Low Bit-Rate Video Coding with 3D Lower Trees (3D-LTW)

Otoniel López*, Miguel Martínez-Rach, Pablo Piñol, Manuel P. Malumbres,
and
José Oliver

Miguel Hernández University,
Avda. Universidad s/n, 03202 Elche, Spain.
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain.
{otoniel,mmrach,pablop,mels}@umh.es,
{joliver}@disca.upv.es

Abstract. The 3D-DWT is a mathematical tool of increasing importance in those applications that require an efficient processing of volumetric info. However, the huge memory requirement of the algorithms that compute it is one of the main drawbacks in practical implementations. In this paper, we introduce a fast frame-based 3D-DWT video encoder with low memory usage, based on lower-trees. In this scheme, there is no need to divide the input video sequence into group of pictures (GOP), and it can be applied in a continuous manner, so that no boundary effects between GOPs appear.

Key words: 3D-DWT, wavelet-based video coding

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 [1] and 3D coding (e.g., compression of volumetric data [2] or multispectral images [3], 3D model coding [4], and especially, video coding).

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 [5]. Then, an adapted version of an image encoder can be used, taking into account the new dimension. For instance, the two dimensional (2D) embedded zero-tree (IEZW) method has been extended to 3D IEZW for video coding by Chen and Pearlman[6], and showed promise of an effective and computationally simple video coding system without motion compensation, and obtained excellent numerical and visual results. A 3D zero-tree coding through modified EZW has also been used with good results in compression of volumetric images[7]. In

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

[5], instead of the typical quad-trees of image coding, a tree with eight descendants per coefficient is used 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) [8, 9]. 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.

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 this section we propose an extension to a three-dimensional wavelet transform of the classical line-based approach [10], 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 subband frames) as soon as it has enough frames to compute them. This approach works as follows:

The algorithm starts requesting LLL frames to the last level (*nlevel*). As seen in Fig. 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

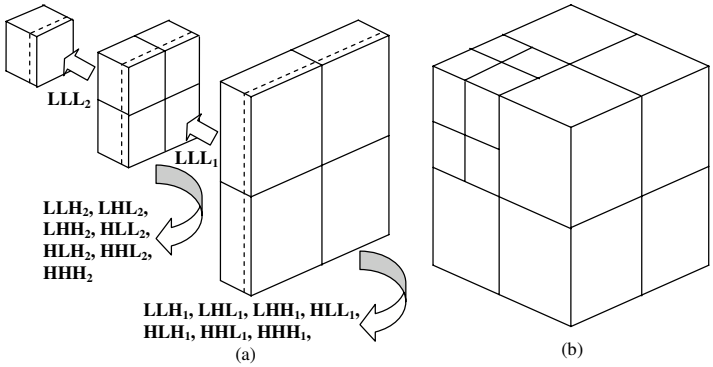


Fig. 1. Overview of the 3D-DWT computation in a two-level decomposition, (a) following a frame-by-frame scheme as shown in Fig. 2; or, (b) the regular 3D-DWT algorithm

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.

```
function LowMemUsage3D_FWT(nlevel)
  set FramesRead_level = 0 ∀level ∈ nlevel
  set FramesLines_level = Nframes / 2^level ∀level ∈ nlevel
  set buffer_level = empty ∀level ∈ nlevel
  repeat
      LLL = GetLLLframe(nlevel)
      if (LLL != EOF) ProcessLowFreqSubFrame(LLL)
  until LLL = EOF
end of fuction
```

Fig. 2. Perform the 3DFWT by calling GetLLLFrame recursive function

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 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 (HHL_{level}, HLH_{level}, HHH_{level}, HLL_{level}, LHL_{level}, LLH_{level} and LHH_{level}), and an LLL frame. The high-frequency coefficients are compressed and this function returns the LLL frame (see Fig. 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.

```
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 ... 2N
          buffer_level(i) = 2DFWT(GetLLframe(level − 1))
      FullSymmetricExtension(buffer_level)
  else
      repeat twice
          Shift(buffer_level)
          frame = GetLLLframe(level − 1)
          if frame = EOF
              buffer_level(2N) = SymmetricExt(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
```

Fig. 3. GetLLLFrame Recursive function

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 [10]. 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 Lower Tree Wavelet Encoder (LTW)

Since the proposed video coder is based on the LTW image coding algorithm [11] the basic principles of LTW will be described briefly in this section. In LTW,

the quantization process is performed by two strategies: one coarser and another finer. The finer one consists in 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 [12]) 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 $2^{rplanes}$.

The 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). The symbol set employed in the LTW altorithm is the following one: a *LOWER* symbol represents a coefficient that is the root of a lowertree, 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 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. For a significant coefficient, we simply use a symbol indicating the number of bits needed to represent it.

Let us describe the coding algorithm. In the first stage (symbol computation), all wavelet subbands are scanned in $2x2$ 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 $2x2$ 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 $2x2$ 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.

Finally, in the second stage, subbands are encoded from the LL_N subband to the first-level wavelet subbands. Observe that this is the order in which the decoder needs to know the symbols, so that lower-tree roots are decoded before its leaves. In each subband, for each $2x2$ 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.

4 3D LTW

This section introduces the extension of the concept of LTW still image coding to 3D video coding. Our main concern is to keep the same simplicity of the 2D LTW, still giving high performance and low memory requirements.

However, some changes must be done in the LTW algorithm so that it can be incorporated in this efficient wavelet transform. The main changes are:

- Global knowledge of the video frame is no longer available, and therefore an estimation of the highest coefficient that may appear should be made, mainly depending on the type of wavelet normalization and the pixel resolution of the source video (in bpp). Finally, to ensure the correctness of the encoder, an escape code should be used for values outside the predicted range.
- Since coefficients from different subband levels are interleaved (due to the computation order of the proposed wavelet transform), instead of a single bitstream, we should generate a different bitstream for every subband level. These bitstreams can be held in memory or saved in secondary storage, and are employed to form the final ordered bitstream.
- Now, the root of a tree has eight descendants, instead of the four descendants in the 2D-LTW.

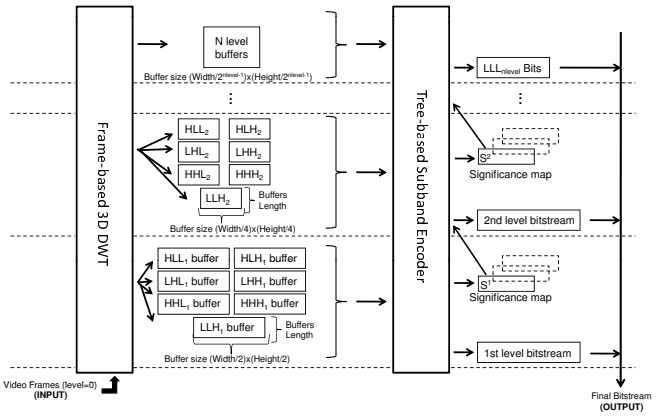


Fig. 4. Overview of the proposed tree-based encoder with efficient use of memory

Fig. 4 shows our overall system. A binary significance map (that will be described later) is needed at each level. In this scheme, when the 3D-DWT releases subband frames, they are inserted into an encoder buffer, which is passed to the tree-based encoder once it is full. The encoder buffer have to store two subband frames for each subband type.

An important difference between this version and the LTW presented previously is that the new adapted encoder must process coefficients in only one-pass,

and therefore symbols must be computed and output at once. However, in this case, it is not an important drawback because the order of the wavelet coefficients is later arranged for the decoder with an independent bitstream generation at each level.

The adapted encoding algorithm is formally described in Fig. 5. Let us see it with some detail. The encoder has to determine if each $2x2$ block of coefficients of both subband frames stored in the encoding buffer is part of a lower-tree. If the eight coefficients in these blocks are lower than the quantization threshold $2^{rplanes}$, and their descendant offspring are also insignificant, they are part of a lower-tree and do not need to be encoded. In order to know if their offspring are significant, we need to hold a binary significance map of every encoder buffer (S^L in the figure) because the encoder buffer is overwritten by the wavelet transform once it is encoded, and hence the significance for their ascendant coefficients is not automatically held. Obviously, this significance map was not needed in the original LTW because the whole image was available for the encoder. The width of each significance map is sized eigth the size of the encoder buffer that it represents, since the significance is held for both $2x2$ block. The significance of both $2x2$ blocks can be held with a single bit. Therefore, the memory required for these significance maps is almost negligible when compared with the rest of buffers.

As in original LTW encoder, when there is a significant coefficient in both $2x2$ block or in its descendant coefficients, we need to encode each coefficient separately. Recall that in this case, if a coefficient and all its descendants are insignificant, we use the *LOWER* symbol to encode the entire tree, but if it is insignificant, and the significance map of its eight direct descendant coefficients shows that it has a significant descendant, the coefficient is encoded as *ISOLATED_LOWER*. Finally, when a coefficient is significant, it is encoded with a numeric symbol along with its significant bits and sign.

At the last level (N), the tree cannot be propagated upward, and for this reason, we always encode all the coefficients at this level. Moreover, we can keep the compressed bit-stream in memory, which allows us to invert the order of the bitstream for the inverse procedure.

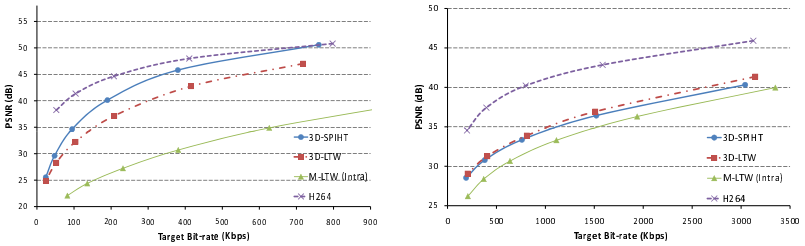
5 Results

In this section we analyze the behavior of the proposed encoder (3D-LTW). We will compare the 3D-LTW encoder versus the fast M-LTW Intra video encoder[13], 3D-SPIHT [14] and H.264 (JM16.1 version), 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 1 Gbyte RAM memory.

In Table 1, 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-LTW encoder (using Daubechies 9/7F filter for both spatial and temporal filtering) uses up to 3.4 times less memory than 3D-SPIHT for

function SubbandCode($level$, $Buffer$, $S^{level-1}$, S^{level})
  Scan Buffer in $2x2$ blocks ($B_{x,y}$) in horizontal raster order
  for each block $B_{x,y} = \{c_{2x,2y}, c_{2x+1,2y}, c_{2x,2y+1}, c_{2x+1,2y+1}\}$
  if $level \neq N \wedge \left(c_{i,j} < 2^{rplanes} \wedge S_{i,j}^{level-1} isInsignif. \forall c_{i,j} \in B_{x,y}\right)$
    set $S_{x,y}^{level} = Insignif.$
  else
    set $S_{x,y}^{level} = Signif.$
  for each $c_{i,j} \in B_{x,y}$
    if $\left|c_{i,j} < 2^{rplanes}\right|$
      if $S_{i,j}^{level-1} isInsignif.$
        arithmetic_output $LOWER$
      else
        arithmetic_output $ISOLATED\_LOWER$
    else
      $nbits_{i,j} = \lceil log_2 (|C_{i,j}|) \rceil$
        if $S_{i,j}^{level-1} isInsignif.$
          arithmetic_output $nbits_{i,j}^{LOWER}$
        else
          arithmetic_output $nbits_{i,j}$
          output $bit_{nbits_{(i,j)}-1} (|C_{i,j}|) \ldots bit_{rplane+1} (|C_{i,j}|)$
          output sign$(c_{i,j})$
        endif
    endif
end of fuction
Note: $bit_n\,(C)$ is a function that returns the n^{th} bit of C

Fig. 5. Lower tree wavelet coding with reduced memory usage



(a) Container (b) Foreman

Fig. 6. PSNR (dB) for all evaluated encoders for a) Container sequence in QCIF format and b) Foreman sequence in CIF format

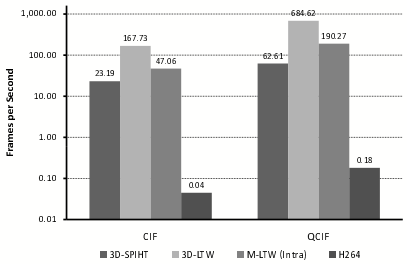


Fig. 7. Execution time comparison of the encoding process

Codec/Format	H.264	3D-SPIHT	3D-LTW	M-LTW
QCIF	35824	10152	4008	**1104**
CIF	86272	34504	10644	**1540**

Table 1. Memory requirements for evaluated encoders (KB) (results obtained with Windows XP task manager, peak memory usage index)

CIF sequence size and up to 9 times less memory than H.264 for QCIF sequence size.

Regarding R/D, in Fig. 6 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-LTW that do not include any ME/MC stage. It is interesting to see the improvement of 3D-SPIHT and 3D-LTW when compared to an INTRA video encoder. As mentioned, no ME stage is included in 3D-SPIHT and 3D-LTW, so this improvement is accomplished by exploiting only the temporal redundancy among video frames. The R/D behavior of 3D-SPIHT and 3D-LTW is similar for images with moderate-high motion activity, with a better behavior of 3D-LTW than 3D-SPIHT (up to 0.5 dB), but for sequences with low movement, 3D-SPIHT outperforms 3D-LTW, mainly due to the further dyadic decompositions applied in the temporal high frequency.

Regarding coding delay, in Fig. 7 we can see that the 3D-LTW encoder is the fastest one, being up to 10 times faster than 3D-SPIHT for QCIF size sequences, 3.5 times faster than the M-LTW INTRA video encoder and up to 3800 times faster than H.264. The decoding process is also faster in 3D-LTW than in the other encoders.

6 Conclusions

In this paper a fast and very low memory demanding 3D-DWT encoder has been presented. The new encoder reduces the memory requirements compared with 3D-SPIHT (3.5 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 11 dB). 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.

References

1. Campisi, P., Neri, A.: Video watermarking in the 3D-DWT domain using perceptual masking. In: IEEE International Conference on Image Processing. (September 2005) 997–1000

2. Schelkens, P., Munteanu, A., Barbariend, J., Galca, M., Giro-Nieto, X., Cornelis, J.: Wavelet coding of volumetric medical datasets. IEEE Transactions on Medical Imaging **22**(3) (March 2003) 441–458
3. Dragotti, P., Poggi, G.: Compression of multispectral images by three-dimensional SPITH algorithm. IEEE Transactions on Geoscience and Remote Sensing **38**(1) (January 2000) 416–428
4. Aviles, M., Moran, F., Garcia, N.: Progressive lower trees of wavelet coefficients: Efficient spatial and SNR scalable coding of 3D models. Lecture Notes in Computer Science **3767** (2005) 61–72
5. Kim, B., Xiong, Z., Pearlman, W.: 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** (December 2000) 1374–1387
6. Chen, Y., Pearlman, W.: Three-dimensional subband coding of video using the zero-tree method. In: Visual Communications and Image Processing. Volume Proc. SPIE 2727. (March 1996) 1302–1309
7. Luo, J., Wang, X., Chen, C., Parker, K.: Volumetric medical image compression with three-dimensional wavelet transform and octave zerotree coding. In: Visual Communications and Image Processing. Volume Proc. SPIE 2727. (March 1996) 579–590
8. Secker, A., Taubman, D.: Motion-compensated highly scalable video compression using an adaptive 3D wavelet transform based on lifting. IEEE Internantional Conference on Image Processing (October 2001) 1029–1032
9. Cheng, P., J.W.Woods: Bidirectional MC-EZBC with lifting implementation. IEEE Transactions on Circuits and Systems for Video Technology (October 2004) 1183–1194
10. Chrysafis, C., Ortega, A.: Line-based, reduced memory, wavelet image compression. IEEE Transactions on Image Processing **9**(3) (March 2000) 378–389
11. Oliver, J., Malumbres, M.P.: Low-complexity multiresolution image compression using wavelet lower trees. IEEE Transactions on Circuits and Systems for Video Technology **16**(11) (2006) 1437–1444
12. Said, A., Pearlman, A.: 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
13. Lopez, O., Martinez-Rach, M., Piñol, P., Malumbres, M., J.Oliver: M-LTW: A fast and efficient intra video codec. Signal Processing: Image Communication (23) (July 2008) 637–648
14. Kim, B., Xiong, Z., Pearlman, W.: Very low bit-rate embedded video coding with 3D set partitioning in hierarchical trees (3D SPIHT) (1997)