Noname manuscript No.
(will be inserted by the editor)

GPU-based 3D Lower tree Wavelet Video Encoder

Vicente Galiano · Otoniel López-Granado · Manuel P. Malumbres · L. Anthony Drummond · Hector Migallón

Received: date / Accepted: date

Abstract The 3D-DWT is a mathematical tool of increasing importance in those applications that require an efficient processing of huge amounts of volumetric info. Other applications like professional video editing, video surveillance applications, multi-spectral satellite imaging, HQ video delivery, etc, would rather use 3D-DWT encoders to reconstruct a frame as fast as possible. In this paper, we introduce a fast GPU-based encoder which uses 3D-DWT transform and lower trees. Also, we present an exhaustive analysis of the use of GPU memory. Our proposal shows good trade off between R/D, coding delay (as fast as MPEG-2 for High definition) and memory requirements (up to 6 times less memory than x264).

Keywords 3D-DWT · video coding · GPU · manycore

1 Introduction

At video content production stages, digital video processing applications require fast frame random access to perform an undefined number of real-time decompressing-editing-compressing interactive operations, without a significant loss of original video content quality. Intra-frame coding is desirable

This research was supported by the Spanish Ministry of Education and Science under grant TIN2011-27543-C03-03 and the Spanish Ministry of Science and Innovation under grant number TIN2011-26254 and TEC2010-11776-E.

V. Galiano, O. López-Granado, M.P. Malumbres, H. Migallón
Physics and Computer Architecture Department
Miguel Hernández University. Elche, Spain 03202
Tel.: +34-966658392
E-mail: {vgaliano,otoniel,mels,hmigallon}@umh.es
L. Anthony Drummond
Lawrence Berkeley National Laboratory
One Cyclotron Road, Berkeley, CA 94703, USA
E-mail: LADrummond@lbl.gov

as well in many other applications like video archiving, high-quality high-resolution medical and satellite video sequences, applications requiring simple real-time encoding like video-conference systems or even for professional or home video surveillance systems [13] and Digital Video Recording systems (DVR). However, intra coding does not take profit of the temporal redundancy between frames.

In the last years, most of all in areas such as video watermarking [3] and 3D coding (e.g., compression of volumetric medical data [15] or multispectral images [5], 3D model coding [2], and especially, video coding), three-dimensional wavelet transform (3D-DWT) based encoders have arisen as an alternative between simple intra coding and complex inter coding solutions that applies motion compensation between frames to exploit temporal redundancy.

In [12], authors utilized 3-D spatio-temporal subband decomposition and geometric vector quantization (GVQ). In [17] a full color video coder based on 3-D subband coding with camera pan compensation was presented. In [4] an extension to 3D of the well known embedded zerotree wavelet (EZW) algorithm developed by Shapiro [16] was presented. Similarly, an extension to 3D-DWT of the set partitioning in hierarchical trees (SPIHT) algorithm developed by Said and Pearlman [14] was presented in [10], using a tree with eight descendants per coefficient instead of the typical quad-trees of image coding. All of this 3D-DWT based encoders are faster than complex inter coding schemes but slower than simple intra coding solutions. So we will try in this work to speed up 3D video encoders to achieve coding delays as closer as possible to the ones obtained by intra video encoders but with a clearly superior compression performance. In order to achieve this goal, we will focus on GPU-based platforms.

Wide research have been carried out to accelerate the DWT, specially the 2D DWT, exploiting both multicore architectures and graphic processing units (GPU). In [19], a Single Instruction Multiple Data (SIMD) algorithm runs the 2D-DWT on a GeForce 7800 GTX using Cg and OpenGL, with a remarkable speed-up. A similar effort has been performed in [18] combining Cg and the 7800 GTX to report a 1.2 - 3.4 speed-up versus a CPU counterpart. In [6], authors present a CUDA implementation for the 2D-FWT running more than 20 times as fast as the sequential C version on a CPU, and more than twice as fast as the optimized OpenMP and Pthreads versions implemented on a multicore CPU. In [7], authors present GPU implementations for the 2D-DWT obtaining speed-ups up to 20 when compared to the CPU sequential algorithm.

In this work, we present a GPU 3D-DWT based video encoder using lower trees as the core coding system. The proposed encoder requires less memory than 3D-SPIHT [10] and has a good R/D behavior. Furthermore, we present an in-depth analysis of the use of GPU's to accelerate the 3D-DWT transform. Using these strategies, the proposed encoder is able to compress a Full-HD video sequence in real time.

The rest of the paper is organized as follows. Section 2 presents the proposed 3D-DWT based encoder. In Section 3 a performance evaluation in terms of R/D, memory requirements and coding time is presented. Section 4 describes several optimization proposals based on CUDA to process the 3D-DWT transform, while in Section 5 we analyze these proposals when applied to the proposed encoder. Finally in Section 6 some conclusions are drawn.

2 Encoding system

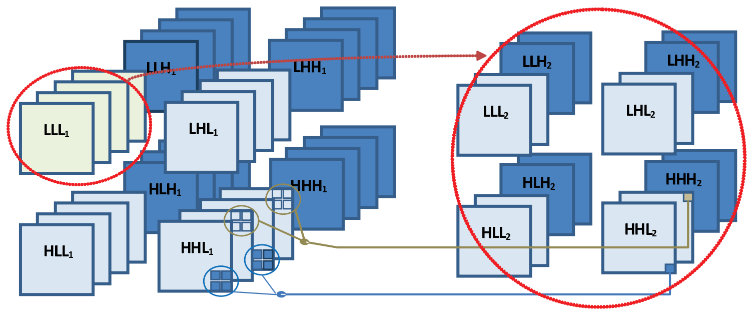


Fig. 1 Example of a two decomposition level 3D-DWT and the relationship between parent-child subbands

In this section we present a 3D-DWT based encoder with low complexity and good R/D performance. As our main concern is fast encoding process, no R/D optimization, motion estimation/motion compensation (ME/MC) or bit-plane processing is applied. This encoder is based on both the 3D-DWT transform and lower-trees (3D-LTW).

First of all, the 3D-DWT is applied to a group of pictures (GOP). In Fig. 1 an example of a two level decomposition of the 3D-DWT transform is applied to a eigth-frame video sequence. As it can be seen on the left side, spatial decomposition to all video frames is performed resulting in four subbands ($LL{*}_1$, $LH{*}_1$, $HL{*}_1$, $HH{*}_1$). After applying the temporal decomposition, we will obtain the high-frequency temporal subbands ($**H_1$ labeled subbands with a dark blue color), and the low-frequency ones ($**L_1$ labeled subbands with a light blue color). On the right side of Fig. 1, we show the second decomposition level of the 3D-DWT transform. So, we will perform the same process to the frames belonging to the LLL_1, performing the spatial and temporal DWT filtering to obtaing the corresponding subbands. Finally, we also show the wavelet coefficients offspring relationship, that the coefficient encoder will exploit. As it can be seen each coefficient of a particular subband at N-th decomposition level will have eigth descendants in the (N-1)-th decomposition level as shown at figure.

After all 3D-DWT decomposition levels are applied, all the resulting wavelet coefficients are quantized and then, the encoding system compresses the input data to obtain the final bit-stream corresponding to that GOP. It is important to remark that the compressed bit-stream is ordered in such a way that the decoder obtains the bit-stream in the correct order.

2.1 Lower-tree wavelet coding

The proposed video coder is based on the LTW image coding algorithm [11]. As in LTW encoder, the proposed video codec uses a scalar uniform quantization by means of two quantization parameters: *rplanes* and Q. The finer quantization consists in applying a scalar uniform quantization, Q, to all wavelet coefficients. The coarser quantization is based on removing the least significant bit planes, *rplanes*, from wavelet coefficients.

The encoder uses a tree structure to reduce data redundancy among subbands (similar to that of [10]), and also as a fast way of grouping coefficients, reducing the number of symbols needed to encode the image. This structure is called lower tree, and all the coefficients in the tree are lower than $2^{rplanes}$. In Fig. 1 a example of the relationship between subbands is presented.

Let us describe the coding algorithm. In the first stage (symbol computation), all wavelet subbands are scanned from the first decomposition level to the N-th (to be able to build the lower-trees from leaves to root) and the encoder has to determine if each 2×2 block of coefficients of both subband frames is part of a lower-tree. In the first level subband (see Fig. 1), if the eight coefficients in these blocks (2 blocks of 2×2 coefficients) 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 both 2×2 blocks have eighth insignificant coefficients and all their direct descendants are *LOWER_COMPONENT*, the coefficients in that blocks are labeled as *LOWER_COMPONENT*, increasing the lower-tree size.

As in the original LTW image encoder, when there is at least one significant coefficient in one of the two blocks of 2×2 coefficients 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 the coefficient is insignificant, and it has a significant descendant, the coefficient is encoded as *ISOLATED_LOWER*. However, if all descendants of a significant coefficient are insignificant (*LOWER_COMPONENT*), we use a special symbol indicating the number of bits needed to represent it and a superscript L (4^L).

Finally, in the second stage, subbands are encoded from the LLL_N subband to the first-level wavelet subbands and symbols computed in the first stage are entropy coded by means of an arithmetic encoder. Recall that no *LOWER_COMPONENT* is encoded. The value of significant coefficients and their sign are raw encoded.

3 Performance Evaluation

In this section we will compare the performance of our proposed encoder (3D-LTW) using Daubechies 9/7F filter for both spatial and temporal domain with the following video encoders:

– 3D-SPIHT [9]
– H.264 (JM16.1 version) (high quality profile) [1]
– MPEG-2 (ffmpeg-r25117) - GOP size 15, sequence type IBBPBBP... [8]
– x264 (mingw32-libx264 r1713-1 high quality profile) in both Inter and Intra mode [8]

The performance metrics employed in the tests are 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.

The test video sequences used in the evaluation are: Foreman (QCIF and CIF) 300 frames, Container (QCIF and CIF) 300 frames, News (QCIF and CIF) 300 frames, Hall (QCIF and CIF) 300 frames, Mobile (ITU D1 576p30) 40 frames, Station2 (HD 1024p25) 312 frames, Ducks (HD 1024p50) 130 frames and Ducks (SHD 2048p50) 130 frames.

It is important to remark that MPEG-2 and x264 evaluated implementations are fully optimized, using CPU capabilities like Multimedia Extensions (MMX2, SSE2Fast, SSSE3, etc.) and multithreading, whereas 3D-DWT based encoders (3D-SPIHT and 3D-LTW) are non optimized C++ implementations.

3.1 Memory requirements

In Table 1, memory requirements of different encoders under test are shown. The 3D-LTW encoder running over a GOP size of 16 frames uses up to 6 times less memory than 3D-SPIHT, up to 22 times less memory than H.264 for QCIF sequence resolution and up to 6 times less memory than x264 which is an optimized implementation of H.264, for small sequence resolutions. It is important to remark that 3D-SPIHT keeps the compressed bit-stream of a 16 GOP size in memory until the whole compression is performed, while encoders like MPEG-2, H.264, 3D-LTW and x264 output the bit-stream inline. Block based encoders like MPEG-2 require less memory than the others encoders, specially at high definition sequences.

3.2 R/D performance

Regarding R/D, in Table 2 we can see the R/D behavior of all evaluated encoders for different sequences. As shown, x264 is the one that obtains the best results for sequences with high movement, mainly due to the exhaustive ME/MC stage included in this encoder, contrary to 3D-SPIHT and 3D-LTW that do not include any ME/MC stage. The R/D behavior of 3D-SPIHT and

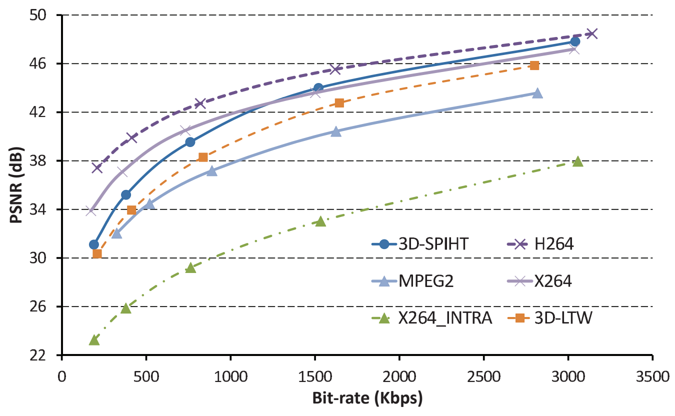
Format/ Codec	**QCIF**	**CIF**	**ITU-D1**	**Full-HD**
H264	35824	86272	227620	489960
x264	10752	18076	36600	178940
MPEG-2	4696	6620	**9164**	32820
3D-SPIHT	10152	34504	118460	645720
3D-LTW	**1611**	**6390**	20576	123072

Table 1 Memory requirements for evaluated encoders (KB)

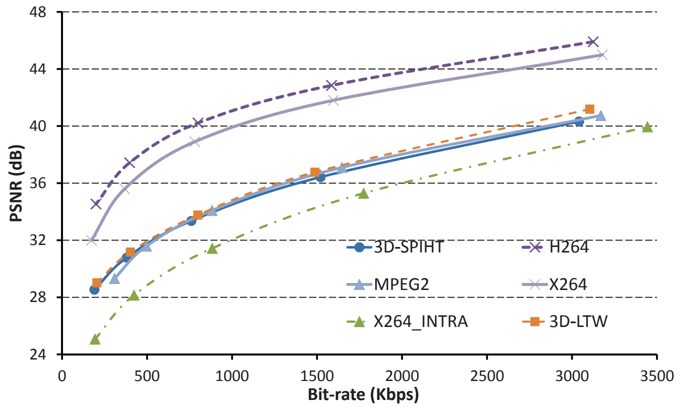
Codec/Bit rate Kbps/dB	**x264**	**MPEG-2**	**x264 Intra**	**3D-SPIHT**	**3D-LTW**
			Foreman (CIF)		
3040	**44.99**	40.74	39.95	40.32	41.38
1520	**41.80**	37.10	35.29	36.42	36.67
760	**38.90**	34.09	31.43	33.35	33.42
380	**35.60**	31.59	28.15	30.78	30.68
190	**31.99**	29.32	25.07	28.53	28.54
			Container (CIF)		
3040	47.20	43.59	37.97	**47.82**	46.54
1520	43.60	40.43	33.04	**43.99**	41.93
760	**40.50**	37.19	29.22	39.54	37.39
380	**37.09**	34.48	25.88	35.20	33.31
190	**33.89**	32.05	23.27	31.10	29.79
			Hall (CIF)		
3040	42.92	42.29	41.19	**44.68**	44.46
1520	40.55	39.89	36.60	**42.27**	41.66
760	38.94	37.95	31.89	**40.11**	38.93
380	37.25	35.95	27.32	**37.39**	35.43
190	**34.80**	33.59	23.88	33.56	31.90
			Mobile (ITU-D1)		
6400	**40.33**	37.82	35.56	38.24	38.86
3598	**38.82**	36.09	32.53	35.07	35.59
2100	**37.57**	34.37	30.12	32.53	32.69
1142	**35.51**	32.58	27.87	30.52	30.64
542	**31.82**	30.68	25.65	28.82	29.26
			Ducks (Full-HD) 50fps		
98304	37.34	**38.49**	36.26	37.77	36.07
49152	34.48	35.27	32.61	**35.39**	32.85
24576	32.46	32.28	29.16	**33.68**	31.49
12288	30.55	29.32	26.43	**31.63**	30.23
6144	28.47	27.82	24.19	28.99	**29.19**

Table 2 Average PSNR (dB) with different bit rate and coders

3D-LTW is similar for images with moderate-high motion activity, but for sequences with low movement, 3D-SPIHT outperform 3D-LTW, mainly due to the extra decomposition levels applied in high frequency subbands. Fig. 2 shows an example of this effect in two different sequences, one with low motion activity like Container and other with moderate motion activity like Foreman. Notice that the proposed 3D-LTW encoder improves the performance of the old-fashioned MPEG-2 inter video encoder. Also, it is worth to highlight the



(a) Container



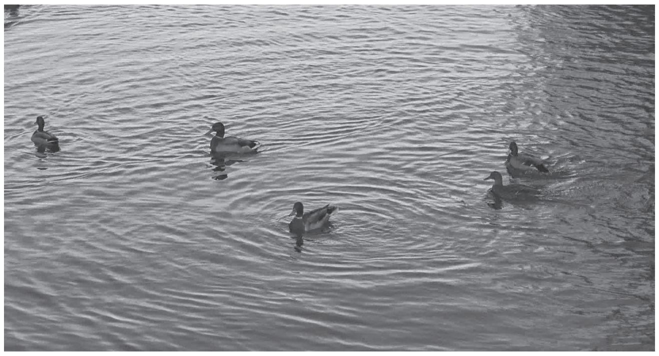
(b) Foreman

Fig. 2 PSNR (dB) for all evaluated filters for Container and Foreman sequences in CIF format

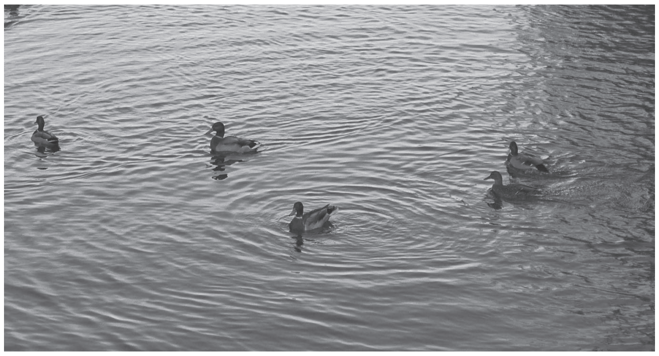
significant R/D improvement of both 3D-LTW and 3D-SPIHT over the x264 intra encoder (up to 11dB). This R/D improvement is accomplished by exploiting only the temporal redundancy among video frames when applying the 3D-DWT. It is also interesting the behavior of 3D-DWT based encoder for high frame rate video sequences like Ducks. As it can be seen all 3D-DWT based encoders have a similar behavior than the other encoders, even better than x264 in INTER mode.

3.3 Subjective Evaluation

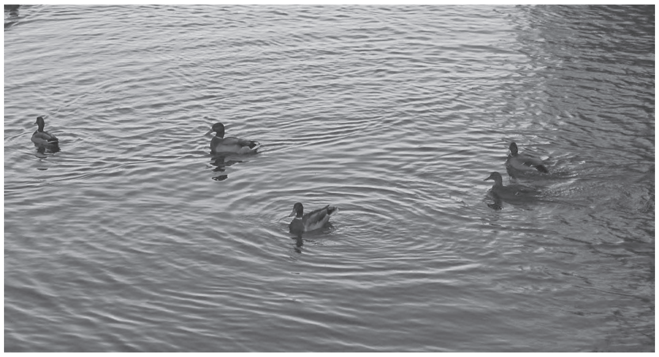
We have also performed a subjective evaluation of the proposed encoder. Fig. 3 and Fig. 4 show the 33^{rd} frame of the Ducks sequence in Full-HD format compressed at 13000 Kbps. As we can see, both 3D-LTW and x264 obtain the



(a) 29.21 dB



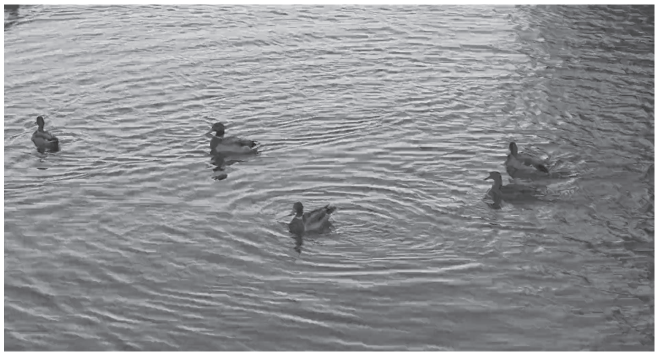
(b) Original

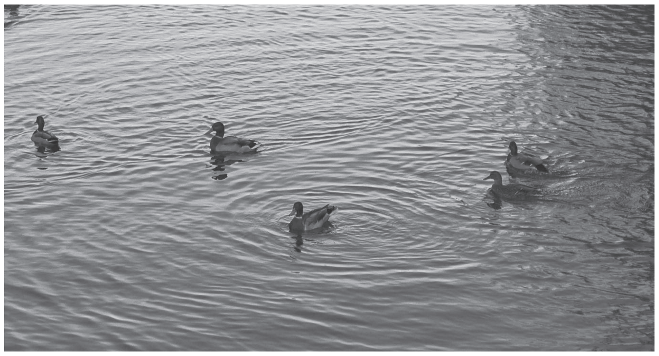


(c) 31.57 dB

Fig. 3 Subjective comparison between a) MPEG-2 and c) 3D-LTW for Ducks (Full-HD) at 13000 Kbps, frame # 33

best results. MPEG-2 obtain lower performance. Also, in Fig. 4, we can see the poor performance of x264 Intra in this frame where disturbing blocking artifacts appear. Its interesting to see the great behavior of 3D-LTW which


(a) 26.55 dB


(b) Original

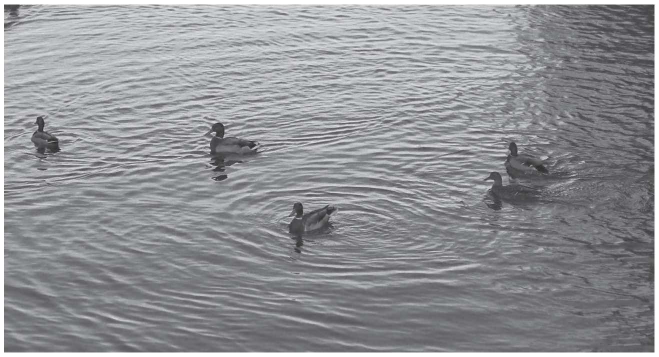

(c) 30.71 dB

Fig. 4 Subjective comparison between a) x264-Intra and c) x264 for Ducks (Full-HD) at 13000 Kbps, frame # 33

is even better than x264, even when no ME/MC is applied in the proposed encoder.

3.4 Coding/Decoding time

In Fig. 5 we present the total coding time (excluding I/O) of all evaluated encoders and for different sequence resolutions. As it can be seen, MPEG-2 encoder is the fastest one due to its block-based processing algorithm. Regarding 3D-DWT based encoders, the proposed encoder 3D-LTW is up to 7 times as fast as 3D-SPIHT and up to 6 times as fast as x264 encoder.

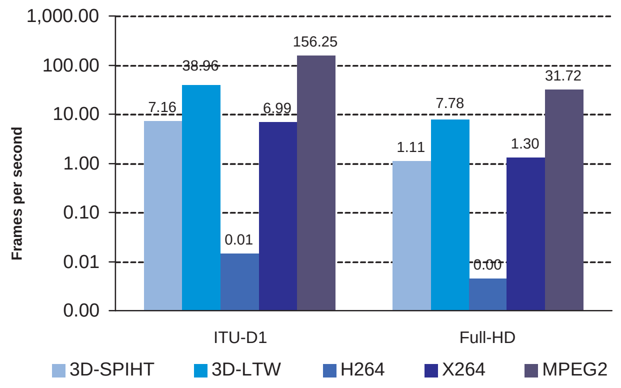


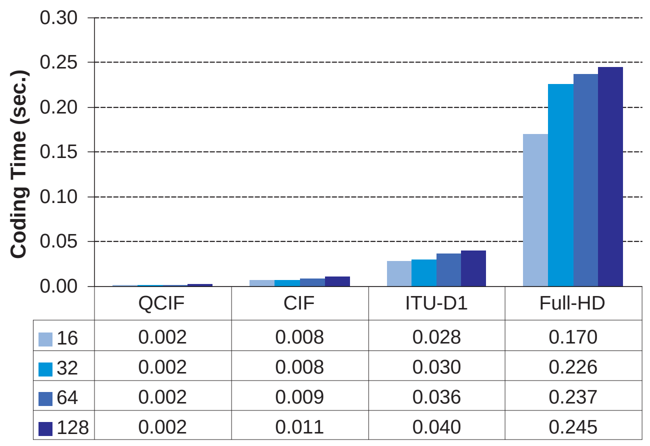
Fig. 5 Coding time in frames per second for all evaluated encoders

Also, in Fig. 6(a) we present the total coding time of a frame for different video sequence resolutions as a function of the GOP size for the 3D-LTW encoder. As it can be seen, for low resolution sequences there are near no differences in the total coding time, but for high resolution video sequences, the total coding time will increase up to 40% as the GOP size increases. Furthermore, its interesting to see that the time required to perform 3D-DWT stage ranges between 45% and 80% of the total coding time depending on the GOP size, as seen in Fig. 6(b). So, improvements in the 3D-DWT computation will drastically reduce the total coding time of the proposed encoder.

4 3D-DWT optimizations

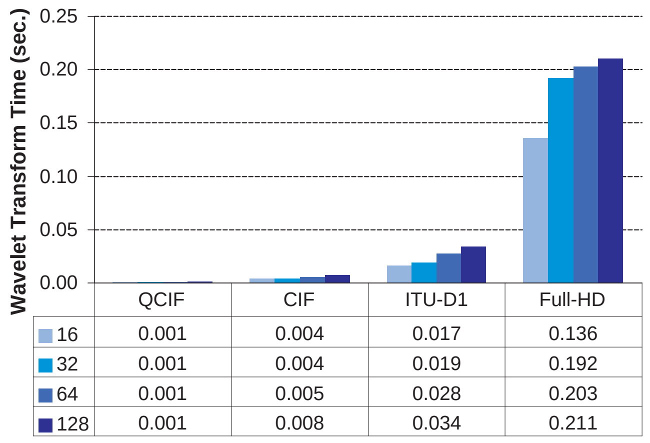
As 3D-DWT computation requires more than 45% and up to 80% of the total coding time in the proposed encoder, in this section we present several GPU based strategies to improve the 3D-DWT computation time.

Two different GPUs architectures are used in this work. The first one is a GTX280 which contains 240 CUDA cores with 1 GB of dedicated video memory. The other one is a laptop GPU (GT540M) with 96 CUDA cores and 2 GB of dedicated video memory. We can appreciate significant differences between both devices that will be reflected in the results shown in this section.



	QCIF	CIF	ITU-D1	Full-HD
■ 16	0.002	0.008	0.028	0.170
■ 32	0.002	0.008	0.030	0.226
■ 64	0.002	0.009	0.036	0.237
■ 128	0.002	0.011	0.040	0.245

(a) Total coding time



	QCIF	CIF	ITU-D1	Full-HD
■ 16	0.001	0.004	0.017	0.136
■ 32	0.001	0.004	0.019	0.192
■ 64	0.001	0.005	0.028	0.203
■ 128	0.001	0.008	0.034	0.211

(b) Wavelet time

Fig. 6 Total coding time and Wavelet transform time per frame of the 3D-LTW encoder for different video sequence resolutions and GOP sizes

The algorithm used to compute the 3D-DWT in the GPUs is illustrated in Fig. 7. Before the first computation step, image data must be transfered from host memory to the global memory of the device. We must transfer the number of frames indicated by the GOP size. As we increase the GOP size, more amount of global memory is needed in the GPU. All frames are stored in adjacent memory positions. In this way, the memory requirements for the GPU is $Width\times\ Height\times frames\times sizeof(float)$ bytes. As showed in Fig. 7, in this implementation we are using two memory spaces in the global memory of the GPU: one for the input data and other for the output data after applying the filtering process. In the first step, each thread computes the row convolution and stores the result in the output memory. For computing the second step,

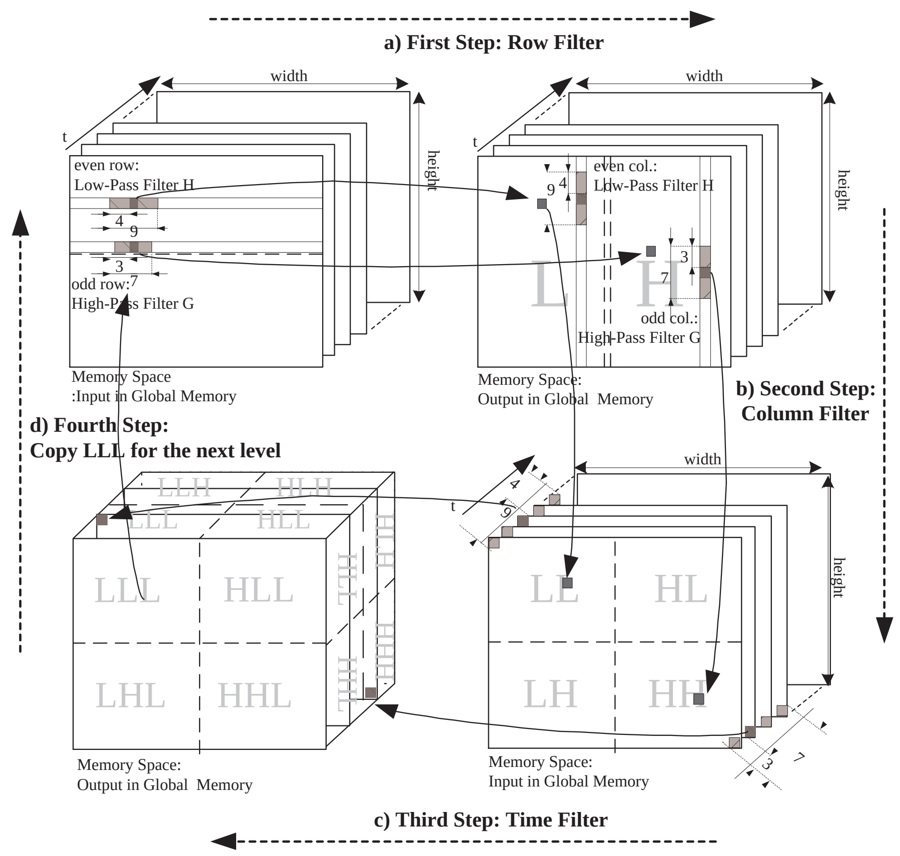
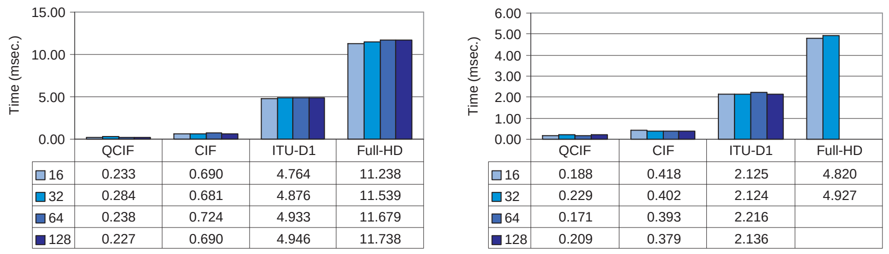


Fig. 7 Steps for Computing 3D-DWT in GPUs

the source data is now the output data obtained in the previous step, so it is not needed a copy of memory data for preparing this step. So, in the second step, the column filter is applied and the 2D-DWT is completed for each frame and after that, the output is again in the source space memory. After that, in the third step, a 1D-DWT is performed to consider the temporal dimension. At the last step, data must be transfered to the host memory to proceed with the next GOP. The first level 3D-DWT is performed in the output space memory and if we want compute a second level we must copy data from output to input space. Then, in the second level only half of the resolution (LLL subband) must be computed, iterating the same steps that for the first level.



	QCIF	CIF	ITU-D1	Full-HD
■ 16	0.233	0.690	4.764	11.238
■ 32	0.284	0.681	4.876	11.539
■ 64	0.238	0.724	4.933	11.679
■ 128	0.227	0.690	4.946	11.738

(a) GT540M

	QCIF	CIF	ITU-D1	Full-HD
■ 16	0.188	0.418	2.125	4.820
■ 32	0.229	0.402	2.124	4.927
■ 64	0.171	0.393	2.216	
■ 128	0.209	0.379	2.136	

(b) GTX280

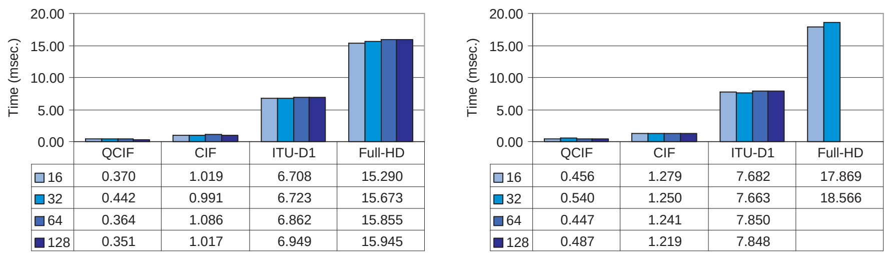
Fig. 8 GPU computational wavelet transform times per frame over GT540M and GTX280 for different video resolutions

4.1 Performance evaluation of the GPU 3D-DWT

In this section we present the performance evaluation of our GPU-based 3D-DWT algorithm in terms of computational and memory transfer times and the speed-ups obtained when compared to the CPU sequential algorithm. We present results for both previously mentioned GTX280 and GT540M platforms.

In Fig. 8 we present the computational times for both GPU platforms used in this work and for two different video sequence resolutions considering GOP sizes varying from 16 to 128 and computing four wavelet decomposition levels. As shown in Fig. 8, for ITU-D1 video frame resolution, the GTX280 is 2.3 times as fast as the GT540M regarding the GPU computational time. This is mainly due to the greater number of cores available in the GTX280 (2.5 times more cores). Moreover, in Fig. 12(a) we compare computational times in GPU shown in Fig. 8, versus the times needed to compute the wavelet transform in CPU, shown in Fig. 6(b)), for a GOP size of 32 and we obtain an speed-up around 16.6 in GT540M, and 38 in GTX280. Computational times for Full-HD resolution over GTX280 are not available due to global memory constraints.

However, only computational time has been considered in this analysis. In Fig. 9, we show total times including transfer times between host memory and GPU memory. We must notice that these times including transfer times are higher than the ones showed in Fig. 8, being 1.3 in GT540M and 3.73 in GTX280. The global computational time including the memory transfer time is lower in the GT540M than in the GTX280 due to the significantly lower memory transfer time, thanks to a second generation of PCI Express bus which improves data transfers. As shown, data transfer between device memory and host memory introduce a significant penalty when using GPUs for general puporse computing. Comparing times from Fig. 9 with the measured times in CPU, note that we continue obtaining a good speed-up of 12 in GT540M and over 10 in GTX280 as shown in Fig. 12(a).



	QCIF	CIF	ITU-D1	Full-HD
■ 16	0.370	1.019	6.708	15.290
■ 32	0.442	0.991	6.723	15.673
■ 64	0.364	1.086	6.862	15.855
■ 128	0.351	1.017	6.949	15.945

(a) GT540M

	QCIF	CIF	ITU-D1	Full-HD
■ 16	0.456	1.279	7.682	17.869
■ 32	0.540	1.250	7.663	18.566
■ 64	0.447	1.241	7.850	
■ 128	0.487	1.219	7.848	

(b) GTX280

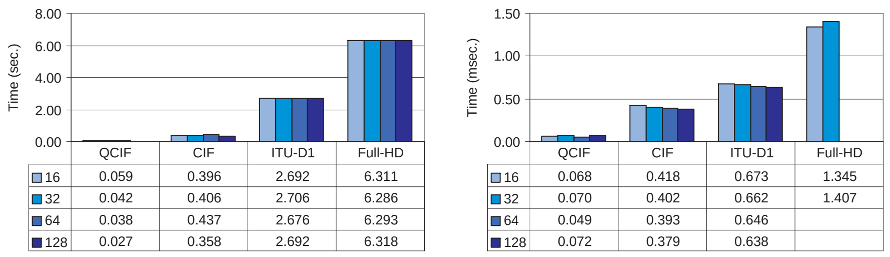
Fig. 9 GPU Computation wavelet transform and transfer times per frame over GT540M and GTX280 for different video resolutions

4.2 Memory access optimization

The previously presented algorithm uses the global memory to store both source and output data in wavelet computation. A reasonable speed-up (13) has been obtained with high video resolutions. However, we can achieve better performance if we compute the filtering steps from the shared memory. A block of the frame (row/column or temporal array) can be loaded into a shared memory array with BLOCKSIZE pixels. The number of thread blocks, NBLOCKS, depends on BLOCKSIZE and video frame resolution. We must note that around the loaded video frame block there is an apron of neighbor pixels that it is also required to load in the shared memory in order to properly filter the video frame block. We can reduce the number of idle threads by reducing the total number of threads per block and also using each thread to load multiple pixels into shared memory. This ensures that all threads are active during the computation stage. Note that the number of threads in a block must be a multiple of the warp size (32 threads on GTX280 and GT540M) for optimal efficiency. To achieve better efficiency and higher memory throughput, the GPU attempts to coalesce accesses from multiple threads into a single memory transaction. If all threads within a warp (32 threads) simultaneously read consecutive words then single large read of the 32 values can be performed at optimum speed.

In this new approach, each row/column/temporal filtering stage is separated into two sub-stages: a) the threads load a block of pixels of one row/column/temporal array from the global memory into the shared memory, and b) each thread computes the filter over the data stored in the shared memory and stores the results in the global memory. We must not forget about the cases when a row or column processing tile becomes clamped by video frame borders, and initialize clamped shared memory array indices with correct values. In this case, threads also must load in shared memory the values of adjacent pixels in order to compute the pixels located in borders.

In Fig. 10, we evaluate the new algorithm for computing the wavelet transform using the shared memory. As we can see, both GPUs have reduced con-

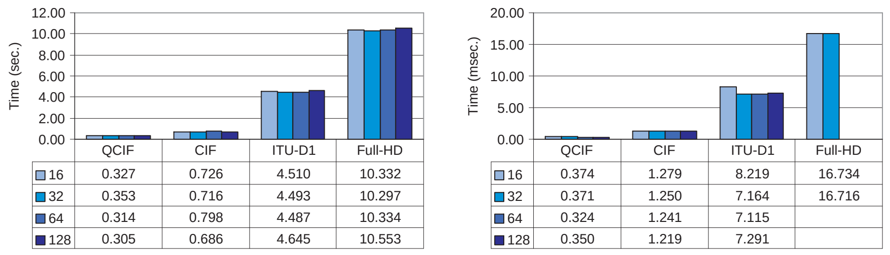


	QCIF	CIF	ITU-D1	Full-HD
▨ 16	0.059	0.396	2.692	6.311
▨ 32	0.042	0.406	2.706	6.286
▨ 64	0.038	0.437	2.676	6.293
■ 128	0.027	0.358	2.692	6.318

	QCIF	CIF	ITU-D1	Full-HD
▨ 16	0.068	0.418	0.673	1.345
▨ 32	0.070	0.402	0.662	1.407
▨ 64	0.049	0.393	0.646	
■ 128	0.072	0.379	0.638	

(a) GT540M (b) GTX280

Fig. 10 GPU computational wavelet transform times per frame over GT540M and GTX280 for different video resolutions using optimized shared memory access



	QCIF	CIF	ITU-D1	Full-HD
▨ 16	0.327	0.726	4.510	10.332
▨ 32	0.353	0.716	4.493	10.297
▨ 64	0.314	0.798	4.487	10.334
■ 128	0.305	0.686	4.645	10.553

	QCIF	CIF	ITU-D1	Full-HD
▨ 16	0.374	1.279	8.219	16.734
▨ 32	0.371	1.250	7.164	16.716
▨ 64	0.324	1.241	7.115	
■ 128	0.350	1.219	7.291	

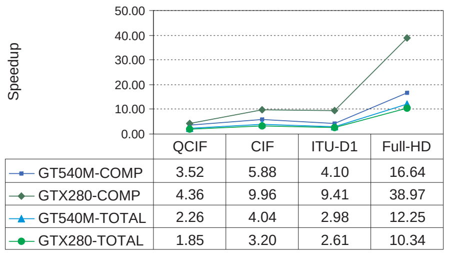
(a) GT540M (b) GTX280

Fig. 11 GPU computational wavelet transform and transfer times per frame over GT540M and GTX280 for different video resolutions using optimized shared memory access

siderably the execution time. As an example, for Full-HD video resolution and with a GOP size of 32, we have improved the computational time up to 1.83x and up to 3.5x for GT540M and GTX280 respectively when compared to the previous algorithm that uses the global memory. Fig. 12(b) compares the times showed in Fig. 10 with times needed to compute the 3D-DWT in CPU, and it shows an speed-up over 30 in GT540M and 136 in GTX280. However, transfer times between host and GPU memory are too high to notice this improvement in total times over GPU. As shown in Fig. 11, total times increase considerably, being the computational wavelet time only the 8% of the total time needed to transfer and compute wavelet. In Fig. 12(b), we show the speed-ups of our proposal taking into account the transfer times. Speed-ups of 19 and 11 were obtained with GT540M and GTX280 GPUs, respectively.

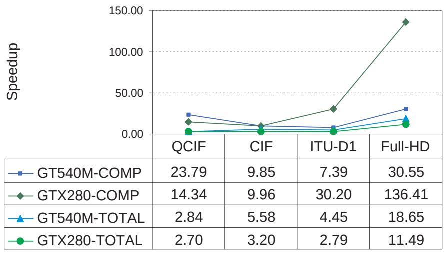
5 Performance evaluation of the proposed encoder using GPUs

After analyzing the performance of the GPU 3D-DWT computation, we will present a comparison of the proposed encoder against the other encoders in terms of coding delay.



	QCIF	CIF	ITU-D1	Full-HD
GT540M-COMP	3.52	5.88	4.10	16.64
GTX280-COMP	4.36	9.96	9.41	38.97
GT540M-TOTAL	2.26	4.04	2.98	12.25
GTX280-TOTAL	1.85	3.20	2.61	10.34

(a) Using Global memory access



	QCIF	CIF	ITU-D1	Full-HD
GT540M-COMP	23.79	9.85	7.39	30.55
GTX280-COMP	14.34	9.96	30.20	136.41
GT540M-TOTAL	2.84	5.58	4.45	18.65
GTX280-TOTAL	2.70	3.20	2.79	11.49

(b) Using optimized shared memory access

Fig. 12 Speed-ups over GT540M and GTX280 for different video resolutions and a GOP size of 32

In Fig. 13 we present the total coding time (excluding I/O) in frames per second of all evaluated encoders and for different sequence resolutions for a quality of 30dB. Now, our proposal uses the GPU to compute the 3D-DWT stage. As it can be seen, 3D-LTW encoder is the fastest one being up to 3.2 times on average as fast as the non-GPU version of the proposed encoder, up to 22 times as fast as 3D-SPIHT and up to 19 times as fast as x264 which is a fully optimized version of H.264. After the GPU optimization of the 3D wavelet transform stage, the proposed encoder is able to compress a Full-HD sequence in real time. Remark, that the optimizations performed are due only to GPU strategies while other encoders like x264, H263, MPEG-2 and MPEG-4 are fully optimized implementations, using CPU capabilities like Multimedia Extensions (MMX2, SSE2Fast, SSSE3, etc.) and multithreading.

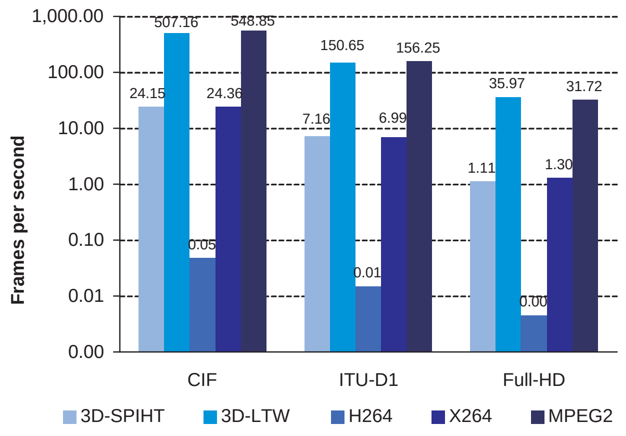


Fig. 13 Average coding time in frames per second for all evaluated encoders after GPU optimization of the proposed encoder

Although, the GPU version of the 3D-LTW encoder has been speeded up to 3.2 times, now, the bottleneck in the global encoder is the coding stage after computing the 3D-DWT transform, specially at low compression rates, where there are lots of significant coefficients to encode. Several strategies could be performed in order to speed up even more the proposed encoder, like overlapping both GPU computation and memory transfer times, overlapping CPU processing times with GPU processing time, or using several GPUs to compute multiple 3D wavelet transforms from different GOPs.

6 Conclusions

In this paper we have presented the 3D-LTW video encoder based on 3D wavelet transform and lower trees with eight nodes. We have compared our algorithm against 3D-SPIHT, H.264, x264, and MPEG-2 encoders in terms of R/D, coding delay and memory requirements.

Regarding R/D, our proposal has a better behavior than MPEG-2. When compared to 3D-SPIHT, our proposal has a similar behavior for sequences with medium and high movement, but slightly lower performance for sequences with low movement like Container. However, our proposal requires 6 times less memory than 3D-SPIHT. Both 3D-DWT based encoders (3D-SPIHT and 3D-LTW) outperforms x264 in Intra mode (up to 11 dB) exploiting only the temporal redundancy among video frames when applying the 3D-DWT. It is also important to see the behavior of 3D-DWT based encoders when applied to high frame rate video sequences obtaining even better PSNR than x264 in Inter mode.

In order to speed up our encoder, we have presented an exhaustive analysis of GPU memory strategies to compute the 3D-DWT transform. As we have seen, the GPU 3D-DWT algorithm obtains good speed-ups, up to 16 in the GT540M platform and up to 39 in the GTX280. Using these optimizations, the proposed encoder (3D-LTW) is a very fast encoder, specially for Full-HD video resolutions, being able to compress a Full-HD video sequence in real time.

The fast coding/decoding process and the avoiding of the use of motion estimation/motion compensation algorithms, makes the 3D-LTW encoder a good candidate for applications where the coding/decoding delay are critical for proper operation or for applications where a frame must be reconstructed as soon as possible. 3D-DWT based encoders could be an intermediate solution between pure Intra encoders and complex Inter encoders.

Although the proposed 3D-LTW encoder has been developed for natural video sequences where Daubechies 9/7F filter for the 3D-DWT stage has been widely used in the literature, other bi-orthogonal filters could be applied, depending on the final application. Even though longer filters capture better the frequency changes on an image, differences on R/D for natural images are negligible with respect to Daubechies 9/7F filter. This effect could be extended to the temporal domain case. However, longer filters introduce an increment on the DWT computation complexity because more operations per pixel must be performed, making the encoder slower. Obviously, if a longer filter is used in the DWT stage, the speed-up will be greater, because more operations per pixel will be performed in a parallel way.

As future work, we pretend to move other parts of the coding stage, like the quantization stage to the GPU to speed up even more the encoder. Furthermore, we pretend to overlap the CPU computation stage with the GPU computation of the 3D-DWT stage. Regarding quantization step over GPU, our first attempts shows that the 3D-DWT stage over GPU will be increased a 12% on average while the coding stage will be reduced a 17% on average, which makes our encoder even faster.

References

1. ISO/IEC 14496-10 and ITU Rec. H.264. Advanced video coding, 2003.
2. 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.
3. 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.
4. Y. Chen and W.A. Pearlman. Three-dimensional subband coding of video using the zero-tree method. In *Visual Communications and Image Processing*, volume Proc. SPIE 2727, pages 1302–1309, March 1996.
5. P.L. 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. J. Franco, G. Bernabé, J. Fernández, M.E. Acacio, and M. Ujaldón. The gpu on the 2d wavelet transform. survey and contributions. In *In proceedings of Para 2010: State of the Art in Scientific and Parallel Computing*, 2010.
7. V. Galiano, O. López, M.P. Malumbres, and H. Migallón. Improving the discrete wavelet transform computation from multicore to gpu-based algorithms. In *In proceedings of International Conference on Computational and Mathematical Methods in Science and Engineering*, 2011.
8. http://ffmpeg.arrozcru.org/autobuilds/blog/2010/09/14/ffmpeg-r25117-swscale-r32222-ok/. ffmpeg, September 2010.
9. B.J. Kim, Z. Xiong, and W.A. Pearlman. Very low bit-rate embedded video coding with 3D set partitioning in hierarchical trees (3D SPIHT), 1997.
10. B.J. Kim, Z. Xiong, and W.A. 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.
11. J. Oliver and M. P. Malumbres. Low-complexity multiresolution image compression using wavelet lower trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(11):1437–1444, 2006.
12. C.I Podilchuk, N.S. Jayant, and N. Farvardin. Three dimensional subband coding of video. *IEEE Tran. on Image Processing*, 4(2):125–135, February 1995.
13. Jang-Seon Ryu and Eung-Tea Kim. Fast intra coding method of h.264 for video surveillance system. *International Journal of Computer Science and Network Security*, 7(10):76–81, 2007.
14. A. Said and 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):243–250, 1996.

15. 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.
16. J.M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12), December 1993.
17. D. Taubman and A. Zakhor. Multirate 3-D subband coding of video. *IEEE Tran. on Image Processing*, 3(5):572–588, September 1994.
18. C. Tenllado, J. Setoain, M. Prieto, L. Pinuel, and F. Tirado. Parallel implementation of the 2d discrete wavelet transform on graphics processing units: Filter bank versus lifting. *Parallel and Distributed Systems, IEEE Transactions on*, 19(3):299 –310, march 2008.
19. Tien-Tsin Wong, Chi-Sing Leung, Pheng-Ann Heng, and Jianqing Wang. Discrete wavelet transform on consumer-level graphics hardware. *Multimedia, IEEE Transactions on*, 9(3):668 –673, april 2007.