Signal Processing: *Image Communication* 23 (2008) 637–648

Contents lists available at ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

M-LTW: A fast and efficient intra video codec

Otoniel M. López [a,*], Miguel O. Martínez-Rach [a], Pablo Piñol [a], Manuel Perez Malumbres [a], José Oliver [b]

[a] *Universidad Miguel Hernández, Avda. Universidad s/n, 03202 Elche, Spain*
[b] *Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain*

ARTICLE INFO

Article history:
Received 30 November 2007
Received in revised form
26 June 2008
Accepted 4 July 2008

Keywords:
Intra-video coding
DWT integer transform
Rate control
Performance evaluation

ABSTRACT

Intra-video coding is a common way to process video material for applications like professional video editing systems, digital cinema, video surveillance applications, multi-spectral satellite imaging, HQ video delivery, etc. Most practical intra-coding systems employ JPEG encoders due to their simplicity, low coding delay and low memory requirements. JPEG2000 is the main candidate to replace JPEG in this kind of application due to its excellent rate/distortion (R/D) performance and high coding flexibility. However, its complexity and computational resource requirements for proper operation could be a limitation for certain applications. In this work, we propose an intra-video codec, M-LTW, which is able to obtain very good R/D performance results, as good as JPEG2000 or H.264 INTRA, with faster processing and lower memory usage.

1. Introduction

A wide variety of video compression schemes have been reported in the literature. Most of them are based on the DCT transform and motion estimation and compensation techniques. Nevertheless, a lot of research interest has been focused on developing still image and video wavelet coders due to the excellent properties of wavelet transform. Most wavelet-based video encoding proposals are strongly based on inter-coding approaches, which require high-complexity encoder designs as a counterpart to the excellent R/D (rate/distortion) performance benefits. On the other hand, some applications like professional video editing, digital cinema, video surveillance applications, multi-spectral satellite imaging, HQ video delivery, etc. would rather use an intra-coding system that is able to reconstruct a specific frame of a video sequence as fast as possible and with high visual quality.

The strength of an intra-video coding system relies on the ability to efficiently exploit the spatial redundancies of each video sequence frame avoiding complexity in the design of the encoding/decoding engines. There are several still image codecs that get very good R/D results. Unfortunately, most of them propose complex algorithms to achieve the pursued R/D performance. As a consequence of the higher computational complexity demanded by these coders, their software (even hardware) implementations require powerful processors with enough computational resources to cope with the algorithm requirements. For example, the JPEG2000 [10] standard uses a large number of contexts and an iterative time-consuming optimization algorithm (called PCRD) to improve coding efficiency, increasing the complexity of the encoding engine. Something similar happens with H.264/AVC [11] INTRA coding, where a powerful spatial prediction scheme with context modeling and R/D optimization is employed in order to efficiently exploit spatial redundancies.

In this paper, we propose a new lightweight and efficient intra-video coder, M-LTW (motion lower-tree wavelet), based on the LTW algorithm [15]. The main

* Corresponding author. Tel.: +34 96 665 8392.
E-mail addresses: otoniel@umh.es (O.M. López), mmrach@umh.es (M.O. Martínez-Rach), pablop@umh.es (P. Piñol), mels@umh.es (M. Perez Malumbres), joliver@disca.upv.es (J. Oliver).

0923-5965/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2008.07.001

contribution of LTW is the way that it builds the significance map when coding each video frame. As other tree-based wavelet coders, it is based on the construction and efficient coding of wavelet coefficient trees. Nevertheless, it does not use an iterative loop in order to determine the significant coefficients and to assign bits to them. It builds the significance map in only one step by using two symbols for pruning tree branches, and also codes the significant coefficients in one step. Other encoders (like the one proposed in [20]) 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 [16,5]) results in a 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 software implementations.

Another very fast non-embedded image encoder has been proposed in [4]. This encoder is called PROGRES. It follows the same ideas of [15], avoiding bit-plane coding and using coefficient trees to encode wavelet coefficients in only one pass. In this encoder, all 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 subband level 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.

Recently, the BCWT image encoder [9] was proposed. It offers high coding speed, low memory usage and good R/D performance. The basis of BCWT algorithm is building a map of the maximum quantization levels of descendants (MQD map), which is a variation of Shapiro's zerotree map [17]. MQD map calculation and coefficient encoding are all carefully integrated avoiding memory waste and reducing computational cost.

1.1. Contributions and paper organization

In this paper we propose a new lightweight intra-video codec, which is competitive with other state-of-the-art intra-video coders, requiring low memory resources and showing very low coding/decoding delay.

Since the LTW encoding engine is non-SNR-embedding, we have proposed a low complexity rate control tool to encode the original video sequence at a user-defined target bitrate, in order to increase the flexibility of M-LTW video encoder and allowing M-LTW to work with rate-adaptive applications. Also, we have changed the whole codec to work with fixed-point arithmetic. Consequently, the discrete wavelet transform (DWT) implemented may use the original lifting floating-point approach or an equivalent DWT integer lifting version which will speed up the DWT transform step. As a secondary benefit, the required memory space is halved (16-bit integer data types instead of 32-bit floats).

In Section 2 the LTW algorithm is described in depth, making special emphasis in the LTW Integer version. In Section 3, we describe a rate control algorithm for non-embedded wavelet image encoders based on the LTW coding engine. In Section 3.2, an extension for video coding of the proposed rate control and its evaluation is presented. In Section 4 we show some evaluation results using R/D, complexity and memory requirements as performance metrics. For further evaluation, we have compared the performance of M-LTW with a fully optimized version of JPEG2000 (Kakadu). Finally, in Section 5 some conclusions are drawn.

2. Texture coding: fast and efficient LTW proposals

For the most part, digital images are represented with a set of pixel values, P. The LTW encoder 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 [2], therefore an element $C_{i,j} \in C$, is called transform coefficient. In a wavelet transform, we denote the subbands resulting from the first level of the image decomposition as LH_1, HL_1 and HH_1, corresponding to horizontal, vertical and diagonal frequencies. The rest of the image transform is performed 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).

One of the main drawbacks in previous wavelet image encoders is their high complexity. Many times, this is mainly due to bit-plane processing, which is performed across different iterations, using a threshold that focuses on a different bit plane in each iteration. In this way, it is easy to achieve an embedded bit-stream with progressive coding, since the more bit planes are added the more SNR resolution is obtained in the recovered image.

Although embedded bit-stream with SNR scalability 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 the LTW algorithm, which 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. Furthermore, in this section we present a very fast integer version of LTW.

In LTW, the quantization process is performed by means of two strategies: one coarser and another finer. The finer one consists of applying a scalar uniform quantization, Q, to wavelet coefficients. The coarser one is based on removing the least significant bit planes, rplanes, from wavelet coefficients.

A tree structure (similar to that of [16]) 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 two stages. In the first one, the significance map is built after quantizing the wavelet coefficients (by means of both Q and rplanes parameters). In Fig. 1 (right), we show the significance map built from wavelet decomposition shown in Fig. 1 (left). The symbol set employed in our proposal is the following one: a *LOWER* symbol (L) represents 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. If a coefficient is insignificant (i.e., lower than 2^{rplanes}) but it does not belong to a lower-tree because it has at least one significant descendant, it is labeled as an *ISOLATED_LOWER* symbol (I). For a significant coefficient, we simply use a symbol indicating the number of bits needed to represent it (i.e. 4). For a significant coefficient that is root of a lower-tree we use a special symbol indicating the number of bits needed to represent it with an L superscript (i.e. 4^L).

Let us describe the coding algorithm. In the first stage (symbol computation), all wavelet subbands are scanned in 2×2 blocks of coefficients, from the first decomposition 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, 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 labeled as *LOWER_COMPONENT*, increasing the lower-tree size.

However, when at least one coefficient in the block is significant, the lower-tree cannot continue growing. In that case, a 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 symbol indicating the number of bits needed to represent that coefficient is employed. However, if all descendants of a significant coefficient are insignificant (*LOWER_COMPONENT*), we use a special symbol indicating the number of bits needed to represent it and a superscript L (4^L in Fig. 1 (right)).

Finally, in the second stage, subbands are encoded from the LL_N subband to the first-level wavelet subbands, as shown in Fig. 2. 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 entropy coded by means of an arithmetic encoder. Recall that no *LOWER_COMPONENT* is encoded. In addition, significant bits and sign are needed for each significant coefficient and therefore binary encoded.

For more details about LTW, and a formal description of the algorithm, the reader is referred to [15].

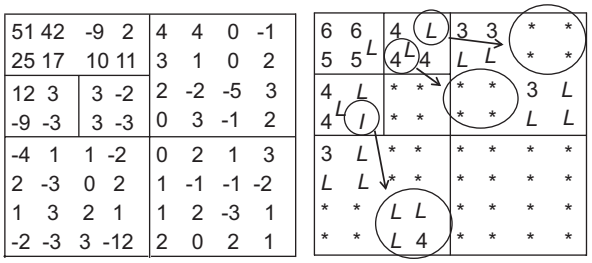

Fig. 1. Left: 2-level wavelet transform of an 8×8 example image, right: map symbols.

2.1. LTW_Int: lower tree wavelet integer

To carry out a fast integer version of LTW, we have developed the DWT with an integer-to-integer lifting scheme based on [3,7]. We have implemented the normalization factor of the lifting scheme (K) as an approximation to integer operations (multiply and shift). In this manner we avoid three extra lifting steps at the expense of making the DWT not reversible. Since we are interested in lossy compression, the fact of performing that approximation does not introduce a meaningful error, being the difference respect to the regular lifting scheme negligible.

Concerning the LTW encoder engine, we have converted all float operations to integer ones.

Relating to the quantization process, this is similar to the one used by LTW described previously. The main difference lies in the scalar uniform quantization process, which is performed using only fixed-point arithmetic operations.

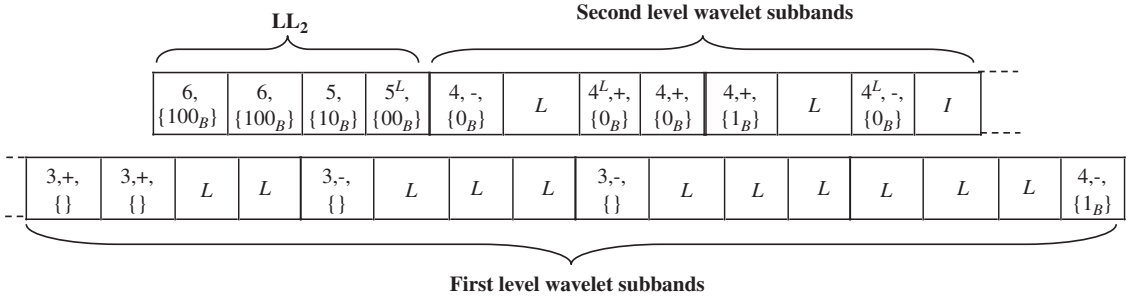

Fig. 2. Example image encoded using LTW.

Before adopting this approach, we tried other integer-to-integer DWT filters so as to obtain a fast version of DWT with similar R/D values. From the study carried out by Adams and Kossentini in [1], we decided to implement 13/7T and 9/7M filters because these filters only use two lifting steps instead of the four lifting states required by 9/7F filter. After evaluating the filters we concluded that the use of other filters lead us to a R/D loss of approximately 1 dB with respect to 9/7F filter, as shown in Fig. 3. However, we do not use the original 9/7F filter but an approximation (9/7F_int_Aprox.), which replaces the three extra lifting steps required for the normalization factor (K) by an integer approximation that truncates the wavelet coefficients to integers and thus introducing a small quantization noise. Contrary to what could be thought, it does not introduce any R/D loss, even showing a slightly better behavior than the original one at low compression rates. This behavior is similar in all tested images and appears only at low to very low compression rates (rplanes $= 2$).

This curious effect is due to the asymmetric error produced by the proposed integer approximation of the normalization factor. In the DWT, the introduced error is slightly greater in low frequency subbands than in high frequency subbands (error propagation in successive decomposition levels). However, this error has higher influence over the high frequency subbands producing a bit plane change in several significant coefficients (notice that at low compression rates there are many small significant coefficients in those subbands), and as a consequence fewer bits are needed to represent them. On the other hand, the inverse discrete wavelet transform (IDWT) introduces a greater error over high frequency subbands. So, the previously DWT introduced error is further compensated, obtaining a slightly better behavior than in the full reversible DWT version. This behavior disappears at higher compression rates (rplanes > 2) when the encoder quantization noise is greater than the error introduced by the DWT.

In order to determine the most appropriate DWT kernel, we have implemented three versions of the LTW encoder. The first one implements the DWT 9/7F filter with a traditional convolution, the second one implements a lifting scheme of DWT with floating-point arithmetic operations and the third one implements an integer-to-integer lifting scheme of DWT with an integer approximation of the normalization factor (K). So as to measure the R/D loss due to fixed-point arithmetic operations, in Figs. 4 and 5 we compare the R/D performance for both floating-point and fixed-point implementations. As shown in Fig. 4, for Barbara test image there is almost no difference in R/D between all proposals (note that both floating-point DWT implementations obtain the same PSNR because they use the same filter D(9/7F), so only convolution implementation is represented in Figs. 4 and 5), but if we focus in Fig. 5, for Lena test image there is a loss of approximately 0.5 dB at a compression rate of 0.5 bpp. These results are in concordance with the ones presented by Grangetto in [8].

With regard to execution time, as we can see in Fig. 6, both lifting scheme DWT implementations are faster than traditional convolution. In this figure, the execution time of the floating-point implementations also includes a cast from float to integer, because the rate control algorithm presented in the next section operates only with fixed-point values. This fact causes that differences in execution time between lifting scheme implementation and traditional convolution are not as significant as the results presented by Grangetto in [8]. The integer-to-integer lifting scheme implementation of DWT is the fastest one, being a 50% faster than floating-point arithmetic

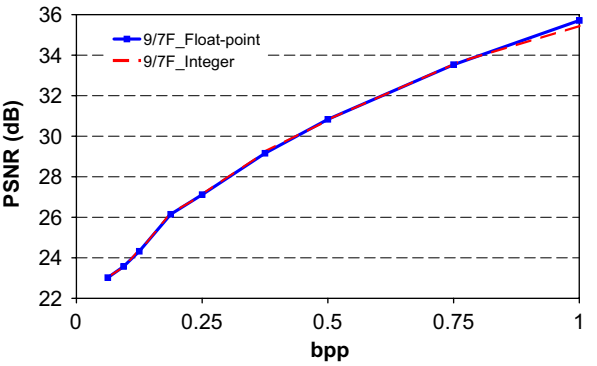

Fig. 4. R/D evaluation for different DWT proposals for Barbara (512 × 512) test image.

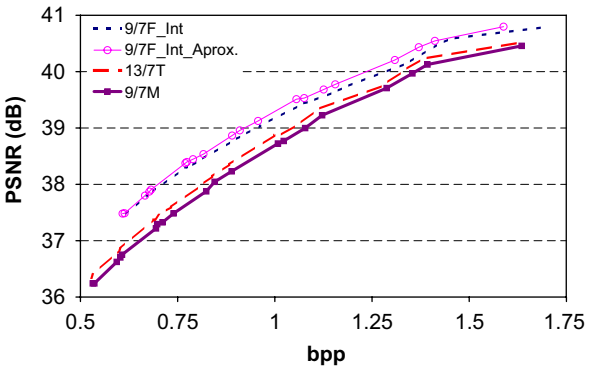

Fig. 3. R/D evolution using different filters for Lena (512 × 512).

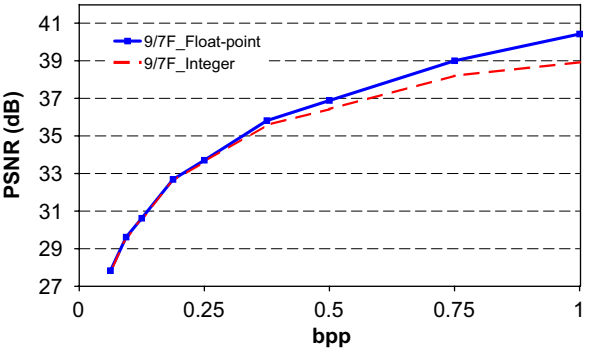

Fig. 5. R/D evaluation for different DWT proposals for Lena (512 × 512) test image.

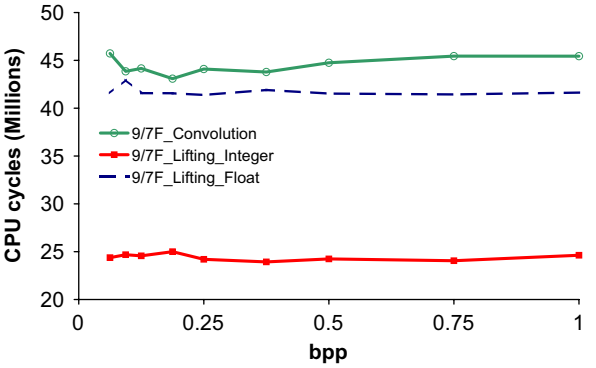

Fig. 6. Execution time comparison between different proposals of DWT for Barbara test image.

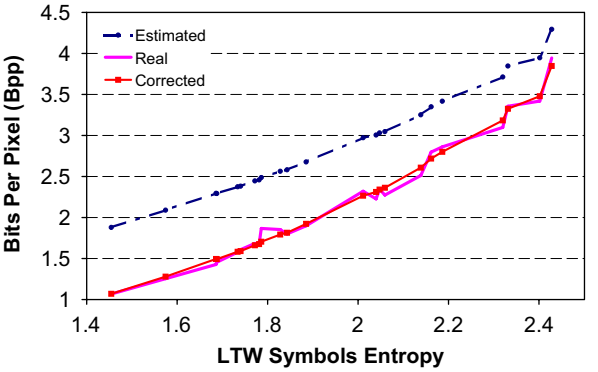

Fig. 7. Estimated vs. real bits per pixel for all Kodak set images for rplanes = 2.

lifting scheme implementation due to the cast to integer differences mentioned previously.

As a consequence of this study, we have finally planned to carry out two versions of LTW, one based on floating-point arithmetic operations that implements DWT with a lifting scheme and another based on fixed-point arithmetic operations that implements DWT with an integer-to-integer lifting scheme approximation.

3. LTW rate control support

Since M-LTW coding engine is based on a non-embedded wavelet image encoder and also it is based on DWT transform, we have chosen a lightweight rate-control algorithm for non-embedded encoders presented in [13]. This algorithm will predict the proper quantization values that lead to a final bitrate close to the target one.

3.1. Rate control based on a trivial coding model

We decided to study how the LTW encoder works in order to define a simplified model of the encoding engine. This model will lead us to an initial and fast estimation of the resulting bitrate for different values of the rplanes parameter (from 2 to 7). In this model, for each specific value of rplanes, the probability distribution of significant and non-significant symbols (when a wavelet coefficient is lower than 2^{rplanes}) is calculated. From the obtained symbol probability distribution, and using the first order entropy, we obtain the bitrate estimation of the arithmetic encoder. This estimation is a lower bound of the real bitrate. This way, the estimation of the bitrate produced by the arithmetic encoder and the number of bits required to store the signs and significant bits (which are binary coded) form the bitrate estimation (E_{bpp}). The resulting estimation gives a biased measure of the real bitrate as shown in Fig. 7. The error found is due to the way we perform the probability distribution of LTW symbols where we do not distinguish the probability distribution of the LTW non-significant symbols *LOWER* and *ISOLATED_LOWER*. These symbols are not known until the encoding step, so we decided to perform a simple symbol

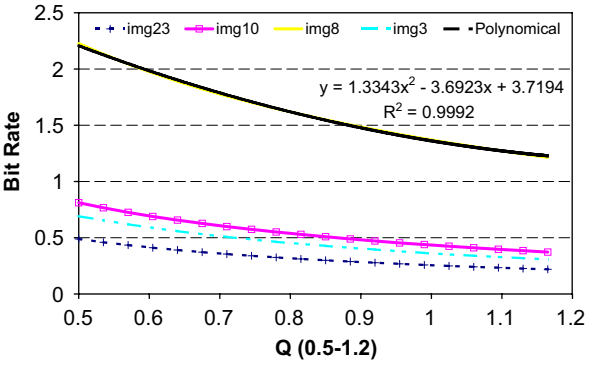

Fig. 8. Bitrate progression of five images from Kodak set from 3 to 4 rplanes.

probability estimation in order to get a fast bitrate prediction. We reduce the error through a scaling function obtained from the Kodak image set [6]. This function is based on the LTW symbol entropy of the image as shown in Fig. 7.

After that, the target bitrate, T_{bpp}, will determine the proper value of rplanes ($E_{\text{bpp}}(\text{rplanes}) > T_{\text{bpp}} > E_{\text{bpp}}(\text{rplanes}+1)$). The bitrate progression from the current rplane to the next one is governed by the Q parameter which follows a second order polynomial curve. All curves of the test images in the Kodak set have a very similar curve minimum value ($K_{\min}$) (as shown in Fig. 8). In this figure, polynomial line (simple-black) and its quadratic equation has been obtained by means of curve fitting (R^2 is the coefficient of determination and it indicates the fitting goodness). Since we know three points of that curve $E_{\text{bpp}}(\text{rplanes})$, $E_{\text{bpp}}(\text{rplanes}+1)$ and the curve minimum ($K_{\min}$), we can build the corresponding expression that will supply the estimated value of Q for a given target bitrate.

In the algorithm in Fig. 9, we show the steps of the rate control algorithm. In step E1, the symbol probability distribution is computed to find the arithmetic encoder bitrate estimation. In step E2, we correct the estimation error by means of a scaling function based on LTW symbol

Input: Wavelet Coefficients ($C_{i,j}$), Target bitrate (T_{bpp}), Curve minimum (K_{min})
Output: Q, $rplanes$
(E1) for each rp ($rplanes$) in [2..7]
for each $C_{i,j}$ coefficient
$nbits_{i,j} = \lceil \log(|C_{i,j}|) \rceil$
if $nbits_{i,j} > rp$
$Symbol_{(nbits_{i,j} - rp)} += 1$
$Bits_{total} += nbits_{i,j} - rp - 1$
else
$Symbol_{non-significant} += 1$
Calculate the Symbols Entropy, S_e
$E_{bpp} = (Bits_{total}/size\ of\ (image)) + S_e$
(E2) for each rp in [2..7]
Apply_Correction_Factor(E_{bpp});
(E3) Determine rp
$E_{bpp}(rp) > T_{bpp} > E_{bpp}(rp+1)$
(E4) Determine quantization parameter (Q)
Obtain A,B,C using $E_{bpp}(rp)$, $E_{bpp}(rp+1)$ and K_{min} for $T_{bpp} = A.Q^2 + B.Q + C$ and solve equation

Fig. 9. Model-based algorithm.

entropy. And finally in steps E3 and E4 we obtain the rplane value (coarser quantization parameter) and the Q value (finer quantization parameter), respectively.

In order to reduce the implementation complexity, we have partially merged the algorithm in the DWT process. So, every time a new wavelet coefficient is computed we perform the E1 step. Just after finishing the DWT, the rest of algorithm steps are executed to obtain the proper quantization value for LTW encoding engine.

3.2. Extension of the proposed rate control to intra-video coding

Since the main goal of this work is to develop a fast intra-video encoder, in order to perform the rate control in the overall video sequence, we have extended the rate control algorithm explained in previous section (Fig. 9) using a very simple approach. Firstly, we apply the rate control algorithm to the first frame to estimate the values of rplanes and Q quantization parameters that fit the frame bitrate budget. After coding the first frame, we compute the estimation error, so we will try to compensate it when coding the following frames. We will do that keeping the same value of rplanes and estimating the appropriate value for Q based on the observed error. During the video sequence coding, when the observed error reaches a threshold (SC_{th}), the algorithm launches the initial estimation algorithm to re-estimate more suitable rplanes and Q parameters so as to converge to the desired bitrate as fast as possible; then the accumulated error will be corrected gradually so as to avoid great R/D alterations. The threshold fixed on a 20% of the target bitrate has been obtained from the model-based algorithm inherent estimation error that tends to be below 10% as concluded in [12].

In Fig. 10 we show the accuracy of the proposed algorithm for Container sequence with a CIF size and we compare it with respect to the embedded encoder SPIHT and JPEG2000. Both SPIHT and JPEG2000 are embedded encoders, so they always obtain the exact target bitrate. With respect to the proposed rate control implemented on M-LTW and M-LTW_Integer, its accuracy was always better than 98.5% being the worst case at very low target bitrates where the average relative error is approximately of 0.4%. Although not shown here, the proposed rate control has a similar behavior with other video sequences and with other frame sizes.

In Fig. 11 we can see the effect of the rate-control algorithm over the CIF Coastguard sequence. As shown, during the most part of the sequence, the bitrate remains constant, but when significant alterations in the scene like camera movements, appearing objects or illumination changes occur, the algorithm produces a bitrate that reaches the fixed threshold. Then, the algorithm detects this situation and converges quickly to the desired bitrate. If we focus on frames between 65 and 75 in Fig. 11 we could see this behavior. This figure also shows the M-LTW encoder behavior when no rate control is applied.

Note that with the proposed rate-control algorithm, we choose the quantization parameters of the last encoded frame as a reference to encode the actual frame and reduce the accumulated rate-control error. This works fine while consecutive frames have similar contents (general case). However, when there are dramatic content changes with respect the previous frame (frames 65–75 at Fig. 11), large bit rate oscillations are produced because our

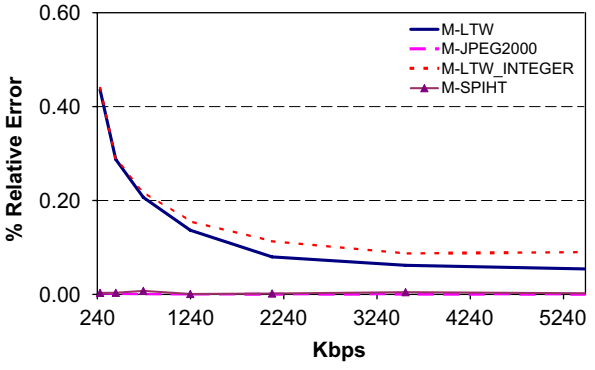

Fig. 10. Rate-control accuracy for Container CIF sequence.

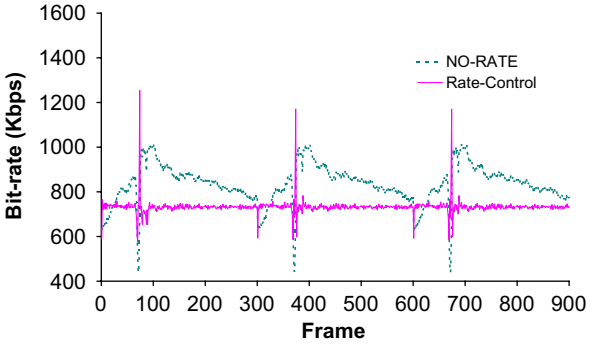

Fig. 11. Rate-control progression for three concatenated Coastguard sequences (CIF).

proposal does not know the complexity or similarity of the next frame.

In order to reduce the rate oscillations, several approaches may be followed: (1) apply the algorithm presented in Fig. 9 to all video frames, so rate fluctuations are confined to the rate-control algorithm precision and as consequence the encoder becomes slower (45% complexity increment), (2) obtain the mean absolute difference (MAD) of the lowest frequency subband (LL_N) between the current frame and the previously encoded one so as to detect scene changes. As Fig. 12 shows the use of MAD works fine and large rate oscillations are avoided at the expense of a 1.5% of complexity increment.

The video rate-control algorithm (Fig. 13) will lead as follows:

- Firstly, we obtain rplanes and Q parameters for the first frame by means of model-based algorithm (see Fig. 9).
- Secondly, we encode and evaluate the estimation error (P_{Err}).
- If the previously evaluated error (P_{Err}) is greater than the threshold input parameter (SC_{th}), then we force a new rplanes and Q estimation. Then, in the following frames we will gradually correct the bitrate error (Adjust%(P_{Err})).
- On the other hand, if the previously evaluated error (P_{Err}) is lower than the threshold input parameter (SC_{th}), we correct the bitrate error estimating only a new Q value and fixing the rplanes parameter to the one obtained in previous encoded frames.

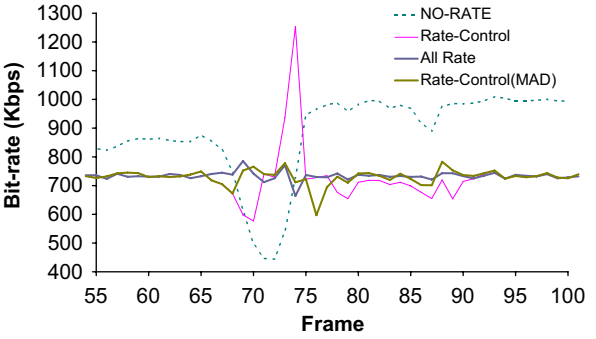

Fig. 12. Rate-control proposals progression for Coastguard sequences (focused on frames 65–75) (CIF).

Input: Video Sequence Frames, Target bitrate (T_{bpp}) and Threshold(SC_{th})
(E1) Obtain rplanes and Q using Algorithm 9 for the first frame.
P_{Err} = Encode_And_Evaluate_Error
Initialize SC_{Change}= **false**
(E2) For all remaining frames
if $P_{Err} > SC_{th}$
Obtain **new** rplanes and Q using Algorithm 9.
SC_{Change}= true
else
if SC_{Change}== true
$NewT_{bpp} = T_{bpp}$+Adjust%(P_{Err})
else
$NewT_{bpp} = T_{bpp} + P_{Err}$
P_{Err}= Encode_And_Evaluate_Error

Fig. 13. Video rate-control algorithm.

4. Performance evaluation

In addition to R/D performance we will also employ other performance metrics like coding delay and memory consumption. All the evaluated encoders have been tested on an Intel PentiumM Dual Core 3.0 GHz with 1 GB RAM memory. We have selected H.264 (Baseline, JM10.2) working in intra-mode, M-JPEG2000 (Jasper 1.701.0), M-SPIHT (Spiht 8.01), M-LTW and M-LTW_Int (integer version of M-LTW), since their source code is available for testing. The correspondent binaries were obtained by means of Microsoft Visual C++ (2005 version) compiler with the same project options and under the above-mentioned machine.

The test video sequences used in the evaluation are: Foreman (QCIF and CIF) 300 frames, Hall (QCIF and CIF) 300 frames, Container (QCIF and CIF) 300 frames, News (QCIF and CIF) 300 frames, Mobile (ITU 576p30) 40 frames and Station2 (HD 1024p25) 312 frames.

4.1. Objective/subjective quality evaluation

Table 1 shows the R/D evaluation of all proposed encoders. Although H.264 obtains better results for sequence sizes smaller than CIF at several compression rates, it is for ITU and HD sizes where encoders based on DWT can exploit optimal DWT decompositions obtaining better results (up to 2 dB with respect to H.264 in Station2 HD at high compression rates). The M-LTW_Int encoder produces slightly lower PSNR results than H.264. The lower performance of integer version is mainly due to the arithmetic precision loss, which is more noticeable at lower compression rates.

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

Codec/bitrate (kb/frame)	H.264	M-JPEG 2000	M-SPIHT	M-LTW	M-LTW_Int
Foreman (QCIF, 30 Hz)					
2.36	22.86	21.10	**24.16**	23.03	23.01
7.40	**28.72**	28.21	28.69	28.69	28.58
20.49	**35.36**	34.68	34.59	34.99	34.40
33.73	39.24	38.89	38.47	**39.37**	37.62
News (CIF, 30 Hz)					
9.27	24.43	25.21	25.54	**25.57**	25.51
14.91	27.37	27.34	**27.76**	27.73	27.63
36.76	**33.97**	33.11	33.01	33.37	33.01
89.91	**41.14**	40.63	40.08	41.00	38.91
Mobile (ITU, 30 Hz)					
38.08	27.04	28.48	28.53	**28.59**	28.48
119.93	32.29	32.41	32.36	**32.57**	32.26
213.36	35.29	35.09	35.05	**35.40**	34.75
386.23	38.59	38.43	38.29	**38.87**	37.21
Station2 (HD, 25 Hz)					
93.92	30.49	32.37	32.29	**32.45**	32.19
180.00	32.58	34.38	34.25	**34.49**	34.06
604.64	37.55	38.67	38.39	**39.02**	37.73
1117.53	40.37	40.78	40.44	**41.38**	39.08

We have also compared all proposed encoders using the VIF distortion metric [18]. We have chosen this metric because it is one of the best 'full reference' metrics as concluded in [14]. In Figs. 14 and 15 we can observe the behavior of evaluated encoders for Foreman (CIF) and Mobile (ITU) sequences. As it can be seen, all encoders based on DWT have a similar performance at high compression rates with a lower DMOSp (predicted differential mean opinion score value) (better quality) than H.264. Only at lower compression rates H.264 outperforms M-LTW_Int, although at these DMOSp values, differences on DMOSp lower than 3 are not visually perceptible as concluded in [14]. The rest of coders show very similar results under VIF quality metric, so they can be considered equivalent in terms of R/D performance.

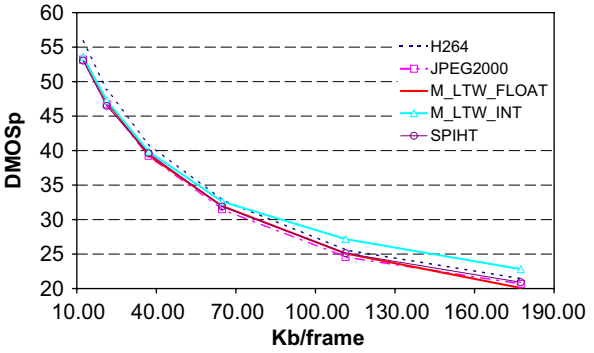

Fig. 14. R/D evaluation with VIF metric on DMOSp space for Foreman (CIF) sequence.

In Fig. 16 we present a subjective evaluation between M-LTW and M-LTW_Int in order to determine the R/D loss in the fixed-point version. Although differences on PSNR between both M-LTW and M-LTW_Int encoders are approximately of 0.4 dB, it is difficult to determine which one has better subjective quality at low compression rates. On the other hand, at high compression rates differences of 1 dB on PSNR are visually perceptible as shown in Fig. 17. All wavelet-based encoders show a similar behavior, but if we focus on the calendar (number 15), we could asses that M-LTW (c) and M-LTW_Int (d) have better subjective quality than JPEG2000 (a) and SPIHT (b). In spite of the fact that SPIHT shows a better PSNR value, M-LTW_Int is visually slightly better at this compression rate.

4.2. Execution time and memory consumption comparison

In Table 2 we show the coding delay for all encoders under evaluation. As expected, H.264 is the slowest encoder and M-LTW is one of the fastest. All M-LTW versions are faster than M-JPEG2000, especially the fixed-point version that performs the encoding process six

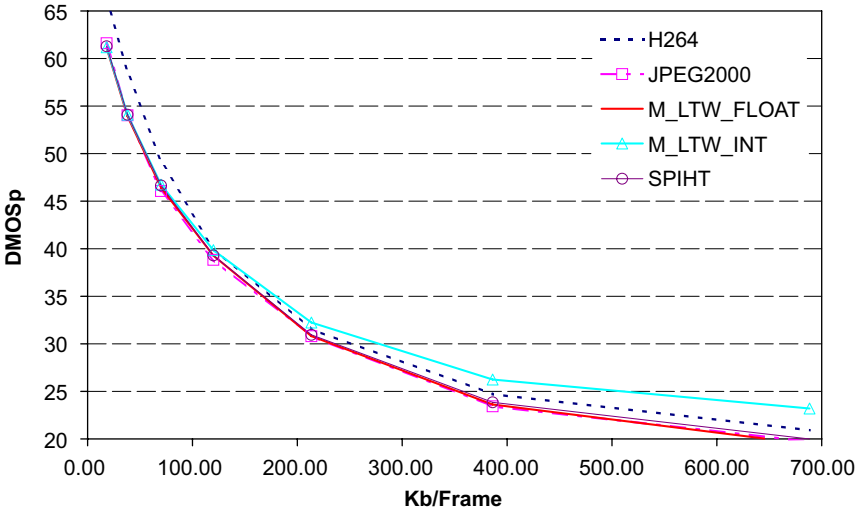

Fig. 15. R/D evaluation with VIF metric on DMOSp space for Mobile (ITU) sequence.

Fig. 16. Subjective comparison between (a) M-LTW and (b) M-LTW_Int for Foreman (QCIF) at 20.49 kb/frame, frame # 33. (a) 34.66db, (b) 34.24db and (c) original.

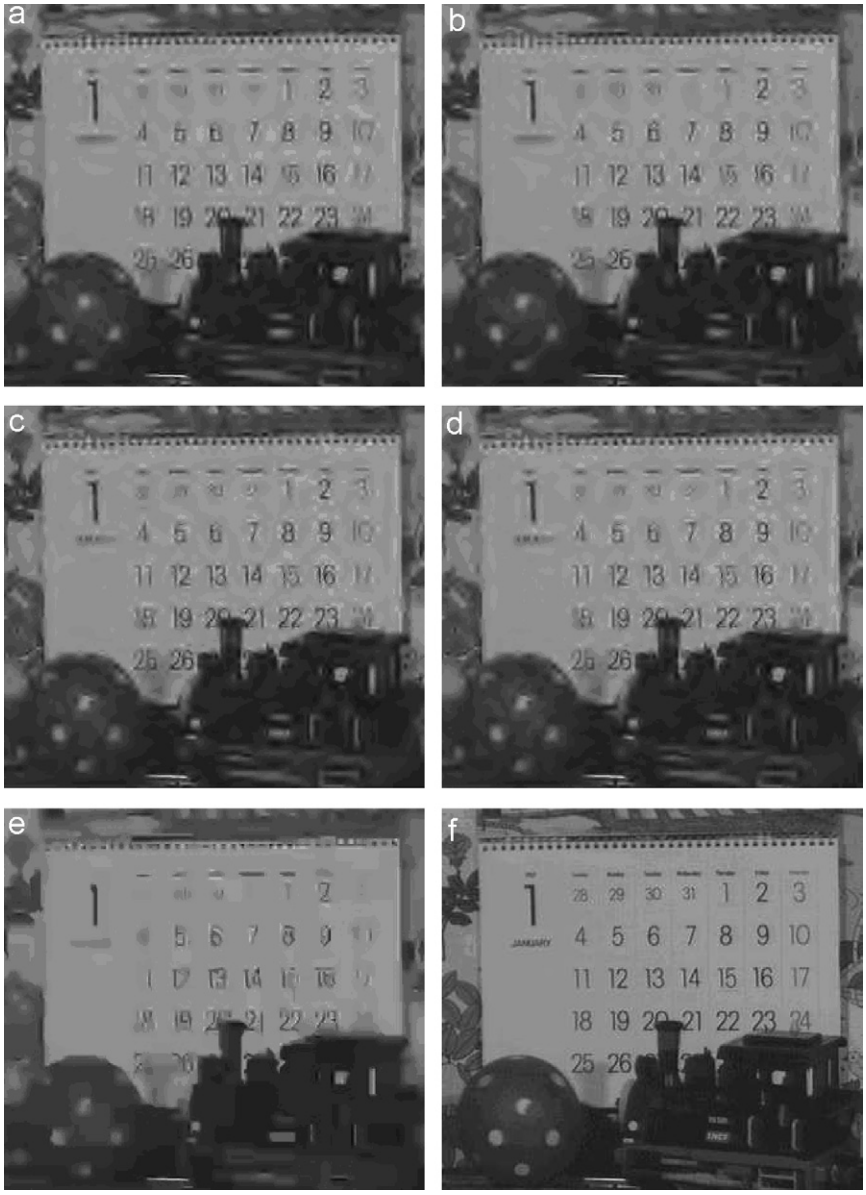

Fig. 17. Subjective comparison between (a) JPEG2000, (b) SPIHT, (c) M-LTW, (d) M-LTW_Int, (e) H.264 and (f) Original for Mobile (ITU) at 38.08 kb/frame, frame # 20.

times faster on average than M-JPEG2000. On the other hand, M-SPIHT is faster than M-LTW only at HD video format.

Fig. 18 shows the maximum frame rate for all evaluated encoders at different sizes for an average PSNR video quality of 30 dB. Integer version of M-LTW is one of the fastest encoders and it can encode an ITU size sequence in real time. For HD images, M-LTW is slower than M-SPIHT. This behavior is due to the cache page miss fail of the lifting DWT implementation where a lazy transform is carried out for both rows and columns. In the lazy transform, the input samples are split into two data sets, one with the even samples of a row or column and the other one with the odd ones. This causes a significative cache page miss fail increase, being more noticeable for columns, as the frame size becomes larger. In order to measure the impact of the cache page miss fail, in Fig. 19 we evaluate the performance of M-LTW implemented with both lifting and convolution DWT in a processor with an L2 cache size of 1 MB. As it can be seen, lifting DWT is 16.6% slower than convolution DWT for HD format and 10% slower for ITU format. With the increase of the L2 cache size, these differences are significatively reduced, being the difference for HD format of 9% as shown in Fig. 20.

In Table 3 the memory requirements of different encoders under test are shown. M-LTW needs only the amount of memory to store the source image (in-line processing) and an extra of 100 kb basically used to store the histogram of significant symbols required by the

Table 2
Execution time comparison of the coding process including DWT (time in seconds)

Codec/bitrate (kb/frame)	H.264	M-JPEG 2000	M-SPIHT	M-LTW	M-LTW_Int
CODING Hall (QCIF, 30 Hz)					
2.70	121.92	4.04	1.62	0.86	**0.51**
7.77	137.18	4.39	2.01	1.07	**0.71**
19.54	165.67	4.55	2.70	1.57	**1.10**
29.50	184.67	4.87	3.22	1.97	**1.41**
CODING News (CIF, 30 Hz)					
14.91	531.40	15.63	3.77	3.96	**2.62**
23.62	559.45	15.20	4.33	4.26	**2.81**
57.73	650.47	15.54	6.67	5.98	**3.94**
89.91	720.44	16.43	8.23	7.17	**4.95**
CODING Mobile (ITU, 30 Hz)					
38.08	233.27	8.99	**1.06**	1.83	1.25
119.94	266.94	7.88	1.83	2.38	**1.68**
213.37	297.11	8.02	2.63	2.93	**2.13**
386.23	351.41	8.29	4.24	3.97	**2.96**
CODING Station2 (HD, 25 Hz)					
93.93	11840.89	326.05	**34.13**	75.78	53.86
180.01	12106.46	327.18	**41.12**	79.24	56.59
604.64	13573.54	326.04	**73.75**	104.67	76.13
1117.53	15067.72	330.81	113.66	129.25	**93.13**

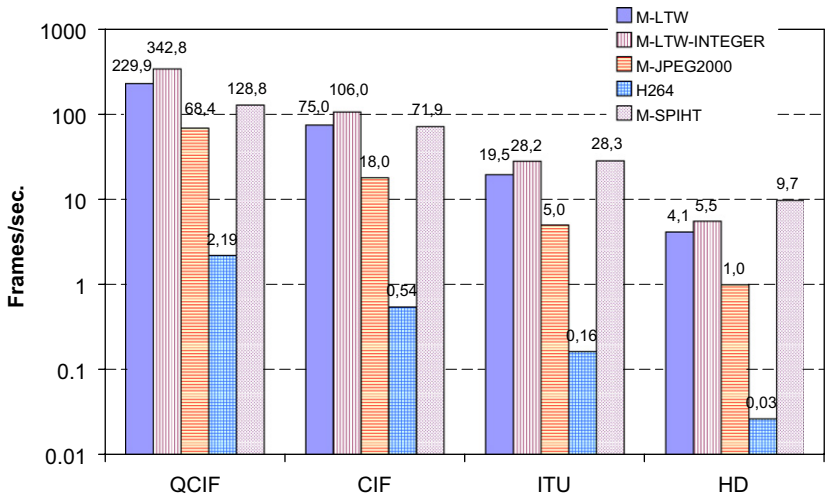

Fig. 18. Maximum frames/s for an average R/D of 30 dB.

rate-control algorithm, variables and structures needed to accomplish the coding process. M-JPEG2000 requires twice the memory amount of M-LTW, and H.264 needs six times the memory amount of M-LTW for QCIF format and eight times for CIF format. M-SPIHT uses 1.7 times the memory amount of M-LTW. Note that M-LTW_Int could be implemented using 16-bit integer, reducing to the half the amount of memory requirements.

4.3. Optimized encoders

The M-LTW implementation was developed finding the optimizations for maximizing R/D performance, so its software code is not optimized, just like H.264 and JPEG2000 reference software. However, we have compared its performance with respect to a full optimized implementation of JPEG2000: Kakadu [19], in order to evaluate whether a full optimization of M-LTW is worth the effort. For that purpose, we have used two versions of Kakadu software: (a) version 2.2.3, compiled without optimization options and (b) the last version 5.2.5 which is fully optimized including multi-thread and multi-core hardware capabilities, processor intrinsics like MMX/SSE/SSE2/SIMD and fast multi-component transforms.

As shown in Fig. 21, M-LTW is a very fast encoder even though not being fully optimized. The speed of M-LTW lies on the simple engine coding model. M-LTW is

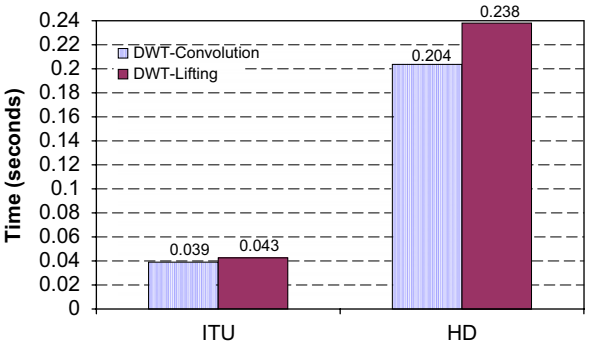

Fig. 19. Execution time comparison between DWT lifting and DWT convolution (1 frame, 1 MB L2 cache).

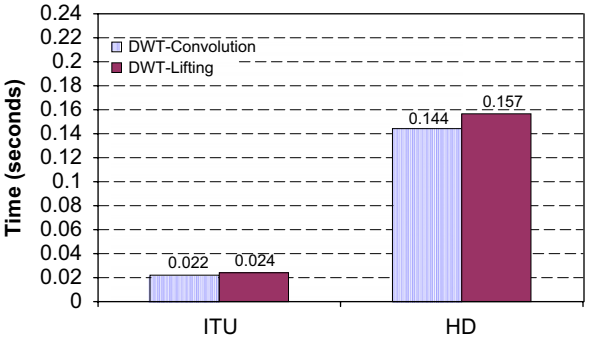

Fig. 20. Execution time comparison between DWT lifting and DWT convolution (1 frame, 2 MB L2 Cache).

Table 3
Memory requirements for evaluated encoders (kb) (results obtained with the Windows XP task manager, peak memory usage column)

Codec/format	H.264	M-JPEG 2000	M-SPIHT	M-LTW	M-LTW_Int
QCIF	6508	2264	1864	**1104**	**1104**
CIF	13016	3920	2880	**1540**	**1540**

approximately twice faster than Kakadu-5.2.5 for News CIF sequence for a PSNR of 32 dB. For HD images, M-LTW is slower than Kakadu-2.2.3, due to the cache page miss fail of the lifting DWT implementation as shown in the previous subsection. Therefore, if we use a convolution implementation of DWT, M-LTW would be slightly faster than Kakadu-2.2.3 and 1.8 times faster than Kakadu-5.2.5 for Station2 HD sequence.

Regarding to memory requirements, M-LTW needs only the amount of memory to store the source image as it was said before, while Kakadu memory requirements are independent of the image size due to its DWT block-based implementation and it is on average 1420 kb.

In terms of R/D, there are slightly differences between all codecs as Table 4 shows. For small and medium size images, M-LTW outperforms Kakadu at medium and high compression rates. For larger images, M-LTW provides slightly lower PSNR than both versions of Kakadu.

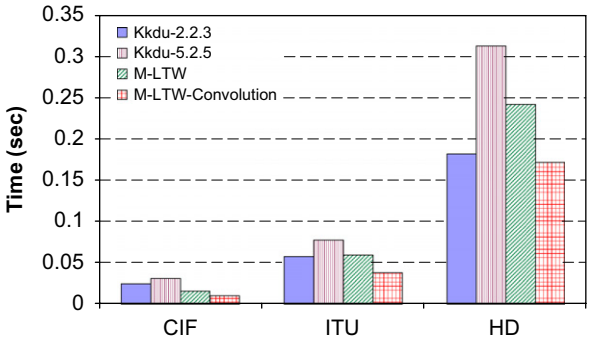

Fig. 21. Execution time comparison (end-to-end) of the coding process.

Table 4
PSNR (dB) comparison between Kakadu and M-LTW

Codec/(kb/frame)	KAKADU 2.2.3	KAKADU 5.2.5	M-LTW
News (CIF, 30 Hz)			
14.91	27.63	27.44	**27.74**
23.62	30.27	29.96	**30.42**
36.75	33.33	33.31	**33.36**
57.73	**37.26**	37.10	36.89
Mobile (ITU, 30 Hz)			
38.08	28.59	28.39	**28.61**
119.93	32.56	32.52	**32.62**
213.36	35.34	35.34	**35.47**
386.23	38.85	38.89	**38.90**
Station2 (HD, 25 Hz)			
93.92	**33.79**	33.70	33.62
180.00	**36.16**	36.15	36.08
604.64	**41.11**	41.11	40.96
1117.53	**43.18**	43.18	42.94

So, a full optimization of M-LTW codec will certainly increase coding speed and will reduce even more memory requirements, making the codec a very competitive intra-video coding solution.

5. Conclusions

In this paper we have presented a fast an efficient intra-video coder, M-LTW, which is based on the non-embedded LTW image coder. We have proposed a fast and lightweight rate-control algorithm for both M-LTW encoder versions, a float-point implementation and another one implemented with integers. After evaluating M-LTW performance in terms of R/D, execution time and memory consumption, it exhibits the best trade-off between R/D performance, coding delay (three times faster than M-JPEG2000 and 108 times faster than H.264) and overall memory usage (half the amount of memory of M-JPEG2000 and six times fewer than H.264). In addition, the M-LTW coder is able to encode an ITU video signal in real time with very low memory demands and good R/D performance at moderate to high compression rates (2 dB better than H.264 for HD video format).

For further evaluation, we have compared M-LTW coder with a highly optimized version of JPEG2000 (Kakadu), being also competitive in terms of coding delay (up to two times faster than Kakadu for small and medium size images) and R/D performance (0.4 dB for CIF, and 0.1 dB for ITU at medium and high compression rates). So, a fully optimization process will make M-LTW even faster and with lower memory requirements. These optimizations will be mainly focused on the DWT coding step by using fast and low memory demanding DWT techniques like line-based or block-based ones and exploiting the parallel capabilities of modern processors (like multithreading and SIMD instructions).

Acknowledgments

This work was funded by Spanish Ministry of education and Science under grant DPI2007-66796-C03-03 and by Valencia Government under grant ARVIV/2007/045.

References

[1] M. Adams, F. Kossentini, Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis, IEEE Trans. Image Process. 9 (6) (June 2000) 1010–1024.
[2] M. Antonini, M. Barlaud, P. Mathieu, I. Daubechies, Image coding using wavelet transform, IEEE Trans. Image Process. 1 (2) (1992) 205–220.
[3] A. Calderbank, I. Daubechies, W. Sweldens, B.-L. Yeo, Wavelet transforms that map integers to integers, Appl. Comput. Harmonic Anal. 5 (3) (July 1998) 332–369.
[4] Y. Cho, W.A. Pearlman, A. Said, Low complexity resolution progressive image coding algorithm: PROGRES (progressive resolution decompression), in: IEEE International Conference on Image Processing, September 2005.
[5] C. Chrysafis, A. Said, A. Drukarev, A. Islam, W. Pearlman, SBHP-a low complexity wavelet coder, in: IEEE International Conference on Acoustics, Speech and Signal Processing, 2000.
[6] CIPR, ⟨http://www.cipr.rpi.edu/resource/stills/kodak.html⟩, center for Image Processing Research.
[7] I. Daubechies, W. Sweldens, Factoring wavelet transforms into lifting steps, J. Fourier Anal. Appl. 4 (3) (1998) 247–268.
[8] M. Grangetto, E. Magli, M. Martina, G. Olmo, Optimization and implementation of the integer wavelet transform for image coding, IEEE Trans. Image Process. 11 (6) (2002).
[9] J. Guo, S. Mitra, B. Nutter, T. Karp, A fast and low complexity image codec based on backward coding of wavelet trees, in: Data Compression Conference, 2006.
[10] ISO/IEC 15444-1, JPEG2000 image coding system, 2000.
[11] ISO/IEC 14496-10:2003, Coding of audio-visual objects part 10: advanced video coding, 2003.
[12] O. López, M. Martínez-Rach, J. Oliver, M. Malumbres, A heuristic bitrate control for non-embedded wavelet image encoders, in: ELMAR-2006, June 2006.
[13] O. López, M. Martinez-Rach, J. Oliver, M. Malumbres, Impact of rate control tools on very fast non-embedded wavelet image encoders, in: Visual Communications and Image Processing 2007, January 2007.
[14] M. Martinez-Rach, O. Lopez, P. Piñol, J. Oliver, M. Malumbres, A study of objective quality assessment metrics for video codec design and evaluation, in: IEEE International Symposium on Multimedia, 2006, pp. 517–524.
[15] J. Oliver, M. Malumbres, Low-complexity multiresolution image compression using wavelet lower trees, IEEE Trans. CSVT 16 (11) (November 2006) 1437–1444.
[16] A. Said, A. Pearlman, A new, fast and efficient image codec based on set partitioning in hierarchical trees, IEEE Trans. CSVT 6 (3) (1996) 243–250.
[17] J. Shapiro, A fast technique for identifying zerotrees in the EZW algorithm, in: Proceedings of the IEEE International Conference Acoustic, Speech, Signal Processing, vol. 3, May 1996, pp. 1455–1458.
[18] H. Sheikh, A. Bovik, Image information and visual quality, IEEE Trans. Image Process. 15 (2) (February 2006) 430–444.
[19] Software Kakadu, ⟨http://www.kakadusoftware.com⟩.
[20] X. Wu, Compression of wavelet transform coefficients, in: The Transform and Data compression Handbook, CRC Press, 2001, pp. 347–378.