Fast low memory demanding 3D video encoder

O.López, M.Martínez-Rach, P.Piñol M.P. Malumbres, J.Oliver

Dept. Física y Arquitectura de Computadores
Universidad Miguel Hernández
03202 Elche
(otoniel,mmrach,pablop,mels)@umh.es

Dept. DISCA
Universidad Politécnica de Valencia
46222 Valencia
joliver@disca.upv.es

Abstract

In recent years, several authors have begun to develop 3D-DWT based video encoders in order to exploit the benefits of the wavelet transform. However, the regular 3D-DWT implementation requires a huge amount of memory, and this is one of the main drawbacks in real applications. In this paper, we introduce a fast run-length based encoding stage over a low memory demanding frame-based 3D-DWT scheme. Our proposal shows good trade off between R/D, coding delay (up to 10 times faster than 3D-SPIHT) and memory requirements (up to 5 times less memory than 3D-SPIHT).

1 Introduction

In recent years, three-dimensional wavelet transform (3D-DWT) has focused the attention of the research community, most of all in areas such as video watermarking [2] and 3D coding (e.g., compression of volumetric data [11] or multispectral images [5], 3D model coding [1], 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 [7]. 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 [7] to extend the SPIHT image encoder to 3D video coding. A more efficient strategy for video coding with time filtering is Motion Compensated Temporal Filtering (MCTF) [12, 3]. 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 containing only a few images hinders the decorrelation of the temporal dimension and causes boundary effects between GOPs.

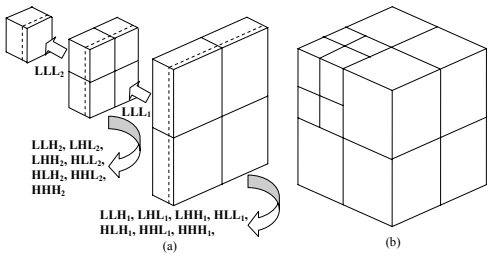

Figure 1: Overview of the 3D-DWT computation in a two-level decomposition, (a) following a frame-by-frame scheme; or, (b) the regular 3D-DWT algorithm

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 introduce a fast run-length based encoding stage over a low memory demanding frame-by-frame 3D-DWT scheme which does not require a GOP division. Also, we evaluate the behaviour of different wavelet filters in both spatial and temporal domain.

2 3D-DWT with low memory usage

In [9], the authors propose an extension to a three-dimensional wavelet transform of the classical line-based approach [4]. In this approach, using a recursive implementation, frames are continuously input without the division of the video sequence into GOPs.

In this algorithm, there is a buffer associated to each decomposition level (see Figure 1. Each buffer must keep 2N+1 frames, where 2N+1 refers to the number of taps for the largest analysis filter bank in the temporal direction. Remark that at a level *i*, the buffer memory requirements are a quarter compared to the level *i-1*.

For the first decomposition level the algorithm directly receives frames one by one and on every input frame, a one-level 2D-DWT is applied and finally, the transformed frames are stored in the associated buffer. Once we have enough frames in the buffer to perform one step of a wavelet transform in the temporal direction (z-axis), the convolution process is applied twice, first using the low-pass filter and then the high-pass filter. In this manner 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 are obtained. At this moment, we can process the first frame of each wavelet subbands except for the LLL_1 subband, which does not belong to the final result, and it will be the incoming data for the following decomposition level. Finally, once the frames at the first level buffer have been used, this buffer is shifted twice, discarding two frames and two new frames are inputted. Once the buffer is updated, the process can be repeated and more subband frames are obtained.

At the second decomposition level, its associated buffer is filled up with the LLL_1 frames computed in the first decomposition level. After that, it is processed in the same way as for the first decomposition level. This process is repeated until the desired decomposition level (*nlevel*) is reached.

The main drawback in this algorithm 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.

So as to solve the synchronization problem, the authors propose a recursive function that obtains the next low-frequency subband frame (LLL) from a contiguous level in a similar way as authors in [10] proposed for the 2D-DWT.

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 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 made, 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 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 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 symbols, we could have used a single symbol to encode every 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.

Despite of use of run-length coding, the compression performance is increased if a specific symbol is used for every insignificant coefficient, since an arithmetic encoder stores 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 symbol for all the sequence. The threshold to enter the run-length mode and start using run-length symbols is defined by the *enter_run_mode* parameter. The formal description of the depicted algorithm can be found in Figure 2.

```
function RLW_Code_Subband(Buffer, L)
  Scan Buffer in horizontal raster order
  for each C_{i,j} in Buffer
    nbits_{i,j} = ⌈log2(|C_{i,j}|)⌉
  if nbits_{i,j} ≤ rplanes
      increase run_length_L
  else
      if run_length_L ≤ enter_run_mode
          repeat run_length_L times
              arithmetic_output LOWER
      else
          arithmetic_output RUN
          rbits = ⌈log2(run_length_L)⌉
          arithmetic_output rbits
          output bit_{nbits_{(i,j)}-1}
          (|C_{i,j}|)...bit_{rplane+1}(|C_{i,j}|)
          output sign(c_{i,j})
end of fuction
Note: bit_n (C) is a function
that returns the n^th bit of C
```

Figure 2: Run-length coding of the wavelet coefficients

4 Results

In this section we analyze the behavior of the proposed encoder (3D-RLW) and we evaluate

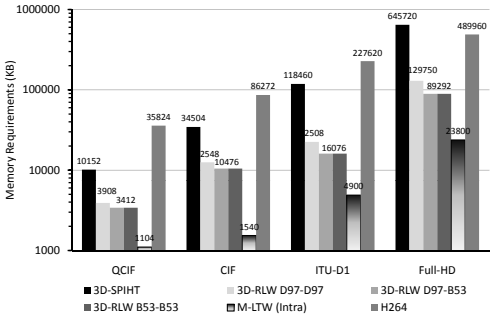

Figure 3: Memory requirements for evaluated encoders (KB) (results obtained with Windows XP task manager, peak memory usage index)

the performance when we use different separable 1D filters in both spatial and temporal domain. For our simulation we have three different options for the 3D decomposition, as shown in Table 1. We will compare the three 3D-RLW encoder versions versus the fast M-LTW Intra video encoder [8], 3D-SPIHT [6] and H.264 (JM16.1 version), in terms of R/D performance, coding 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.

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

Table 1: Filter choices for 3D decomposition of video

In Figure 3, 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 encoder (using Daubechies 9/7F time filter) uses up to 5 times less memory than 3D-SPIHT for Full-HD sequences and up to 10 times less memory than H.264 for QCIF and ITU-D1 sequence sizes. 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.

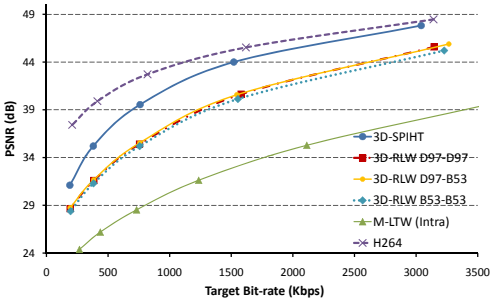

Figure 4: PSNR (dB) for all evaluated encoders for Container sequence in CIF format

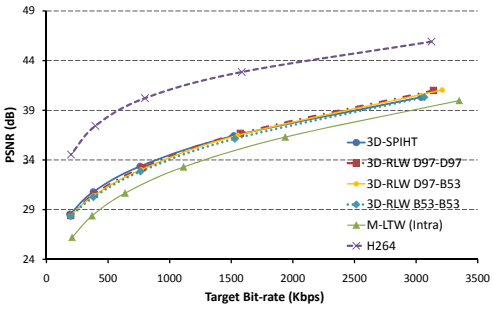

Figure 5: PSNR (dB) for all evaluated encoders for Foreman sequence in CIF format

Regarding R/D, in Figures 4 and 5 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 motion estimation/motion compensation (ME/MC) stage included in this encoder, contrary to 3D-SPIHT and 3D-RLW versions that do not include any ME/MC stage. It is interesting to see the improvement of 3D-SPIHT and 3D-RLW versions when compared to an INTRA video encoder. As mentioned, no ME stage is included in 3D-SPIHT and 3D-RLW versions, so this improvement is accomplished by exploiting only the temporal redundancy among video frames. 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 3D-RLW version using

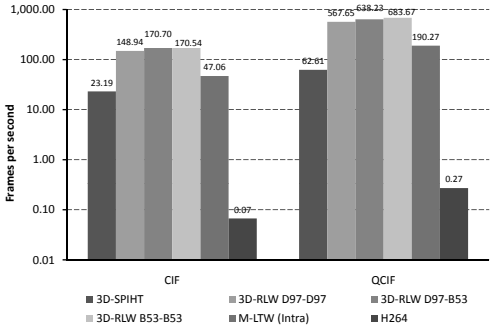

Figure 6: Execution time comparison of the encoding process

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.

Regarding coding delay, in Figure 6 we can see that all 3D-RLW encoder versions are faster than the other evaluated encoders, being up to 10 times faster than 3D-SPIHT for QCIF size sequences and 3 times faster than the M-LTW INTRA video encoder. The decoding process is also faster in 3D-RLW than in the other encoders.

5 Conclusions

In this paper a fast and low memory demanding 3D-DWT video encoder has been presented. We have implemented three encoder versions using different wavelet filters for both spatial and temporal domain. All three versions reduce the memory requirements compared with 3D-SPIHT (up to 5 times less memory) and H.264 (up to 10 times less memory). The new 3D-DWT encoder is very fast (up to 10 times faster than 3D-SPIHT) and it has better R/D behavior than the INTRA video coder M-LTW (up to 5 dB). 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 shows the best trade off between R/D, coding delay (the fastest one) and memory requirements (up to 1.5 less memory than the one using Daubechies 9/7F filter in the temporal domain). 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.

Acknowledgements

Thanks to Spanish Ministry of education and Science under grants DPI2007-66796-C03-03 and TIN2009-05737-E for funding.

References

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

[2] P. Campisi and A. Neri. Video watermarking in the 3D-DWT domain using perceptual masking. In *IEEE International Conference on Image Processing*, pages 997–1000, September 2005.

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

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

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

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

[7] B. Kim, Z. Xiong, and 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*, 10:1374–1387, December 2000.

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

[9] J. Oliver, O. Lopez, M. Martinez-Rach, and M. Malumbres. A general frame-by-frame wavelet transform algorithm for a three-dimensional analysis with reduced memory usage. In *IEEE International Conference on Image Processing*, pages 469–472, October 2007.

[10] J. Oliver, E. Oliver, and 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*, pages 485–488, September 2005.

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

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