Energy-Aware Light-Weight DMM-1 Patterns Decoders with Efficiently Storage in 3D-HEVC

[1,2]Gustavo Sanchez, [3]Luciano Agostini, [1]César Marcon
[1] PPGCC – Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil
[2] IFFarroupilha – Instituto Federal Farroupilha, Brazil
[3] GACI/PPGC – Universidade Federal de Pelotas, Pelotas, Brazil
gustavo.sanchez@acad.pucrs.br, cesar.marcon@pucrs.br, agostini@inf.ufpel.edu.br

***Abstract*—One of the major problems in Depth Modeling Mode 1 (DMM-1) hardware design is the large memory area required for storing all wedgelet patterns. This work presents four energy-aware light-weight coding techniques to reduce the memory usage and power dissipation. Experimental results show that the proposed techniques are capable of reaching a compression rate of 76.9%, representing a reduction of 140 Kbits. In the DMM-1 hardware design, this compression rate results in smaller memory, fewer memory accesses and, thus a power dissipation decrease, reaching 678 mW for ST 65nm standard cells technology.**

Keywords—3D-HEVC; DMM-1; Pattern Compression; Memory Size Reduction

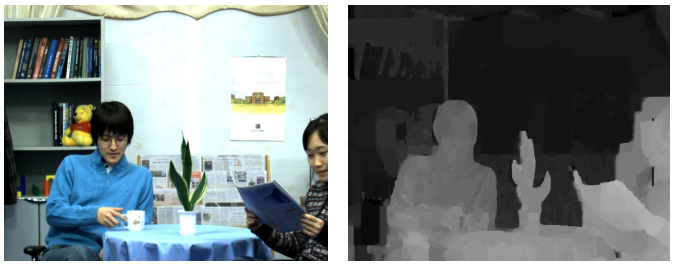
I. INTRODUCTION

Due to the current increase in 3D video coding usage, the Joint Collaborative Team on 3D Video Coding Extension Development (JCT-3V) [1] spends significant effort in research and development to extend the High-Efficiency Video Coding (HEVC) standard [2] to 3D video applications, aiming a bandwidth reduction for 3D video transmission/storage, while maintaining its quality. By February 2015, the JCT-3V has finalized the 3D-HEVC standard, which uses the most advanced features provided by HEVC and inserts many 3D-specific features aiming to obtain higher efficiency.

The adoption of Multiview Video plus Depth (MVD) [3] representation is one of the key factors for 3D-HEVC being capable of achieving high levels of compression rate. In MVD, each texture view is associated with a depth map. The same camera captures the texture images and depth maps, which provides geometrical information according to the distance between the objects and the camera. These maps are composed of 8-bits samples, where the closer the object is from the camera, the lighter is the shade of gray the object is represented. Fig. 1 presents a (a) texture view and its associated (b) depth map extracted from *Newspaper_CC* video sequence.

The motivation for MVD usage is to reduce the bandwidth for a 3D video transmission. This reduction is possible because, in the decoder, virtual views can be synthesized by interpolating texture and depth data with the use of techniques such as Depth Image Based Rendering (DIBR) [3]. These techniques allow synthesizing a dense set of texture views of the scene [4] and, consequently, to reduce the number of transmitted texture views.

The quality of the depth maps is crucial to allow generating high-quality virtual views, which is one of the main challenges in this scenario. Fig. 1 shows that depth maps contain characteristics that contrast with texture frames; i.e., they contain large areas of constant values (background or body of objects) and sharp edges (border of objects) while texture frames have smooth transitions between its pixels [5].



(a) (b)

Fig. 1. (a) Texture view and its associated (b) depth map extracted from *Newspaper_CC* video sequence.

It is important to emphasize that distortions in the depth maps indirectly impact on the video quality since they are used to synthesize new texture views of the same scene. Then, it is important to encode the depth maps as precisely as possible, preserving these edges (i.e., without smoothing them) and avoiding errors in the video synthesis process.

The standard HEVC intra prediction [2] was designed considering the smooth transition between texture pixels, and this prediction is not efficient in the presence of depth maps sharp edges. Taking into account the importance of edge preservation, the JCT-3V has designed a coding tool called bipartition modes, which should be applied in the 3D-HEVC intra prediction to encode better depth maps edges [6]. The bipartition modes are composed by two Depth Modeling Modes (DMMs) – DMM-1 and DMM-4. Notice that DMM-2 and DMM-3 were removed in 3D-HEVC standard since they present low coding efficiency [7].

One of the major problem for DMM-1 hardware design is related to the huge memory area required to store the wedgelet patterns [8]. This work proposes four light-weight approaches that reduce the area, quantity of accesses and energy consumption of the memory used for DMM-1 wedgelet patterns storage. The remainder of this paper is organized as follows: Section II presents the 3D-HEVC depth intra prediction algorithm. Section III describes the proposed

978-1-5090-2736-1/16/$31.00 ©2016 IEEE

solutions. Section IV presents the simulation results comparing with the standard HEVC storage requirements. Finally, Section V renders the conclusions of this work.

II. Depth Maps Intra Prediction

Fig. 2 illustrates a high-level block diagram of the depth maps intra prediction implemented in 3D-HEVC reference software [9]. In the intra prediction, depth maps can be encoded by two modules: the (i) **HEVC intra prediction** that implements the same intra algorithms used for texture videos, and the (ii) **bipartition modes**, which explore depth maps properties that are not used on HEVC intra prediction.

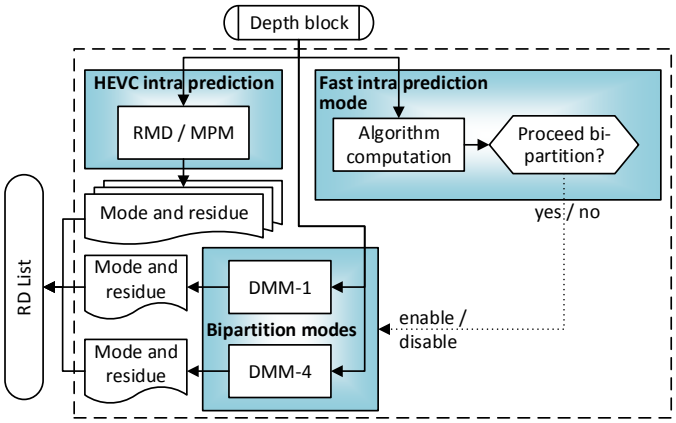


Fig. 2. Main blocks and flow of 3D-HEVC depth maps intra prediction.

The bipartition modes should be evaluated in parallel with the basic HEVC intra prediction step. However, since depth coding blocks are very flat or smooth, in general, and bipartition modes obtain better results in edges of sharp transitions, the **Fast intra prediction mode** proposed in [10] is applied to enable/disable the use of bipartition modes, reducing the computational complexity of the 3D-HEVC depth intra prediction.

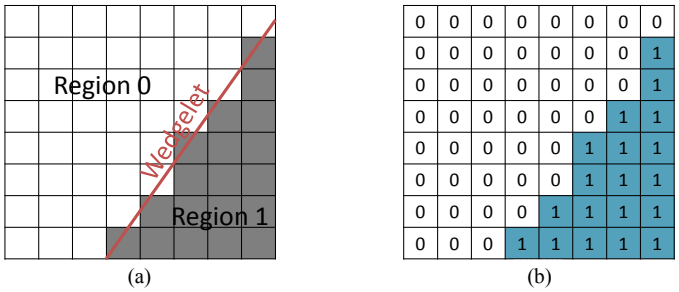
The **Fast intra prediction mode** only enables the use of the bipartition modes when the first mode obtained by HEVC intra prediction in the Rate-Distortion list (RD-list) is not the planar mode, and the encoding block variance is higher than a pre-defined threshold [10], meaning that the encoding block has a high probability of containing an edge. When this technique fails, it means that the encoding block represents a flat or a smoothing area of the scene, which is the case where the HEVC intra prediction obtains better results. This procedure reduces the computational complexity of the intra prediction process without reducing encoding efficiency, because it reduces the unnecessary bipartition modes evaluation significantly. When the bipartition modes are enabled, the DMM-1 and DMM-4 are processed, and their results are added in the RD-list. In the following steps inside the encoder, the RD-cost is computed for all results of HEVC intra modes and bipartition modes added to this list.

The bipartition modes divide a given depth block into two regions. The encoded block encompasses a pattern containing the segmentation information, which specifies the region each sample belongs to, and a constant value that represents each region [11]. This pattern is composed of an array with N×N elements (N is the block width), containing 0 or 1 when the element belongs to region 0 or 1, respectively.

The DMM-4 algorithm dynamically creates the segmentation pattern from the texture data information, while the DMM-1 algorithm, which is the focus of this work, requires an evaluation over a predefined pattern set, which is a memory consuming approach.

A. Depth Modeling Mode 1 (DMM-1)

The DMM-1 algorithm segments blocks in two regions using a straight line named wedgelet. Regarding a continuous space, as exemplified in Fig. 3(a), the wedgelet could be represented and stored by the straight line equation. However, taking into account the discrete samples space, the wedgelet is stored in an N×N-array containing the binary pattern that defines the regions 0 and 1, as exemplified in the 8×8 block of the Fig. 3(b).



(a) (b)

Fig. 3. Wedgelet segmentation model of a depth block: (a) the straight line dividing Region 0 and Region 1, and (b) discretization with constant values.

Fig. 4 presents a high-level diagram of the DMM-1 encoding algorithm, which is composed of three stages: (i) **Main Stage**, (ii) **Refinement Stage**, and (iii) **Residue Stage**. The **Main Stage** evaluates the entire initial wedgelet set (i.e., all wedgelets that must be assessed before the refinement) and finds identifies the best wedgelet pattern among the available ones.

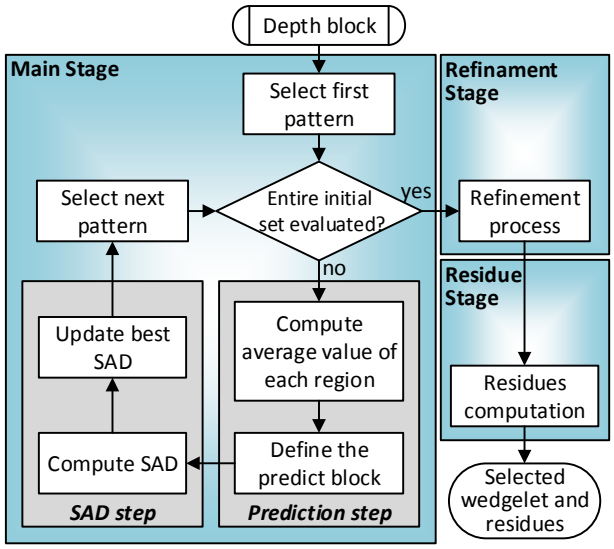


Fig. 4. The main blocks of the DMM-1 encoding algorithm.

For each wedgelet pattern, Fig. 5 shows the pseudo-code applied to obtain the predicted block, where the $block_{i,j}$ represents the samples of the encoding block, $wedge\_pattern_{i,j}$ is the binary array of the wedgelet pattern, and $pred\_block_{i,j}$ is the resulting predicted block.

The DMM-1 algorithm starts computing avg_0 and avg_1, which are the average values of all samples mapped into regions 0 and 1, respectively (lines 1-11). Subsequently, the algorithm generates the predicted block inserting the avg_0 or avg_1 in the pred_block matrix, according to the wedgelet pattern (**Prediction step**). Next, Equation (1) computes the Sum of Absolute Differences (SAD) for each wedgelet pattern, where $| P_{i,j} - O_{i,j} |$ is the absolute value of the residue between the predicted and original depth samples at position (i, j). Finally, all SADs are compared, and the pattern with the lowest SAD defines the best wedgelet (**SAD step**).

```
1.  avg_0 ← 0, avg_1 ← 0, num_0 ← 0, num_1 ← 0
2.  for i ← 1 to N
3.      for j ← 1 to N
4.          if wedge_pattern_i,j = 1
5.              avg_0 ← avg_0 + block_i,j
6.              num_0 ← num_0 + 1
7.          else
8.              avg_1 ← avg_1 + block_i,j
9.              num_1 ← num_1 + 1
10. avg_0 ← avg_0 / num_0
11. avg_1 ← avg_1 / num_1
12. for i ← 1 to N
13.     for j ← 1 to N
14.         if wedge_pattern_i,j = 1
15.             pred_block_i,j ← avg_1
16.         else
17.             pred_block_i,j ← avg_0
```

Fig. 5. Pseudo-code for DMM-1 prediction block generation.

$$SAD = \sum_{i=1}^{N} \sum_{j=1}^{N} |P_{i,j} - O_{i,j}| \quad (1)$$

The **Refinement Stage** evaluates up to eight wedgelets around of the selected one (i.e., with a similar pattern) in the previous operation. Again, the wedgelet that obtained the lowest SAD among these eight possibilities, along with the first wedgelet selected in **Main Stage,** is elected as the best one. Finally, the **Residue Stage** subtracts the predicted block of the elected wedgelet from the original one and adds this wedgelet into the RD-list.

Fig. 6 exemplifies the encoding of a 4×4 depth block along with the evaluation of three wedgelet patterns. 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 predicted block step computes the average value of all pixels in the region (e.g., the average value of regions 0 and 1 of pattern **a** are 64 and 76, respectively). The residue step annotates the position corresponding of each pixel with the difference between the predicted and original depth sample. The SAD of all patterns is attained and finally, the pattern **b** is selected since it has the lowest SAD.

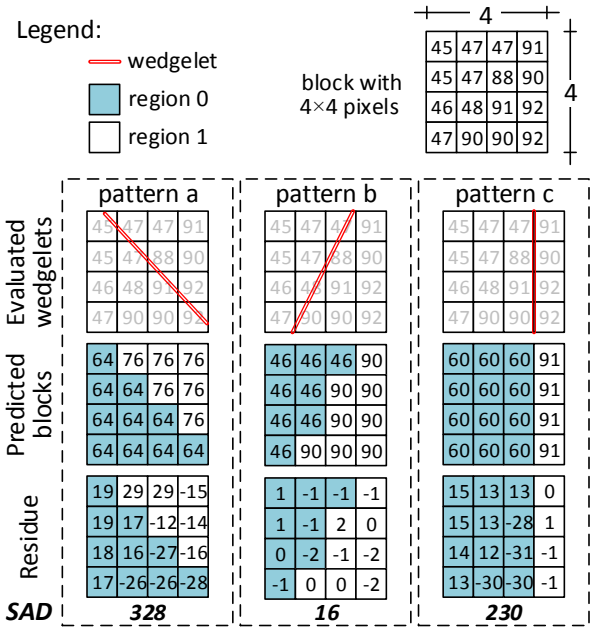


Fig. 6. Example of a 4×4 depth block encoding with the DMM-1 algorithm.

One of the major problems in DMM-1 hardware design is efficiently storing all patterns because there are a lot of possible wedgelets [8]. Table I shows the number of wedgelets and its storage requirements, considering that the memory stores all patterns using N×N bits; i.e., without any compression, which is done in 3D-HEVC Test Model (HTM) version 16.0 reference software due to the required performance.

TABLE I. NUMBER OF EVALUATED WEDGELETS IN DMM-1

Block size	Wedgelets	Storage requirements (bits)
4×4	86	1,376
8×8	802	51,328
16×16	510	130,560
32×32	510	-
Total		183,264

Table I depicts that the DMM-1 hardware requires 183,264 bits to store all wedgelet patterns for all available block sizes. Notice that the DMM-1 wedgelet patterns of 32×32 blocks are not stored because they are obtained up-scaling 16×16 wedgelets. Furthermore, the standard DMM-1 algorithm requires frequent access to the wedgelet patterns implying large energy consumption. This work explores and compares light-weight techniques for wedgelet patterns compression to reduce the storage area and energy consumption.

III. PROPOSED COMPRESSION TECHNIQUES

This section describes four techniques proposed to reduce the wedgelets pattern memory size: (i) First Bit and Change (FB&C); (ii) Huffman Code; (iii) Block Change Map (BCM); and (iv) Line Change Map (LCM).

A. First Bit and Change (FB&C)

FB&C algorithm is grounded in the fact that DMM-1 is a bipartition approach that divides each block into two and only two regions. Therefore, the capacity of the standard DMM-1 to represent interlaced ones and zeros in a single line is unnecessary producing an inefficient storage model. For example, regarding a 4×4 block, each line should never contain the 0110 pattern because any wedgelet is capable of describing such pattern. All forbidden lines patterns for 4x4 blocks are presented in Table II along with the allowed ones and its proposed FB&C representation. For an N×N block, there are 2×N allowed lines in DMM-1 patterns.

TABLE II. FB&C REPRESENTATION FOR 4×4 PATTERN LINES

Forbidden lines	Allowed lines	Proposed representation
0010	0111	000
0100	0011	001
0101	0001	010
0110	0000	011
1001	1000	100
1010	1100	101
1011	1110	110
1101	1111	111

Fig. 7(a) illustrates the FB&C technique that codifies each line of the block copying the 1st bit content of the uncoded line to the coded one. Let *RB* be the remaining bits required for codifying a line of the N×N block, which can be obtained applying Equation (2). Then, the remaining *RB* bits of the coded line indicate how many subsequent positions, the line will change to the other region.

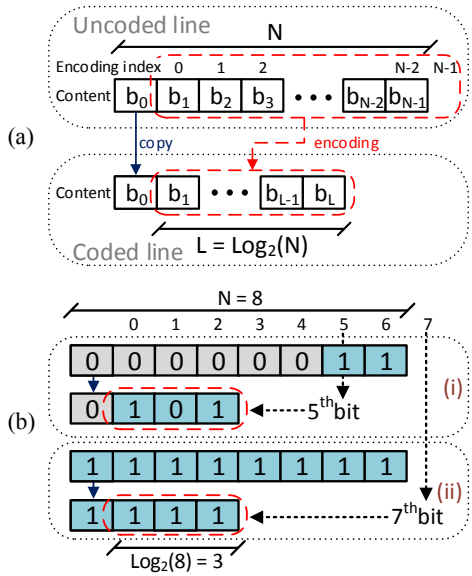


Fig. 7. (a) FB&C encoding model; and (b) two examples with 8×8 blocks.

$$RB = log_2(N) \tag{2}$$

Note in the examples of Fig. 7(b) that the encoded value of N-1 means that the line contains only a single region. Moreover, one can notice that the amount of bits required by the FB&C algorithm grows logarithmically, while the same storage requirement increases linearly for the standard DMM-1 technique. The Equation (3) analytically computes the compression rate obtained with FB&C technique.

$$Compression\ Rate = 1 - \frac{\log_2(N) + 1}{N} \tag{3}$$

B. 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. This algorithm works reducing the representation of the most frequent lines and increasing the representation of the less frequent lines. The statistical analysis to create Huffman code tree is obtained offline because wedgelets are predefined before the encoding execution.

Table III shows the original samples with all available lines, its probability and the Huffman code for all available 4×4 pattern lines. One can notice that the samples 0000 and 1111, which are the most often samples, are represented by only 2 bits, while 0011 and 0001, which are the less often samples, are represented by 5 bits, resulting in a reduction in storage requirements.

TABLE III. PROBABILITY AND HUFFMAN CODE FOR 4×4 PATTERN LINES

Original Sample	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

The Average Line Size (ALS) used to represent each line when applying Huffman Code is obtained using Equation (4), where *p(i)* is the probability of the sample *i* appear.

$$ALS = \sum_{i=1}^{2\times N} p(i) * length(HuffmanCode(i)) \tag{4}$$

The compression rate can be analytically obtained applying the Equation (5). For the 4×4 block example, the ALS is 2.884 bits, and the compression rate is 27.98%.

$$Compression\ Rate = 1 - \frac{ALS}{N} \tag{5}$$

C. Block Change Map (BCM) Algorithm

In BCM algorithm, the first wedgelet pattern can be encoded with any algorithm or even be stored without compression. Next, for each wedgelet pattern, a matrix called *Block Change Map* is created indicating when this wedgelet pattern has a bit change comparing with the previous pattern. The positions that contain a change are signalized with 1 while the remaining positions are signalized with 0.

This Block Change Map creation helps to increase the entropy of the wedgelet patterns and then, Huffman algorithm is applied per line, reducing the storage requirements considerably. An example of its application for two 4×4 wedgelets patterns is presented in Fig. 8.

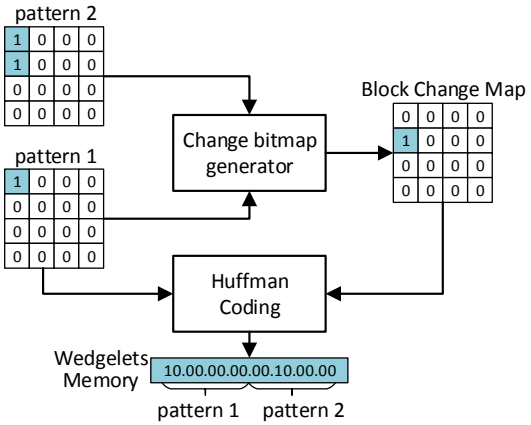


Fig. 8. Example of BCM application for two 4×4 block patterns.

In this example, the Huffman Coding algorithm has been applied to encode the first wedgelet pattern. A comparison is performed bit by bit in Change bitmap generator block to encode the second pattern, where it resulted in 0 in the positions that identically are 1 in the positions that presented a change. The same Huffman Coding block that encoded the first pattern is used to encode the change bitmap. Finally, all encoded data are inserted into the wedgelets memory.

D. Line Change Map (LCM) Algorithm

The LCM algorithm presents high similarity with the BCM, however, instead of generating a block change map, it generates a line change bitmap for each line. Similar to the previous solution, the first line can be encoded using any technique or even be stored uncompressed. When next blocks are being encoded, the LCM between its first line and the last line of previous block is generated.

Fig. 9 presents an example of this technique, where the first line has been stored using the same Huffman coding tree employed to store the line change bitmap. The following lines are compared against the previous line, where every bit change is indicated with 1 and non-bit change is indicated with 0. This resulting line change map is encoded using Huffman code algorithm, and all data is stored in wedgelet memory.

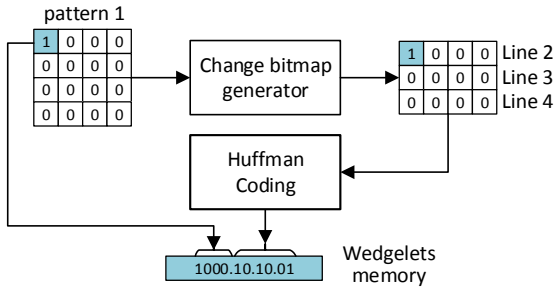


Fig. 9. Example of LCM application for a 4×4 block pattern.

IV. Experimental Results and Discussion

A. Simulation Results

All wedgelets patterns were extracted from 3D-HTM 16.0 and the proposed solutions were implemented using Matlab. Table IV presents the quantity of bits required to store all available wedgelets of all block sizes regarding standard DMM-1 and the four proposed solution.

TABLE IV. Quantity of bits required to store all wedgelets

Algorithm	4×4	8×8	16×16	Total
Standard	1,376	51,328	130,560	183,264
FB&C	1,032	25,664	40,800	67,496
Huffman	991	23,503	34,298	58,792
BCM	761	14,428	27,175	42,364
LCM	1,086	22,301	27,108	50,495

Fig. 10 shows the impact of the compression techniques having as reference the Standard DMM-1. The FB&C and Huffman techniques, which are the simpler solutions, are capable of achieving a compression rate of 63.17% and 67.92%, respectively. The highest compression rates are achieved applying BCM and LCM algorithms, where memory reduction of 76.88% and 72.45% are obtained, respectively.

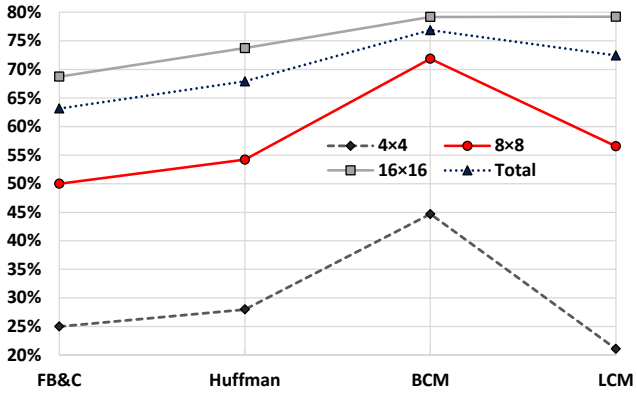


Fig. 10. Compression rates for all stored patterns of all proposed technique.

B. Implementation Impact on the Target Architecture

Memory reduction techniques usage should care about the impact on the target architecture. In this direction, it is important to evaluate the area and power consumption overhead. We synthesized the architectures that implement the decoder of the proposed solutions using ST 65nm standard cells technology for a frequency capable to achieve real-time passing by the entire initial wedgelet set for HD 1080p video processing. Table V depicts the area of the additional hardware that is necessary to implement each technique.

TABLE V. Area of the target architecture of each proposed technique (in gates)

Technique	4×4	8×8	16×16	32×32	Total
FB&C	19	25	67	65	176
Huffman	17	47	92	93	249
BCM	210	1089	6257	6257	13813
LCM	39	244	406	324	1013

The standard DMM-1 storage does not require additional hardware resources in its implementation while all of the proposed solutions requires an increase in hardware usage with a considerable amount reduction in memory size. The hardware implementation of FB&C and Huffman Code are the smaller, requiring only a small lookup table capable of converting the encoded lines into the uncoded line.

BCM is the solution that requires the highest area among our solutions, however, it is the only solution that delivers an entire NxN block per cycle. It requires N lookup tables to convert the encoded Huffman data to the change bitmap and NxN 1-bit registers to store the previous block pattern. Finally, an N×N-array of X-ORs should convert the previous pattern to the current pattern.

The LCM hardware design presents high similarity compared to BCM. However, it delivers a patterns line per cycle, requiring only one lookup table, N 1-bit registers and, N-array of X-ORs to convert the previous line to the current line.

Along with area evaluation, energy consumption results for memory access of each proposed technique have been analyzed, considering that 100pJ are consumed each time a byte is accessed in a DDR3 memory, according to [12]. Table VI presents some results of this evaluation.

TABLE VI. POWER DISSIPATION OF THE STANDARD DMM-1 AND THE PROPOSED TECHNIQUES (IN mW)

Technique	Resource	4×4	8×8	16×16	32×32	Total
Standard	Memory	49.77	250.39	304.82	304.82	909.79
FB&C	Memory	37.32	125.19	95.26	95.26	353.03
	Hardware	0.39	1.60	2.16	0.68	4.83
	Total	37.71	126.79	97.41	95.94	357.86
Huffman	Memory	35.84	114.65	80.08	80.08	310.65
	Hardware	0.14	1.19	1.08	0.43	2.84
	Total	35.99	115.84	81.16	80.51	313.49
BCM	Memory	27.53	70.38	63.43	63.43	224.78
	Hardware	0.42	2.30	3.23	1.05	7.00
	Total	27.95	72.69	66.66	64.49	231.78
LCM	Memory	39.28	108.79	63.28	63.28	274.63
	Hardware	0.47	3.42	3.82	0.63	8.34
	Total	39.75	112.21	67.10	63.91	282.97

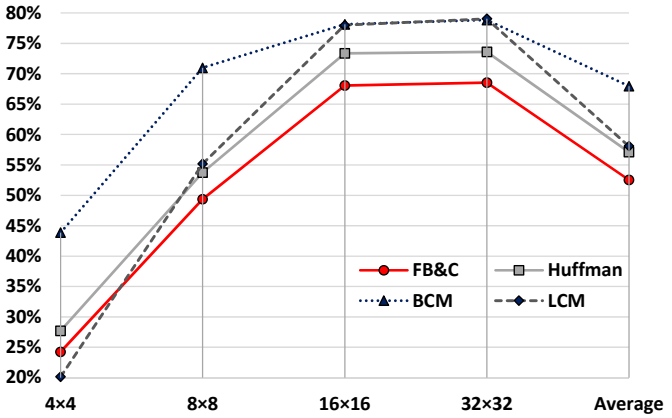


Fig. 11. Power dissipation decrease when using the proposed techniques.

The power dissipated from hardware overhead required by the proposed techniques is small, when compared to the power dissipated by the memory accesses. Fig. 11 illustrates the how much each proposed technique enable to reduce power when compared with the standard DMM-1 implementation.

As one can notice, the proposed techniques are capable of reducing from 20% to 45% when taking account small blocks, and from 68% to 80% for the larger blocks implying a total power dissipation decrease between 551 mW and 678 mW.

C. *Comparison with Related Work*

We only find the work [13] that proposed simplifications on the wedgelet set. In [13], a reduction of 27.8% in the memory size is obtained with a loss in the compression rate of 0.03%. This result is obtained storing the entire 16×16 wedgelet patterns and dynamically down-sampling these patterns for other block sizes. No energy information is given in [13]. The techniques proposed here are capable of providing more memory area reduction without impacting the compression rate of the encoding videos. Moreover, the proposed techniques provide a high decrease in power consumption, which is not demonstrated in [13].

V. CONCLUSIONS

This work presented four techniques to reduce the storage requirements of DMM-1 wedgelets. The standard technique store all patterns bit by bit, without considering the entropy of this information, resulting more than 180 Kbits to store the entire wedgelet set. The proposed techniques are capable of achieving a compression rate ranging between 63.17% and 76.88%, which represents a reduction between 112 and 140 Kbits using only simple and light-weight techniques. Additionally, the implementations of the proposed techniques result in a smaller area, fewer memory accesses and, consequently, a power consumption decrease.

REFERENCES

[1] JCT-3V. Available at //phenix.int-evry.fr/jct2, access in Ago. 2015.

[2] G. Sullivan et al. "Overview of the high efficiency video coding (HEVC) standard," Transactions on circuits and systems for video technology, v. 22, n. 12, pp. 1649-1668, Dec. 2012.

[3] P. Kauff, et al. "Depth map creation and image based rendering for advanced 3DTV services providing interoperability and scalability," *Image Communication*, v. 22, n. 2, pp. 217-234, Feb. 2007.

[4] C. Fehn. "Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV," *SPIE, Stereo-scopic Displays and Virtual Reality Syst*, v. 5291, pp. 93-104, May 2004.

[5] A. Smolic et al. "Intermediate view interpolation based on multiview video plus depth for advanced 3D video systems," *IEEE International Conference on Image Processing (ICIP)*, pp. 2448-2451, 2008.

[6] G. Sanchez et al. "Complexity reduction for 3D-HEVC depth maps intra-frame prediction using simplified edge detector algorithm", *International Conference on Image Processing*, pp. 3209-3213, 2014.

[7] G. Tech et al. "Overview of the Multiview and 3D extensions of High Efficiency Video Coding," *IEEE Transactions on Cicuits and Systems for Video Technology (TCSVT)*, v.26, n.1, pp. 35-49, Sep. 2015.

[8] G. Sanchez et al. "Real-time scalable hardware architecture for 3D-HEVC bipartition modes," *Journal of Real-Time Image Processing*, 2016.

[9] L. Zhang et al. "3D HEVC Test Model 6," ISO/IEC *JTC1/SC29/WG11*, Geneva, Oct. 2013.

[10] Z. Gu et al. "Fast Intra Prediction Mode Selection for Intra Depth Map Coding," *ISO/IEC JTC1/SC29/WG11*, Vienna, Aug. 2013.

[11] P. Merkle et al. "3D video: Depth coding based on inter-component prediction of block partitions," *Picture Coding Symposium (PCS)*, pp. 149-152, 2012.

[12] T. Vogelsang. "Understanding the Energy Consumption of Dynamic Random Access Memories," *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 363-374, 2010.

[13] S. Ma et al. "Reducing Wedgelet lookup table size with down-sampling for depth map coding in 3D-HEVC," *IEEE International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1-5, 2015.