Journal of Electronic Imaging

JElectronicImaging.org

Exploration of depth modeling mode one lossless wedgelets storage strategies for 3D-high efficiency video coding

Gustavo Sanchez
César Marcon
Luciano Volcan Agostini

Gustavo Sanchez, César Marcon, Luciano Volcan Agostini, "Exploration of depth modeling mode one lossless wedgelets storage strategies for 3D-high efficiency video coding," *J. Electron. Imaging* **27**(1), 013025 (2018), doi: 10.1117/1.JEI.27.1.013025.

Journal of Electronic Imaging 27(1), 013025 (Jan/Feb 2018)

Exploration of depth modeling mode one lossless wedgelets storage strategies for 3D-high efficiency video coding

Gustavo Sanchez,[a,b,*] César Marcon,[a] and Luciano Volcan Agostini[c]
[a]Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil
[b]Instituto Federal Farroupilha, Alegrete, Brazil
[c]Group of Architecture and Integrated Circuits—Universidade Federal de Pelotas, Pelotas, Brazil

Abstract. The 3D-high efficiency video coding has introduced tools to obtain higher efficiency in 3-D video coding, and most of them are related to the depth maps coding. Among these tools, the depth modeling mode-1 (DMM-1) focuses on better encoding edges regions of depth maps. The large memory required for storing all wedgelet patterns is one of the bottlenecks in the DMM-1 hardware design of both encoder and decoder since many patterns must be stored. Three algorithms to reduce the DMM-1 memory requirements and a hardware design targeting the most efficient among these algorithms are presented. Experimental results demonstrate that the proposed solutions surpass related works reducing up to 78.8% of the wedgelet memory, without degrading the encoding efficiency. Synthesis results demonstrate that the proposed algorithm reduces almost 75% of the power dissipation when compared to the standard approach. *© 2018 SPIE and IS&T* [DOI: 10.1117/1.JEI.27.1.013025]

Keywords: 3D-high efficiency video coding; depth modeling mode-1; wedgelets compression; memory size reduction; hardware design.

Paper 170711 received Aug. 23, 2017; accepted for publication Jan. 23, 2018; published online Feb. 22, 2018.

1 Introduction

The 3-D video services have become very popular in recent years, leading to the intensive investigation of innovative technologies to execute them effectively and efficiently. The Joint Collaborative Team on 3-D Video Coding Extension Development (JCT-3V)[1] has finished the standardization of 3D-high efficiency video coding (3D-HEVC),[2,3] which is state of the art in 3-D video coding. 3D-HEVC is an extension of the HEVC[4] standard, providing all features for 2-D coding and inserting new coding tools to explore redundancies among encoding views and depth maps characteristics.

The 3D-HEVC standard adopted the multiview video plus depth (MVD) data representation[5] to enhance the data compression rate.[6] In the MVD format, each texture view is associated with a depth map, which is captured to provide the geometrical information of the scene, according to the distance between the objects and the camera. The depth maps are composed of 8-bits samples, where darker shades of gray (values near 0) represent distant objects, whereas lighter shades of gray (values near of 255) express near objects. Figure 1 shows (a) a texture view and its associated (b) depth map extracted from MicroWorld video sequence.[7] The motivation for MVD usage is to reduce the bandwidth for a 3-D video transmission. Instead of requiring a transmission of many views, only a few views are transmitted, along with their associated depth maps. In the decoder, a dense set of virtual views can be synthesized by interpolating texture and depth data using techniques such as depth image-based rendering.[6]

One of the main challenges in the 3D-HEVC development process was the generation of high-quality encoded depth maps. Figure 1 shows that depth maps contain vast areas of constant values in image background or objects bodies and sharp edges in objects borders, while texture shows smooth transitions among the pixels. If only the conventional HEVC tools are used to encode the depth maps, then depth map edges tend to be smoothed. The HEVC tools use low-pass filters, which tend to smooth high-frequency edge regions. The view synthesis process demands a precise edge information since border distortions should introduce blocking artifacts, mosquito noise, and edge blurring.[8]

JCT-3V included new intraframe prediction tools to deal with the distinct characteristics between texture and depth maps. These tools include depth modeling modes (DMMs),[9] which are composed of DMM-1 and DMM-4, depth intra-skip (DIS),[10] and segment-wise direct component coding (SDC).[11] DMMs were designed to maintain a high quality in the encoded edges, DIS focuses on obtaining significant bitrate savings in homogeneous areas, and SDC is an alternative to the transforms followed by quantization (TQ) encoding flow.

The introduction of these tools in 3D-HEVC specification has allowed the 3D-HEVC to obtain higher efficiency when encoding depth maps. However, these tools increased the 3D-HEVC video encoder[12] and decoder complexity. The high complexity and the extremely intensive required data flow demand dedicated hardware designs to allow the processing of 3-D videos in embedded consumer electronic devices, such as cell phones and video cameras. The memory

*Address all correspondence to: Gustavo Sanchez, E-mail: gustavo.sanchez@acad.pucrs.br

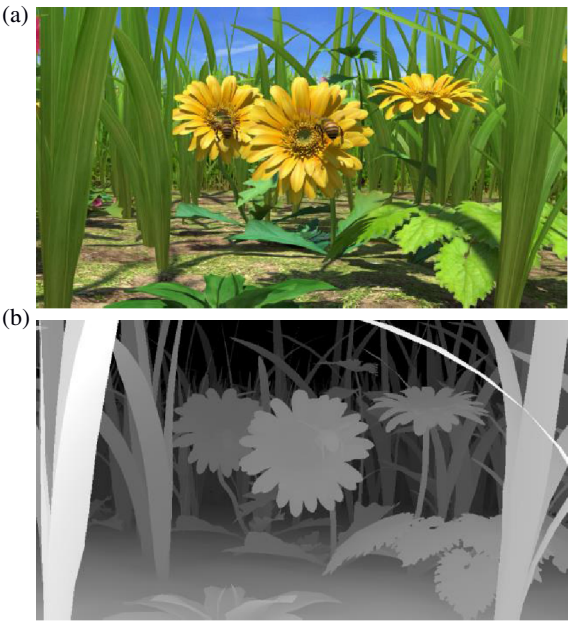


Fig. 1 (a) Texture view and its associated (b) depth map extracted from MicroWorld video sequence.[7]

system is a central challenge in embedded systems designs, including video coding applications. Memory is one of the main circuits responsible for energy consumption, and the memory communication is commonly a bottleneck of these systems.[13]

The DMM-1 encoder and/or decoder increases this memory problem since they require a high memory communication and storage as highlighted in our previous work[14] (see Sec. 4 for further details). This article presents a set of high-efficient algorithms to reduce the storage requirement for the DMM-1 wedgelets pattern, which can store all wedgelet patterns required by the 3D-HEVC. A hardware design of the algorithm with the best compression rate results is also presented.

The remainder of this article is organized as follows. Section 2 presents the related work. Section 3 describes the 3D-HEVC depth intraframe prediction algorithm, focusing on the DMM-1 encoder and decoder. Section 4 shows the main challenges faced in the DMM-1 implementation related to the memory issue along with our previous designed solution to reduce the memory area. Section 5 presents the algorithmic solutions proposed to reduce the memory required for wedgelets storage. Section 6 shows the algorithm evaluation results comparing the solutions proposed in this article with related works. Section 7 presents the hardware design for the best algorithm presented in Sec. 6 and discusses the impacts of this solution when integrates into a DMM-1 encoder or decoder. Finally, Sec. 8 renders the main conclusions of this work.

2 Related Works

The literature presents few works proposing hardware designs of DMM-1 tools, including such that require fewer memory accesses as proposed in this article. Our previous work[15] presented a complete DMM architecture that simplifies DMM-1 by removing the refinement step due to the high amount of memory required to store the refinement wedgelet patterns. Amish and Bourennane[16] designed an architecture for encoding bipartition modes in real time; however, this work does not consider the problem of DMM-1 wedgelets storage. To the best of our knowledge, these are the unique published hardware designs regarding the DMM-1 encoder.

The storage of all wedgelet patterns implies a DMM-1 hardware implementation with high area and power dissipation.[15] To overcome this problem, some works (e.g., Refs. 14 and 17) proposed approaches to reduce the area required for wedgelet patterns storage. Reference 17 proposed an architecture that reduces 27.8% of the memory size with 0.03% Bjontegaard Delta-rate[18] (BD-rate) loss. This result was obtained storing the entire 16×16 wedgelet patterns and dynamically downsampling these patterns for other block sizes. This technique can be applied only to the encoder hardware since it cannot create all available wedgelets required for decoding a 3D-HEVC compliant bitstream. Our previous work[14] proposes four lightweight approaches that reduce area consumption, number of accesses, and power dissipation of the memory used for the DMM-1 wedgelet patterns storage. All solutions proposed in Ref. 14 can generate the entire wedgelet set and, consequently, no BD-rate (encoding efficiency) impact is noticed when those approaches are applied.

3 Depth Maps Intraframe Prediction

The JCT-3V provides a 3D-HEVC Test Model (3D-HTM)[19] containing an implementation of the 3D-HEVC encoder and decoder. The DMM-1 encoding flow described in this work is based on version 16.2 of the 3D-HTM reference software. The 3D-HTM was also used in this article to extract the DMM-1 wedgelets patterns.

The 3D-HEVC depth maps intraframe prediction follows a similar encoding strategy as HEVC does, using the quadtree structure.[20] The encoding depth map frame is divided into several coding tree units (CTUs), which are encoded one by one. Each CTU should be split into four coding units (CUs), which can be recursively divided into smaller CUs until reaching a minimum size defined by the encoder. Moreover, a CU can also be split into prediction units (PUs), where intra- and interframe prediction algorithms are applied to encode it. An intraframe PU can assume sizes ranging from 4×4 to 64×64, using only quadratic sizes (i.e., blocks with the same width and height).

For a given encoding block, Fig. 2 shows a high-level block diagram of the depth maps intraframe prediction process, considering the execution of the 3D-HTM. The encoding process evaluates all the encoding possibilities considering their rate-distortion-cost (RD-cost), which is a function that evaluates both the image quality and the required number of bits to encode each block. The predictions generated by all encoding tools are evaluated, and the one with the lowest RD-cost is selected as the best encoding mode. This process is called rate-distortion optimization (RDO) and, since it needs information about the number of required bits and the reached quality, then all encoder process must be done for all candidate modes to define which one will be used to encode each block, implying in a highly complex process.

The DIS encoding tool[21] predictions are directly processed by the entropy coding to be evaluated by the RDO.

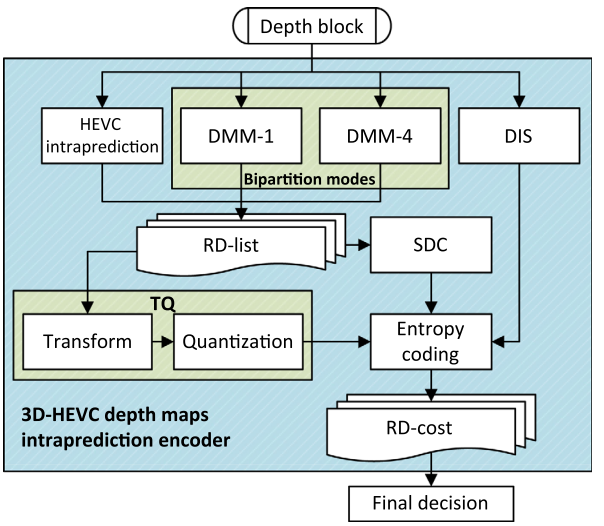


Fig. 2 Main blocks and flow of the 3D-HEVC depth maps intraframe prediction.[12]

The DIS tool defines four prediction modes: (i) vertical intraprediction mode, (ii) horizontal intraprediction mode, (iii) vertical single depth mode, and (iv) horizontal single depth mode. These prediction modes are properly described in Ref. 21.

The remaining encoding tools require the creation of a rate-distortion list (RD-list), where the predictions that will be fully evaluated by the RDO must be inserted. Two encoding tools can insert modes into this list: (i) HEVC intraframe prediction[22] and (ii) DMMs (including DMM-1 and DMM-4).

The HEVC intraframe prediction specifies 35 prediction modes, i.e., planar, DC, and 33 directional modes. However, instead of inserting all these modes into the RD-list, the reference software uses two heuristics to locally evaluate the modes with a higher probability to have a small RD-cost. These heuristics are rough mode decision (RMD) and most probable modes (MPMs). The RMD heuristic compares the available modes employing the sum of absolute transformed differences (SATD). Only the modes with the lowest SATDs are inserted into the RD-list. MPM gets the three MPMs, which can be the modes used in neighbor blocks or the modes more frequently selected, and inserts them into the RD-list. Therefore, only a few modes are evaluated by the intense calculation effort required by the full RD-cost computation.

DMM-1 segments the encoding block in two regions through a straight line called a wedgelet. All wedgelets partitions are predefined in the 3D-HEVC specification. The DMM-1 algorithm evaluates some of the wedgelets and inserts in the RD-list only the best wedgelet (i.e., the wedgelet that best-fit with the region evaluated in the depth map) found among the evaluated ones.

DMM-4 applies a technique called intercomponent prediction, which uses previously encoded information from the texture view during the depth maps prediction to find the best partition. The DMM-4 prediction dynamically creates a partition from the texture information consisting of arbitrary shapes or even disconnected regions. The dynamic partitioning increases the compression efficiency since the bitstream does not need to carry partitioning information.

After generating the DMM-1 and DMM-4 predictions, the prediction with best DMM-1 mode and the result of DMM-4 can be inserted into the RD-list. All modes inserted into the RD-list are evaluated using TQ and SDC.[23] Subsequently, the RD-costs are obtained using entropy coding in the TQ, SDC, and DIS results.

3.1 *Depth Modeling Mode-1 Encoding Algorithm*

As previously described, the DMM-1 algorithm segments the blocks into two regions using a wedgelet. Regarding a continuous space, as exemplified in Fig. 3(a), a straight-line equation can represent and store a wedgelet. However, regarding a discrete space, the wedgelet is stored in an array with $N \times N$ elements (N is the width and height of the block). Each element has the constant partition value (CPV) "0" or "1" to represent the regions "0" or "1," respectively.[24] Figure 3(b) exemplifies a binary pattern defining the regions "0" and "1" in an 8×8 block.

Figure 4 shows a high-level diagram of the DMM-1 encoding algorithm following 3D-HTM.[19] For a didactic purpose, this article splits DMM-1 flow into three stages: (i) main stage, (ii) refinement stage, and (iii) residue stage.

The main stage evaluates the initial wedgelet set (i.e., wedgelets that must be assessed before the refinement stage) and finds the best wedgelet pattern among the available ones, which are stored into the wedgelet memory (WMem).

In the prediction step, the DMM-1 algorithm starts computing the average values of all samples mapped into regions "0" and "1." Subsequently, the algorithm generates the predicted block mapping the average values obtained earlier, according to the wedgelet pattern.

Next, the distortion step applies a distortion criterion to identify the similarity of the predicted block compared to the encoded one. The 3D-HTM reference software uses the synthesized view distortion change[25] as the distortion criterion; however, real-time hardware implementations often use the sum of absolute differences (SAD) as distortion criterion due to its coding efficacy allied to the reduced area and power costs.[15] Finally, all distortions are compared, and the pattern with the lowest distortion defines the best wedgelet.

The refinement stage evaluates up to eight wedgelets around the selected one in the previous operation, i.e., with the smallest distortion among the patterns. Again, a predicted block is generated for each wedgelet pattern, and they are compared using a distortion criterion. The wedgelet

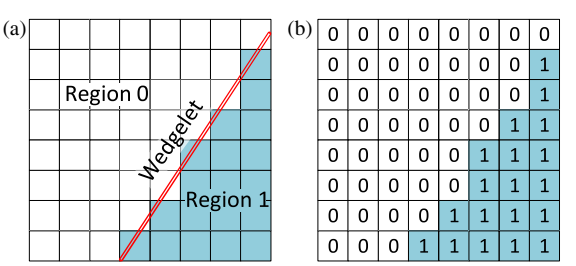


Fig. 3 Wedgelet segmentation model of a depth block with (a) a straight-line splitting regions "0" and "1" and (b) a discretization with constant values.[15]

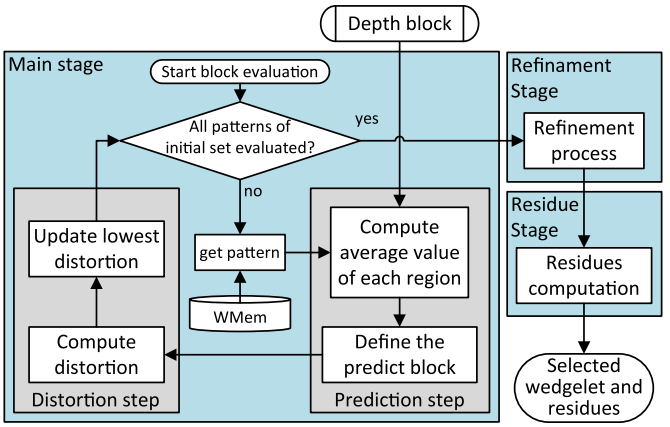


Fig. 4 Main blocks of the DMM-1 encoding algorithm.

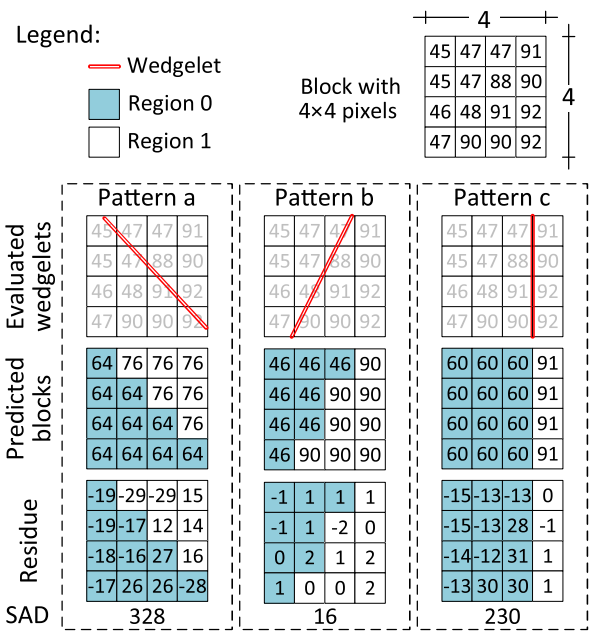


Fig. 5 Example of a 4×4 depth block encoding with the DMM-1 algorithm.[15]

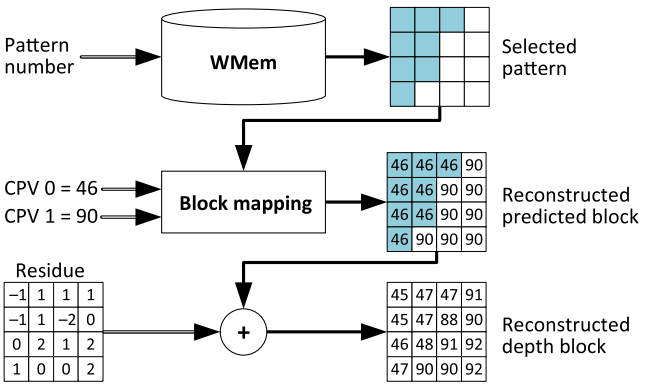


Fig. 6. Example of a 4×4 depth block decoding with the DMM-1 algorithm.

obtaining the lowest distortion among these evaluated wedgelets is elected as the best one. Finally, the residue stage subtracts the original block from the predicted one and adds this wedgelet into the RD-list.

Figure 5 exemplifies the DMM-1 main stage for a 4×4 depth block, along with the evaluation of three wedgelet patterns supposing SAD as the distortion criterion. The DMM-1 prediction process encodes the depth block sample according to the evaluated wedgelet (i.e., patterns a, b, and c). This procedure maps the pixels of the block sample in one of the two regions.

Subsequently, the prediction step computes the average value of all pixels in each region, obtaining the CPV of each region (e.g., the CPV of regions "0" and "1" of pattern a are 64 and 76, respectively). The distortion step calculates the difference between the original and the predicted depth block. The SAD of all patterns is attained, and finally, the pattern b is selected since it has the lowest SAD.

3.2 *DMM-1 Decoding Algorithm*

The DMM-1 decoding algorithm is much simpler than the encoding one. Figure 6 exemplifies the decoding of the 4×4 depth block employed in Sec. 3.1. For a given block size, the input of the DMM-1 decoding algorithm requires the number of the selected pattern, the CPV of each region, and the residue block.

While the encoding process requires the evaluation of the entire WMem or at least the initial set, the decoding process requires the access of only the wedgelet indexed by the received pattern number, retrieving the DMM-1 pattern selected in the encoding process. The CPV of each region is mapped into this DMM-1 pattern to reconstruct the predicted block. Finally, the residues are added to the predicted block, generating the reconstructed depth block.

4 Memory Challenges and Previous Works

This section aims to describe some motivational context behind the design of our algorithms. Section 4.1 describes the memory challenges imposed by DMM-1 when designing a system and applying this algorithm, demonstrating that memory reduction solution is necessary. Moreover, Sec. 4.2 details our previous work that implements lossless DMM-1 memory reduction techniques.

4.1 *Memory Challenges*

One of the bottlenecks in DMM-1 hardware design is the efficient storage of all wedgelet patterns due to the large number of allowed patterns. The 3D-HTM software stores all wedgelet patterns without compression using $N \times N$ bits for each pattern to fulfill the beginning of the encoder or decoder execution. 3D-HTM uses this strategy to keep high performance during software execution. However, this strategy is highly inefficient when the DMM-1 encoder or decoder is implemented in a dedicated hardware design.

Figure 7 shows that the DMM-1 implemented inside 3D-HTM requires 183,264 bits to store all wedgelet patterns of all block sizes. Notice that the wedgelet patterns of 32×32 blocks are not stored because they are obtained upscaling the wedgelets of 16×16 blocks. Furthermore, the DMM-1 encoder algorithm implemented in 3D-HTM requires frequent access to the wedgelet patterns, implying in a significant power dissipation caused by the memory accesses. Thus, essential requirements for dedicated DMM-1 encoder design are memory size reduction and efficient memory access.

A DMM-1 encoder implementation can remove wedgelets from WMem to reduce the total size. Examples of this approach are in (i) Ref. 17 that removes some wedgelets downsampling the wedgelets of bigger block sizes and (ii) Ref. 14 that excludes wedgelet refinements. However, the encoder wedgelets reduction degrades the coding efficiency, compromising the BD-rate. Additionally, the correct decoding of a 3D-HEVC bitstream demands the knowledge of all wedgelet patterns; therefore, these two solutions cannot be applied to the decoder.

4.2 *Proposed Solutions in Our Previous Works*

Our previous work[14] proposed four lossless techniques to reduce the WMem size required to store the wedgelet patterns: (i) first bit and change (FB&C), (ii) Huffman code, (iii) block change map (BCM), and (iv) line change map (LCM). Independently of the used encoding technique, the wedgelets are statically encoded, and once stored into the WMem, they remain the same during the entire system execution.

4.2.1 *First bit and change algorithm*

FB&C algorithm is grounded in the fact that DMM-1 divides each block into two and only two regions. Thus, DMM-1 does not represent interlaced ones and zeros in a single line leading to an inefficient storage model if $N \times N$ bits are stored per pattern. For example, in a 4×4 block, each line should never contain the pattern "0010" because no wedgelet can describe such line. It happens because the DMM-1 algorithm was designed for partitioning and encoding blocks into two regions using a straight line. Consequently, patterns containing more than one change of region in a line are forbidden. Thus, $N \times N$ blocks allow only $2 \times N$ lines in the DMM-1 patterns.

Table 1 FB&C coding for line patterns of a 4×4 block.

Allowed line	FB&C code
0111	000
0011	001
0001	010
0000	011
1000	100
1100	101
1110	110
1111	111

Table 1 presents all allowed lines for 4×4 blocks along with the respective FB&C codes. Note that half of the bits combination (patterns) is forbidden.

Figure 8(a) shows the FB&C technique that codifies each pattern copying the first bit content of the original pattern to the coded one. Let RB be the number of remaining bits required for coding a line of the $N \times N$ block, which can be obtained applying Eq. (1). Then, RB specifies in how many subsequent bits the coded line will change to the other region. Figure 8(b) shows two encoding examples of 8×8 block lines. The first example shows how to encode the "00000011" line with FB&C representation. The first bit "0" is just copied, and the encoding algorithm seeks for the first occurrence of "1," i.e., in the fifth position. Then, the value 5 (related to the fifth position) is stored after the first bit, completing the FB&C representation of the first example. The second example shows a line containing only a single region of "1s." Therefore, the FB&C codifies this line copying the first bit followed by the $N - 1$ value (7)

$$\mathrm{RB} = \log_2(N). \tag{1}$$

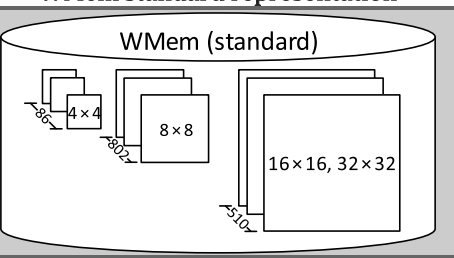
Block size	Wedgelets	Storage requirements (bits)	WMem standard representation
4×4	86	1376	
8×8	802	51,328	
16×16	510	130,560	
32×32	510	-	
Total		183,264	

Fig. 7 Number of wedgelets evaluated by the DMM-1 algorithm and the storage requirements.

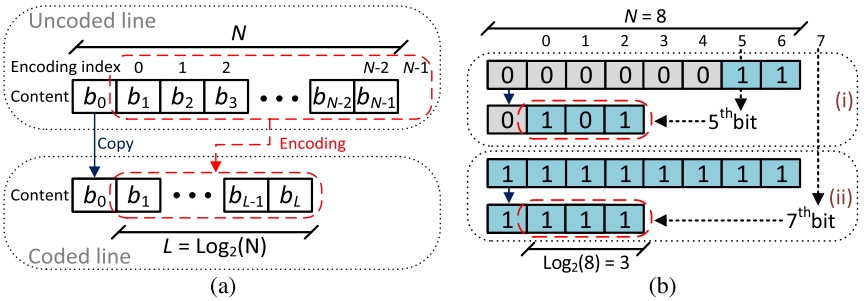


Fig. 8 (a) FB&C encoding model and (b) two pattern examples for 8×8 blocks.

4.2.2 *Huffman code*

Huffman code is a traditional algorithm in data compression field, which creates a prefix binary code tree with minimum expected code-word length.[26] In this work, Huffman works reducing the representation of the most-frequent used lines and increasing the representation of the less-frequent used lines. The statistical analysis to create the Huffman code tree is performed statically because wedgelets are predefined before the encoding execution and then a statistical analysis at runtime is not necessary.

Table 2 shows the allowed lines, its occurrence probability (extracted from the static analysis of all wedgelets patterns), and the Huffman code for all possible patterns of 4×4 block lines. One can notice that "0000" and "1111" are the most often used patterns being represented by only 2 bits, while "0011" and "0001," which are the less often patterns, are represented by 5 bits, resulting in a reduction in the storage requirements.

Equation (2) describes the average line size (ALS) used to represent each line when applying Huffman code, where $p(i)$ is the probability of appearing sample i and length [HuffmanCode(i)] is the number of bits used for representing the Huffman code of the sample i. For instance, ALS is 2.884 bits in the 4×4 block example, while FB&C algorithm required 3 bits per encoded line

$$\mathrm{ALS} = \sum_{i=1}^{2\times N} p(i) \times \mathrm{length}[\mathrm{Huffman\ Code}(i)]. \quad (2)$$

Table 2 Occurrence probability and Huffman code for the allowed lines of 4×4 blocks.

Allowed lines	Probability (%)	Huffman code
0111	6.10	0001
0011	5.81	00001
0001	6.10	00000
0000	21.52	10
1000	12.50	010
1100	15.12	001
1110	12.50	011
1111	20.35	11

4.2.3 *Block change map algorithm*

BCM is an algorithm planned to reduce the entropy of the wedgelet patterns employing an auxiliary matrix containing the differences between subsequent wedgelets.

For each wedgelet coding, the BCM algorithm stores the wedgelet patterns read from the standard WMem into the pattern$_{k+1}$ buffer. Additionally, the content of the pattern$_{k+1}$ is copied to the pattern$_k$ buffer. Consequently, pattern$_{k+1}$ contains the most recently read wedgelet, whereas pattern$_k$ contains the previous one.

The algorithm fills an auxiliary matrix of bits with the comparison between pattern$_k$ and pattern$_{k+1}$. The positions containing a change are signalized with "1" while the remaining positions are signalized with "0." Subsequently, the algorithm applies Huffman code to the content of this matrix, producing the wedgelet coded in BCM format. The exception occurs only in the first wedgelet since at the first time only pattern$_{k+1}$ contains a valid pattern; thus, the algorithm applies Huffman code on the content of pattern$_{k+1}$ instead of the auxiliary matrix.

The coded patterns are stored into the coded WMem reducing the storage requirement significantly when compared to the approach that uses only Huffman code, as demonstrated in the section of experimental results.

Figure 9 shows a block diagram of the BCM architecture, exemplifying the coding of the first two 4×4 wedgelet patterns. This example applies "Huffman" to encode the first wedgelet pattern. Subsequently, the "change bitmap generator" fills the auxiliary matrix comparing bit by bit the first wedgelet with the subsequent one, resulting in a single bit difference. The content of the matrix is encoded using Huffman. Finally, all encoded data are inserted into the coded WMem. Remark that the output of Huffman code is different from the one presented in Table 2 because BCM produces other probabilities of patterns occurrence.

4.2.4 *Line change map algorithm*

LCM algorithm works similarly to the BCM algorithm; however, LCM generates a bitmap change for each line, exploring the vertical redundancies between all lines of all blocks of the same size, i.e., the algorithm connects all wedgelets considering that the first line of a subsequent wedgelet needs to be compared with the last line of the previous wedgelet pattern. Instead of having pattern$_k$, pattern$_{k+1}$, and auxiliary matrix, LCM algorithm requires only vectors as buffers and, thus, corresponding the analogous line$_k$, line$_{k+1}$, and auxiliary vector, respectively. Notice that BCM algorithm reads only lines from the standard WMem; in addition, it stores only coded lines into the coded WMem.

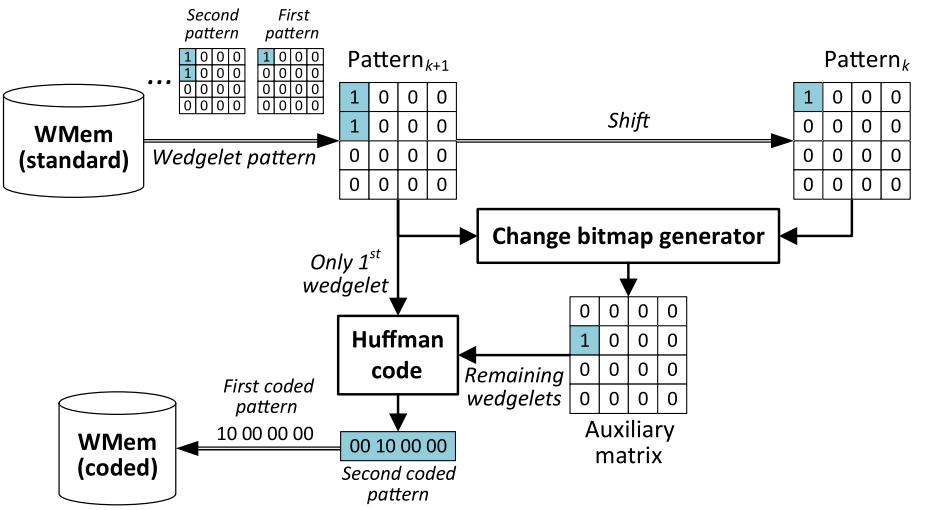


Fig. 9 Block diagram of the BCM architecture exemplifying the coding of two 4×4 wedgelet patterns.

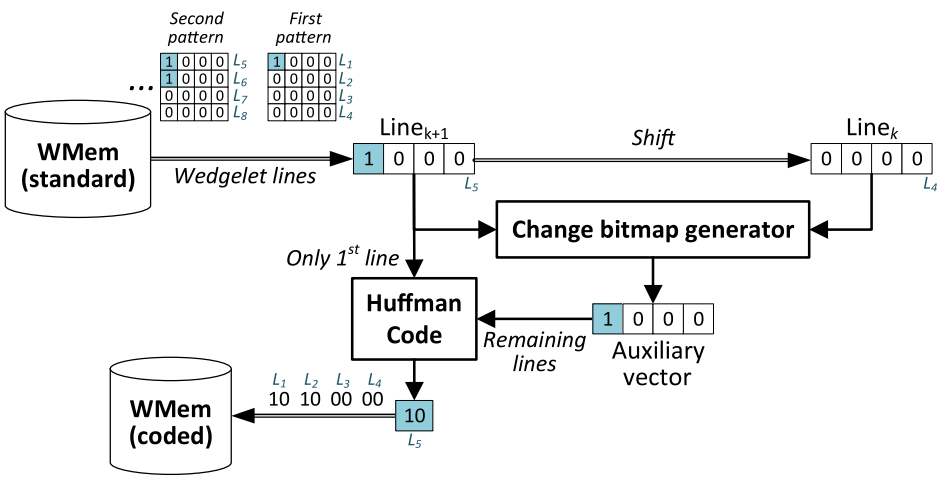


Fig. 10 Block diagram of the LCM architecture exemplifying the coding of five lines of the 4×4 wedgelet patterns. Line buffers are filled for generating the L_5 coded line.

Figure 10 shows an example of this technique applied to the five first lines of the standard WMem regarding 4×4 block sizes, which implies the coding of two subsequent wedgelets.

5 DMM-1 Memory Reduction Techniques

This section presents three advanced techniques proposed in this article to increase the compression of the wedgelet patterns when compared with the previous solutions. The techniques are: (i) block line change map (B-LCM), (ii) dual first bit and change (D-FB&C) algorithm, and (iii) ending rows removal (ERR). All these techniques reduce the memory size maintaining all wedgelet patterns in the WMem, i.e., without affecting the encoded video quality, bitrate, or decoding process. Section 6 presents a detailed evaluation of these techniques.

5.1 *Block Line Change Map Algorithm*

LCM algorithm[14] groups logically the wedgelet list before performing the bitmap change. When the first line pattern of a subsequent wedgelet differs significantly from the last line pattern of its predecessor wedgelet, the change bitmap generator produces more bit-changes, reducing the encoding efficiency. B-LCM was proposed to mitigate this problem, as an improvement of the LCM algorithm. B-LCM applies FB&C to encode the first line of all wedgelets, and the remaining lines of each wedgelet are encoded using LCM. This procedure makes the encoding of the wedgelet pattern independent from previous or subsequent patterns.

5.2 *Dual First Bit and Change Algorithm*

FB&C[14] is grounded on the fact that DMM-1 cannot contain interlaced ones and zeros in a single row, enabling to explore horizontal redundancies. However, the same restriction (inexistence of interlaced ones and zeros) occurs in the patterns columns. Thus, D-FB&C was proposed to explore both horizontal and vertical redundancies. Figure 11(a) exemplifies this coding technique in an 8×8 block size, where the first bit of the block is stored without compression; then, the first column of the block is encoded with the position

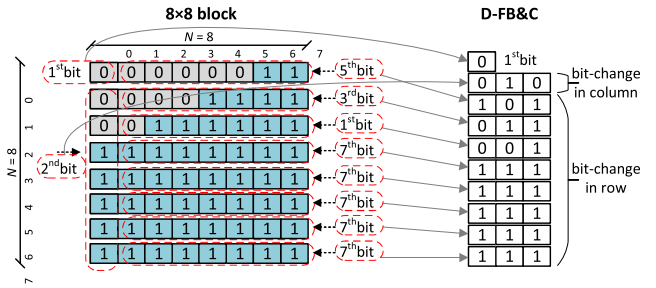


Fig. 11 D-FB&C encoding example with an 8×8 wedgelet pattern.

of the first bit-change (similar to the explanation of FB&C), which happens in the second bit of the first column of the example. Therefore, each line of the pattern is inserted into the memory being encoded with the position of the bit-change as FB&C does; however, without requiring the storage of the first bit of each row, since the encoded first column can provide this information.

5.3 *Ending Rows Removal Technique*

ERR is a technique planned to reduce the storage area removing consecutive vectors with the same bit pattern, especially when large blocks are encoded. The technique is based on two aspects: (i) DMM-1 defines only a single bit variation in each row or column and (ii) this variation may result in many consecutive rows or columns with the same pattern. Specifically, the ERR technique is applied for rows removal, but it could be applied to columns reduction, due to the symmetrical characteristics of the blocks and wedgelets.

Let (R_j, C_k) be the row–column pair representing the position of a bit inside a block with $1 \leq j \leq N$ and $1 \leq k \leq N$, then C_1 and C_N represent the leftmost and rightmost columns of the block, respectively. Whenever a wedgelet crosses C_1 or C_N, there is at least one row with a pattern where all bits are equal (i.e., all bits of the row are "0s" or "1s"). Additionally, there is a bit-change (i.e., "0" → "1" or "1" → "0") for each crossing. Considering these bit-changes represented by the pairs (R_a, C_1) and (R_b, C_N), then any and all row between $\text{Max}(R_a, R_b)$ and R_N have the same pattern where all bits are equal, allowing to apply the ERR technique. Based on this observation, the ERR algorithm skips adding information into WMem when all subsequent rows have the same pattern of the current one.

Figure 12 exemplifies three wedgelet patterns for 8×8 blocks. Figure 12(a) shows a wedgelet pattern that crosses only one column, the rightmost one (C_N), whose bit-change is placed in the pair (R_5, C_8); thus, all rows from R_5 to R_8 have the same pattern "11111111." Figure 12(b) exemplifies a wedgelet pattern with bit-changes placed on pairs (3, 1) and (5, 8); since Max(3, 5) is 5, rows 5 to 8 have the same pattern. However, Fig. 12(c) shows a wedgelet pattern that does not cross the columns C_1 and C_N; in this case, there is no occurrence of repeated patterns.

The ERR technique can be applied together with all other encoding algorithms. This article signalizes this technique inserting the symbol (+) after the basic encoding algorithm; for instance, D-FB&C+ means that ERR technique is applied together with D-FB&C.

Figure 13 shows the same wedgelet pattern of Fig. 11 to compare FB&C, D-FB&C, and D-FB&C+, considering an 8×8 block. The example shows that D-FB&C+ reduces 12 bits in relation to D-FB&C, representing 42.8% of compression for this 8×8 block. In addition, when D-FB&C+ is

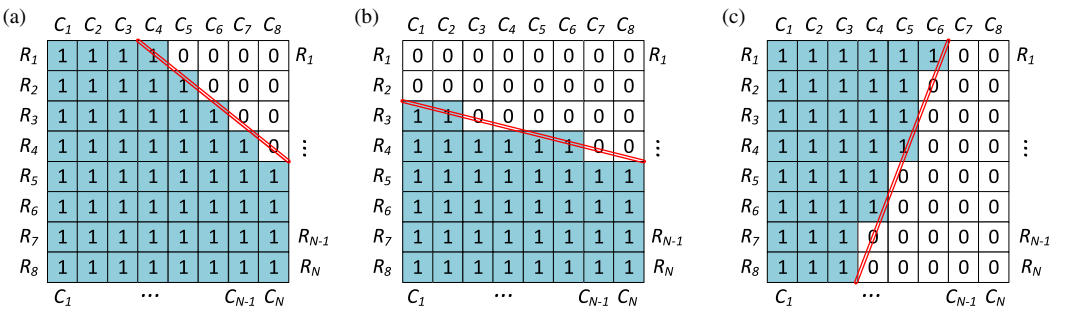


Fig. 12 (a)–(c) Example of wedgelet patterns for 8×8 blocks.

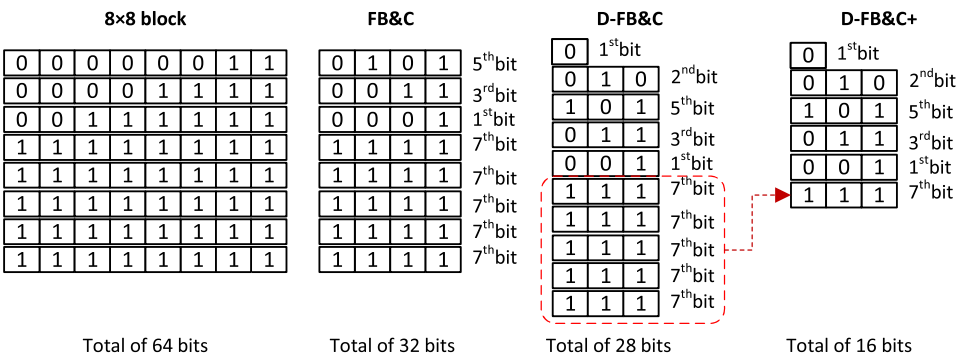


Fig. 13 The same wedgelet pattern of Fig. 11 codified with FB&C, D-FB&C, and D-FB&C+.

Table 3 Number of bits required to store all wedgelets.

Work	Algorithm	4×4	8×8	16×16	Total	BD-rate (%)
	Standard	1376	51,328	130,560	183,264	0.00
17	Downsampling	0	0	130,560	130,560	0.05
15	No refinement	928	20,096	98,304	119,328	0.25
14	Huffman	991	23,503	34,298	58,792	0.00
This work	BCM	761	14,428	27,175	42,364	0.00
	LCM	1086	22,301	27,108	50,495	0.00
	FB&C	1032	25,664	40,800	67,496	0.00
	B-LCM	865	19,944	25,259	46,068	0.00
	D-FB&C	946	22,456	35,190	58,592	0.00
	FB&C+	825	17,256	24,225	42,306	0.00
	B-LCM+	755	17,107	21,700	39,562	0.00
	D-FB&C+	808	16,150	21,930	38,888	0.00

compared to FB&C, which does not explore vertical redundancies, it reduces by half the memory required to store that pattern (i.e., from 32 to 16 bits).

6 Evaluation of the Proposed Algorithms

All wedgelets patterns employed on the experimental results were extracted from 3D-HTM, and the proposed solutions were implemented using MATLAB®. Additionally, D-FB&C+ was implemented using 65-nm CMOS technology to extract synthesis results.

Table 3 shows the number of bits required to store all wedgelets patterns of all block sizes of the standard DMM-1, the proposals of Refs. 14, 15, and 17, and the solutions proposed in this work. In addition, the ERR technique was evaluated with FB&C, B-LCM, and D-FB&C (i.e., FB&C+, B-LCM+, and D-FB&C+). Notice that all memory compression techniques proposed in this article do not change the 3D-HEVC processing flow. Our solutions only reduce the WMem size maintaining 0% of BD-rate and, consequently, minimizing the required bandwidth between the encoder/decoder with the WMem.

Figure 14 shows the impact of the presented compression solutions using the standard DMM-1 as reference. In the downsampling solution proposed in Ref. 17, only the total compression rate is presented since it removes the memory area required for storing 4×4 and 8×8 wedgelet patterns, maintaining all wedgelet patterns for 16×16 blocks. B-LMC improves the LCM algorithm increasing more than 2% the compression rate. D-FB&C provides almost 5% gains when compared to FB&C because of its vertical-aware propriety. The ERR technique applied to FB&C, B-LCM, and D-FB&C increases 9.3% the compression rate, in average. Additionally, D-FB&C+ reaches a total memory reduction of 78.8% compared to the standard approach; consequently, D-FB&C+ is the best compression result among all solutions proposed in this work and also among all published works.

D-FB&C+ provides 1.9% of compression gains in relation to BCM, which is the best-designed algorithm in our previous work.[14] Also, the D-FB&C+ implementation is much more straightforward than BCM, because BCM needs to implement the BCM block, which requires lots of arithmetical operations, while D-FB&C+ can be implemented with a simple lookup table (for D-FB&C) combined with control mechanisms to implement ERR. Therefore, the results of the techniques presented in this article surpass all solutions available in the literature focusing on WMem

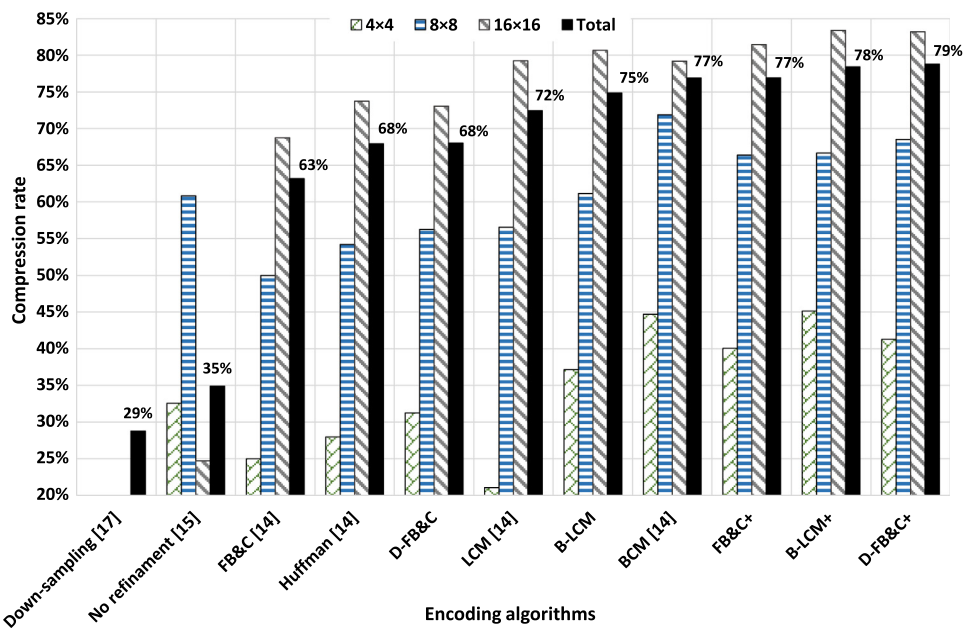


Fig. 14 Compression rates for all stored patterns of all proposed techniques.

reduction, including our previous works, and without encoding efficiency losses (i.e., 0% of BD-rate impacts). Considering the described solutions and the evaluated results, we recommend the designers to use the D-FB&C+ when developing a DMM-1 encoder and/or decoder hardware design to obtain more WMem compression with low complex implementation.

7 D-FB&C+ Hardware Implementation

This section presents the D-FB&C+ hardware implementation. The D-FB&C+ algorithm was chosen to be implemented in hardware because it is the most efficient technique for wedgelets compression, as presented in Sec. 6.

The D-FB&C+ can be implemented with a lightweight hardware due to the low complexity of this algorithm. The D-FB&C+ coded wedgelet is generated in an offline process, as explained previously. These codes are stored in the WMem and remain the same during the system execution. Then, any DMM-1 encoder or decoder designed in hardware requires only an additional hardware to support the D-FB&C + decoding, since when the DMM-1 encoder or decoder requires a wedgelet from WMem, the coded wedgelet must be decoded to be used.

The standard WMem contains all wedgelets in the uncompressed format defined by the 3D-HEVC standard, as shown in Fig. 7. These wedgelets can be directly assessed by the DMM-1 encoder or decoder. The solution proposed in this article defines that the wedgelets are stored in a compressed way inside WMem. Then, an extra hardware (the D-FB&C+ decoder) is necessary between the WMem and the DMM-1 encoder and decoder. The D-FB&C+ decoder, besides reducing the WMem, also reduces the memory communication since the flow between the memory and the encoder/decoder is of compressed wedgelets. Notice that both the DMM-1 encoder and decoder can share a single hardware containing the decoder of the compressed data (i.e., D-FB&C+ decoder) if they are inside the same integrated circuit.

A D-FB&C+ decoder architecture was designed using very high speed integrated circuits hardware description language to evaluate the area consumption and power dissipation, considering the DMM-1 encoder scenario. Figure 15 shows the block diagram of the proposed architecture.

As described in Secs. 5.2 and 5.3, the D-FB&C+ algorithm stores the wedgelet pattern in an irregular format. The first bit of the D-FB&C+ format is the bit of the top left corner of the wedgelet pattern block, and the following bits are grouped into sets of $\log_2(N)$-bit ("lCodes") to code bit-changes in the rows and columns of the block. Notice that lCode size depends on the block size is being decoded; for instance, lCode is 3-bit size for 8×8 blocks since N is 8 and $\log_2(N)$ is 3. Therefore, this section exemplifies the D-FB&C+ decoder considering 8×8 blocks.

The D-FB&C+ decoder has a one-byte register ("InputReg") that receives 8 bits from the coded WMem. Due to the irregular format of the D-FB&C+, in the first reading, the InputReg stores the first coded bit of the block, two lCodes, plus one bit that is part of the third lCode. The first lCode indicates the line of the first column where the bit-change occurs and the second lCode indicates the position of the bit-change in the first row. This last bit of InputReg is not handled in this first reading since the third lCode is incomplete. However, this remaining bit together with the next byte reading allows the construction of three more lCodes.

The "ControlUnit" circuit controls the bitstream read from the coded WMem to keep updated the information being received and check if a rereading of the WMem is required to complete the wedgelet pattern decoding. Thus, the reading of the coded WMem follows as long as there are wedgelet patterns that should be read by the D-FB&C+ decoder.

Decoding the pattern requires an output matrix ("OutMatrix") that must have the size of the block being decompressed. In practice, this matrix must have the largest block size to be decompressed (i.e., 16×16), since smaller blocks are stored within this same area.

ControlUnit copies the first bit of the InputReg into a one-bit auxiliary register ("AuxB") and the next lCode into an auxiliary register ("AuxCol") that stores the number of the row where occurs the bit-change in the first column. All subsequent lCodes encode the bit-change position in each row of

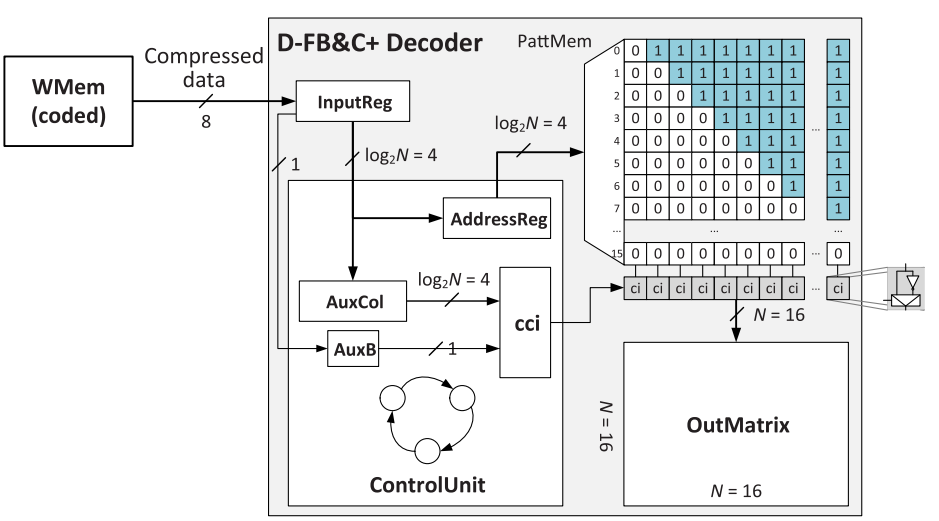


Fig. 15 Partial block diagram of the D-FB&C+ decoder architecture.

Table 4 D-FB&C+ synthesis results compared to the standard approach.

		Area (μm^2)				Power (mW)			
Algorithm	Resource	4×4	8×8	16×16	32×32	4×4	8×8	16×16	32×32
Standard	Memory	1523	30,118	68,378	68,378	4.2	217.3	325.5	81.4
D-FB&C+	Memory	1021	8496	11,833	11,833	1.8	40.1	38.5	9.6
	Hardware	710	2617	7256	6526	2.3	13.4	34.7	16.8
	Total	1731	11,113	19,089	18,359	4.1	53.5	73.2	26.4
Gain (%)		−13.7	63.1	72.1	73.2	2.4	75.4	77.5	67.6

the block (rowBlock); thus, ControlUnit copies this data into a register ("AddressReg") that addresses a pattern memory ("PattMem") containing the wedgelet pattern of a row. Similarly, to the coded WMem, the PattMem is also filled only once during the design time.

The output row of PattMem passes by a vector of controlled inverters (ci) that maintain the pattern or invert all bits. The vector of cis is managed by a circuit (cci) that reads AuxB and AuxCol and, according to both information, decides if the patterns have to be inverted. For instance, if AuxB $=0$ and the AuxCol $>$ rowBlock, the pattern is preserved; however, if AuxB $=1$ and the AuxCol $>$ rowBlock, the pattern is inverted. We chose to implement this inversion circuit to reduce in half the PattMem size because half of the wedgelet patterns are complementary to the ones presented in Fig. 15.

All patterns passing by the vector of cis are stored into OutMatrix, and the process of reading new lCodes is repeated until the pointer of the OutMatrix row is smaller than the height of the block is being decoded.

The D-FB&C+ decoder architecture was synthesized using standard cells ST 65-nm technology, and the area consumption and power dissipation of the WMem were estimated using CACTI.[27] The parameters used in the synthesis were focused on achieving real-time processing in the DMM-1 main stage. Table 4 shows the area consumption and power dissipation results and compares the proposed solution to the standard approach.

First, notice that the WMem area required for 16×16 and 32×32 blocks storage is the same because the wedgelet patterns for 32×32 blocks are obtained from the same memory information employed on 16×16 blocks, and it is just required to upscale the patterns. Moreover, the circuits employed on 32×32 blocks encoding dissipate less power than 16×16 blocks because there are four times less 32×32 blocks in a frame than 16×16 blocks and both require the same computation effort.

Regarding 4×4 blocks, our proposal reduces the dissipated power by only 2.4% when compared to the standard approach, because the power dissipation of the decoder is considerably high. Additionally, the hardware of the D-FB&C+ decoder has an extra area overhead implying our solution consumes 13.7% more area than the standard approach.

Regarding blocks $\geq8\times8$, the gains of the D-FB&C+ solution are expressive reducing between 63.1% and 73.2% the area consumption and between 67.6% and 77.5% the power dissipation, even considering the hardware overhead of the D-FB&C+ decoder. Considering the coding of all block sizes, the D-FB&C+ approach reduces the power dissipation from 628.4 to 157.2 mW when compared to the standard approach, which represents a reduction of almost 75%.

8 Conclusions

This article presented three lossless techniques to compress the memory used for wedgelet storage in 3D-HEVC. The standard 3D-HEVC DMM-1 WMem requires more than 180 kbits, which hinder the design of a low-power and/or low-area DMM-1 encoder or decoder architecture. Therefore, we proposed three techniques to achieve good results in reducing the hardware power dissipation and the hardware consumption. Our best results were obtained using D-FB&C+ that combines the D-FB&C technique, which explores both vertical and horizontal redundancies inside the wedgelet pattern, and the ERR technique, which focuses on removing repeated lines at the end of the block containing the wedgelet pattern. D-FB&C+ reduces 78.8% of the WMem size, representing more than 144 kbits of reduction. This huge compression rate, besides reduces the wedgelets memory size, also reduces the bandwidth required for memory communication. The presented hardware design for the D-FB&C+ algorithm showed a power reduction of up to 77.5% when compared with a standard implementation and with a smaller area consumption.

Acknowledgments

This paper was achieved in cooperation with Hewlett-Packard Brazil Ltd. using incentives of Brazilian Informatics Law (Law No. 8.248 of 1991). Authors also would like to thank CNPq (processes 309707/2015-3 and 486136/2013-2), CAPES (processes 88881135737/2016-01 and 88881119481/2016-01), and FAPERGS (process 16/2551-0000241-0) Brazilian research agencies to support the development of this work.

References

1. JCT-3V, //phenix.int-evry.fr/jct2 (July 2017).
2. K. Muller et al., "3D high-efficiency video coding for multi-view video and depth data," *IEEE Trans. Image Process.* **22**(9), 3366–3378 (2013).
3. G. Tech et al., "Overview of the multiview and 3D extensions of high efficiency video coding," *IEEE Trans. Circuits Syst. Video Technol.* **26**(1), 35–49 (2016).

4. G. Sullivan et al., "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.* **22**(12), 1649–1668 (2012).
5. K. Muller, P. Merkle, and T. Wiegand, "3D video representation using depth maps," *Proc. IEEE* **99**(4), 643–656 (2011).
6. P. Kauff et al., "Depth map creation and image-based rendering for advanced 3DTV services providing interoperability and scalability," *Signal Process. Image Commun.* **22**(2), 217–234 (2007).
7. Mitsubishi Electric Research Laboratories, ftp://ftp.merl.com/pub/tian/NICT-3D/ (July 2017).
8. L. Vosters, C. Varekamp, and G. Haan, "Overview of efficient high-quality state-of-the-art depth enhancement methods by thorough design space exploration," *J. Real-Time Image Process.* 1–21 (2015).
9. P. Merkle et al., "Depth intra coding for 3D video based on geometric primitives," *IEEE Trans. Circuits Syst. Video Technol.* **26**(3), 570–582 (2016).
10. J. Lee, M. Park, and C. Kim, "3D-CE1: depth intra skip (DIS) mode," Document JCT3V-K0033, JCT-3V, Switzerland (2015).
11. H. Liu and Y. Chen, "Generic segment-wise DC for 3D-HEVC depth intra coding," in *Proc. of the IEEE Int. Conf. on Image Processing (ICIP)*, pp. 3219–3222 (2014).
12. G. Sanchez et al., "3D-HEVC depth maps intra prediction complexity analysis," in *Proc. of the IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS)*, pp. 348–351 (2016).
13. J.-C. Tuan, T.-S. Chang, and C.-W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," *IEEE Trans. Circuits Syst. Video Technol.* **12**(1), 61–72 (2002).
14. G. Sanchez, C. Marcon, and L. Agostini, "Energy-aware light-weight DMM-1 patterns decoders with efficiently storage in 3D-HEVC," in *Proc. of the Symp. on Integrated Circuits and Systems (SBCCI)*, pp. 1–6 (2016).
15. G. Sanchez, C. Marcon, and L. Agostini, "Real-time scalable architecture for 3D-HEVC bipartition modes," *J. Real-Time Image Process.* **13**(1), 71–83 (2017).
16. F. Amish and E. Bourennane, "An efficient hardware solution for 3D-HEVC intra-prediction," *J. Real-Time Image Process.* 1–13 (2017).
17. M. Shuying et al., "Reducing wedgelet lookup table size with down-sampling for depth map coding in 3D-HEVC," in *Proc. of the IEEE Int. Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–5 (2015).
18. G. Bjontegaard, "Calculation of average PSNR differences between RD curves," ITU-T SC16/SG16 VCEG-m33, Austin (2001).
19. JCT-3V Experts, "3D-high efficiency video coding test model 16.2," https://hevc.hhi.fraunhofer.de/svn/svn_3DVCSoftware/tags/HTM-16.2/ (July 2017).
20. D. Marpe et al., "Video compression using nested quadtree structures, leaf merging, and improved techniques for motion representation and entropy coding," *IEEE Trans. Circuits Syst. Video Technol.* **20**(12), 1676–1687 (2010).
21. K. Oh, J. Lee, and D. Park, "Depth intra skip prediction for 3D video coding," in *Proc. of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conf. (APSIPA ASC)*, pp. 1–4 (2012).
22. J. Lainema et al., "Intra coding of HEVC standard," *IEEE Trans. Circuits Syst. Video Technol.* **22**(12), 1792–1801 (2012).
23. F. Jager, "Simplified depth map intra coding with an optional depth lookup table," in *Proc. of the Int. Conf. on 3D Imaging (IC3D)*, pp. 1–4 (2012).
24. P. Merkle et al., "3D video: depth coding based on inter-component prediction of block partitions," in *Proc. of the Picture Coding Symp. (PCS)*, pp. 149–152 (2012).
25. B. Oh, J. Lee, and D. Park, "Depth maps coding based on synthesized view distortion function," *IEEE J. Sel. Top. Signal Process.* **5**(7), 1344–1352 (2011).
26. D. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE* **40**(9), 1098–1101 (1952).
27. N. P. Jouppi et al., "CACTI-IO: CACTI with OFF-chip power-area-timing models," *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **23**(7), 1254–1267 (2015).

Gustavo Sanchez has been a professor at the Federal Institute Farroupilha, Brazil, since 2014. He received his master's degree in computer science from the Federal University of Pelotas in 2014. Currently, he is pursuing his PhD in computer science at the Pontifical Catholic University of Rio Grande do Sul. He has more than 8 years of experience in designing algorithms and hardware for video coding. His interests include complexity reduction algorithms and dedicated hardware design for video coding.

César Marcon has been a professor of Graduate Program in Computer Science at Pontifical Catholic University of Rio Grande do Sul, Brazil, since 1995. He received his PhD in computer science from Federal University of Rio Grande do Sul, Brazil, in 2005. He is a member of IEEE. He has more than 100 papers published in prestigious journals and conference proceedings. Since 2005, he coordinated nine research projects in the areas of telecom and telemedicine.

Luciano Volcan Agostini has been a professor since 2002 at Federal University of Pelotas, Brazil, where he leads the Video Technology Research Group. He received a PhD in computer science from Federal University of Rio Grande do Sul, Brazil, in 2007. He is a Brazilian distinguished researcher (PQ-2). He is a senior member of IEEE and ACM, and he is also a member of the IEEE CAS Multimedia Systems and Applications Technical Committee.