A GENERAL FRAME-BY-FRAME WAVELET TRANSFORM ALGORITHM FOR A THREE-DIMENSIONAL ANALYSIS WITH REDUCED MEMORY USAGE

Jose Oliver[a], *Otoniel López*[b], *Miguel Martínez-Rach*[b], *and Manuel P. Malumbres*[b]

[a]Technical University of Valencia, Camino de Vera sn, 46022 Valencia, Spain
[b]Miguel Hernández University, Avda. Universidad s/n, 03202 Elche, Spain

ABSTRACT

The 3D-DWT is a mathematical tool of increasing importance. 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 frame-by-frame algorithm to calculate the 3D-DWT with low memory usage. This algorithm is general, in the sense that it can be employed with any wavelet transform and, contrary to other proposals, it gets the same results as the regular wavelet transform. In addition, there is no need to divide the input video sequence into group of frames, and it can be applied in a continuous manner, so that coding efficiency is increased and no blocking artifacts appear.

Index Terms— 3D-DWT, wavelet-based video coding

1. INTRODUCTION

In the recent years, the 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, instead of the typical quad-trees of image coding, a tree with eight descendants per coefficient is used in [5] to extend SPIHT [6], the well-known image encoder, to video coding. However, the coding efficiency of these video encoders is poor in moderate-to-high motion sequences due to the appearance of misaligned objects in the time direction, causing an energy increase in high-frequency subbands, and thereby preventing energy concentration in low-frequency subbands. A more efficient strategy for video coding with time filtering is Motion Compensated Temporal Filtering (MCTF) [7] [8]. In these techniques, in order to compensate the object (or pixel) misalignment between frames, and hence to 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 temporal 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 more number of frames are needed in memory. Another drawback is the need to group images in small Group of Pictures (GOPs) to prevent very high memory usage, because the 3D-DWT must be computed along a set of images which are held in memory. This division of the video sequence in 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, a software implementation is not always easy. In this paper, we extend to video the line-based approach introduced in [9] to compute the 2D-DWT. This approach is general so any wavelet transform can be computed. To ease a software implementation, we use the same recursive strategy as in [10].

2. THREE-DIMENSIONAL DWT ALGORITHMS

In the regular 3D-DWT, the wavelet transform is applied in the three directions, i.e., in the spatial directions (horizontal and vertical) and in the time direction (which is known as temporal filtering), resulting in eight first level wavelet subbands (typically named as LLL_1, LHL_1, LLH_1, LHH_1, HLL_1, HHL_1, HLH_1, HHH_1). Afterwards, the same decomposition can be done, focusing on the low-frequency subband (LLL_1), achieving in this way a second-level wavelet decomposition, and so on (see example in Fig 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 [11] 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

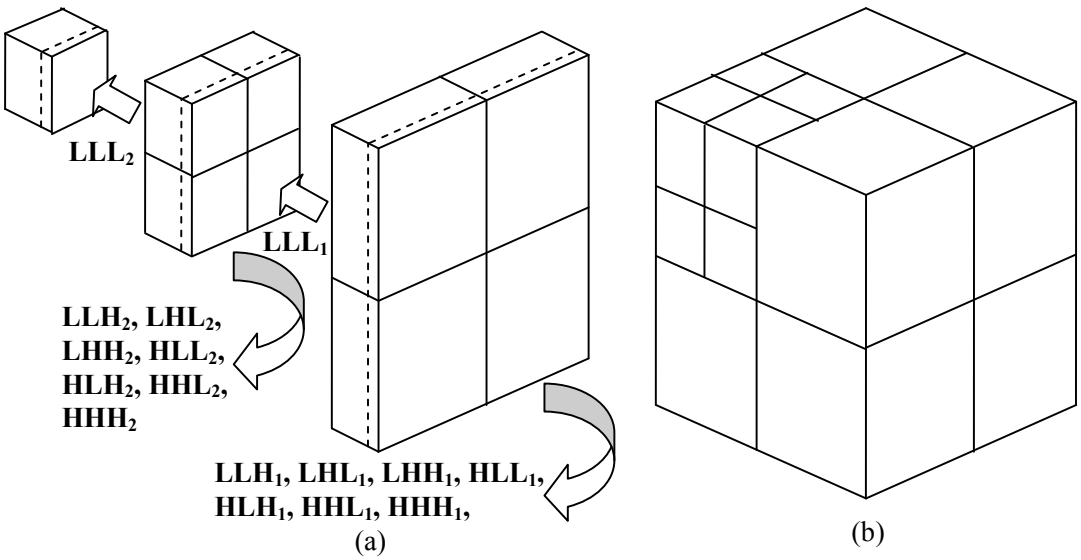

Fig. 1: Overview of the 3D DWT computation in a two-level decomposition, (a) following a frame-by-frame scheme as shown in Algorithm 1 or (b) the regular 3D DWT algorithm.

along with a few more frames so as to continue to compute the wavelet transform in the time direction. A similar example is [12], 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 ([11] and [12]) require less memory, the resulting coefficients are far from being the same as in the regular algorithm.

Other proposals rely on blocking algorithms [13], 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 [7] [8], the temporal decomposition is usually carried out with a very simple transform based on the lifting scheme [14]. 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. However, if longer filters with several decomposition levels are applied in the temporal dimension, memory handling becomes difficult.

3. FRAME-BY-FRAME 3D-DWT

In this section, we propose to extend the line-based approach [9], which computes the 2D-DWT with reduced memory consumption, to a three-dimensional wavelet transform. 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.

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 is the number of taps for the largest analysis filter bank. 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 subband (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, but it is needed as incoming data for the following decomposition level. On the other hand, once the frames in 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 input at the other end. Once the buffer is updated, the process can be repeated and more subband frames are obtained.

At the second 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 it is 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

```
function GetLLFrame ( 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 = GetLLframe ( 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
```

Algorithm 1.1: Recursive function

have different delays. Moreover, all the buffers exchange their result at different intervals, according to their level.

Handling several buffers with different delay and rhythm becomes a hard task. The next section proposes a recursive algorithm that clearly specifies how to perform this communication between buffers.

4. A RECURSIVE IMPLEMENTATION OF THE FRAME-BY-FRAME 3D-DWT

In this section, we present an algorithm based on [10] that automatically solves the synchronization problem among levels that has been addressed in the previous section. To solve this problem, this algorithm defines a recursive function that obtains the next low-frequency subband frame (LLL) from a contiguous level.

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 call itself until the level 0 is reached. At this

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

Algorithm 1.2: Perform the 3DFWT by calling Algorithm 1.1

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 complete recursive algorithm is formally described in the frame entitled *Algorithm 1.1*, while *Algorithm 1.2* sets up the variables and performs the DWT by calling the recursive algorithm. Let us see the first algorithm.

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 (the N+1 frame is copied into the N-1 position,…, the 2N is copied into the 0). 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 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 processed according to the application (compressed, saved to secondary storage, etc.) and this function returns the LLL frame.

Every recursive function needs at least one base case to stop backtracking. This function has two base cases. The first case is when all the frames at this level have been read. It is detected by keeping an account of the number of frames read and the maximum number of frames that can be read at every level. In this case, the function returns EOF. An alternative when the number of frames in the sequence is unknown a priori is propagating the EOF label. The second base case is reached when *level* gets 0 and then no further recursive call is need since a frame can be directly read from the input video sequence.

The inverse DWT algorithm is similar to the forward DWT, but applied in reverse order. An important difference between this proposal and those based on GOPs is how the

video can be decoded from the middle of the bitstream, that is, if the user begins to receive the video broadcast while it is already in progress. In the regular algorithm, the current group of frames being received is ignored, and then, the following group is stored in memory. After it has been entirely received, it can be decoded, and the 3D-IWT can be applied. On the other hand, for the inverse transform in the frame-by-frame scheme, the decoding process begins immediately by filling up the highest-level buffer (*nlevel*) with the information received from the bitstream. During this process, other information from the bitstream is ignored. Afterwards, once this buffer is full, we begin to accept also 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 determinist 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 [9]. Other simpler solutions are: to save every level in secondary storage separately so that it can be read in a different order and, if the WT is used for compression, to keep the compressed bitstream in memory.

5. MEMORY CONSUMPTION COMPARISON

In this new algorithm, at every level, each buffer must be able to keep either 2N+1 low frequency frames (recall that 2N+1 is the filter length), or even less if the lifting scheme is used as shown in [10]. As presented in Figure 1(a), each buffer at a level i needs a quarter of coefficients if compared with the previous level ($i-1$). Therefore, for a frame size of ($w \times h$) and an *nlevel* time decomposition, the number of coefficients required by this algorithm is:

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

which is asymptotically (as *nlevel* approaches infinity)

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

independently 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. For instance, in a C implementation, a 3D-DWT of a CIF sequence (with B5/3 and three decomposition levels) has required 2.5 MB with our new proposal, while the regular algorithm with 32 frames/GOP needs 12.4 MB, and introduces discontinuities in the transform, being less efficient for coding purposes.

6. CONCLUSIONS

A frame-by-frame transform algorithm has been presented, considering the existing problems about different delay and rhythm among the buffers. The new algorithm reduces the memory requirements compared with the regular one, computing exactly the same coefficients. In addition, there is no need to artificially divide the video sequence in constant-size group of pictures. As future work, a motion estimation stage will be included to align video motion, improving coding efficiency by approaching a t+2D scheme.

7. REFERENCES

[1] P. Campisi, A. Neri, "Video watermarking in the 3D-DWT domain using perceptual masking," *IEEE International Conference on Image Processing (ICIP)*, pp. 997-1000, Sept 2005.

[2] P. Schelkens, A. Munteanu, J. Barbariend, M. Galca, X. Giro-Nieto, Jan Cornelis, "Wavelet Coding of Volumetric Medical Datasets," *IEEE Tr. Medical Imaging*, pp. 441-458, March 2003.

[3] P.L.Dragotti, G.Poggi, "Compression of multispectral images by three-dimensional SPITH algorithm," *IEEE Transactions on Geoscience and Remote Sensing*, pp. 416-428, Jan 2000.

[4] M. Avilés, F. Morán, N. García, "Progressive Lower Trees of Wavelet Coefficients: Efficient Spatial and SNR Scalable Coding of 3D Models," *LNCS, vol 3767*, pp. 61-72, 2005.

[5] B. J. Kim, Z. Xiong, W. A. Pearlman, "Low bit-rate scalable video coding with 3D set partitioning in hierarchical trees (3D SPIHT)," *IEEE Tr. on Cir. and Sys. for Video Tech.*, Dec 2000.

[6] A. Said, A. Pearlman. "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, June 1996.

[7] A. Secker, D. Taubman, "Motion/compensated highly scalable video compression using an adaptive 3D wavelet transform based on lifting," *IEEE ICIP*, pp. 1029-1032, Oct 2001.

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

[9] C. Chrysafis, and A. Ortega, "Line-based, reduced memory, wavelet image compression," *IEEE Transactions on Image Processing.*, March 2000.

[10] J. Oliver, E. Oliver, M.P.Malumbres, "On the efficient memory usage in the lifting scheme for the two-dimensional wavelet transform computation," *IEEE ICIP*, Sept 2005.

[11] E. Moyano, F.J. Quiles, A. Garrido, L. Orozco-Barbosa, J. Duato, "Efficient 3D wavelet transform decomposition for video compression," *Int. Work. Digital & Computational Video*, Oc2001.

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

[13] G. Bernabé, J. González, J.M. García, "Memory Conscious 3D Wavelet Transform," *Euromicro Conference*, Sept 2002.

[14] Sweldens, "The lifting scheme: a custom-design construction of biorthogonal wavelets," *Appl. Comput. Harmon. Anal.*, 1996.