Low-complexity wavelet-based scalable image & video coding for home-use surveillance

Low-Complexity Wavelet-Based Scalable Image & Video Coding for Home-Use Surveillance

Marijn J. H. Loomans, *Student Member*, IEEE, Cornelis J. Koeleman, and Peter H. N. de With, *Fellow*, IEEE

Abstract — *We study scalable image and video coding for the surveillance of rooms and personal environments based on inexpensive cameras and portable devices. The scalability is achieved through a multi-level 2D dyadic wavelet decomposition featuring an accurate low-cost integer wavelet implementation with lifting. As our primary contribution, we present a modification to the SPECK wavelet coefficient encoding algorithm to significantly improve the efficiency of an embedded system implementation. The modification consists of storing the significance of all quadtree nodes in a buffer, where each node comprises several coefficients. This buffer is then used to efficiently construct the code with minimal and direct memory access. Our approach allows efficient parallel implementation on multi-core computer systems and gives a substantial reduction of memory access and thus power consumption. We report experimental results, showing an approximate gain factor of 1,000 in execution time compared to a straightforward SPECK implementation, when combined with code optimization on a common digital signal processor. This translates to 75 full color 4CIF 4:2:0 encoding cycles per second, clearly demonstrating the real-time capabilities of the proposed modification.*[1]

Index Terms — Scalable, Image Compression, Wavelet Transforms, Embedded Systems, Image Coding.

I. INTRODUCTION

Real-world applications of scalable image and video coding include surveillance of rooms and home premises. Part of this application, is real-time compression using inexpensive cameras and remote viewing on hand-held battery-powered devices. Scalable coding based on wavelets [1], provides intrinsic scalability in quality, resolution and complexity and therefore matches well with home-use and hand-held applications. Wavelets are efficiently implemented using the lifting framework [2] and several integer-to-integer wavelets have been proposed [3]-[4] to support lossless transformation and enable efficient implementation on fixed-point arithmetic. Zhang [5] applied integer scaling factors to the 5/3 integer wavelet coefficients to maintain a lossless transformation and improve rate-distortion performance. The wavelet coefficients can effectively be coded using dedicated encoding algorithms, such as EZW [6], SPIHT [7], SPECK [8] and EBCOT [9], of which the latter is utilized in the JPEG2000 standard [10]. Despite the ongoing developments for wavelet transformation and efficient coding techniques, a coding standard has not been broadly employed in practice by a large group of users. This is explained by the relatively high implementation complexity and the absence of key applications. Both drawbacks have limited the wide-spread use in consumer electronics, especially in hand-held and battery-operated devices, as these devices have stringent boundaries on energy usage and thus computational complexity. Therefore, we have concentrated on efficient implementation techniques of this technology. Taking efficiency and power consumption as leading principles, we reconsider the design of such coding systems up to the algorithm level since this will yield a better performance. This paper introduces algorithmic improvements on specific places in the processing to obtain significant gains in implementation efficiency and power consumption. One of the initial processing steps is multi-level 2D dyadic wavelet decomposition, featuring an accurate low-cost integer wavelet implementation with lifting. An efficient embedded-systems implementation of this was discussed by the authors in [11]. The primary contribution of this paper is a modification of the SPECK codec, to make it more suitable for embedded applications and obtain a significant improvement in memory usage efficiency and avoid multiple coding iterations. This modification embodies splitting the SPECK algorithm in two stages, leading to the name of Two-Stage SPECK (TSSP). In TSSP, special care is taken in the quadtree partitioning, to facilitate encoding of non power-of-two image sizes. We furthermore include two extensions: the first is a highly scalable mode, which allows full scalability in all dimensions, without a need to decode any part of the bit stream. The second is an energy correction mode, that improves the rate-distortion performance of the 5/3 integer wavelet significantly. When all measures are included and code is further optimized, we show that it is possible to achieve real-time performance on a standard low-cost DSP.

The sequel of this paper is as follows. Section II introduces the dyadic wavelet decomposition and lifting implementation of the integer filters with energy correction. In Section III, we provide the two processing stages in the TSSP coding algorithm. Section IV presents two extensions to the TSSP

[1] M. J. H. Loomans and C.J. Koeleman are with VDG Security B.V., 2718 TA, Zoetermeer, The Netherlands (e-mail: marijn.loomans@vdg-security.com and rick.koeleman@vdg-security.com).

P. H. N. de With is with CycloMedia Technology B.V., 4181 AE, Waardenburg, The Netherlands. (e-mail: p.h.n.de.with@tue.nl).

M. J. H. Loomans and P. H. N. de With are also with the Electrical Engineering Department, Eindhoven University of Technology, 5600 MB, Eindhoven, The Netherlands.

Contributed Paper
Manuscript received 04/12/11
Current version published 06/27/11
Electronic version published 06/27/11.

implementation: a highly scalable mode and a special 5/3 energy correction mode. Experimental results are discussed in Section V and we conclude the paper in Section VI.

II. WAVELET-BASED IMAGE CODING

A. Multi-Level 2D Dyadic Wavelet Decomposition

In wavelet-based image coding, the input image is transformed into the wavelet domain by 2D separable wavelet filters. The result of the 2D wavelet transform consists of four frequency bands, commonly referred to as the *LL*, *HL*, *LH* and *HH* bands. The *LL* band represents the low-pass image in both horizontal and vertical direction and can be seen as a down-scaled version of the original image. The *HL*, *LH* and *HH* bands contain high-pass image information. The *LL* band still contains a large amount of spatial correlation and therefore, the 2D wavelet transform is applied to this band several times, up to a predetermined number of iterations. The number of iterations can be limited by the resolution of the *LL* band, the required number of bits to store the coefficients in memory and/or the impact on coding performance. The dyadic wavelet decomposition for three levels is represented in Figure 1(a).

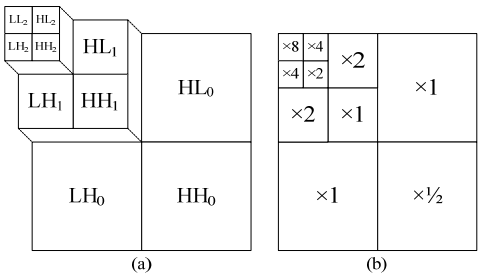


Fig. 1. (a) Three level dyadic wavelet decomposition with (b) accumulated energy correction factors for the 5/3 integer wavelet.

B. Lifting Framework and Integer Wavelets

Using the lifting framework [2], the wavelet can be implemented with less multiplication and add operations than the straightforward FIR implementation. Figure 2 shows the lifting implementation.

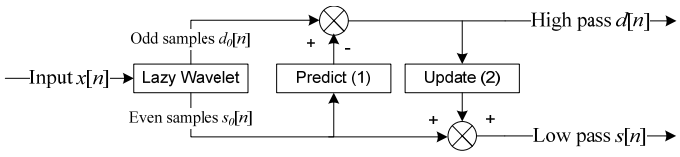


Fig. 2. Lifting implementation of wavelet filtering.

Input samples are split into odd and even samples, after which the even samples are filtered and used to adjust the odd samples in the predict step. Likewise, the odd samples are then filtered and used to adjust the even samples in the update step. The figure shows a single set of update and predict steps, which is sufficient to implement the 5/3 wavelet filter. For more complex wavelet filters, more sets of update and predict steps should be cascaded. For example, the 9/7-F integer filter has two sets of update and predict steps. The multiplication factor can be seen as a form of energy correction of the low- and high-pass outputs. The 5/3 integer wavelet can be implemented by a single lifting iteration, without any output multiplication. Following the notation from [4], the predict step is defined by (1), and the update step by (2), which are specified as

$$d[n] = d_0[n] - \left\lfloor \frac{s_0[n+1] + s_0[n]}{2} \right\rfloor, \tag{1}$$

$$s[n] = s_0[n] + \left\lfloor \frac{d[n] + d[n-1]}{4} + \frac{1}{2} \right\rfloor. \tag{2}$$

In these equations, $d[n]$ and $s[n]$ are the high-pass and low-pass output, respectively. Parameter $d_0[n]$ is the result of the lazy wavelet, and represents the odd samples of the input $x[n]$, hence $d_0[n] = x[2n+1]$. Similarly, $s_0[n]$ represents the even input samples $x[2n]$.

As can be derived from these formulas, the multiplication factors in the predict and update steps are powers of two, and therefore the 5/3 wavelet can be implemented with adders and simple bit-shift arithmetic, thereby eliminating multiplier arithmetic completely. This makes the 5/3 integer wavelet a perfect candidate for fixed-point embedded implementations, where complexity is a key design constraint.

Both the 5/3 and 9/7-F integer wavelet are fully reversible and therefore lossless. Note that there is no energy correction or scaling factor utilized for these integer wavelets, so that they retain perfect reconstruction.

C. Multi-level 2D Energy Correction for 5/3 Int. Wavelets

To balance the energy between the low- and high-pass output of the 5/3 wavelet, we utilize a scaling factor K at the end of the lifting process. For the standard 1D wavelet, a good factor would be $\sqrt{2}$ for the low-pass output, and consequently, the reciprocal value for the high-pass output. These factors are well-defined real numbers and thus not suited for fixed-point implementation.

Since the wavelet transform is separable, two 1D wavelet filtering steps are performed in succession. We combine both 1D scaling factors into four alternative 2D scaling factors *after* the 2D transform has completed. Using the initial 1D scaling factor of $K=\sqrt{2}$, this results in the 2D scaling factors of 2, 1, 1 and ½ for the LL, HL, LH and HH bands respectively. These 2D scaling factors can be efficiently implemented in fixed-point arithmetic with bit-shifting.

The dyadic wavelet decomposition consists of multiple filtering operations of the 2D wavelet transform at several scales, as discussed in Section II-A. This computing structure enables the extension of the 2D energy corrections to the multi-level framework, which results in the scaling factors of Figure 1(b), identical to those proposed by Zhang [5].

III. TWO-STAGE SPECK (TSSP)

Wavelet coefficients can be effectively coded using zero-tree coders, such as EZW [6] and SPIHT [7]. These coders utilize the property that the high-pass wavelet coefficients are

sparsely distributed non-zero coefficient values, which also have inter-band correlations. The zero-tree can be used to efficiently code large regions of zeros, which occur at most bit-levels of the wavelet tree. Since wavelets maintain a certain form of spatial information in their frequency analysis (unlike the FFT), the correlation between bands of the same resolution and/or correlation across resolutions can be exploited. An example of such correlation phenomena is created by a sharp signal transient that introduces significant wavelet coefficients across different frequency scales at approximately the same spatial location.

SPECK [8] uses a different approach, where only the spatial correlations between wavelet coefficients within a single frequency band are utilized, by using quadtree partitioning. At first glance, it seems unfortunate that the correlation of coefficients between frequency and/or resolution bands is not utilized. However, this has the benefit of *local data utilization* employing cache memories more efficiently, which speeds up the algorithm significantly. Moreover, it offers the possibility of parallel implementations, in which different frequency bands and resolutions can be processed simultaneously. Although SPECK does not utilize the cross-frequency band correlations and cross-resolution correlations, it still outperforms EZW and SPIHT in terms of coding efficiency.

However, SPECK is still very data-dependent in decision making, testing the significance of pixels sequentially. To reduce this data-dependency, we propose to split SPECK in two stages, as visualized in Figure 3: one *data-independent* stage and one *data-dependent* stage, leading to the name Two-Stage SPECK (TSSP). The first data-independent stage has a fixed pre-determined access pattern, and supplies data to a temporary buffer (bottom of Figure 3) with information about the significance of quadtree partitioning elements. The first stage has strong optimization possibilities and it can easily be split in several independent processing areas having identical computing structures and thus equal computation times, thereby enabling parallelization on multi-core architectures.

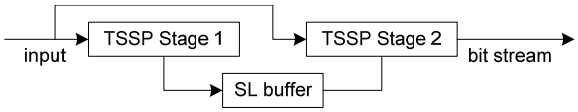


Fig. 3. Schematic representation of the two-stage processing in TSSP.

The second, or data-dependent stage does not need to investigate the significance of individual pixels, but utilizes the temporary buffered data from the first stage. From this buffer, significance information of the quadtree is used to create sorting information for significant coefficient regions, and to skip large insignificant coefficient regions. Similar to the first stage, the computing structure can also be equalized for areas, but the computation depth will contain data-dependencies, leading to variations in computation time.

A. TSSP Quadtree Partitioning

To facilitate arbitrary image sizes, special care is taken in TSSP to provide quadtree partitioning for non power-of-two image sizes. At a certain point in the partitioning process, when the width or height of blocks becomes odd, they cannot be split into 4 blocks of equal size, and a decision needs to be made on how the block is partitioned. TSSP supports two quadtree partitioning methods. First, a partition called top/left floor, which means that the size of the top-left partition is determined with the floor operator ($\lfloor x \rfloor$). The second partition is called top/left ceiling, and is based on the ceiling operator ($\lceil x \rceil$). These two partitioning methods are visualized in Figure 4 for an image of 1920×1088 resolution, for which the quadtree partitioning becomes irregular at level 6, where the block size is 30×17. For the wavelet transform, the same rounding of the size of low- and high-pass bands should be used. The top/left floor partitioning is recommended for regular use, so that the low-pass band of the wavelet will have the smallest size, thereby improving compression efficiency. However, for shape-adaptive wavelets, it is desirable to generate a larger low-pass band to preserve DC information, and the top/left ceiling partitioning is preferred.

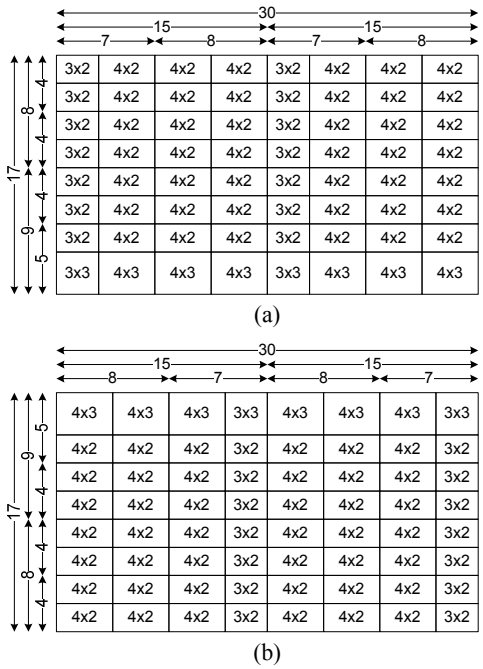


Fig. 4. Size-invariant quadtree partition for a 30×17 region, based on (a) floor and (b) ceiling functions for calculation of the top/left partition size. The 30×17 region is the 6th decomposition step of a 1920×1088 image.

The depth of the quadtree $N_{QTdepth}$ is defined by the number of quadtree partitioning steps that can be made until the remaining area of coefficients has a width or height of 2 or 3 coefficients. This can be calculated iteratively using the following pseudo code, with *QTdepth* denoting the depth of the quadtree:

$QTdepth = 0$
while $width \geq 4$ **and** $height \geq 4$ **do**
  $width \Leftarrow \lfloor width/2 \rfloor$, $height \Leftarrow \lfloor height/2 \rfloor$
  $QTdepth \Leftarrow QTdepth + 1$
end while.

The number of elements in the quadtree N_{QTelem} with l quadtree levels can be calculated recursively by

$$N_{QTelem}(l) = 4 \cdot N_{QTelem}(l-1) + 1, \quad (7)$$

with $N_{QTelem}(0) = 1$ and $l > 0$. For several common video resolutions, the quadtree depth $N_{QTdepth}$ and quadtree size $N_{QTelem}(N_{QTdepth})$ is displayed in Table I. For wavelet coefficients of `int16` precision, the significance level of the quadtree nodes can be stored in a `uint4` element. The resulting required buffer size for the whole significance level buffer is also displayed in Table I.

TABLE I
SIZE OF TSSP QUADTREE AND SIGNIFICANCE LEVEL BUFFER.

Resolution	$N_{QTdepth}$	N_{QTelem}	*SL* buffer size
1920×1088	9	349,525	171 kB
1920×1080	9	349,525	171 kB
1280×720	8	87,381	42.7 kB
704×576	8	87,381	42.7 kB
704×480	7	21,845	10.7 kB

B. TSSP Stage 1

At the first stage of the TSSP, the Significance Level (*SL*) of every node of the quadtree is stored in a buffer. Since the *SL* of a block is equal to the maximum of its four quadrants, we generate this buffer from the quadtree leafs up to the trunk. The buffer is organized such that the first element contains the *SL* of the whole image, and the second element the *SL* of the top-left quadrant, etc. By starting the calculations at the bottom-right of the image and progressing in a reversed Morton-order, we generate the *SL* buffer backwards. This ordering process is visualized in Figure 5 for a quadtree of depth 3 with 85 elements. The *SL* of the bottom-right 4 leafs is calculated first, after which the *SL* of the node can be calculated, indicated in the figure by white dots. Once all four nodes at that level are calculated, the *SL* of the parent node can be calculated, indicated in the figure by the gray dots. Finally, the *SL* of the whole image is calculated, referring to the black dot in the middle of the figure. This quadtree consists of 64 leafs, 16 level-2 nodes (white dots), 4 level-1 nodes (gray dots) and 1 level-0 node (black dot), leading to a total of 64+16+4+1=85 quadtree elements.

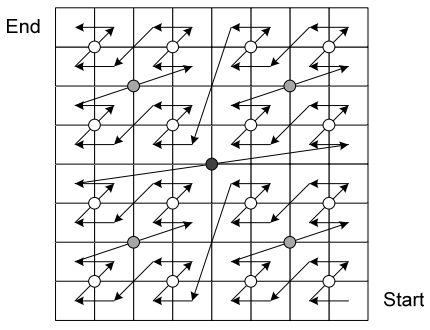


Fig. 5. Reversed Morton-order scanning in Stage 1 of the TSSP.

Processing in Stage 1 of the TSSP can be easily split over multiple cores. For example, the image can be split in four quadrants, and for each quadrant, the *SL* buffer is calculated in parallel. Each computing thread is assigned to an individual independent image region. When all regions are completed, the first value of each of the four buffers is used to calculate the *SL* of the whole image, and this value and the four buffers are cascaded to form the final *SL* buffer. Parallel processing can be implemented at any desired level, as processing can be split in 4, 16, 64, etc. blocks.

C. TSSP Stage 2

In the second stage of the TSSP, the *SL* buffer created at the first stage is used to make data-dependent coding decisions. Based on the *SL* level of a node in the quadtree, it is determined if this node is significant and should be partitioned further, or if it should be skipped. Individual coefficients do not need to be observed, since only information from the *SL* buffer is used for decision making.

The order in the *SL* buffer is designed in such a way that the reading in Stage 2 is always in the forward direction. As long as the blocks and their partitions are significant, the next value from the *SL* buffer is read. Once an insignificant block is encountered, the whole quadtree below this block is insignificant by definition, and will be skipped. The values in the *SL* buffer that represent this insignificant part of the quadtree can be skipped as well, and the number of *SL* values to skip (Δ_{index}), can be calculated directly from the current level in the quadtree ($N_{QTlevel}$) by the following expression

$$\Delta_{index} = \sum_{l=0}^{N_{QTlevel}} (4^l) - 1 = \frac{4^{N_{QTlevel}+1} - 1}{3} - 1. \quad (8)$$

Sorting and refinement data is generated for all bit-planes in parallel for each block and coefficient coding step. When blocks are split or skipped, sorting data is generated, and when a leaf of the quadtree is reached, individual coefficients are encoded, thereby generating sorting and refinement data. As data is generated for multiple bit-planes at once, we require temporary sorting and refinement buffers for each of the bit-planes. These buffers are ordered at the end of the coding stage to create the final progressive-quality or progressive-resolution bit stream. For `int16` wavelet coefficients, we require 15 sorting and 15 refinement buffers, as the sign is stored in the sorting buffer of the top bit-plane. If lossy coding is allowed, buffers for the lower bit-planes can be omitted.

The encoding quality is adjusted by the minimum level of significance of wavelet coefficients, indicated by the Bit-Level Reduction parameter *BLR*. Any wavelet coefficient with an absolute value smaller than 2^{BLR} is considered insignificant. Blocks and coefficients are encoded as follows.

Blocks encoding starts by reading its *SL* from the buffer created in Stage 1. If the block is considered significant ($SL \geq BLR$), a '1' is written to the sorting buffer for the bit-plane that has become significant, and a '0' in the sorting buffers for

each bit-plane above that, to indicate their insignificance at those bit-plane levels. Afterwards, the block is split into 4 quadrants according to the TSSP partitioning scheme, and the process repeats itself, using the next value in the *SL* buffer. If the block is considered not significant (*SL* < *BLR*), a '0' is written to the sorting buffer for all bit-planes, and the underlying tree is skipped. The skip in the *SL* buffer is calculated using the current quadtree depth using (8).

Individual coefficients are encoded when a significant block cannot be split into 4 quadrants, which occurs when we reach a leaf of the quadtree, e.g. the 3×2 top-left block in Figure 4(a). All coefficients in the block are encoded row-by-row, starting from the top-left. The individual coefficient encoding process is explained with an example using a coefficient value of 12,289, and visualized in Figure 6. The value of 12,289 has its first significant bit at bit-plane level 13, and therefore a '1' is written to the sorting buffer for bit-plane 13, followed by a '0' indicating the positive sign. For each of the sorting buffers above, a '0' is written, indicating the coefficient is not yet significant for those levels. For bit-plane levels 0—12, the refinement bits are written to the respective refinement buffers.

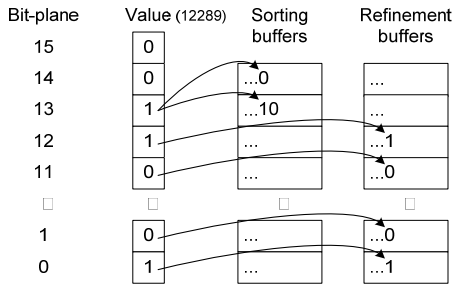


Fig. 6. Individual coefficient encoding process in Stage 2 of the TSSP for an example `int16` coefficient value of 12,289.

After all frequency bands of a single wavelet level are coded, the separate sorting and refinement buffers are ordered. They can be simply cascaded to generate a progressive-quality bit stream identical to the original SPECK bit stream, or they can be placed in separate data blocks for each resolution and bit-plane to facilitate highly scalable coding, which will be further elaborated in the following section.

IV. TSSP Extensions

A. Highly Scalable (HS) Mode

The basic TSSP bit stream only provides fine-grain quality scalability by truncation of the bit stream, and can be decoded using a regular SPECK decoder. To facilitate full scalability in resolution and quality, and to enable the use of the TSSP parser and decoder, the highly scalable TSSP bit stream consists of separate data blocks, each with the bit stream of a particular bit-plane and resolution level. If this Highly Scalable (HS) mode is used, the TSSP decoder can be constructed using the same principles and benefits as the TSSP encoder, e.g. parallel processing of separate data blocks.

For the HS mode, additional header information is included, which indicates the maximum and minimum bit-planes for each resolution level, and the number of bits in each data block, for both the sorting and refinement bits. Alignment of the data blocks is also included, with standard alignment at Byte level, to avoid difficult sub-Byte aligned copy operations. Alternative alignment configurations can be used as well, such as larger alignments that match memory bus width, at the cost of a slightly less efficient bit stream.

The TSSP bit stream can be organized in a progressive-quality or a progressive-resolution order. For the progressive-quality order, data blocks of bit-planes of each resolution layer are stored consecutively, followed by the data blocks for all resolution levels of the next bit-plane. For the progressive-resolution order, data blocks for all bit-planes of one resolution layer are stored consecutively, followed by the bit-planes of the next resolution layer.

Scalability is achieved through the use of the TSSP parser, which prunes and reorders the TSSP bit stream at any time after encoding, to create a bit stream with a desired quality, resolution and progression order. This parser only utilizes information from the header, and does not need to decode the payload data from the data packets. As a result, the parser only needs to create a new header, and reorder the bit stream using simple and efficient memory copy operations. In a network environment, data blocks can also be assigned different Quality-Of-Service (QOS) levels, thereby enabling graceful quality/resolution degradation in case of network congestion.

B. 5/3 Energy Correction (53EC) Mode

TSSP can be extended with energy correction for the well-known 5/3 integer wavelet. The 2D energy correction factors of Section II-C for the 5/3 wavelet are powers of two. Instead of correcting the wavelet coefficients, we alter the bit-plane truncation of the scalable TSSP codec to achieve a similar result. With this codec modification, we can use the original 5/3 integer wavelet, and achieve better lossy coding performance, while still supporting lossless coding.

Figure 7 shows the data order of the regular scalable TSSP codec for a 3-level dyadic decomposition, with frequency-band labels identical to those used in Section II-A. The encoded data for each frequency band and bit-plane level is

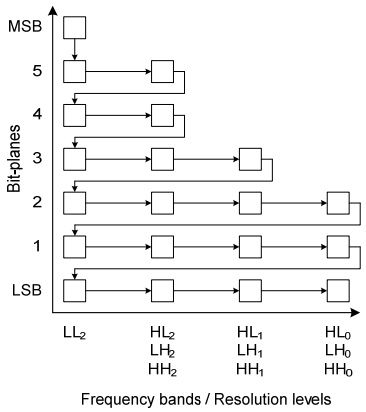


Fig. 7. Visualization of regular data block order in the TSSP bit stream.

visualized by the blocks, and the order in the bit stream by the arrows between the blocks. In the 5/3 energy correction mode, we modify the data order in the TSSP codec to match the 2D energy correction visualized in Figure 1(b). In the regular TSSP, the *LH*, *HL* and *HH* bands carry equal weight, and their bit streams are combined in a single data block. For the proposed 2D energy correction, we need to apply different corrections to the *LH* and *HL* bands, than used for the *HH* band. Therefore, in the modified data order, two separate data blocks occur: one for the *LH* and *HL* bands, and one for the *HH* band. The new data order, based on the additional data blocks and energy correction, is visualized in Figure 8.

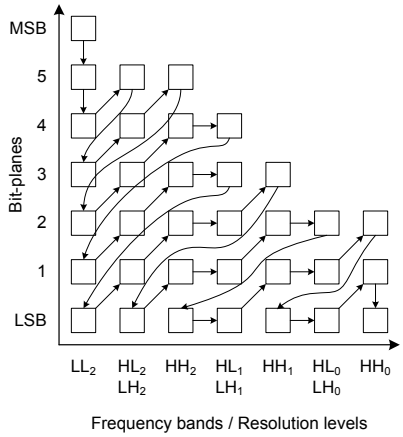


Fig. 8. Visualization of the TSSP bit stream order with codec-based energy correction, with data blocks per band and bit-plane.

Figure 9 shows the 5/3 energy correction mode data order, but now stretched vertically to visualize the effective energy correction of the blocks with factors ×8, ×4, ×2 and ×½.

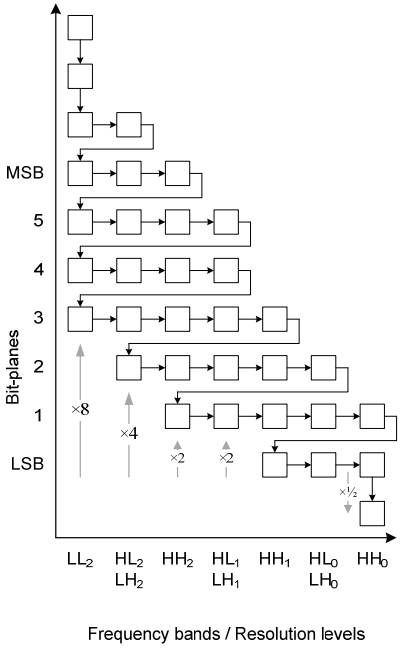


Fig. 9. Visualization of TSSP bitstream with codec-based energy correction, vertically stretched to clarify amount of 2D correction.

These factors are identical to the energy correction shown in Figure 1(b). The energy correction is the same for the HH_1 band and the LH_2 and HL_2 bands, but it should be noted that it is not allowed to combine them into a single combined band, since they originate from different spatial resolution levels. Furthermore, additional descriptive information has to be included for the extra blocks, which will slightly increase the size of the bit stream.

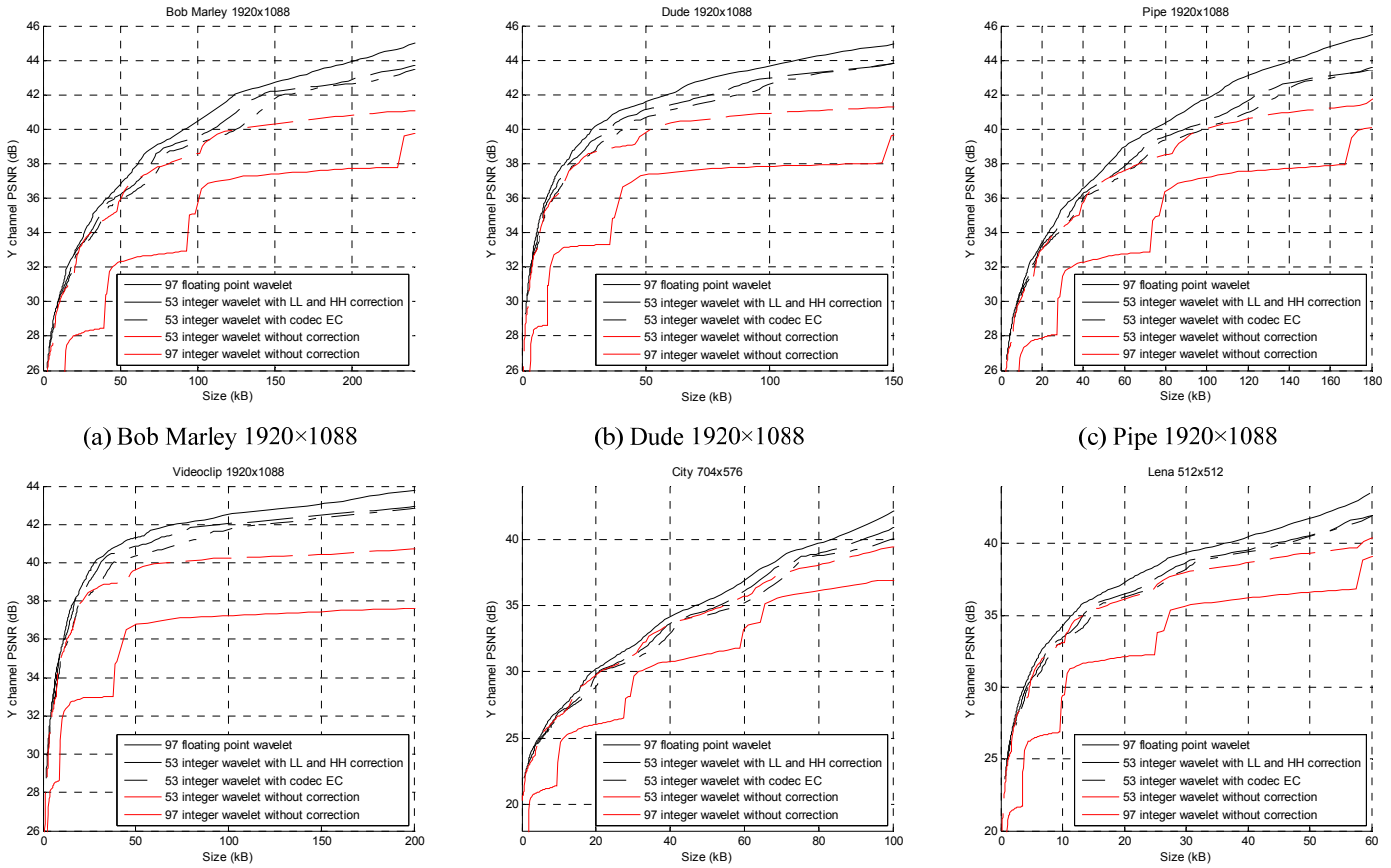
V. Experimental Results

Experiments have been conducted on the set of raw test images depicted in Figure 10, using a YCbCr 4:2:0 color standard. We prefer to utilize 6 levels of dyadic wavelet decompositions, and for full-HD images we are required to enlarge the standard images slightly, due to limitations in the utilized SPECK codec implementation. This enlargement is achieved by mirroring top and bottom image rows. As discussed before, the TSSP codec is capable of encoding arbitrary image sizes.

Fig. 10. Raw images used for experiments with various resolutions.

A. Lossy Performance Evaluation

Figure 11 shows the rate-distortion curves for several images with diverse resolutions and using various lossy and lossless wavelet transforms. The curves are generated using the proposed TSSP codec in the Highly Scalable (HS) mode, and encoded to (near-)lossless quality. The bit stream was then truncated using the TSSP parser for a range of rate points, and decoded with the TSSP decoder. For the lossy wavelets, the 9/7 floating-point and the 5/3 integer wavelet with Energy Correction (EC) are used. For the lossless wavelets, the 9/7-F integer wavelet and the 5/3 integer wavelet are applied, the latter with and without codec-based EC.



(a) Bob Marley 1920×1088 (b) Dude 1920×1088 (c) Pipe 1920×1088

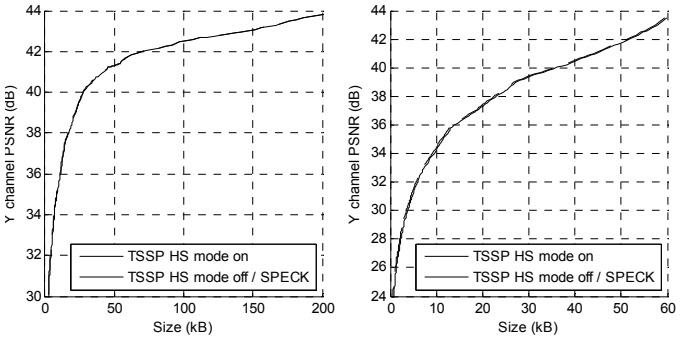
(d) Videoclip 1920×1088 (e) City 704×576 (f) Lena 512×512

Fig. 11. Rate-distortion curves for the 9/7 floating point wavelet and the 5/3 and 9/7 fixed point wavelets with various methods of Energy Correction (EC) for the (a) Bob Marley, (b) Dude, (c) Pipe, (d) Videoclip, (e) City and (f) Lena test images.

For all images, the lossy 9/7 floating-point wavelet yields the best performance, followed by the lossy 5/3 integer wavelet with energy correction applied to the *LL* and *HH* bands during wavelet calculation. For the lossless wavelets, the 5/3 integer wavelet with energy correction performed in the codec provides the best results, which are close to the lossy 5/3 integer wavelet, followed by the 9/7-F integer wavelet. The small loss in coding efficiency for the lossless 5/3 integer wavelet with codec-based energy correction, is achieved with the additional benefit of retaining perfect reconstruction. From these curves, it is also clearly visible that the 5/3 wavelet without any form of EC is impractical for lossy image coding, as it yields a quality degradation of up to 5 dB.

A performance difference exists between applying energy correction in the wavelet and in the codec. This is partly explained by the increase in bit-stream length when using codec-based energy correction, due to the inclusion of extra header information. The remaining performance loss is likely due to the different incomplete wavelet coefficient rounding in the decoder, followed by the inverse wavelet transform, with and without internal energy correction.

Figure 12 shows the rate-distortion curves for two images at the highest and lowest resolution, with and without the HS mode activated. Without the HS mode activated, the TSSP bit stream is identical to the SPECK bit stream and therefore, their curves are interchangeable. For TSSP with the HS mode activated, we observe a slight quality degradation at the same rate, or a small rate increase at the same quality, which is explained by the additional header information, and the Byte-alignment for all data blocks. For larger images and higher rates, the relative performance difference becomes smaller, as the number of extra alignment bits becomes insignificant compared to the number of bits in the larger data blocks. The same applies to the required header bits for each data block.



(a) Videoclip 1920×1088 (b) Lena 512×512

Fig. 12. Rate Comparison between TSSP with Highly Scalable (HS) mode on and off for the (a) Videoclip, (b) Lena test images, using the 9/7 DWT.

B. Lossless Performance Evaluation

Table II shows the compressed data sizes for the whole set of test images generated by the TSSP codec with and without Highly Scalable (HS) mode and for the SPECK codec. Since the bit stream for TSSP without HS mode and the bit stream for SPECK are identical, their lossless compression factors are identical as well. The bit stream of TSSP using the highly scalable mode includes additional header information and alignment bits, increasing the data sizes by only 0.1%.

TABLE II
LOSSLESS PERFORMANCE COMPARISON USING THE 5/3 INTEGER WAVELET FOR TSSP WITH AND WITHOUT HIGHLY SCALABLE MODE, AND SPECK.

Image	TSSP with HS	TSSP without HS	SPECK
Videoclip	1103 kB (2.8×)	1102 kB (2.8×)	1102 kB (2.8×)
Bob M.	1059 kB (2.9×)	1058 kB (2.9×)	1058 kB (2.8×)
Dude	967 kB (3.2×)	966 kB (3.2×)	966 kB (3.2×)
Pipe	915 kB (3.3×)	915 kB (3.4×)	915 kB (3.4×)
Eye	764 kB (2.7×)	763 kB (2.7×)	763 kB (2.7×)
City still	275 kB (2.2×)	274 kB (2.2×)	274 kB (2.2×)
Crew still	220 kB (2.7×)	219 kB (2.7×)	219 kB (2.7×)
Lena	145 kB (2.7×)	144 kB (2.7×)	144 kB (2.7×)

To compare compression efficiency differences between the different wavelet options, we calculated the lossless compressed data sizes using the TSSP codec with highly scalable mode on and evaluated the following three cases: (1) using the 5/3 integer wavelet without Energy Correction (EC), (2) using the 5/3 integer wavelet with codec-based energy correction and (3) applying the 9/7-F integer wavelet. The results for these three cases are listed in Table III.

TABLE III
TSSP LOSSLESS PERFORMANCE EVALUATION FOR SEVERAL INTEGER WAVELETS AND EC OPTIONS AND HIGHLY SCALABLE MODE ACTIVATED.

Image	5/3 DWT, no EC	5/3 DWT, +EC	9/7-F DWT
Videoclip	1103 kB (2.8×)	1104 kB (2.8×)	1142 kB (2.7×)
Bob M.	1059 kB (2.9×)	1060 kB (2.9×)	1099 kB (2.8×)
Dude	967 kB (3.2×)	968 kB (3.2×)	1018 kB (3.0×)
Pipe	915 kB (3.3×)	916 kB (3.3×)	977 kB (3.1×)
Eye	764 kB (2.7×)	764 kB (2.7×)	792 kB (2.6×)
City still	275 kB (2.2×)	276 kB (2.2×)	282 kB (2.1×)
Crew still	220 kB (2.7×)	220 kB (2.7×)	226 kB (2.6×)
Lena	145 kB (2.7×)	145 kB (2.7×)	145 kB (2.7×)

We observe that the 5/3 integer wavelet without any form of energy correction has the best lossless coding performance, but the difference with codec-based energy correction is negligible at only 0.1%. This small difference is due to the small extra overhead caused by the split of the high-pass data into two data blocks, which inevitably increases overhead in header information and alignment bits. The 9/7-F integer wavelet shows a slightly worse lossless performance of 3.5% compared to the 5/3 integer wavelet.

C. Implementation Complexity

TSSP was explicitly designed for enabling real-time implementation on embedded systems. To examine the performance gain compared to a straightforward SPECK implementation, we have ported the TSSP to the well-known DM642 digital signal processor. Optimization techniques such as SIMD (Single Instruction Multiple Data) and DMA (Direct Memory Access) have been utilized to achieve maximum computational and functional parallelism.

The SL buffer is placed in the Level-2 cache, together with a ping-pong style buffer, used for input image caching using DMA in Stage 1. For Stage 2, temporary sorting and refinement bit-stream buffers for each of the bit-planes are placed in the Level-2 cache, which are written to external memory when full. These buffers are used to improve single bit writing routines and have the size of a single `uint32`.

Table IV shows the cycle counts for a fully optimized TSSP implementation, running at a clock speed of 600 MHz. The cycle counts include memory transfers and stalls, and are averaged over 1,000 frames of a typical surveillance type video. The video is processed in YCbCr 4:2:0 color space.

TABLE IV
COMPUTATIONAL COMPLEXITY MEASUREMENTS FOR TSSP, FOR A 4CIF SURVEILLANCE SEQUENCE IN MCYCLES, AVERAGED OVER 1,000 FRAMES.

Function	Y channel	Cb channel	Cr channel
5/3 DWT	3.648 (26.65%)	1.076 (7.86%)	1.083 (7.91%)
Stage 1	1.531 (11.18%)	0.402 (2.94%)	0.402 (2.94%)
Stage 2	3.921 (28.64%)	0.740 (5.41%)	0.828 (6.05%)
Parser	0.037 (0.27%)	0.011 (0.08%)	0.012 (0.09%)
Total	**9.137 (66.74%)**	**2.229 (16.28%)**	**2.325 (16.98%)**

During these experiments, we have observed that the processor never waited for DMA transfers to finish, indicating that computations are currently the main bottleneck in this implementation, not the memory transfers.

From Table IV, we can read the encoding of a single 4CIF 4:2:0 color image requires 13.691 MCycles, of which 7.884 MCycles are required for the TSSP encoding, and the remaining 5.807 MCycles for the 5/3 integer wavelet transform. At 600 MHz, this translates to more than 75 TSSP encoding cycles per second, and more than 43 complete encoding cycles per second.

A straightforward implementation of SPECK without code optimization executed on the digital signal processor requires approximately 10 seconds for encoding a monochrome 4CIF image, while the optimized TSSP requires only 5.489 MCycles, translating to 9.15 ms at 600 MHz. Therefore, together with the discussed code optimizations, the TSSP reduces the execution time with a factor thousand, compared to the straightforward SPECK implementation.

VI. CONCLUSIONS

In this paper, we have studied scalable image and video coding for the surveillance of rooms and personal environments for deployment in inexpensive cameras and portable devices. To this end, we have proposed a Two-Stage SPECK (TSSP) coding algorithm, in which the coding of wavelet coefficients is split into two stages. The first data-

independent stage involves a buffer with Significance Level *SL* values for each of the nodes in the quadtree. The second data-dependent stage employs this *SL* buffer to make coding decisions for quadtree partitioning and the skipping of insignificant portions of the quadtree, both for all bit-planes in parallel.

A major advantage of the proposed processing is that the bit stream for all bit-planes is generated in parallel, thereby reducing the memory access by an order of magnitude. Furthermore, the TSSP is well suited for execution on multi-core systems, as it can be implemented in parallel at any quadtree depth. It features regular access patterns, and in the first processing stage, a fixed computational load.

The proposed extensions of our system generate other benefits, such as improved scalability of the bit stream. First, the Highly Scalable (HS) mode allows parsing of the bit stream without payload decoding, thereby creating a bit stream of any desired quality, resolution and bit stream order. Second, the Energy Correction (EC) mode retains the perfect reconstruction feature of the 5/3 integer wavelet, while significantly improving lossy coding performance with a negligible performance drop of only 0.1% at lossless coding.

To prove that the discussed design decisions make real-time performance possible, we have implemented the proposed TSSP on a commonly used digital signal processor. When all measures are included and code is further optimized, we achieve a gain factor of approximately 1000 in execution time compared to a straightforward SPECK implementation. This translates to a performance of more than 75 TSSP encoding cycles of 4CIF 4:2:0 color images per second, clearly demonstrating TSSP's real-time capabilities.

REFERENCES

[1] I. Daubechies, "Orthonormal bases of compactly supported wavelets," *Journal of Communictions on Pure & Applied Mathemetics*, vol. 41, pp. 909–996, Oct. 1988.

[2] W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets," *Applied and Computational Harmonic Analysis*, vol. 3, no. 2, pp. 186–200, 1996.

[3] A. R. Calderbank, I. Daubechies, W. Sweldens, and Boon-Lock Yeo, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis*, vol. 5, no. 3, pp. 332–369, 1998.

[4] M. Adams and F. Kossentni, "Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis," *IEEE Transactions on Image Processing*, vol. 9, no. 6, pp. 1010–1024, Jun 2000.

[5] Li-Bao Zhang and Ke Wang, "Embedded multiple subbands scaling image coding using reversible integer wavelet transforms," *Intelligent Multimedia, Video and Speech Processing, 2004. IEEE Proceedings of 2004 International Symposium on*, pp. 599–602, 2005

[6] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, no. 41, pp. 3445– 3462, 1993.

[7] A. Said and W. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *CirSysVideo*, vol. 6, no. 3, pp. 243–250, June 1996.

[8] A. Islam and W. A. Pearlman, "An embedded and efficient low complexity hierarchical image coder," *Visual Communications and Image Processing 1999*, vol. 3653, no. 1, pp. 294–305, Jan. 1999.

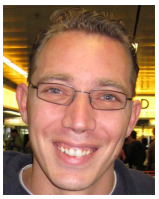
[9] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, 2000.

[10] C. Christopoulos, A. N. Skodras, and T. Ebrahimi, "Jpeg 2000 still image coding system: An overview," *IEEE Transactions on Consumer Electronics*, no. 46, pp. 1103–1127, 2000.

[11] M. J. H. Loomans, C. J. Koeleman, and P. H. N. de With, "Realtime multi-level wavelet lifting scheme on a fixed-point dsp for jpeg 2000 and scalable video coding," in *Digital Signal Processing, 2009 16th International Conference on*, 2009, pp. 1–6.

BIOGRAPHIES

Marijn J. H. Loomans (M'05) received his MSc. in Electrical Engineering from Eindhoven University of Technology (TU/e) and his Master of Technological Design in Embedded Systems from the National University of Singapore (NUS) in 2005 and 2004, respectively. Currently he is pursuing his Ph.D. at VDG Security BV and TU/e, where he is researching scalable video coding for embedded surveillance systems. At the ICCE 2011, he received the first place best paper award. His research interests include scalable video coding and embedded-system implementations.



Cornelis J. Koeleman graduated in Electrical Engineering from Utrecht Polytechnical College, the Netherlands in 2002. Pursuing a career in embedded processing and system design, he joined QEF where he worked on machine vision technology for industrial applications. He joined VDG Security in 2003, where he was involved in several European projects concerning video content analysis and embedded system design, such as CANTATA and VICOMO. During his career, acting as a system architect, he has supervised multiple university students in projects ranging from embedded system to video compression and content analysis.

Peter H. N. de With (F'06) graduated in Electrical Engineering from the University of Technology in Eindhoven and received his Ph.D. degree from the University of Technology Delft, The Netherlands in 1992. He joined Philips Research Labs Eindhoven in 1984, where he was until 1993 involved in several European projects on SDTV and HDTV recording. In this period, he contributed as a principal coding expert to the DV standardization for digital camcording. Between 1994-1997, he was leading the design of advanced programmable video architectures as a senior TV systems architect. In 1997, he was appointed as full professor at the University of Mannheim, Germany, at the faculty Computer Engineering, where he was heading the chair on Digital Circuitry and Simulation. Between 2000 and 2007, he was with LogicaCMG (now Logica) in Eindhoven as a principal consultant. In 2008, he joined CycloMedia Technology, The Netherlands, as vice-president for video technology. Since 2000, he is professor at the University of Technology Eindhoven, at the faculty of Electrical Engineering and leading a research group on 3D video coding and video analysis. He has written and co-authored over 300 papers on video coding, analysis, architectures and their realization. He has received several awards for IEEE CES Transactions papers, SPIE papers and company inventions. Mr. de With is a Fellow of the IEEE, program/technical committee member of the IEEE CES, ICIP and VCIP, a regular scientific board member or advisor to various companies, and of the Dutch Imaging school ASCII, and board member of various working groups.