A NEW FAST LOWER-TREE WAVELET IMAGE ENCODER

J. Oliver, M.P. Malumbres

Department of Computer Engineering (DISCA)
Universidad Politécnica de Valencia
Camino de Vera 17, 46071 Valencia, Spain
E-mail: {joliver, mperez}@gap.upv.es

ABSTRACT

During the last decade a lot of research and develop efforts have been made to design competitive still image coders for several kinds of applications.

In this paper, we present a new wavelet still-image coder, called LWT (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 (in 0.8 dB PSNR) and stack-run (in 0.13 dB). Also, we have tested the temporal complexity of LTW algorithm, resulting 3.5 times faster than an optimized EZW.

1. INTRODUCTION

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-bands 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 own 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 lineal combination of compacted energy in both domains the frequency and space.

One of the most widely used technique from the above presented ones is tree encoding. However, this kind of coders 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 coders, by nature, are not able to work efficiently in 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 reduces the complexity on the encoder stage in such manner that it is similar to de decoder stage. So, another important feature of LTW is its symmetric behavior.

In section 2 a description of the proposed algorithm is shown. In section 3, we show a performance evaluation of our proposed scheme in terms of rate/distortion and computation complexity performance metrics. Finally, in section 4 some conclusions and future work are drawn.

2. THE LOWER-TREE WAVELET (LTW) 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] (with a lot of advantages over other transform methods as the DCT), 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 a 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). And 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, 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), the following 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 (2c, 2d), (2c+1, 2d), (2c, 2d+1), (2c+1, 2d+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 on 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 last 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 $plane = \max_{\forall c_{i,j} \in C} \{\lceil \log_2 (|c_{i,j}|) \rceil\}$

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

$D_{i,j} = \{\text{descendant}(c_{i,j})\}$

$\text{nmaxdesc} = \max_{\forall c_{x,y} \in D_{i,j}} \{\lceil \log_2 (|c_{x,y}|) \rceil\}$

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 *n*th 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 n^{th} bit of *c*, and $\text{setbit}_n(c)$ set one the n^{th} bit of *c*.

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

2.1. Comparison with other tree-based 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 (see performance comparison 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 MPEG standard.

3. SIMULATION RESULTS

We have implemented the LTW encoder and decoder algorithm in order to verify the expected results. 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 [2], although other filters like 9/7 [3] have shown similar behavior. 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 bpp at low rates, and others codecs not tree-oriented, 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).

Codec\rate	LTW	EZW	Stack-run	SPIHT
2	44.79	N/a	n/a	n/a
1	40.12	39.55	n/a	n/a
0.5	37.01	36.28	36.89	37.21
0.25	33.93	33.17	33.80	34.11
0.125	31.04	30.23	n/a	n/a

Table 1: PSNR (dB) with different bit rates and codecs

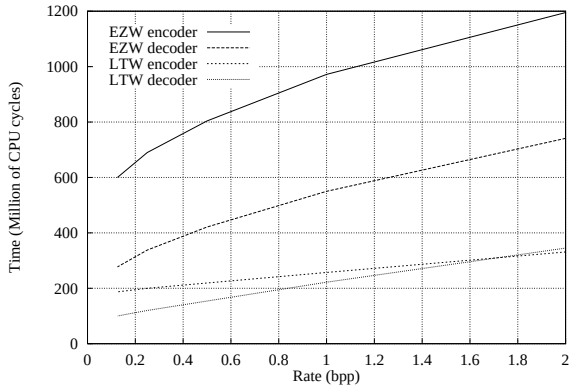

Figure 1: Execution time comparison (EZW and LTW)

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 root of zero tree or as isolated zero (see [4]). Figure 1 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).

4. CONCLUSIONS

In this paper, we have presented the LTW encoder, a new 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 EZW, SPIHT and stack-run algorithms. Results show that LTW improves EZW and stack-run in 0.8 and 0.13 dB at most bit rates but it is not able to reach the SPIHT performance, being 0.2 dB behind it. 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 on common video sequences.

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 Transac ions on circ i s an s s ems or i eo ec no og* , 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