Low-Complexity 3D-DWT video encoder suitable for IPTV

O. López[a,*], P. Piñol[a], M.O. Martínez-Rach[a], M.P. Malumbres[a], J. Oliver[b]

[a] *Miguel Hernandez University, Avda. Universidad s/n, 03202 Elche (Alicante), Spain*
[b] *Universidad Politécnica de Valencia, Camino de Vera s/n, 46222 Valencia, Spain*

Abstract

3D-DWT encoders are good candidates for applications like professional video editing, IPTV video surveillance applications, live event IPTV broadcast, multi-spectral satellite imaging, HQ video delivery, etc, in order to reconstruct a frame as fast as possible. However, the main drawback of the algorithms that compute the 3D-DWT is the huge memory requirement in practical implementations. In this paper, we present a fast frame-based 3D-DWT video encoder with low memory usage. Furthermore, we evaluate the behavior of the encoding system when different separable 1D filters are applied, both in the spatial and temporal dimension.

Keywords: 3D-DWT, wavelet-based video coding, IPTV video surveillance

1. Introduction

Internet Protocol Television (IPTV) is the use of an IP broadband network to deliver television services to the end user. Nowadays, IPTV makes use of H.264 encoding [1] to deliver the media content, although MPEG-4 Part II [2] and MPEG-2 [3] encoding systems still are used. However, these encoders have a great computational complexity, specially for real-time applications or devices with power or memory comsumption constraints.

In recent years, three-dimensional wavelet transform (3D-DWT) has focused the attention of the research community, most of all in areas such as

[*] Corresponding author

Email addresses: otoniel@umh.es (O. López), pablop@umh.es (P. Piñol), mmrach@umh.es (M.O. Martínez-Rach), mels@umh.es (M.P. Malumbres), joliver@disca.upv.es (J. Oliver) video watermarking [4] and 3D coding (e.g., compression of volumetric data [5] or multispectral images [6], 3D model coding [7], and specially, video coding). These encoders are good candidates for some applications like professional video editing, IPTV video surveillance applications (Traffic cameras, child/day care, mall cctv surveillance), live event IPTV broadcast, multi-spectral satellite imaging, HQ video delivery, etc., where a specific frame of a video sequence must be reconstructed as fast as possible and with high visual quality.

In video compression, some early proposals were based on merely applying the wavelet transform on the time axis after computing the 2D-DWT for each frame [8]. Then, an adapted version of an image encoder can be used, taking into account the new dimension. For instance, instead of the typical quad-trees of image coding, a tree with eight descendants per coefficient is used in [8] to extend the SPIHT image encoder [9] to 3D video coding. Other strategy for video coding with time filtering is Motion Compensated Temporal Filtering (MCTF) [10, 11]. In these techniques, in order to compensate object (or pixel) misalignment between frames, and hence avoid the significant amount of energy that appears in high-frequency subbands, a motion compensation algorithm is introduced to align all the objects (or pixels) in the frames before being temporally filtered.

In all these applications, the first problem that arises is the extremely high memory consumption of the 3D wavelet transform if the regular algorithm is used, since a group of frames must be kept in memory before applying temporal filtering, and in the case of video coding, we know that the greater temporal decorrelation, the greater number of frames are needed in memory. Another drawback is the necessity of grouping images in small Group Of Pictures (GOP) to prevent very high memory usage, because the 3D-DWT must be computed along a set of images which are held in memory. This video sequence division into GOPs causes boundary effects between GOPs.

Even though several proposals have been made to avoid the aforementioned problems, most of them are not general (for any wavelet transform) and/or complete (the wavelet coefficients are not the same as those from the usual dyadic wavelet transform). In addition, software implementation is not always easy. In this paper, we propose a video encoder based on a frame-by-frame 3D-DWT scheme which does not require a GOP division, significantly reduces the memory usage and performs the 3D-DWT much faster than traditional algorithms.

2. 3D-DWT with low memory usage

In the regular 3D-DWT, the wavelet transform is applied in the three directions, i.e., in the horizontal, vertical and time directions, resulting in eight first level wavelet subbands (typically named as HHL_1, HLH_1, HHH_1, HLL_1, LHL_1, LLH_1, LHH_1 and LLL_1). Afterwards, the same decomposition can be done, focusing on the lowest-frequency subband (LLL_1), achieving in this way a second-level wavelet decomposition, and so on (see example in Figure 1(b)).

Because this algorithm is clearly memory-intensive, with very high memory requirements, and exhibits high coding delay (the whole 3D-DWT needs to be computed before starting the coding stage) several alternative proposals have been made.

Some of these alternatives are based on modifying the order in which the temporal filtering is calculated. E.g., in [12] the authors propose to compute the wavelet transform in the time direction with only a few frames; then the resulting high-frequency frames are released as a part of the final result, and the low-frequency frames are employed along with a few more frames so as to continue to compute the wavelet transform in the time direction. A similar example is [13], where the temporal decomposition is done by interleaving frames in small groups, getting a low-frequency frame per group, which is stored to be decomposed later with the low-frequency frames from the rest of groups. Although both algorithms ([12] and [13]) require less memory, the resulting coefficients are far from being the same as in the regular algorithm.

Other proposals rely on blocking algorithms [14], in which the transform is computed in working subsets to reduce memory usage and exploit data locality. Despite the use of overlapping techniques to avoid typical blocking artifacts, the coding efficiency decreases because the redundancy among neighboring blocks is not exploited.

In MCTF [10][11], the temporal decomposition is usually carried out with a very simple transform based on the lifting scheme [15]. When using filters with only a prediction and an update step (or even sometimes the update step is skipped), only a few frames need to be handled in MCTF.

In this section we propose an extension to a three-dimensional wavelet transform of the classical line-based approach [16], which computes the 2D-DWT with reduced memory consumption. In the new approach, frames are continuously input with no need to divide the video sequence into GOPs. Moreover, the algorithm yields slices of wavelet subbands (which we call

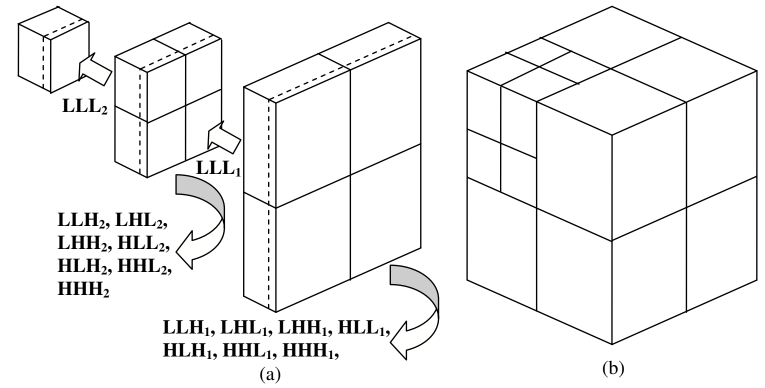

LLL₂
LLL₁
LLH₂, LHL₂, LHH₂, HLL₂, HLH₂, HHL₂, HHH₂
LLH₁, LHL₁, LHH₁, HLL₁, HLH₁, HHL₁, HHH₁,
(a)
(b)

Figure 1: Overview of the 3D-DWT computation in a two-level decomposition, (a) following a frame-by-frame scheme as shown in Figure 2; or, (b) the regular 3D-DWT algorithm

subband frames) as soon as it has enough frames to compute them. This approach works as follows:

For the first decomposition level the algorithm directly receives frames one by one. On every input frame, a one-level 2D-DWT is applied. Then, this transformed image is stored in a buffer associated to the first decomposition level. This buffer must be able to keep 2N+1 frames, where 2N+1 correspond with the number of taps for the largest analysis filter bank in the temporal direction. We only consider odd filter lengths because they have higher compression efficiency; however, this analysis could be extended to even filters as well.

When there are enough frames in the buffer to perform one step of a wavelet transform in the temporal direction (z-axis), the convolution process is calculated twice, first using the low-pass filter and then the high-pass filter. The result of this operation is the first frame of each high-frequency subbands (the HHL_1, HLH_1, HHH_1, HLL_1, LHL_1, LLH_1 and LHH_1 wavelet subbands), and the first frame of the LLL_1 subband. At this moment, for a dyadic wavelet decomposition, we can process and release the first frame of the wavelet subbands. However, the first frame of the LLL_1 subband does not belong to the final result, since it represents the incoming data for the following decomposition level. On the other hand, once the frames at the first level buffer have been used, this buffer is shifted twice (using a rotation operation) so that two frames are discarded while another two frames are inputted at the other end. Once the buffer is updated, the process can be repeated and more subband frames are obtained.

At the second decomposition level, its buffer is filled with the LLL_1 frames that have been computed in the first level. Once the buffer is completely filled, it is processed in the very same way as we have described for the first level. In this manner, the frames of the second level wavelet subbands are achieved, and the low-frequency frames from LLL_2 are passed to the third level. As depicted in Figure 1(a), this process can be repeated until the desired decomposition level (*nlevel*) is reached.

In this algorithm a major problem arises when it is implemented. This drawback is the synchronization among buffers. Before a buffer can produce frames, it must be completely filled with frames from previous buffers, therefore they start working at different moments, i.e., they have different delays. Moreover, all the buffers exchange their result at different intervals, according to their level.

Handling several buffers with different delays and rhythms becomes a hard task. To solve the synchronization problem, the algorithm depicted at Figure 2 defines a recursive function that obtains the next low-frequency subband frame (*LLL*) from a contiguous level in a similar way as authors in [17] proposed for the 2D-DWT.

```
function LowMemUsage3D_FWT(nlevel)
   set $FramesRead_{level} = 0 \ \forall level \in nlevel$
   set $FramesLines_{level} = \frac{Nframes}{2^{level}} \ \forall level \in nlevel$
   set $buffer_{level}$ = empty $\forall level \in nlevel$
   repeat
         LLL = GetLLLframe(nlevel)
         if (LLL != EOF) ProcessLowFreqSubFrame(LLL)
   until LLL = EOF
end of fuction
```

Figure 2: Perform the 3DFWT by calling GetLLLFrame recursive function

The algorithm starts requesting *LLL* frames to the last level (*nlevel*). As seen in Figure 1, the *nlevel* buffer must be filled with subband frames from the *nlevel*-1 level before it can generate frames. In order to get them, this function recursively calls itself until level 0 is reached. At this point, it no longer needs to call itself since it can return a frame from the video sequence, which can be directly read from the input/output system.

The first time that the recursive function is called at every level, it has its buffer ($buffer_{level}$) empty. Then, its upper half (from N to 2N) is recursively filled with frames from the previous level. Recall that once a frame is received, it must be transformed using a 2D-DWT before being stored. Once the upper half is full, the lower half is filled by using symmetric extension. On the other hand, if the buffer is not empty, it simply has to be updated. In order to update it, it is shifted one position so that the frame contained in the first position is discarded and a new frame can be introduced in the last position (2N) by using a recursive call. This operation is repeated twice.

However, if there are no more frames in the previous level, this recursive call will return End Of Frame (EOF). That points out that we are about to finish the computation at this level, but we still need to continue filling the buffer. We fill it by using symmetric extension again.

Once the buffer is filled or updated, both high-pass and low-pass filter banks for the time direction (z-axis) are applied to the frames in the buffer. As a result of the convolution, we get a frame of every wavelet subband at this level, and an *LLL* frame. The high-frequency coefficients are compressed and this function returns the LLL frame which is the lowest frequency subband frame (see Figure 3).

The inverse DWT algorithm is similar to the forward DWT, but applied in reverse order. The decoding process begins immediately by filling up the highest-level buffer (*nlevel*) with the information received from the bit-stream. During this process, other information from the bit-stream is ignored. Afterwards, once this buffer is full, we also begin to accept information from the previous level, and so forth, until all the buffers are full. At that moment, the video can be sequentially decoded as usual. The latency of this process is deterministic and depends on the filter length and the number of decomposition levels (the higher they are, the higher latency). However, for the regular 3D algorithm, the latency depends on the remaining number of frames in the current group when the process begins, and the GOP size. A drawback that has not been considered yet is the need to reverse the order of the subbands, from the forward DWT to the inverse one. This problem can be solved by using some buffers at both ends, so that data are supplied in the right order [16]. Other simpler solutions are to save every level in secondary storage separately so that it can be read in a different order or to keep the compressed bit-stream in memory if the 3D-DWT is used for compression.

3. Run-Length encoder

In order to have low memory consumption, once a wavelet subband is calculated, it has to be encoded as soon as possible to release memory. The

```
function GetLLLFrame ($level$)
1) First base case: No more frames to read at this level
  if $FramesRead_{level} = MaxFrames_{level}$
      return EOF
2) Second base case: The current level belongs
to the space domain and not to the wavelet domain
  else if $level = 0$
      return InputFrame()
  else
3) Recursive case
3.1) Recursively fill or update the buffer for this level
  if $buffer_{level}$ is empty
      for $i = N \ldots 2N$
          $buffer_{level}(i) = 2DFWT(GetLLLframe(level - 1))$
      SymmetricExtension($buffer_{level}$)
  else
      repeat twice
          Shift($buffer_{level}$)
          $frame$ = GetLLLframe($level - 1$)
          if $frame$ = EOF
              $buffer_{level}(2N)$ = SymmetricExtension($buffer_{level}$)
          else
              $buffer_{level}(2N)$ = 2DFWT($frame$)
3.2) Calculate the WT for the time direction from the frames
in buffer, then process the resulting high frequency subband frames
  $\{LLL, LLH, LHL, LHH\}$ =Z-axis_FWT_LowPass($buffer_{level}$)
  $\{HLL, HLH, HHL, HHH\}$ =Z-axis_FWT_HighPass($buffer_{level}$)
  ProcessSubFrames($\{LLH, LHL, LHH, HLL, HLH, HHL, HHH\}$)
  set $FramesRead_{level}$=$FramesRead_{level} + 1$
  return $LLL$
end of fuction
```

Figure 3: GetLLLFrame Recursive function

encoder cannot use global video information since it does not know the whole video. Moreover, we aim at fast execution, and hence no R/D optimization or bitplane processing can be applied, because it would turn it even slower. In the next subsection, a Run-Length Wavelet (RLW) encoder with the aforementioned features is proposed.

3.1. Fast run-length coding

In the proposed coding algorithm, 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 using the Q parameter. The coarser one is based on removing bit planes from the least significant part of the coefficients. We define *rplanes* as the number of less significant bits to be removed, and we call significant coefficient to those coefficients $c_{i,j}$ that are different to zero after discarding the least significant *rplanes* bits, in other words, if $c_{i,j} \geq 2^{rplanes}$. The wavelet coefficients are encoded as follows. The coefficients in the subband buffer are scanned row by row (to exploit their locality). For each coefficient in that buffer, if it is not significant, a run-length count of insignificant symbols at this level is increased (run_length_L). However, if it is significant, we encode both the count of insignificant symbols and the significant coefficient, and run_length_L is reset.

The significant coefficient is encoded by means of a symbol indicating the number of bits required to represent that coefficient. An arithmetic encoder with two contexts is used to efficiently store that symbol. As coefficients in the same subband have similar magnitude, an adaptive arithmetic encoder is able to represent this information in a very efficient way. However, we still need to encode its significant bits and sign. They are raw encoded to speed up the execution time.

In order to encode the count of insignificant symbols, we encode a *RUN* symbol. After encoding this symbol, the run-length count (run_length_L) is stored in a similar way as in the significant coefficients. First, the number of bits needed to encode the run value is arithmetically encoded (with a different context), afterwards the bits are raw encoded.

Instead of using run-length count symbols, we could have used a single symbol to encode each insignificant coefficient. However, we would need to encode a larger amount of symbols, and therefore the complexity of the algorithm would increase (most of all in the case of large number of insignificant contiguous symbols, which usually occurs in moderate to high compression ratios). However, the compression performance is increased if a specific symbol is used for every insignificant coefficient, since an arithmetic encoder processes more efficiently many likely symbols than a lower amount of less likely symbols. So, for short run-lengths, we encode a *LOWER* symbol for each insignificant coefficient instead of coding a run-length count symbol for all the sequence. The threshold to enter the run-length mode and start using run-length count symbols is defined by the *enter_run_mode* parameter. The formal description of the depicted algorithm can be found in Figure 4.

4. Results

4.1. Wavelet Filter Evaluation

In this section we analyze the behavior of the proposed encoder (3D-RLW) and we evaluate the performance when we use different separable 1D

```
function RLW_Code_Subband(Buffer, L)
  Scan Buffer in horizontal raster order
  for each $C_{i,j}$ in Buffer
    $nbits_{i,j} = \lceil log_2 (|C_{i,j}|) \rceil$
  if $nbits_{i,j} \leq rplanes$
      increase $run\_length_L$
  else
      if $run\_length_L \leq enter\_run\_mode$
          repeat $run\_length_L$ times
              arithmetic_output $LOWER$
      else
          arithmetic_output $RUN$
          $rbits = \lceil log_2 (run\_length_L) \rceil$
          arithmetic_output $rbits$
          output $bit_{nbits_{(i,j)}-1} (|C_{i,j}|) \ldots bit_{rplane+1} (|C_{i,j}|)$
          output $\mathrm{sign}(c_{i,j})$
end of fuction
Note: $bit_n (C)$ is a function that returns the $n^{th}$ bit of $C$
```

Figure 4: Run-length coding of the wavelet coefficients

filters in both spatial and temporal domain. For our simulation we have three different options for the 3D decomposition, as shown in Table 1. The first one, D97-D97, uses Daubechies 9/7F filter in both spatial and temporal dimension. The second one, D97-B53, uses Daubechies 9/7F filter for the spatial dimension and LeGall B5/3 filter for the temporal dimension. Finally, the B53-B53 option uses the LeGall B5/3 filter for both spatial and temporal dimension. We will compare the three 3D-RLW encoder versions versus the fast M-LTW Intra video encoder [18], in terms of R/D performance and memory requirements.

Option	Spatial	Temporal
D97-D97	Daubechies 9/7F	Daubechies 9/7F
D97-B53	Daubechies 9/7F	LeGall B5/3
B53-B53	LeGall B5/3	LeGall B5/3

Table 1: Filter choices for 3D decomposition of video

In this new algorithm (frame-by-frame 3D wavelet transform), each buffer must be able to keep either $2N + 1$ low frequency frames at every level (recall that $2N + 1$ is the filter length), or even less if the lifting scheme is used as shown in [17]. As presented in Figure 1(a), each buffer at a level i needs a quarter of coefficients if compared with the previous decomposition level $(i - 1)$. Therefore, for a frame size of $(w \times h)$ and an *nlevel* time

Format/Codec	QCIF	CIF	ITU-D1	Full-HD
D97-D97	3908	12548	22508	129750
D97-B53	3412	10476	16076	89292
B53-B53	3412	10476	16076	89292
M-LTW	1104	1540	4900	23800

Table 2: Memory requirements for evaluated filters (KB) (results obtained with Windows XP task manager, peak memory usage index)

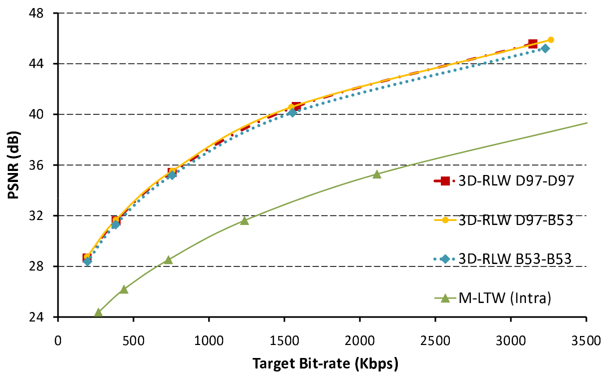


Figure 5: PSNR (dB) for all evaluated filters for Container sequence in CIF format

decomposition, the number of coefficients required by this algorithm is:

$$(2N+1)\times(w\times h)+(2N+1)\times(w\times h)/4 \\ +\ldots+(2N+1)\times(w\times h)/4^{nlevel-1} \qquad (1)$$

which is asymptotically (as *nlevel* approaches infinity) independent of the number of frames to be encoded, less than the regular case, which needs $(w\times h\times G)$, being G the number of frames in a GOP.

$$\sum_{n=0}^{\infty}\frac{(2N+1)\times(w\times h)}{4^n}=(2N+1)\times(w\times h)\times\frac{4}{3} \qquad (2)$$

For an objective evaluation, in Table 2, the memory requirements of different encoders under test are shown. Obviously, the M-LTW encoder only uses the memory needed to store one frame. The 3D-RLW version using LeGall 5/3 temporal filter requires up to 1.5 times less memory than the one using Daubechies 9/7F time filter.

Regarding R/D, in Figure 5 we can see the behavior of all evaluated encoders. As shown, the 3D-RLW version using LeGall B5/3 filter in both spatial and temporal domain obtains slightly lower R/D performance compared to the other 3D-RLW versions using Daubechies 9/7F filter in the spatial domain. It is interesting to see the improvement of 3D-RLW versions when compared to an INTRA video encoder (up to 9 dB). In these encoders no ME/MC stage is included, so the improvement is accomplished by exploiting only the temporal redundancy among video frames when applying the 3D-DWT.

Among the three 3D-RLW versions, the one using Daubechies 9/7F filter for the spatial domain and LeGall 5/3 filter for the temporal domain (D97-B53) shows the best trade off between R/D (similar behavior than the one using Daubechies 9/7F filter in the temporal domain) and memory requirements (up to 1.5 less memory than the one using Daubechies 9/7F filter in the temporal domain).

4.2. Global Evaluation

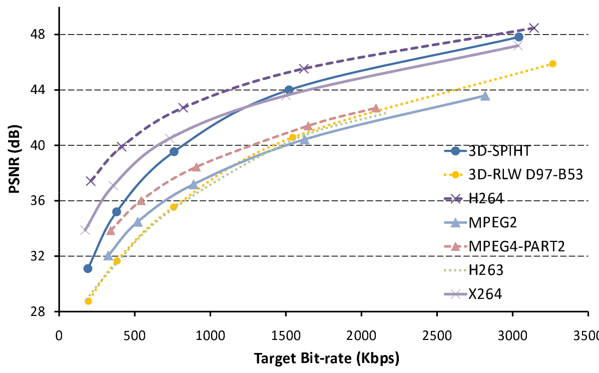
For an extensive evaluation, in this section we analyze the behavior of the proposed encoder (3D-RLW) using Daubechies 9/7F filter for the spatial domain and LeGall 5/3 filter for the temporal domain (D97-B53). We will compare the 3D-RLW encoder versus 3D-SPIHT [19], H.264 (JM16.1 version), H.263 (ffmpeg-r25117), MPEG-2 (ffmpeg-r25117), MPEG-4 Part II (ffmpeg-r25117) and X.264 (mingw32-libx264 r1713-1 high quality profile) [20] in terms of R/D performance, coding and decoding delay and memory requirements. All the evaluated encoders have been tested on an Intel PentiumM Dual Core 3.0 GHz with 2 Gbyte RAM memory.

It is important to remark that H.263, MPEG-2, MPEG-4 and X.264 evaluated implementations are fully optimized, using CPU capabilities like Multimedia Extensions 2 (MMX2), Single Instruction Multiple Data Extension 2 (SSE2Fast), Supplemental Streaming SIMD Extension 3 (SSSE3) and multithreading, whereas 3D-SPIHT and 3D-RLW are not optimized implementations.

In Table 3, the memory requirements of different encoders under test are shown. Obviously, encoders like MPEG-2, H.263 and MPEG-4 only using P frames, require to keep in memory just 2 frames to acomplish the ME/MC stage, whereas encoders based on 3D-DWT like 3D-SPIHT and 3D-RLW need to keep more frames in memory to apply the time filter. The 3D-RLW encoder uses up to 7 times less memory than 3D-SPIHT, up to 14 times less

Format/Codec	**QCIF**	**CIF**	**ITU-D1**	**Full-HD**
H264	35824	86272	227620	489960
X264	10752	36468	36600	178940
3D-RLW	**3412**	**10476**	**16076**	**89292**
3D-SPIHT	10152	34504	118460	645720

Table 3: Memory requirements for evaluated encoders (KB) (results obtained with Windows XP task manager, peak memory usage index)



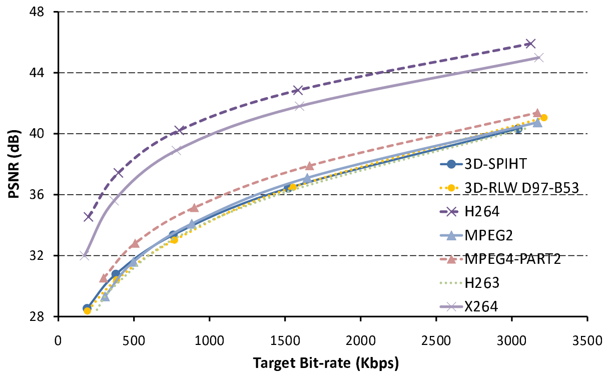
(a) Container

Figure 6: PSNR (dB) for all evaluated encoders for Container sequence in CIF format

memory than H.264 for ITU-D1 sequence size and up to 3 times less memory than X.264 which is an optimized version of H.264.

Regarding R/D, in Figures 6 and 7 we can see the R/D behavior of all evaluated encoders. As shown, H.264 is the one that obtains the best results, mainly due to the exhaustive motion estimation/motion compensation (ME/MC) stage included in this encoder, contrary to 3D-SPIHT and 3D-RLW that do not include any ME/MC stage. The optimized version of H.264 (X.264) has lower R/D performance than H.264 because it uses a fast ME/MC stage which is less accurate than the used in the H.264 standard version (up to 3 dB). The R/D behavior of 3D-SPIHT and 3D-RLW is similar for images with moderate-high motion activity, but for sequences with low movement, 3D-SPIHT outperforms 3D-RLW, showing the power of tree encoding system. The proposed encoder (3D-RLW) has a similar behavior than H.263 and MPEG-2 and slightly lower performance than MPEG-4.

Regarding coding delay, in Figure 8 we can see that the 3D-RLW encoder



(b) Foreman

Figure 7: PSNR (dB) for all evaluated encoders for Foreman sequence in CIF format

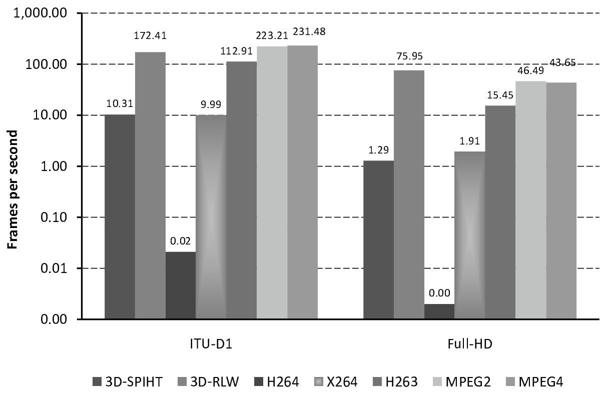


Figure 8: Execution time comparison of the encoding process

is one of the fastest encoders, being up to 16 times faster than 3D-SPIHT for ITU-D1 size sequences, 1.5 times faster than MPEG-2 for Full-HD size sequences and up to 39 times faster than X.264 for Full-HD size sequences. The decoding process is also very fast in 3D-RLW, having a similar behavior than MPEG-2 and MPEG-4 encoders for Full-HD size sequences.

5. Conclusions

In this paper a fast and low memory demanding 3D-DWT encoder has been presented and several separable 1D filters has been tested. The new encoder using Daubechies 9/7F for the spatial domain and LeGall 5/3 filter for the temporal domain (D97-B53), reduces the memory requirements compared with 3D-SPIHT (7 times less memory), H.264 (up to 14 times less memory) and X.264 (up to 3 times less memory). The new 3D-DWT encoder is faster than 3D-SPIHT (up to 16 times faster for Full-HD), MPEG-2 (up to 1.5 times faster for Full-HD) and X.264 (up to 39 times faster for Full-HD).

Regarding R/D, our proposal has a similar behavior than MPEG-2 and H.263 and slightly lower performance than MPEG-4. When compared with 3D-SPIHT, our proposal has a similar behavior for sequences with medium and high movement, but lower performance for sequences with low movement like Container. In order to improve the coding efficiency, an ME/MC stage could be added. In this manner, the objects/pixels of the input video sequence will be aligned, and so, fewer frequencies would appear at the higher frequency subbands, improving the compression performance. Also, a full optimization process exploiting the parallel capabilities of modern processors (like multithreading and SIMD instructions) will make 3D-RLW even faster.

The low memory requirements and the fast coding/decoding process, makes the 3D-LTW encoder a good candidate for IPTV applications where the coding delay is critical for proper operation.

6. Acknowledgements

Thanks to Spanish Ministry of education and Science under grants DPI2007-66796-C03-03 for funding.

References

[1] I. 14496-10, I. R. H.264, Advanced video coding (2003).

[2] ISO/IEC JTC1. ISO/IEC 14496-2, Coding of audio-visual objects (April 2001).

[3] ISO/IEC JTC1. ISO/IEC 13818-2, Generic coding of moving pictures (2000).

[4] P. Campisi, A. Neri, Video watermarking in the 3D-DWT domain using perceptual masking, in: IEEE International Conference on Image Processing, 2005, pp. 997–1000.

[5] P. Schelkens, A. Munteanu, J. Barbariend, M. Galca, X. Giro-Nieto, J. Cornelis, Wavelet coding of volumetric medical datasets, IEEE Transactions on Medical Imaging 22 (3) (2003) 441–458.

[6] P. Dragotti, G. Poggi, Compression of multispectral images by three-dimensional SPITH algorithm, IEEE Transactions on Geoscience and Remote Sensing 38 (1) (2000) 416–428.

[7] M. Aviles, F. Moran, N. Garcia, Progressive lower trees of wavelet coefficients: Efficient spatial and SNR scalable coding of 3D models, Lecture Notes in Computer Science 3767 (2005) 61–72.

[8] B. Kim, Z. Xiong, W. Pearlman, Low bit-rate scalable video coding with 3D set partitioning in hierarchical trees (3D SPIHT), IEEE Transactions on Circuits and Systems for Video Technology 10 (2000) 1374–1387.

[9] A. Said, A. Pearlman, A new, fast and efficient image codec based on set partitioning in hierarchical trees, IEEE Transactions on Circuits, Systems and Video Technology 6 (3) (1996) 243–250.

[10] A. Secker, D. Taubman, Motion-compensated highly scalable video compression using an adaptive 3D wavelet transform based on lifting, IEEE Internantional Conference on Image Processing (2001) 1029–1032.

[11] P. Cheng, J.W.Woods, Bidirectional MC-EZBC with lifting implementation, IEEE Transactions on Circuits and Systems for Video Technology (2004) 1183–1194.

[12] E. Moyano, F. Quiles, A. Garrido, L. Orozco-Barbosa, J. Duato, Efficient 3D wavelet transform decomposition for video compression, in: Int. Work. Digital and Computational Video, 2001.

[13] Y. Nian, L. Wu, S. He, Y. Gu, A new video coding based on 3D wavelet transform, in: IEEE International Conference on Intelligent Systems Design and Applications, 2006.

[14] G. Bernabe, J. Gonzalez, J. Garcia, Memory conscious 3D wavelet transform, in: Euromicro Conference, 2002.

[15] W. Sweldens, The lifting scheme: a custom-design construction of biorthogonal wavelets, Applied and Computational Harmonic Analysis 3 (2) (1996) 186–200.

[16] C. Chrysafis, A. Ortega, Line-based, reduced memory, wavelet image compression, IEEE Transactions on Image Processing 9 (3) (2000) 378–389.

[17] J. Oliver, E. Oliver, M.P.Malumbres, On the efficient memory usage in the lifting scheme for the two-dimensional wavelet transform computation, in: IEEE International Conference on Image Processing, 2005, pp. 485–488.

[18] O. Lopez, M. Martinez-Rach, P. Piñol, M. Malumbres, J.Oliver, M-LTW: A fast and efficient intra video codec, Signal Processing: Image Communication (23) (2008) 637–648.

[19] B. Kim, Z. Xiong, W. Pearlman, Very low bit-rate embedded video coding with 3D set partitioning in hierarchical trees (3D SPIHT) (1997).

[20] http://ffmpeg.arrozcru.org/autobuilds/blog/2010/09/14/ffmpeg-r25117-swscale-r32222-ok/, ffmpeg (September 2010).