# HUFFMAN CODING OF WAVELET LOWER TREES FOR VERY FAST IMAGE COMPRESSION

*Jose Oliver*

Department of Computer Engineering (DISCA)
Technical University of Valencia
Camino de Vera 17, 46017 Valencia, Spain
E-mail: joliver@disca.upv.es

*Manuel P. Malumbres*

Departamento de Física y ATC
Miguel Hernández University
Avd. Universidad s/n, 03202 Elche, Spain
E-mail: mels@umh.es

## ABSTRACT

In this paper, a very fast variation of the Lower-Tree Wavelet (LTW) image encoder is presented. LTW is a fast non-embedded encoder with state-of-the-art compression efficiency, which employs a tree structure as a fast method of coding coefficients, being faster than other encoders like SPIHT or JPEG 2000. The alternative Huffman-based encoder presented in this paper serves to largely reduce the execution time, at the expense of loss in coding efficiency. Experimental results show that this encoder is more efficient than other very fast wavelet encoders, like the recently proposed PROGRESS (which is surpassed in up to 0.5 dB), and faster than them (from 4 to 9 times in coding). Compared with the JPEG 2000 reference software, the encoder is from 18 to 38 times faster, while PSNR is similar at low bit-rates, and about 0.5 lower at high bit-rates.[1]

## 1. INTRODUCTION

Great efforts have been made to improve coding efficiency of wavelet-based image encoders, achieving in this way a reduction in the bandwidth or amount of memory needed to transmit or store a compressed image. Unfortunately, many of these coding optimizations involve higher complexity, requiring faster and more expensive processors. For example, the JPEG 2000 [1] standard uses a large number of contexts and an iterative time-consuming optimization algorithm (called PCRD) to improve coding efficiency. Other encoders (like the one proposed in [2]) achieve very good coding efficiency with the introduction of high-order context modeling, being the model formation a really slow process. Even bit-plane coding employed in many encoders (like [3] and [4]) results in slow coding process since an image is scanned several times, focusing on a different bit-plane in each pass, which in addition causes a high cache miss rate. In [5], a tree-based wavelet encoder (LTW) is presented, which avoids all these complex techniques to minimize the execution time, although without loss of coding efficiency. In this paper, some proposals to reduce the complexity of LTW still more, at the expense of moderate loss of compression efficiency, are introduced.

## 2. PREVIOUS VERY FAST IMAGE ENCODERS

Other very fast wavelet image encoders have been reported in the literature. Basically, these encoders do not present any type of iterative method, and each coefficient is encoded as soon as it is visited. This results in the impossibility to perform SNR scalability and precise rate control. They simply apply a constant quantization to all the wavelet coefficients, encoding the image at a constant and uniform quality, as it happened in the former JPEG standard, where only a quality parameter was available (and no rate control was performed). In our encoder, we will also take this approach.

One of these very fast encoders, called SBHP, was introduced in [4]. In this proposal, the wavelet subbands are divided into blocks, and each block is partitioned depending on its significance with respect to a threshold value. The significance of each new sub-block is Huffman encoded, so that blocks are partitioned until the significant coefficients (with respect to the threshold) are located (note that it is similar to SPIHT [3] but it uses rectangular structures instead of zerotrees). Although the basic SBHP algorithm is embedded, the really very fast version of SBHP (*non-embedded* SBHP) is not, because coefficients are entirely encoded as soon as they are found to be significant.

Another very fast non-embedded encoder has been recently proposed in [6]. This encoder is called PROGRESS. It follows the same ideas of [5], avoiding bit-plane coding, using coefficient trees to encode wavelet coefficients in only one-pass, and arranging the coefficients in order to achieve resolution scalability. In this encoder, all

[1] This paper was supported by the Spanish *Ministerio de Ciencia y Tecnología* under grant MEC TIC2003-00339.

the coefficients (and not only the zero coefficients) are arranged in trees. The number of bits needed to encode the highest coefficient in each tree is computed, and all the coefficients at the current subband level are binary encoded with that number of bits. Then, the following level subband is encoded (in decreasing order), simply by computing again the number of bits needed to represent each sub-tree at that level and using that number of bits again.

## 3. LOWER TREE WAVELET CODING USING HUFFMAN CODES

For the most part, digital images are represented with a set of pixel values, *P*. The encoder proposed in this paper can be applied to a set of coefficients *C* resulting from a dyadic decomposition $\Omega(\cdot)$, so that $C=\Omega(P)$. The most commonly used dyadic decomposition for image compression is the hierarchical wavelet subband transform, therefore an element $c_{i,j} \in C$ is called transform coefficient. In a wavelet transform, we call $LH_1$, $HL_1$ and $HH_1$ to the subbands resulting from the first level of the image decomposition, corresponding to horizontal, vertical and diagonal frequencies. The rest of image transform is computed by 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).

As we saw in the introduction of this paper, one of the main drawbacks in previous wavelet image encoders is their high complexity. Many times, that is mainly due to bit plane coding, which is performed along different iterations, using a threshold that focuses on a different bit plane in each iteration. This way, it is easy to achieve an embedded bit-stream with progressive coding, since more bit planes add more SNR resolution to the recovered image.

Although embedded bit-stream is a nice feature in an image coder, it is not always needed and other alternatives, like spatial scalability, may be more valuable according to the final purpose. In this section, we describe a very fast algorithm that is able to encode wavelet coefficients without performing one loop scan per bit plane. Instead of it, only one scan of the transform coefficients is needed. This algorithm was first presented in [5], but in this Section we simplify it (e.g., by adapting it to Huffman coding instead of adaptive arithmetic coding) in order to achieve faster execution time at the expense of compression efficiency.

In this algorithm, the quantization process is performed by two strategies: one coarser and another finer. The finer one consists in applying a scalar uniform quantization to the coefficients, and it is carried out along with the DWT, simply by varying the normalization factor in the lifting transform, and then rounding the coefficient. On the other hand, the coarser one is based on removing bit planes from the least significant part of the coefficients, and it is performed while our algorithm is applied. Related to this bit plane quantization, we define *rplanes* as the number of least significant bits to be removed. In addition, we consider that a coefficient $c_{i,j}$ is insignificant if it is 0 after removing *rplanes* bits, in other words, if $|c_{i,j}| < 2^{rplanes}$.

A tree structure (similar to that of [3]) is used not only to reduce data redundancy among subbands, but also as a simple and fast way of grouping coefficients. As a consequence, the total number of symbols needed to encode the image is reduced, decreasing the overall execution time. This structure is called lower tree, and it is a coefficient tree in which all its coefficients are lower than $2^{rplanes}$.

Our algorithm consists of three stages. In the first one, all the symbols needed to efficiently represent the transform image are calculated. During this stage, statistics can be collected in order to compute a Huffman table in a second stage. Finally, the last stage consists in coding the symbols computed during the first one by using Huffman coding.

Let us describe the symbol set employed in our proposal. First, we will describe the symbols corresponding to insignificant coefficients, and then to the significant ones. Note that we assume that $s_{i,j} \in S$ is a symbol used to represent a coefficient $c_{i,j} \in C$.

A *LOWER* symbol is used to represent a coefficient that is the root of a lower-tree. The rest of coefficients in a lower-tree are labeled as *LOWER_COMPONENT*, but they are never encoded because they are already represented by the root coefficient. On the other hand, if a coefficient is insignificant but it does not belong to a lower-tree because it has at least one significant descendant, it is an *ISOLATED_LOWER*.

For a significant coefficient, we use a symbol indicating the number of bits needed to represent that coefficient ($nbits_{i,j}$). We call it a *numeric symbol*. Thereby, for a significant coefficient, if its corresponding *numeric symbol* is Huffman coded, and its significant bits and sign are binary coded ("raw coded"), the quantized coefficient is fully represented. However, there is a special "*LOWER numeric symbol*" (represented as $nbits_{i,j}^{LOWER}$) that not only indicates the number of bits of a coefficient, but also the fact that all its descendants are labeled as *LOWER_COMPONENT*. 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.

Let us describe now the coding algorithm. In the first stage (symbol computation), all the wavelet subbands are scanned in 2×2 blocks of coefficients, from the first level to the $N^{th}$ (to be able to build the lower-trees from leaves to root). In the first level subband, if the four coefficients in each 2×2 block are insignificant (i.e., lower than $2^{rplanes}$), they are considered to be part of the same lower-tree, being labeled as *LOWER_COMPONENT*. Then, when scanning upper 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, the symbol for each coefficient is computed one by one. Each insignificant coefficient in the block is assigned a *LOWER* symbol if all its descendants are *LOWER_COMPONENT*, otherwise it is assigned an *ISOLATED_LOWER* symbol. On the other hand, for each significant coefficient, a *numeric symbol* is employed, being a *LOWER numeric symbol* ($nbits_{i,j}^{LOWER}$) if its direct descendants are *LOWER_COMPONENT*.

As an optimization in this first pass, in order to increase the appearance of 2×2 blocks of *LOWER_COMPONENT*, whenever the four coefficients have insignificant descendants, the threshold to compare these four coefficients is increased from $2^{rplanes}$ to $2^{rplanes+1}$ to extend an existing lower-tree more easily.

In the second stage, Huffman codes are built with the probability model for the source (the symbols computed in the first stage), once this probability model has been acquired during the first stage. The computed table containing the Huffman codes is output so that the decoder can use it to decode the encoded symbols.

Finally, in the third stage, subbands are encoded from the $LL_N$ subband to the first-level wavelet subbands. Observe that this is the order in which the decoder needs to know the symbols, so that lower-tree roots are decoded before its leaves. In addition, this order provides resolution scalability, because $LL_N$ is a low-resolution scaled version of the original image, and as more subbands are being received, the low-resolution image can be doubled in size. In each subband, for each 2×2 block, the symbols computed in the first stage are Huffman encoded using the codes computed in the second stage. Recall that no *LOWER_COMPONENT* is encoded, and that significant bits and sign are needed, and therefore binary encoded, for each significant coefficient.

Observe that since no adaptive coding and context-modeling is performed, the order in which coefficient blocks are scanned in each subband does not affect compression efficiency, and therefore, a typical raster scan order is followed because it avoids cache miss, being faster. In the original LTW, a scan in clusters was used in order to take advantage of spatial locality with an adaptive arithmetic encoder with two-contexts, increasing the PSNR but being slower. In addition, in order to speed up Huffman decoding, lookup tables are built in the decoder.

The proposed coding algorithm is formally described in the frame entitled Algorithm 1.

**function** HuffmanLTWCoding( $C$ )
*1) Symbol computation:*
Scan the subbands (scan $C$, from 1 to N, in 2×2 blocks)
**For each** block $B_n$
**if** $|c_{i,j}| < 2^{rplanes} \wedge$
$(\text{descendant}(c_{i,j}) = LOWER\_COMPONET \vee$
$\neg\text{descendant}(c_{i,j}))$ $\forall c_{i,j} \in B_n$
set $s_{i,j}$=*LOWER_COMPONENT* $\forall s_{i,j} \in B_n$
**else for each** $c_{i,j} \in B_n$
**if** $|c_{i,j}| < 2^{rplanes} \wedge$ descendant( $c_{i,j}$)=*LOWER_COMP.*
set $s_{i,j}$=*LOWER*
**else**
**if** $|c_{i,j}| < 2^{rplanes} \wedge$ descendant( $c_{i,j}$) $\neq$ *LOWER_COMP.*
set $s_{i,j}$=*ISOLATED_LOWER*
**else**
$nbits_{i,j} = \lceil \log_2(|c_{i,j}|) \rceil$
**if** descendant( $c_{i,j}$)=*LOWER_COMPONENT*
set $s_{i,j} = nbits_{i,j}^{LOWER}$
**else**
set $s_{i,j} = nbits_{i,j}$
*2) Huffman computation:*
Build Huffman codes with statistics from $S$
**output** Huffman Table
*3) Coefficient coding:*
Scan the subbands (scan $S$, from N to 1, in 2×2 blocks)
**For each** $s_{i,j}$ in a subband
**if** $s_{i,j} \neq$ *LOWER_COMPONENT*
**Huffman_output** $s_{i,j}$
**if** $s_{i,j} \neq LOWER \wedge s_{i,j} \neq ISOLATED\_LOWER$
**output** $\text{bit}_{nbits_{i,j}-1}(|c_{i,j}|)\ldots\text{bit}_{rplane+1}(|c_{i,j}|)$
**output** sign( $c_{i,j}$)

**end of fuction**

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

Algorithm 1: Proposed coding algorithm.

## 4. NUMERCIAL RESULTS

In Table 1, we compare the compression efficiency of the new proposed variation of LTW with the original one, with JPEG 2000, and with other fast wavelet encoders, namely PROGRESS [6] and SBHP [4]. Obviously, the original LTW is more efficient (from to 0.2 to 0.7 dB in PSNR, depending on the bit-rate), mainly due to the use of adaptivity, context-modeling and arithmetic coding. However, our new proposal is more efficient than PROGRESS (up to 0.5 dB at low bit-rates) and than SBHP in slightly detailed images, like Café (in low-frequency

images, like Woman, SBHP works slightly better, ranging from 0.05 to 0.2 dB).

However, the main advantage of this new proposal is its very fast execution time. In table 2, execution time is compared with the original LTW and PROGRESS[2] (excluding the DWT). For the coding process, the original LTW is faster than PROGRESS (up to 3 times) except in high bit-rates. If we compare the new fast LTW proposal with PROGRESS, this advantage is increased, being from 4 to 9 times faster, depending on the bit-rate. The decoding process is much faster than the coding process in PROGRESS (because it does not need to compute the highest coefficient in each sub-tree), and consequently only an improvement of about 20% is achieved with the proposed LTW at 1 bpp, while this improvement becomes smaller as the bit-rate is reduced. Although there are no execution time results (or reference software) available to compare our proposal with SBHP, the use of bitplane coding and a sorting algorithm in SBHP probably make it slower. In fact, our encoder, when coding Woman at a range from 1 bpp to 0.125 bpp, is from 18 to 38 times faster than JPEG 2000 reference software (in particular, Jasper, written in C), while in [4] authors state that SBHP coding was 4 times faster than JPEG 2000 VM. In addition, our algorithm only needs the amount of memory required to hold the image in memory. For more tests, our implementation is available at http://www.disca.upv.es/joliver/LTWhuff.

| codec\ bitrate | SBHP | PRO-GRESS | LTW Huffman | LTW Orig. | JPEG 2000 |
|---|---|---|---|---|---|
| Lena (512×512) | | | | | |
| 0.125 | n/a | 30.59 | 31.06 | 31.27 | 30.84 |
| 0.25 | n/a | 33.71 | 34.03 | 34.31 | 34.04 |
| 0.5 | n/a | 36.85 | 37.03 | 37.35 | 37.22 |
| 1 | n/a | 39.89 | 40.11 | 40.50 | 40.31 |
| Café (2560×2048) | | | | | |
| 0.125 | 20.49 | n/a | 20.56 | 20.76 | 20.74 |
| 0.25 | 22.64 | n/a | 22.90 | 23.24 | 23.12 |
| 0.5 | 26.01 | n/a | 26.31 | 26.85 | 26.80 |
| 1 | 31.08 | n/a | 31.30 | 32.03 | 32.04 |
| Woman (2560×2048) | | | | | |
| 0.125 | 27.09 | 26.89 | 27.23 | 27.52 | 27.33 |
| 0.25 | 29.59 | 29.40 | 29.70 | 30.16 | 29.98 |
| 0.5 | 33.11 | 33.02 | 33.15 | 33.82 | 33.63 |
| 1 | 37.98 | 37.75 | 37.76 | 38.53 | 38.43 |

Table 1: PSNR (dB) with different bit-rate and coders

[2] For the execution time comparison, similar processors have been employed. Results for PROGRESS were published in [6] with an Intel Xeon 2 Ghz Processor, and results for LTW and JPEG 2000 are obtained in this paper with an Intel PentiumM 1.6 Ghz Processor. On the other hand, all the implementations (including PROGRESS) are written in C language and compiled with MS Visual C++ 6.0 and the same speed optimization level.

## 5. CONCLUSIONS

We have presented a very fast version of the lower-tree wavelet encoder using Huffman coding and other strategies to reduce execution time. The loss of coding efficiency is compensated by the reduction of execution time. In fact, the encoder is less complex than some of the fastest wavelet encoders reported in the literature, being up to 9 times faster than PROGRESS (and much more symmetric than it), while the PSNR is from 0.3 to 0.5 dB higher at low bit-rates. In addition, in-place symbol computation is performed, and therefore, there is no memory overhead. As a conclusion, we think that our encoder can be considered one of the fastest wavelet-based image encoders, and therefore it is a good candidate for real-time interactive multimedia communications, allowing simple implementations both in hardware and software.

| codec\ bitrate | PRO-GRESS | LTW Huffman | LTW Orig. | PRO-GRESS | LTW Huffman | LTW Orig. |
|---|---|---|---|---|---|---|
| | CODING | | | DECODING | | |
| Lena (512×512) | | | | | | |
| 0.125 | 23.7 | 2.7 | 8.2 | 1.6 | 1.6 | 4.8 |
| 0.25 | 26.1 | 3.5 | 12.1 | 2.6 | 2.4 | 8.6 |
| 0.5 | 29.0 | 5.0 | 19.7 | 4.6 | 3.9 | 15.8 |
| 1 | 34.8 | 8.1 | 36.4 | 8.3 | 6.7 | 30.8 |
| Woman (2048×2048) | | | | | | |
| 0.125 | 378.4 | 51.3 | 149.5 | 24.1 | 26.1 | 83.4 |
| 0.25 | 404.3 | 68.8 | 217.2 | 41.9 | 40.5 | 147.3 |
| 0.5 | 450.1 | 100.2 | 337.3 | 74.7 | 63.2 | 266.6 |
| 1 | 528.4 | 140.0 | 568.7 | 128.4 | 101.5 | 484.2 |

Table 2: Execution time comparison of the coding process (excluding DWT) (time in million of CPU cycles)

## 6. REFERENCES


[1] ISO/IEC 15444-1, JPEG2000 image coding system, 2000.

[2] X. Wu, "Compression of Wavelet Transform Coefficients," *The Transform and Data Compression Handbook*, pp. 347-378, CRC Press, 2001.

[3] 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, 1996.

[4] C. Chrysafis, A. Said, A. Drukarev, A. Islam, W. A. Pearlman, "SBHP- A low complexity wavelet coder," in *Proc. IEEE Int. Conference on Acoustics, Speech, and Signal* Processing, 2000.

[5] J. Oliver, M. P. Malumbres, "Fast and efficient spatial scalable image compression using wavelet lower trees," in *Proc. IEEE Data Compression Conference*, Snowbird, UT, March 2003.

[6] Yushin Cho, W. A. Pearlman, A. Said, "Low complexity resolution progressive image coding algorithm: PROGRES (Progressive Resolution Decompression)", in *Proc. IEEE International Conference on Image Processing*, September 2005.