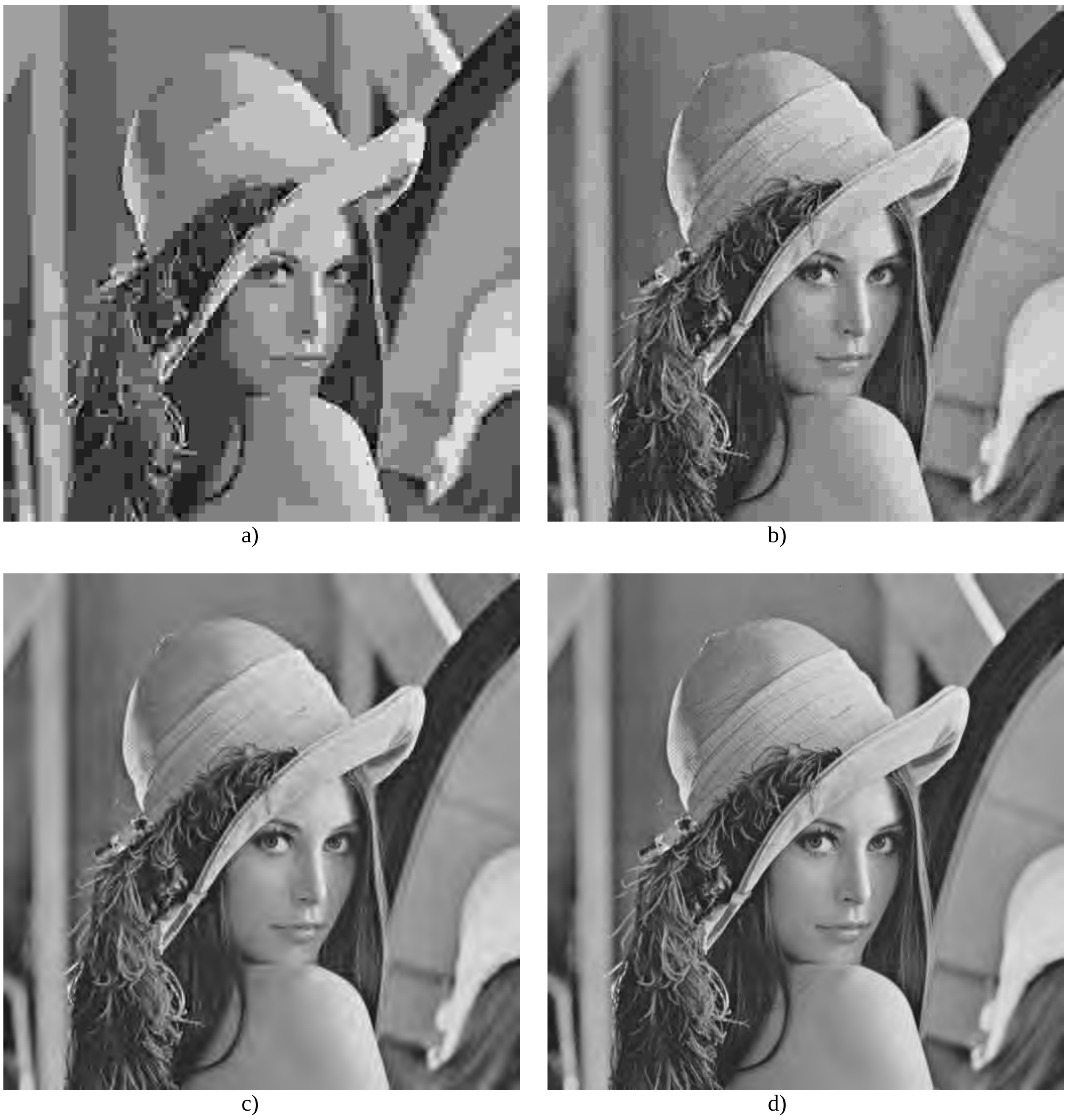

Fig. 1. Lena image compressed using JPEG a) and b), and using wavelets (LTW) c) and d)
(left column at very low bit rates (0.125 bpp approx.) and right column at low bit rates (0.25 bpp))

in figure 1, that clearly states what objective measures have revealed.

TABLE I

PSNR (dB) WITH DIFFERENT BIT RATES AND CODECS

codec\bpp	JPEG	EZW	Stack-run	SPIHT	Preliminary LTW
1	35.70	39.55	n/a	40.41	40.12
0.5	32.87	36.28	36.89	37.21	37.01
0.25	29.12	33.17	33.80	34.11	33.93
0.125	-	30.23	n/a	31.10	31.04

One of the main advantages of the LTW algorithm is its lower temporal complexity. In order to perform a practical comparison between LTW and EZW, we have implemented a version of the EZW. This program runs the EZW in an efficient way. For instance, in the initialization section of the algorithm, the highest descendent of every coefficient is efficiently calculated (cost O(n^2) for a *n*x*n* image), therefore there is no need to explore the trees in the dominant pass to know if a coefficient is encoded as zero-tree root or as isolated zero (see [3]). Figure 2 shows as our algorithm greatly outperforms the EZW in terms of execution time (the encoder is over 3.5 times faster and the decoder about 2.5). On the other hand, the LTW encoder and decoder are much more symmetric than the EZW. Notice that, except at very low bit rates, the execution time for the LTW encoder is very similar to the execution time for the decoder. The exploration of the trees (i.e., looking for significant descendents) is only performed on the encoder side, and its temporal complexity is the same at any rate, that is what makes the LTW really asymmetric at very low bit rates (lower than 0.25 bpp).

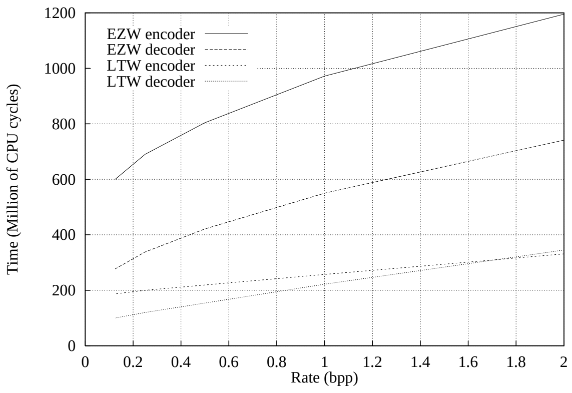

Fig. 2. Execution time comparison (EZW and LTW)

IV. Some design considerations in LTW

During the development of a wavelet image encoder, many design options appear. Some of them involve the wavelet transform process (filter used, number of wavelet decomposition) and others the encoding process.

A. *Basic options*

Choosing a good filter set is crucial in order to achieve good compactness of the image in the LL band, in this way, the amount of nonzero coefficients and its magnitude are reduced, and therefore the image entropy. Shapiro's EZW uses an Adelson 9-tap QMF bank filter, however, it has been proved that biorthogonal filters, like B9/7 and Villasenor 10/18, provide better results. These filters make better energy compaction and are symmetric, what results in lower magnitude of the coefficients on the border of the image on the condition that a symmetric extension of the image is applied. Therefore, biorthogonal Villasenor 10/18 filter [4] will be used, although other biorthogonal filters like B9/7 have shown similar behavior.

Another important aspect in wavelet processing is the number of decomposition levels performed. It mainly depends on the image size and the number of filter taps. With our image, which is 512x512, a six level dyadic decomposition is suitable, resulting in a final 8x8 LL subband.

Let us focus on the quantization process. In section 2 it has been explained how the bit rate and its corresponding distortion factor can be modified by means of two quantization parameters, one finer and another coarser (*rplanes*). LTW is not naturally embedded, it is the price that we have to pay for the lower temporal complexity. Instead of it, the bit rate is adjusted using two quantization parameters in a similar way as in the widely used JPEG standard.

In fact, the finer quantization parameter is used to adjust the bit rate in an extremely accurately way. This parameter is actually a scaling factor, rather than a quantization factor. So, if q is any real number representing the finer quantization (typically within the interval [0, 1]), it is easy to see that the quantization process involving the algorithm is equivalent to multiply all the wavelet transform coefficients by q and then perform an integer division by $2^{rplanes}$. In this sense, two different *rplanes* values may represent the same global quantization whenever we choose the suitable q value, i.e. if *rplanes* is decreased in one q should be divided by two whereas if *rplanes* is increased in one q should be doubled.

Figure 3 shows the relation between the quantization parameters and the final bit rate achieved. In these curves it can be easily seen the equivalence previously mentioned. The bit rate corresponding to q=0.2 and *rplanes*=6 is roughly 0.5 bpp. If we decrease *rplanes* in one, the same bit rate is achieved with q=0.4, and if we decrease it again it is achieved with q=0.8.

The same effect can be observed in figure 4, where the image quality (PSNR) is evaluated in function of the quantization parameters.

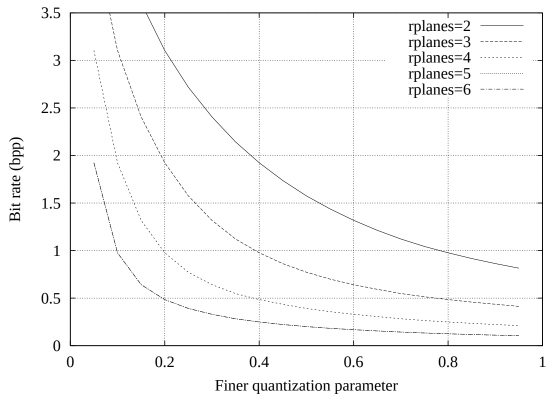

Fig. 3: Relation between quantization parameters and bit rate

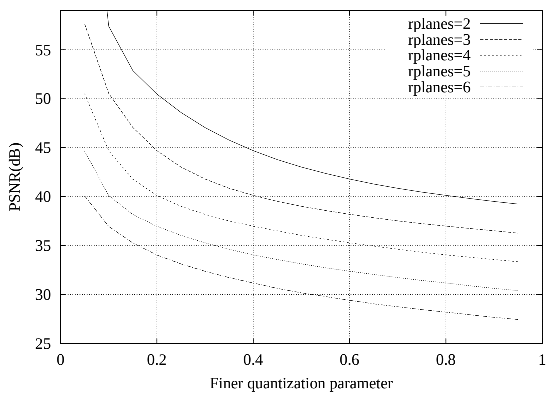

Fig. 4: Relation between quantization parameters and PSNR

B. *Analyzing the adaptive arithmetic encoder*

As coefficients in the same subband have similar magnitude, and due to the order we have established to scan the coefficients, an adaptive arithmetic encoder [6] is able to encode very efficiently the number of bits of the transform coefficients (i.e. the significance map used by the LTW algorithm). That is why this mechanism is essential in the R/D performance of the encoder.

A regular adaptive arithmetic encoder uses one dynamic histogram in order to estimate the current probability of a symbol. To improve this estimation we can use a different histogram depending on the decomposition level of the wavelet subband. It makes sense because coefficients in different subbands tend to have different magnitude, whereas those in the same decomposition level have similar magnitude.

Table II, column a) shows the performance of the preliminary LTW in terms of image quality (PSNR) for different bit rates (bpp). In column b) we can see the results of using a different histogram on every decomposition level. It results clearly beneficial with no penalty in temporal complexity.

In section 2 we have defined *maxplane* as the maximum number of bits needed to represent the higher coefficient in the wavelet decomposition, and it is the value used to initialize the arithmetic encoder. At this point, we can define *maxplaneL* as the maximum number of bits needed to represent the higher coefficient in the level L. So these values can be used to adjust the initialization of every arithmetic encoder, provided all *maxplaneL* symbols are output to the decoder. The drawback introduced by the needed of encoding these symbols is manifestly compensated by the improvements achieved, just as column c) in table II shows.

Last column in this table presents the very little advantage attained by removing the possibility of appearance of a ISOLATED_LOWER symbol in the last level of the wavelet transform.

Several other actions can be tackled directly on the adaptive arithmetic encoder. On the one hand, the maximum frequency count (see more details in [6]) proposed by the authors is 16384 (when using 16 bits for coding). Practical experiences led Shapiro to reduce this value to 1024, and in LTW a value of 512 has been shown more adequate. On the other hand, another parameter that can be adjusted is how many the histogram is increased with every symbol. If this value is greater than one, the adaptive arithmetic encoder may converge faster to local image features, but increasing it too high may turn the model (pdf estimation) inappropriate, leading to poorer performance.

TABLE II

PSNR(DB) WITH DIFFERENT OPTIONS IN THE ARITH. ENCODER

opt/bpp	a)	b)	c)	d)
1	40.12	40.19	40.25	40.26
0.5	37.01	37.06	37.11	37.12
0.25	33.93	34.00	34.07	34.07
0.125	31.04	31.10	31.17	31.17

TABLE III

PSNR (DB) WITH DIFFERENT BIT RATES AND THE FINAL LTW CODEC

codec\bpp	JPEG	EZW	Stack-run	SPIHT	Final LTW
1	35.70	39.55	n/a	40.41	40.26
0.5	32.87	36.28	36.89	37.21	37.12
0.25	29.12	33.17	33.80	34.11	34.07
0.125	-	30.23	n/a	31.10	31.17

V. CONCLUSIONS

In this paper, we have presented the LTW encoder, a wavelet still-image encoder based on the construction and efficient coding of wavelet coefficient trees. Due to its higher symmetry and lower temporal complexity, we think that the LTW is a good candidate for real-time interactive multimedia communications.

We have evaluated our proposal, comparing its performance in terms of rate/distortion with the JPEG, EZW, SPIHT and stack-run algorithms. According to table III, results show that LTW improves EZW and stack-run in 0.9 and 0.25 dB respectively, and show similar performance to SPIHT algorithm. However, we have shown that the main contribution of this algorithm is its lower temporal complexity. In particular, LTW is able to code the standard Lena image up to 3.5 times faster than EZW.

As future work, we are planning to optimize the LTW encoder and include it in a Motion Wavelet video encoder, testing its performance using common video sequences.

VI. REFERENCES

[1] M. Antonini, M. Barlaud, P. Mathieu, I. Daubechies. “Image coding using wavelet transform,” *IEEE Trans Image Processing*, vol 1. nº 2. pp. 205-220, 1992

[2] A. Said, A. Pearlman. “A new, fast, and efficient image codec based on set partitioning in hierarchical trees,” *IEEE Transactions on circuits and systems for video technology*, vol. 6, nº 3, June 1996

[3] J.M. Shapiro, “Embedded Image Coding Using Zerotrees of Wavelet Coefficients,” *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445-3462, December 1993.

[4] M.J. Tsai, J. Villasenor, F. Chen. “Stack-run image coding,” *IEEE Trans. on Circuits and Systems for Video Technology*, vol 6, nº 10, pp. 519-521, Oct. 1996

[5] Z. Xiong, K. Ramchandran, M.T. Orchard. “Space-frequency quantization for wavelet image coding,” *IEEE Trans. on image processing*, vol.6, nº5, pp.677-693, May 1997

[6] I.H. Witten, R.M. Neal, J.G. Cleary, “Arithmetic coding for compression,” *Commun. ACM*, vol 30. pp. 520-540, 1986.