Initialization, limitation and predictive coding of the depth and texture quadtree in 3D-HEVC

Elie Gabriel Mora, Joel Jung, Marco Cagnazzo, *Senior Member, IEEE*,
and Beatrice Pesquet-Popescu, *Fellow, IEEE*

***Abstract*—The 3D video extension of HEVC (3D-HEVC) exploits texture-depth redundancies in 3D videos using inter-component coding tools. It also inherits the same quadtree coding structure as HEVC, for both components. The current software implementation of 3D-HEVC includes encoder shortcuts that speed-up the quadtree construction process, but those are always accompanied by coding losses. Furthermore, since the texture and its associated depth represent the same scene, at the same time instant and view point, their quadtrees are closely linked. In this paper, an inter-component tool is proposed where this link is exploited to save both runtime and bits through a joint coding of the quadtrees. If depth is coded before the texture, the texture quadtree is initialized from the coded depth quadtree. Otherwise, the depth quadtree is limited to the coded texture quadtree. A 31% encoder runtime saving, a -0.3% gain for coded and synthesized views and a -1.8% gain for coded views are reported for the second method.**

***Index Terms*—3D Video Coding, inter-component prediction, depth quadtree limitation, texture quadtree initialization, predictive coding.**

I. Introduction

THREE dimensional video has undergone a rapid development in the past few years, from the release of blockbuster 3D movies to the emergence of new multimedia services such as 3D television [1] (3DTV) and Free Viewpoint Television [2] (FTV). While 3D video has not yet met its expected success (mainly due to viewing discomforts and the burden of wearing 3D glasses), it is expected to conquer the market with autostereoscopic viewing, which requires more views to be coded and transmitted, hence the multi-view video format (MVV). The multi-view video plus depth (MVD) format is an interesting alternative to MVV since it introduces depth views which are easier to encode, and which can be used to synthesize as many views as required at the receiver side using Depth Image Based Rendering (DIBR).

Up to now, the texture and depth components of a given 3D format have been encoded using traditional 2D encoders, such as H.264/AVC [3], HEVC [4], or the multiview extension of AVC, known as MVC [5]. These standards fail however to exploit all the intricate redundancies in the 3D information, as they were not initially designed to handle this type of data. For instance, depth videos in the MVD format have unique characteristics which are not exploited by these standards. Hence, there is a need for a true 3D video coding standard to achieve efficient coding of 3D data.

In 2011, MPEG addressed this need by issuing a Call for Proposals (CfP) for 3D Video Coding Technologies [6]. Following the response of the CfP, a joint collaborative team between ISO and ITU, called JCT-3V, has been formed. JCT-3V drafted two test models for a 3D video coding standard, one AVC-based (3D-AVC), and the other HEVC-based (3D-HEVC).

HEVC is a hybrid codec like its predecessors, but with added tools that increase coding efficiency. Nearly 50% bitrate reduction compared to H.264/AVC is achieved. HEVC uses a quadtree coding structure [7] which provides block size flexibility and granularity. While an HEVC input consists of only one color video, 3D-HEVC processes MVD data, consisting of multiple texture videos and their associated depth videos. A base view (texture + depth) is coded in HEVC for backward compatibility, while the other views, called dependent views, are coded with additional tools.

3D-HEVC introduces inter-view and inter-component coding tools for efficient coding of dependent views and depth data, respectively. While inter-view coding tools have already been introduced in the MVC standard, inter-component tools are a novel category of tools that are being intensively researched recently. Indeed, the texture and depth components of a 3D signal are highly correlated, and significant gains may be achieved by efficient predictions or inheritances of specific coding information from one component to the other.

3D-HEVC utilizes the quadtree-based coding structure introduced in HEVC [8] for both the texture and depth components. In the current reference software implementation of 3D-HEVC (called HTM), a complex Rate Distortion Optimization (RDO) process is performed at each level of the quadtree to determine the best coding mode and partition size for a coding unit (CU) at that level. The best depth level for the quadtree of a CU is also determined using an R-D check. Tools aimed at speeding up the quadtree construction by skipping some of these R-D checks were proposed in the reference software of HEVC (called HM), and are included in HTM, but these encoder shortcuts are always accompanied by coding losses.

Furthermore, existing inter-component tools are not designed to directly handle this quadtree coding structure. They can only influence its construction indirectly by favoring a specific inherited coding mode or prediction parameter

E.G. Mora is with Telecom ParisTech, 75634 Paris, France and also with Orange Labs, 92794 Issy Les Moulineaux, France (e-mail: elie.mora@orange.com).

J. Jung is with Orange Labs, 92794 Issy Les Moulineaux, France (e-mail: joelb.jung@orange.com).

M. Cagnazzo and B. Pesquet-Popescu are with Telecom ParisTech, 75634 Paris, France (e-mail: {cagnazzo,pesquet}@telecom-paristech.fr).

amongst others, which can lead to a different partitioning of a coding tree unit (CTU). In this paper, we propose a novel inter-component tool that utilizes either the coded texture or depth quadtree to control the construction and coding of the other component's quadtree. The aim is to reduce encoder runtime while simultaneously improving coding efficiency. The main idea is to force a depth CU to be less or equally partitioned than its co-located texture CU. This means that if the depth is coded first, the texture quadtree will be initialized from the depth quadtree. Otherwise, the depth quadtree will be limited to the texture quadtree. Both methods succeed in reducing encoder runtime, as confirmed by our simulation results. Additionally, a predictive coding of the quadtree of a component using the other component's quadtree is performed, and achieves coding gains. The depth quadtree limitation with its predictive coding part has been presented at the 2nd JCT-3V meeting, and was adopted in both the 3D-HEVC working draft and the HTM software [9].

The remainder of this paper is organized as follows. Section II presents inter-component tools that can be found in literature and in the 3D-HEVC test model. This section also details the quadtree coding structure in 3D-HEVC and presents tools designed to speed-up the construction of this structure. Section III analyzes the texture-depth quadtree relationship in order to prove the potential of the proposed texture quadtree initialization (QTI) and the depth quadtree limitation (QTL). Section IV presents QTI, QTL, and their associated predictive coding part. The results for QTI and QTL are given in Section V. Those results are analysed in Section VI. Section VII concludes this paper while underlining possibilities for future work.

II. Background

A. Inter-component coding tools

1) Inter-component tools in 3D-HEVC: For coding depth maps in 3D-HEVC, two inter-component tools are used so far: Depth Modeling Modes (DMMs) 3 and 4, and Motion Parameter Inheritance (MPI).

The DMMs were introduced in 3D-HEVC as four additional Intra modes, added to the list of existing angular and planar modes, for Intra coding of a depth CU [4]. DMMs predict a depth CU using two constant regions, seperated by a straight line [10]. The two prediction values across the constant regions are coded in the bitstream. The partition information can be coded (DMM mode 1), or inferred using spatial depth neighbours (DMM mode 2). In DMM modes 3 and 4, the partition information is inferred using reconstructed co-located CUs in texture (hence the inter-component aspect of the modes).

Furthermore, the motion vector correlations between the texture and the depth component are exploited with the Motion Parameter Inheritance (MPI) tool for depth coding [10]. In MPI, the motion vectors and, when the texture CU is smaller than the depth CU, the quadtree coding structure of the co-located texture CU, are considered for inheritance in the depth CU as an additional Merge candidate [8]. This introduces a dependency when parsing the depth, because of the need to check the sizes of both the texture and depth CUs before deciding to parse the split flag.

These inter-component tools increase coding efficiency as they bring a 1.8% bitrate reduction on synthesized views, and 0.6% on coded texture videos when considering the total bitrate (texture+depth), at the expense of a small increase of about 7% in encoder and decoder runtimes [11].

2) Inter-component tools in literature: View Synthesis Prediction (VSP) is an inter-component coding tool used to efficiently code side views, which is currently in 3D-AVC test model and candidate for adoption in 3D-HEVC. In VSP, the coded central base view is warped using its associated coded depth map to the side view position. This warped version of the base view is used to code the side view in two ways: in [12], the warped image allows to know which CUs in the side view can not be rendered, due to occasional disocclusions, and need to be coded. This implies significant changes in the coder, especially in the prediction stage in Inter and Intra coding modes. In [13], the warped image is added to the reference picture list of the side view, just before the inter-view reference picture, and the remaining coding process remains unchanged.

For depth coding, a tool called Depth Block Skip (DBS) was introduced in [14]. DBS is an inter-component coding tool used to force the coding of a depth block in SKIP mode if the temporal correlation in texture is found to be high enough. A depth block at time instant i is forced to be coded in SKIP mode if the sum of absolute difference (SAD) between the co-located block in texture at i and the co-located texture block at $i-1$ is lower than a certain threshold. Since the process can be exactly repeated at the decoder, the SKIP mode does not need to be signaled. The coding gains brought by this tool are however not significant enough to justify the use of two texture frames to code a depth block (hence increasing memory consumption).

In [15], the SKIP mode for depth is forced whenever the co-located texture block has been coded in SKIP mode. The rationale behind this is that the uniformity of motion in texture is likely to hold also in depth. This tool is able to eliminate the flickering in depth motion due to bad depth estimation and hence increase the quality of the synthesized views, though it still needs evaluation in 3D-AVC along side other inter-component coding tools.

Motion information can also be shared between texture and depth. In [16], a joint texture-depth motion estimation is performed where the Mean Square Error (MSE) involved in R-D Lagrangian cost computations is computed as a weighted sum of the texture and depth MSE. Only one motion vector field is thus transmitted. An improvement of the depth map reconstruction of more than 1 dB, for a small reduction in the texture video quality (between 0.4 and 0.8 dB) is possible when fine tuning the weight parameter.

The texture Intra mode can also be inherited for the currently coded depth prediction unit (PU). In [17], the Intra mode of the co-located texture PU is inherited and systematically placed in the most probable mode candidate list for the depth PU. This leads, in some cases, to the replacement of an efficient spatial Intra predictor. To avoid this, the inheritance can be performed only in PUs where sharp edges exists in

texture as in [18], because it is in those PUs where texture and depth Intra modes are most correlated.

Finally, spatial transforms can be specifically designed for depth. In [19], an adaptive wavelet lifting scheme is proposed wherein short filters are applied to areas in depth containing edges, and long filters applied in homogeneous depth areas. The edge detection is performed on texture for decodability, hence the inter-component aspect of the tool. Compared to the linear 5/3 filter bank with the JPEG2000 codec used for entropy coding, the tool brings significant gains on synthesized views (up to 1.2 dB quality increase).

Apart from the MPI tool, all inter-component tools in 3D-HEVC and in literature do not directly handle neither the construction, nor the compression of the quadtree coding structure for either the texture or the depth component, hence the novelty of our proposed inter-component methods.

B. Quadtree coding in 3D-HEVC

3D-HEVC inherits an advanced quadtree-based coding approach from HEVC [7], wherein a picture is divided into coding tree units (CTU). Those are the equivalent of macroblocks in previous video coding standards. The CTU can then be split into four coding units (CU), and each CU can be further split into four CUs, and so on. A specified maximum depth level is set to limit the CU split recursion.

A CU can further be partitioned into prediction units (PUs) (there are 8 different PU partition sizes, out of which 4 are asymetric, as shown in Figure 1) where each PU has its own prediction information. The PUs however cannot be further partitioned. Note that each CU is also the root of a transform

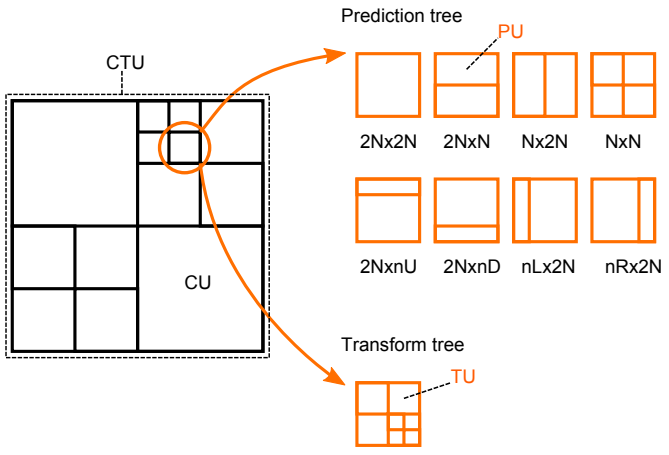


Fig. 1. Quadtree structure of a CTU and possible partition shapes

tree, known as residual quadtree (RQT).

Three syntax elements, sent for each CU, control the quadtree coding structure for texture and depth. First, the **split_flag** syntax element is sent to signal whether a CU has been split into four sub-CUs or not. If the CU is at the highest depth level, it cannot be further split, and hence the split flag is not sent. If the CU is not split, a **skip_flag** syntax element is sent to signal whether the CU has been coded in SKIP mode or not. If not, the **partition_size** syntax element signals the partition shape which has been selected for coding that CU.

In HTM, a Rate Distortion Optimization (RDO) process tests, for each CU, all partition sizes, except the N×N partition for depth levels lower than the maximum depth level. These partition sizes are tested with respective coding modes (Merge, Intra, Inter), and a Lagrangian cost is computed for each partition size - coding mode combination. The cost of splitting the CU is also computed and the configuration yielding the lowest cost is selected for that CU. Note that in HTM-4.0, the asymetric motion partitions (AMP) are disabled. However, the proposed depth quadtree limitation and texture quadtree initialization techniques are compatible with AMP.

C. Fast encoding methods

HTM includes three non-normative tools that directly impact the splitting and partitioning of CTUs. First, an Early Skip tool checks whether the R-D cost of the SKIP mode in a 2N×2N partition is lower than a certain threshold. If that is the case, no other prediction modes are tested and the CU is no longer split (the recursion is stopped at this level). Second, the Coding Block Flag (CBF)-based early termination [20], where after each Inter partition check, if there is no residual to code (i.e. if the CBF equals 0) in the CU, all subsequent Inter checks (except Inter mode in N×N) and Intra checks are no longer tested. Finally, the Early CU Termination tool (ECU) stops the CU split at a certain depth level if the SKIP mode in 2N×2N turned out to be the best mode at that level [21].

In JMVM (reference software of the MVC standard), these three tools have also been implemented and tested [22]. A technique that categorizes macroblocks (MBs) into either simple or complex mode regions MBs, and in which Inter mode checks (including time-consuming motion and disparity estimations) for the first category are skipped, is proposed as well in [23]. Also in JMVM, the Previous Disparity Vector Disparity Estimation (PDV-DE) and Stereo-Motion Consistency Constraint Motion and Disparity Estimation (SMCC-MDE) shortcuts are proposed in [24] to reduce the search range for disparity estimation.

Recently, the Enhanced Depth CU (EDCU) [25] shortcut has been proposed for the depth component only, where the recursive split of the depth CU is stopped if the best mode of the current depth CU is SKIP and if the co-located texture CU was encoded in SKIP mode as well. Other tools [26], [27] attempt to reach a good complexity-performance trade-off by implementing algorithms to make more intelligent early CU termination decisions.

However, all these tools are aimed at reducing encoder runtime at the expense of a decrease in coding efficiency as they do not allow any efficient coding of the quadtree syntax elements. Furthermore, these tools would not be able to exploit the relationship between the texture and the depth quadtrees as they are purely 2D coding tools. The proposed methods exploit this relationship to both reduce the encoder runtime and achieve coding gains.

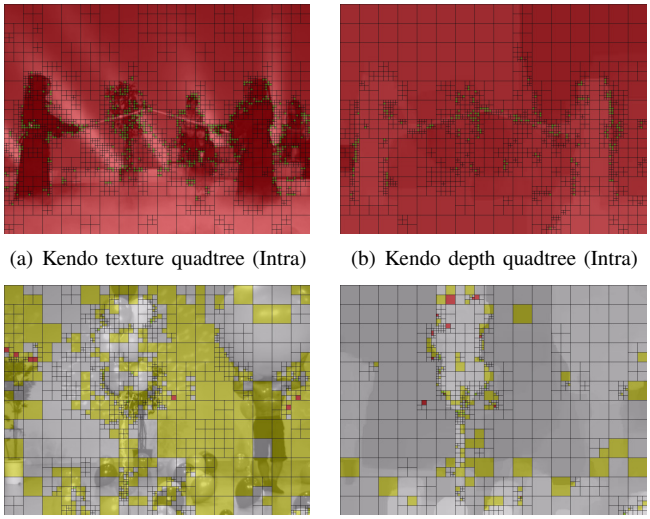
III. Motivation

A. Comparison of the texture and depth quadtree

Our work in this paper is based on the following assumption:

Assumption 1. *A texture CU is at least as partitioned as its co-located depth CU.*

Indeed, a depth map is a grayscale Luma-only image which accounts for objects distance from the camera. The geometric information that the depth map conveys is used to synthesize intermediate views using DIBR. A depth map is essentially composed of large planar regions seperated by sharp edges and is invariant to illumination, patterned textures, and shadows [19], [28]–[32]. Furthermore, fine partitioning is usually performed along edges, to account for the lack of a correct prediction for a CU. Since texture has more edges due to illumination changes, patterned textures and shadows, it is thus in general more partitioned than depth. This can be seen in Figure 2 which presents the texture and depth coding and prediction quadtrees at QP 25 in an Intra and an Inter frame of the Kendo and Balloons sequences respectively.



(a) Kendo texture quadtree (Intra) (b) Kendo depth quadtree (Intra)

(c) Balloons texture quadtree (Inter) (d) Balloons depth quadtree (Inter)

Fig. 2. Texture and depth quadtree partitions for a Kendo Intra frame and a Balloons Inter frame at QP 25 (best viewed in color)

Table I gives the percentage of CUs where Assumption 1 fails for four tested QPs and for one GOP of the sequences considered (assumptions 2 and 3 will be discussed in Section IV). The GOP consists of an 8-frame pyramid, with I, P and hierarchical B pictures. The test was done under HTM-4.0 in the same coding configuration and test conditions as described later in Section V-A. We can see from this table

Sequence	Assumption 1				Assumption 2				Assumption 3			
	25	30	35	40	25	30	35	40	25	30	35	40
Kendo	38	15	6	2	15	4	1	0	43	18	8	3
Newspaper	56	31	16	6	27	13	6	1	60	34	17	7
Balloons	40	19	7	4	17	7	1	0	44	21	8	5
Dancer	15	16	10	4	3	5	3	0	21	20	12	5
GT Fly	21	11	4	2	5	4	1	0	25	12	4	2
Poznan Hall2	32	11	4	2	12	3	1	0	34	13	4	2
Poznan Street	31	13	6	2	9	3	1	0	36	15	7	2

TABLE I
PERCENTAGE OF CUS PER SEQUENCE AND PER QP WHERE ASSUMPTIONS 1, 2 AND 3 FAIL

that the assumption seems reasonable. It holds particularly well for the Dancer and the GT Fly sequences because they are computer-generated sequences with very clean depth maps. Indeed, the depth maps of the other sequences considered are estimated using stereo matching algorithms and thus, they inherently contain artefacts. These artefacts sometimes come out as false edges in the depth map which cause a fine partitioning of an area that is supposed to be flat. This area would however be coarsly partitioned in texture, hence breaking the initial assumption. These false edges are smoothed out at lower bitrates, which is why the assumption failure percentage decreases from QP 25 to QP 40.

Furthermore, the assumption does not hold if we have two adjacent objects in texture with similar luminance and chrominance but with different depths. In that case, the CUs containing both objects will be split in depth and not in texture. Particular motions (zooming for instance) also break the assumption. However these cases are rare, and this is confirmed by the results of Table I.

B. Analysis of the quadtree coding cost

Table II shows the percentage of bits used to code the split flag and the partition size per slice type and for one entire GOP (in the texture and depth bitstreams). The same software basis, test conditions and coding configuration as the ones used to get the results of Table I were used in order to get these percentages. Results shown here are averaged across four QPs (25, 30, 35 and 40).

Sequence	Texture				Depth			
	I	P	B	GOP	I	P	B	GOP
Kendo	3.2	6.1	16.6	7.5	8.2	12.1	19.4	13.3
Newspaper	3.1	7.4	22.0	5.7	9.4	14.3	20.3	13.6
Balloons	3.2	8.0	20.8	6.5	9.3	14.5	18.6	13.4
Dancer	2.9	9.1	17.9	7.0	14.6	22.8	24.2	21.9
GT Fly	4.2	11.1	21.5	8.8	10.6	17.3	18.9	16.1
Poznan Hall2	4.7	8.6	16.2	8.3	13.3	18.2	18.4	17.2
Poznan Street	3.7	7.7	14.8	7.0	10.4	15.8	20.4	15.5
Average	**3.6**	**8.3**	**18.5**	**7.2**	**10.8**	**16.4**	**20.0**	**15.8**

TABLE II
PERCENTAGE OF BITS PER SLICE TYPE FOR THE "SPLIT_FLAG" PLUS THE "PART_SIZE" ELEMENTS IN THE TEXTURE AND DEPTH BITSTREAMS

This table shows that there is a significant amount of bits used to code the split flag and the partition size. The bitrate percentage for these two elements is especially high on P and B slices for depth and on B slices for texture. And although on average, the percentage in the GOP is higher for depth, our experiments show that the texture as a whole represents around 90% of the entire texture+depth bitstream in HTM-4.0. Consequently the quadtree information of the texture represents 6.5% ($7.2\% \times 0.9$) of the entire bitstream, while the quadtree information for depth represents only 1.6% ($15.8\% \times 0.1$).

C. Conclusion

Since the assumption seems reasonable judging from Table I, the encoder runtime is expected to be reduced if the quadtree structure of a texture CTU is directly initialized from

that of the depth or if the quadtree structure of a depth CTU is limited to that of the texture. The encoder runtime reduction can be evaluated fairly in both components since the texture and depth runtimes are roughly the same in HTM-4.0 (coding all texture views takes, on average, 55% of the total encoding time of the sequences, while coding the depth views takes the remaining 45%).

Furthermore, Table II shows that the cost of the quadtree syntax element is significant, especially in texture. There is much to be gained from an efficient coding of these elements.

Consequently, a texture quadtree initialization using depth and a depth quadtree limitation using texture, with a quadtree predictive coding scheme in both methods can achieve both encoder runtime reduction and coding gains.

IV. Proposed methods

A. Texture quadtree initialization

1) Proposed scheme: When the depth is coded before the texture, the texture encoder has access to the quadtree information (split flag and partition size) of the co-located CU in depth to control the quadtree of a currently coded texture CU. Based on our assumption, we propose to force a texture CU to be at least as partitioned as its corresponding depth CU. In other words, the quadtree of the texture CU is *initialized* from the quadtree of the depth CU; it can be further partitioned but not less. This has two benefits: it can reduce encoder runtime since certain coding modes are no longer checked, and allow a predictive coding of the quadtree syntax elements of the texture which, in turn, will increase coding efficiency. The rest of this section will detail the texture quadtree initialization (QTI) and its associated predictive coding part (PC).

Figure 3 shows all possible depth CU partitions and the allowed texture CU partitions in each case, at a specific depth level L. It does not show the allowed texture sub-CU partitions at level $L+1$ if the CU at depth level L was split. Indeed, this scheme is recursive; it can be applied the same way for each sub-CU.

If the depth CU is split or has a partition size of N×N , as in case (a), the texture CU is forced to be split or partitioned in N×N respectively. Other partitions are not checked, and no split flag or partition size is sent for the texture CU. In case (b) and (c), the only partition allowed for the texture CU is N×2N (respectively 2N×N). Splitting the texture CU is also possible. Hence, the encoder only sends the split flag. If the CU is not split, the partition size is inferred from the depth CU. In case (d), all texture modes and partitions are checked, the encoder needs to send both the split flag and the partition size in this case. Note that in all cases, splitting the texture is always an option. Hence, in QTI, the recursion is never stopped at any level, it is the starting level that is set.

2) Flexible QTI+1 variant: The proposed scheme sometimes leads to a sub optimal (R-D wise) partitioning of a texture CU as it can force an unnecessary split or partition when an assumption failure occurs. The texture quadtree is said to be altered, meaning it has changed from what the RDO process intended it to be. We define the severity of a scheme in QTI by the amount of forced split and partitions

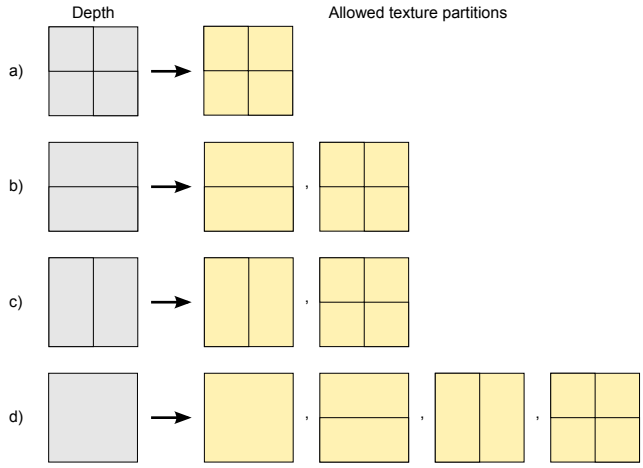


Fig. 3. Allowed texture partitions in QTI

the scheme imposes in a CTU. The least severe scheme is where no forcing is done (reference coding). The most severe scheme is where each CU is forced to be split or partitioned in N×N. Forcing splits and partitions however removes the need of sending split flags and / or partition sizes. Consequently, the more severe the scheme is, the more it alters the texture quadtree, but the more bitrate reduction it allows as well. An efficient coding scheme has a severity level that achieves a good compromise between amount of alteration and amount of bitrate reduction.

In this work, we propose a less severe scheme for QTI that we call QTI+1 which relies on the following assumption:

Assumption 2. *A texture CU is at least one depth level less partitionned than its co-located depth CU*

This scheme forces to split the texture CU at depth level L only if the depth CU is split at depth level $L+1$ (hence the name), as shown in Figure 4. The split flag for the texture CU does not need to be sent in this case. In any other case, the decision is left for the R-D based optimization process, and consequently the split flags and partition sizes need to be sent. Compared to Assumption 1, Assumption 2 fails less often as can be seen in Table I. Hence, this flexible scheme alters the texture quadtree less often but at the expense of a lower bitrate reduction potential. Note that this scheme is recursive as well, it can be applied in the same way for each sub-CU at depth level $L+1$ if the CU at level L is split.

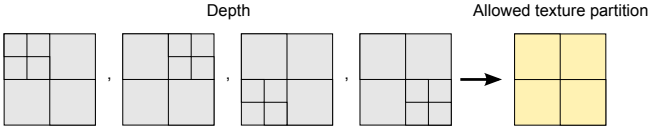


Fig. 4. The QTI+1 variant

B. Depth quadtree limitation

1) Proposed scheme: When texture is coded first, the depth encoder has access to the split flag and the partition size of the co-located texture CU to control the quadtree of a

currently coded depth CU. The same assumption as in QTI is considered: a depth CU is, at most, as partitioned as its co-located texture CU. In other words, the depth quadtree is *limited* to that of the texture. This also has two benefits since encoder runtime can be reduced, and coding efficiency increased. The rest of this section will detail the depth quadtree limitation (QTL) and its associated predictive coding part (PC).

Figure 5 shows the allowed depth partitions per possible texture partition in QTL. If the texture is partitioned either in 2N×2N, N×2N, or 2N×N as in case (a), the depth is forced to be in 2N×2N. The encoder will not check smaller partition sizes, and will not try to split the depth CU. Also, it does not need to send the split flag or partition size for that CU. In case (b), all partition sizes are checked and splitting the depth CU is allowed. The split flag and partition size need to be sent. This case does not yield any encoder runtime reduction nor coding gains compared to a reference coding.

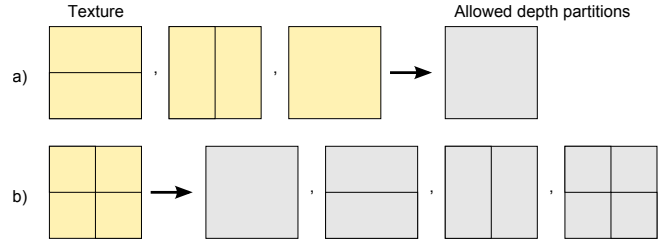


Fig. 5. Allowed depth partitions in QTL

2) Strict QTL-1 variant: The proposed depth quadtree limitation scheme has a certain level of severity. Just like in QTI, we can adjust this severity level to obtain a better trade-off between the amount of quadtree alteration and the potential of bitrate reduction. A more severe scheme for QTL was thus studied in this work, which relies on the following assumption:

Assumption 3. *A depth CU is at most one depth level less partitionned than its co-located texture CU*

In this scheme, the depth CU at depth level L is only allowed to be split or rectangularly partitioned if the texture is split at depth level $L+1$. In any other case, the depth CU is forced to be partitioned in 2N×2N. This scheme, called QTL-1, is more severe than QTL, because no split flags or partition sizes are sent for both cases of the original QTL scheme. However, the alteration level is also higher since there are more assumption failures, as can be seen from Table I.

C. Impact on the codec architecture

QTL+PC and QTI+PC include a predictive coding scheme where the split flag and/or the partition size for a CU are not sent in some cases. Similarly as for the MPI tool, this implies a parsing dependency since the decoder needs to check the texture or depth quadtree to know whether to parse or not the two syntax elements. It is most likely that any realistic implementation of 3D-HEVC will perform at most a parallel parsing with one CTU delay given that parallel decoding is forbidden by the following tools: DMM and MPI. This is possible with both QTL+PC and QTI+PC.

On the encoder side, this whole set of tools enables a one CTU delay parallel encoding, whatever the component coding order. This one CTU delay parallelization strategy also enables a very limited increase of the memory requirements, resulting from the storage of the quadtree information due to QTL, MPI or QTI, tiny compared to other data to be stored at the CTU level.

V. Experimental results

A. Experimental setting

We have implemented QTI, QTL, their predictive coding parts, and their variants in HTM-4.0 [33]. This software first codes a center view 0, then two side views, 1 and 2. Each view is composed of both the texture and depth components. The coding order is the following: T_0 D_0, T_1 D_1, T_2 D_2 (where T_i and D_i are respectively the texture and depth frames in the i^{th} view). This configuration is appropriate to test QTL+PC.

This software includes a tool called Flexible Coding Order (FCO) [34]. The FCO tool allows to change the coding order to T_0 D_0, D_1 T_1, D_2 T_2, which is appropriate to test QTI+PC for the two side views. FCO cannot change the order to D_0 T_0, D_1 T_1, D_2 T_2 since the processing of the base view needs to remain HEVC compatible. Hence QTI+PC cannot be applied for the base view. In our QTI experiments, the coding of the base view remains consequently unchanged. It is important to note that we could have applied QTL+PC in the base view to code D_0, but that would not allow for a fair evaluation of QTI since results would be mixed in with QTL results.

Aside from the changed coding order for QTI, we have strictly followed all the conditions in the Common Test Conditions (CTC) for HTM experiments defined by JCT-3V [35]. A Group Of Pictures (GOP) of 8 was considered with an Intra period of 24. An 8-bit internal processing was chosen. The maximum coding unit depth was set to 4, and the maximum coding unit size was set to 64×64. Furthermore, the following encoder shortcuts were enabled: fast encoder control (FEN), fast decision for Merge RD cost (FDM). For side view coding, disparity compensated prediction and multiview motion vector prediction were enabled. For depth coding, MPI, DMM (inter-component tools) and view synthesis optimization (VSO) were all enabled to conform to CTCs.

The following QP combinations for texture and depth (respectively) were considered: (25;34), (30;39), (35;42) and (40;45). We have tested our tools on seven sequences defined in the CTCs (1920×1088 and 1024×768). The experiments were done on 10 seconds of video length. Each sequence is composed of three views: the left, the center (coded first) and the right view. After encoding, three synthesized views were rendered between each view. PSNRs on synthesized views were measured with respect to views synthesized using uncompressed original views. Coding gains correspond to bitrate reductions evaluated using the Bjontegaard metric (BD-Rate) [36]. The experiments were launched on a cluster of servers, each having 8 CPU cores, and 10 GB of RAM, some even having 16 GB of RAM. To present the results, we have used the standard table template (which defines standard columns) given in the common test conditions [35].

Sequence	Video				Synt.	Coded+Synt.	Runtimes	
	0	1	2	Avg			Enc	Dec
Balloons	0.0	2.6	2.2	0.9	0.8	0.8	91	100
Kendo	0.0	3.4	3.5	1.4	1.1	1.1	87	96
Newspaper	0.0	5.3	3.8	1.7	1.4	1.4	87	99
GT Fly	0.0	3.0	3.5	0.8	0.7	0.7	87	100
Poznan Hall2	0.0	2.8	2.6	1.1	0.9	0.9	92	99
Poznan Street	0.0	1.8	2.0	0.6	0.5	0.5	93	100
Dancer	0.0	2.4	2.5	0.7	0.6	0.6	92	99
Average	**0.0**	**3.0**	**2.9**	**1.0**	**0.9**	**0.9**	**90**	**99**

TABLE III
BD-RATE CODING RESULTS PER SEQUENCE, IN %, OF QTI

Sequence	Video				Synt.	Coded+Synt.	Runtimes	
	0	1	2	Avg			Enc	Dec
Balloons	0.0	1.7	1.4	0.6	0.5	0.5	90	100
Kendo	0.0	1.9	2.1	0.8	0.7	0.7	91	97
Newspaper	0.0	3.0	2.3	1.0	0.9	0.8	87	93
GT Fly	0.0	1.7	2.2	0.5	0.4	0.4	94	103
Poznan Hall2	0.0	1.3	1.6	0.6	0.5	0.5	104	101
Poznan Street	0.0	1.1	1.1	0.3	0.3	0.3	97	101
Dancer	0.0	1.1	1.1	0.3	0.3	0.3	101	108
Average	**0.0**	**1.7**	**1.7**	**0.6**	**0.5**	**0.5**	**95**	**100**

TABLE IV
BD-RATE CODING RESULTS PER SEQUENCE, IN %, OF QTI+PC

Sequence	Video				Synt.	Coded+Synt.	Runtimes	
	0	1	2	Avg			Enc	Dec
Balloons	0.0	0.1	0.0	0.0	0.0	0.0	87	105
Kendo	0.0	0.2	0.2	0.1	0.1	0.1	86	96
Newspaper	0.0	0.1	0.1	0.0	0.1	0.1	86	92
GT Fly	0.0	0.3	0.6	0.1	0.1	0.1	92	117
Poznan Hall2	0.0	-0.1	0.0	0.0	0.0	0.0	105	101
Poznan Street	0.0	-0.2	0.2	0.0	0.0	0.0	102	99
Dancer	0.0	0.1	0.1	0.0	0.0	0.0	101	99
Average	**0.0**	**0.1**	**0.2**	**0.0**	**0.1**	**0.0**	**94**	**101**

TABLE V
BD-RATE CODING RESULTS PER SEQUENCE, IN %, OF QTI+1+PC

Sequence	Depth	Video total	Synt.	Coded+Synt.	Runtimes	
					Enc	Dec
Balloons	-15.7	-1.4	1.3	0.4	68	98
Kendo	-11.5	-1.3	0.9	0.2	64	98
Newspaper	-15.1	-1.8	2.0	0.7	64	100
GT Fly	-20.6	-1.2	0.0	-0.4	62	100
Poznan Hall2	-24.5	-2.3	0.6	-0.3	69	98
Poznan Street	-15.3	-1.0	0.1	-0.2	74	87
Dancer	-37.8	-1.0	0.2	-0.2	65	99
Average	**-20.1**	**-1.4**	**0.7**	**0.0**	**66**	**97**

TABLE VI
BD-RATE CODING RESULTS PER SEQUENCE, IN %, OF QTL

B. QTI results

Tables III, IV and V present the BD-Rate coding results of QTI, QTI+PC, and the variant QTI+1+PC (positive values are losses). The video column shows the BD-Rate coding results for the three coded views. Only the texture PSNR and the texture bitrate are considered in this computation. The "Synth" column shows the BD-Rate coding results for synthesized views. The bitrate considered is the sum of the bitrates of the three coded texture and depth views. The PSNR is the average PSNR for the 6 synthesized views. The "Coded+Synth" column shows the BD-Rate coding results for coded and synthesized views. The same bitrate as in the previous column is considered, but the PSNR is the average of the PSNRs of the 3 coded and the 6 synthesized views. The tables also provides encoding and decoding runtimes (time taken to code / decode both texture and depth) compared to the reference. QTI succesfully reduces encoder runtime by 10% because certain coding modes and partition sizes are no longer checked for CUs, as explained in Section IV-A1. This is however accompanied by around 1% bitrate increase on coded views and on synthesized views.

When the predictive coding part is added, the BD-Rate losses are largely reduced. A bitrate reduction of 1.3% and 0.4% is achieved on side views and on coded and synthesized views respectively compared to QTI alone. This comes with a small increase in encoding runtime due to memory accesses to the depth component and additional checks, making a total runtime reduction of 6%. The bitrate reductions however are not large enough to compensate all the losses in QTI, hence the remaining overall BD-Rate losses in QTI+PC.

The QTI+1 variant relaxes the severity of the scheme by only forcing to split a texture CU if the co-located depth CU is split at the next depth level. Here, the texture quadtree is less altered but the bitrate reduction potential is also reduced. QTI+1+PC still achieves the same encoder runtime savings as QTI+PC but with only a small coding loss of 0.1% on coded and synthesized views.

C. QTL results

1) Objective results: Tables VI, VII and VIII shows the BD-Rate coding results and runtimes of QTL, QTL+PC, and QTL-1+PC. Notice that in these tables, the "Video" columns are removed since QTL does not affect texture data. A new column, "Depth", is added to show BD-Rate coding results evaluated only on the depth component. Only the depth PSNR and depth bitrate are considered in this computation. Another column, "Video total" is added wherein the average PSNR of the coded texture videos is measured against the total texture+depth bitrate. Since QTI does not affect depth, these two columns are not shown in Tables III, IV and V.

As seen in Table VI, QTL achieves a significant 34% encoder runtime reduction, while introducing coding losses on

Sequence	Depth	Video total	Synt.	Coded+Synt.	Runtimes	
					Enc	Dec
Balloons	-18.2	-1.7	1.0	0.1	76	102
Kendo	-14.3	-1.6	0.6	-0.2	68	99
Newspaper	-17.9	-2.3	1.6	0.3	66	100
GT Fly	-23.7	-1.4	-0.2	-0.6	70	99
Poznan Hall2	-28.1	-2.9	0.0	-0.9	65	97
Poznan Street	-18.7	-1.2	-0.1	-0.5	69	87
Dancer	-40.6	-1.3	-0.1	-0.5	69	99
Average	**-23.1**	**-1.8**	**0.4**	**-0.3**	**69**	**97**

TABLE VII
BD-RATE CODING RESULTS PER SEQUENCE, IN %, OF QTL+PC

Sequence	Depth	Video total	Synt.	Coded+Synt.	Runtimes	
					Enc	Dec
Balloons	-24.7	-2.2	1.6	0.4	71	103
Kendo	-15.8	-2.0	1.4	0.3	65	98
Newspaper	-23.4	-2.9	2.8	0.9	66	103
GT Fly	-37.3	-2.0	0.2	-0.5	67	99
Poznan Hall2	-35.3	-3.6	0.6	-0.7	61	99
Poznan Street	-25.5	-1.6	0.2	-0.3	70	88
Dancer	-47.3	-1.9	0.6	-0.2	64	99
Average	**-29.9**	**-2.3**	**1.1**	**0.0**	**66**	**98**

TABLE VIII
BD-RATE CODING RESULTS PER SEQUENCE, IN %, OF QTL-1+PC

Test	Sequence + QP	Viewer									AVG	AVG / Test
		1	2	3	4	5	6	7	8	9		
3D	Balloons 25	0	-1	-1	-1	0	0	-1	-1	0	-0.6	-0.1
	Newspaper 25	0	0	0	1	0	0	0	0	0	0.1	
	Balloons 35	1	1	0	0	0	1	1	0	0	0.4	
	Newspaper 35	0	0	-1	0	0	0	0	-1	0	-0.2	
2D	Balloons 25	1	0	0	0	0	0	0	1	0	0.2	-0.2
	Newspaper 25	-1	-1	-1	-1	0	0	-1	0	-1	-0.7	
	Balloons 35	0	0	0	1	-1	-1	0	1	0	0	
	Newspaper 35	0	-1	0	-1	-1	1	0	-1	0	-0.3	

TABLE IX
CODING SCORES FROM SUBJECTIVE VIEWING EXPERIMENTS FOR 2D AND 3D TESTS

Method	Video				Video total	Synt.	Enc
	0	1	2	Avg			
QTL+PC	0.0	0.0	0.0	0.0	-1.8	0.4	69
ECU-D	0.0	0.0	0.0	0.0	-2.2	1.1	76
EDCU	0.0	0.0	0.0	0.0	-1.4	0.4	82
CBF-D	0.0	0.0	0.0	0.0	0.0	0.3	85
QTI+1+PC	0.0	0.1	0.2	0.0	0.0	0.1	94
ECU-T	0.3	-1.2	-1.1	0.5	0.7	-0.1	63
CBF-T	0.9	0.5	0.5	0.9	1.0	0.7	72

TABLE X
AVERAGE BD-RATE CODING RESULTS OF QTI+1+PC, QTL+PC AND OTHER RECENT ENCODER SHORTCUTS

synthesized views. The predictive coding part brings an additional 0.3% bitrate reduction in both the synthesized and the coded+synthesized columns but with a little runtime penalty due to the additional access to texture data. The QTL-1+PC variant is a more radical scheme: it achieves 34% encoder runtime reduction, but at the expense of a bigger coding loss of 1.1% on synthesized views. Consequently, QTL+PC achieves a better coding gain vs. encoder runtime reduction trade-off than QTL-1+PC.

2) Subjective results: Table VII shows that QTL+PC gives losses, on average, on synthesized views. These are due to smoothing out wrong edges in depth, which is in fact an improvement brought by the tool. To show that QTL+PC does not add any new artefacts on synthesized views, a subjective viewing session was conducted during the 2nd JCT-3V meeting [37]. A 3D and a 2D test were performed. For each test, two sequences (Balloons and Newspaper) at two QPs (25 and 35) each were evaluated. QP 25 was selected because it yields the highest coding losses on synthesized views. QP 35 was selected to provide complementary information for lower bitrates.

In the 3D test, a stereo sequence was constructed from the two synthesized views that are closest to the center view, and was projected onto a 3D screen. In the 2D test, the closest synthesized view to the left of the center view was selected for projection. These synthesized views were rendered from texture and depth coded using the reference software HTM-4.0 on the one hand, and with our tool enabled on the other hand. Hence two sequences, A and B, were projected to nine (5 JCT-3V experts, and 4 non-experts) viewers in this order: A B A B. The viewers did not know what A and B corresponded to. Actually, A was set as the reference for some viewings and as the proposed method for others. The viewers were asked to rate if A is largely better (a score of +3 is given), better (+2), slightly better (+1), same as (0), slightly worse (-1), worse (-2) or largely worse (-3) than B. Table IX shows the coding scores for the 2D and the 3D test, where a negative value represents an improvement resulting from the use of the proposed QTL+PC method.

In both the 2D and the 3D test, no score above 1 or below -1 is reported. It is thus confirmed that no new annoying artifact is noticeable. In the 3D test, results were particularly consistent: for each sequence, there was either no difference, or one of the method claimed as slightly better. Said differently, for one given experiment, it is not possible to find a score of +1 and -1 simultaneously. The proposed method is even favored with an average score of -0.1 and -0.2 in the 3D and 2D case respectively. This confirms that smoothing out the wrong edges in QTL+PC actually tends to be beneficial.

D. Comparison with state-of-the-art encoder shortcuts

Table X shows the average coding gains of QTL+PC and QTI+1+PC compared to state-of-the-art encoder shortcuts: ECU, CBF and EDCU, defined in Section II-C. ECU and CBF are 2D video shortcuts, applicable to both texture and depth components. Thus, for fairness of evaluation, we have compared ECU and CBF applied on texture only (resp. depth only) with QTI+PC (resp. QTL+PC). EDCU is a depth only encoder shortcut and was thus compared only with QTL+PC.

Table X shows that QTL+PC achieved more bitrate and encoder runtime reductions than its competitors. For texture coding, ECU-T and CBF-T achieved more encoder runtime reductions than QTI+1+PC, at the expense of a bigger loss in the "Video Total" column.

VI. RESULTS INTERPRETATION AND ANALYSIS

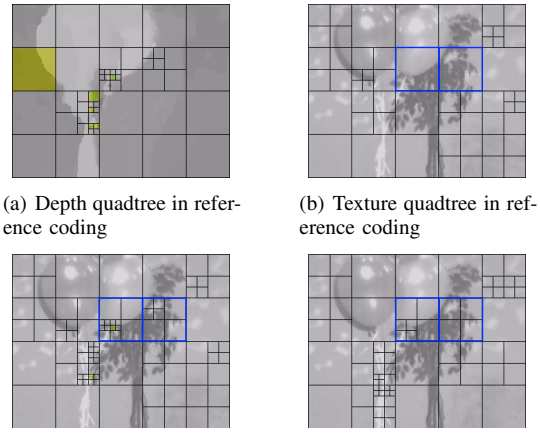
A. Analysis of QTI results

Table XI shows in which cases encoder runtime savings, bitrate savings and coding losses occur in QTI+PC. When a depth CTU is homogeneous (not split), QTI+PC does not impact the texture quadtree construction nor its coding. When a depth CTU contains an edge and is split accordingly, if the texture CTU is split as well, QTI brings encoder runtime savings since coding modes and partition sizes are no longer checked for low depth level CUs. The predictive coding part further brings bitrate reductions since the split flag and partition size for low level CUs are not sent in the bitstream. If

CTU		Texture	
		Homogeneous 64×64	Textured/Edge 8×8
Depth	Homogeneous 64×64	QTI+PC: no impact QTL+PC: runtime saving + bitrate reduction	QTI+PC: no impact QTL+PC: no impact
	Textured/Edge 8×8	QTI+PC: coding loss + runtime saving QTL+PC: runtime saving + bitrate reduction + synth. PSNR drop	QTI+PC: bitrate reduction + runtime saving QTL+PC: bitrate reduction

TABLE XI
ANALYSIS OF IMPACT ON CODING LOSS, BITRATE REDUCTION AND ENCODER RUNTIME IN QTI+PC AND QTL+PC UNDER DIFFERENT COMBINATIONS OF DEPTH AND TEXTURE CTUS

the texture CTU is homogeneous while the depth CTU is split, our initial assumption no longer holds. Runtime savings are achieved but coding losses occur since the texture quadtree is altered. This is shown in Figure 6: while the highlighted texture CTUs are homogeneous in reference coding as seen in Figure 6(b), they become as partitioned as the co-located depth CTUs with QTI+PC, as seen in Figure 6(c). This situation is relatively unfrequent, as can be seen from Table I, but it is responsible for the coding losses shown in Table III for QTI alone.



(a) Depth quadtree in reference coding

(b) Texture quadtree in reference coding

(c) Texture quadtree with QTI+PC

(d) Texture quadtree with QTI+1+PC

Fig. 6. Texture quadtree in reference coding, with QTI+PC and QTI+1+PC. Gray CUs are coded in SKIP mode, and green CUs in Inter (best viewed in color)

We can see from Table XI that encoder runtime savings in QTI only occur when the depth CTU is split. Actually, they are possible only if a depth CU is split or partitioned in N×2N or 2N×N, which correspond to cases (a), (b) and (c) respectively in Figure 3. The percentage of these three cases, that we call FT for Favorable for Texture, is shown in Table XII for each sequence. This table shows that overall, the percentage of CUs wherein encoder runtime savings are possible is only 9.2% which explains why the runtime savings in QTI were not larger than 10%.

Furthermore, the predictive coding part of the tool brings bitrate reductions only on those 9.2% of CUs as well. Consequently, the amount of bitrate reduction is not sufficient to compensate all the losses in QTI that are due to assumption failures. Hence, coding losses are still reported in QTI+PC. In QTI+1+PC, the more flexible scheme reduces the alteration of the texture quadtree while decreasing the FT percentage in the process. Figure 6(d) indeed shows that QTI+1+PC allowed the highlighted texture CTUs, to be one level less partitioned than the depth CTUs, hence reducing the texture quadtree alteration compared to QTI+PC, and the losses that come along with it. The resulting trade-off nearly removes all coding losses (only a small 0.1% loss is reported).

Sequence	**FT**	**FD**
Kendo	10.3	82.1
Newspaper	11.4	87.0
Balloons	8.8	87.5
Dancer	11.4	80.9
GT Fly	7.4	84.5
Poznan Hall2	5.5	89.2
Poznan Street	9.8	82.5
Average	**9.2**	**84.9**

TABLE XII
THE FT AND FD PERCENTAGES PER SEQUENCE

When applied on the texture component only, ECU reduces significantly the encoder runtime by 37% because it is applied often. It gives losses on the central view but significant gains on the dependent views. Indeed, a significant PSNR drop is noticeable in these views, but well compensated by an equally significant bitrate reduction. This PSNR drop is no longer compensated when taking into account the PSNR and bitrate of the central view hence the 0.5% loss in the "Video Avg" column in Table X. When also taking into account the depth bitrate in the BD-Rate computation, the results are even worse (0.7% loss). QTI+1+PC is applied less often than ECU, hence only reducing encoder runtime by 6%. However, it also does not alter the texture enough to cause such a major PSNR drop, hence not presenting any coding loss when evaluating the coded views PSNR against the coded views bitrate or against the coded views + depth bitrate. In mobile applications for instance where encoder runtime savings are primordial, ECU on texture could be preferred over QTI+1+PC. In other applications which do not involve view synthesis, and where only the R-D performance of the coded texture views counts, QTI+1+PC could be more suitable than ECU-T. Note that QTI+1+PC also gives lower losses in general than CBF-T in all evaluation scenarios, although CBF-T gives larger runtime reductions.

B. Analysis of QTL results

Table XI shows that when the texture and depth CTUs are homogeneous, runtime savings are achieved since coding modes and partition sizes are no longer checked for high level CUs in depth. The recursion is stopped at depth level 0, hence bringing significant runtime reduction. Also, a small bitrate saving is achieved with the predictive coding part since neither the split flag or the partition size of the CTU needs to be sent. If only the depth is split to achieve better prediction on

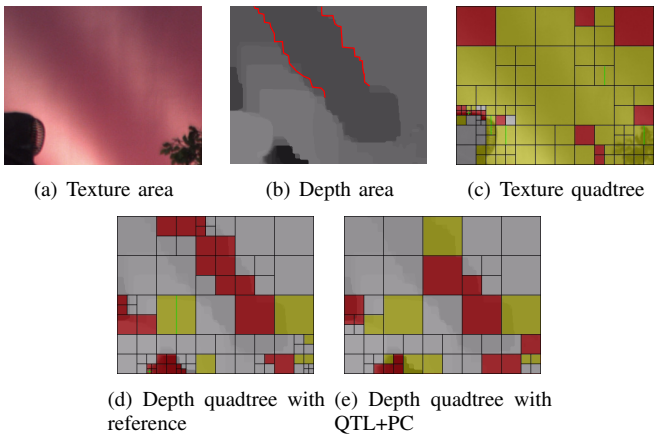
an edge, then our assumption no longer holds. In this case, the runtime and bitrate reductions are accompanied by a loss of quality on synthesized views. This does not happen very frequently however, as can be seen in Table I. When the texture is split, if the depth CTU is homogeneous, QTL+PC has no impact on coding the depth CTU. If the depth CTU is also split, only bitrate reductions are achieved by not sending the quadtree information for high level depth CUs.

We can see from Table XI that the CUs wherein bitrate reductions and runtime savings are possible in depth have an homogeneous co-located texture CU. Actually, these reductions occur when the texture CU is partitioned in 2N×2N, N×2N, or 2N×N, which correspond to case (a) in Figure 5. The percentage of these CUs, that we call Favorable for Depth (FD), is given in Table XII, for each sequence. Overall, the FD percentage equals 84.9%, which is relatively high. This explains the significant runtime reduction and bitrate savings in QTL+PC. The large difference between the FT and FD percentages also explains the performance gap between QTI+PC and QTL+PC in terms of runtime reductions. The predictive coding part achieved however nearly the same coding gains in both tools, as it gave -0.3% and -0.4% on coded+synthesized in QTL and QTI respectively. Since FD is much higher than FT, the predictive coding in QTI should have given less gains. However, the cost of transmitting the split flag and partition size in the bitstream is higher in texture than in depth (6.5% against 1.6%), as shown in Section III-B, so even though the predictive coding is less used in QTI, it has more impact there than in QTL.

The example in Figure 7 shows how QTL successfully smoothes out wrong edges in depth. Indeed, the texture background area, as shown in Figure 7(a) is planar. Its corresponding CTUs are not split (or at most, split once). In depth, the corresponding area should normally be planar as well, but a badly performed stereo matching created a wrong edge in that area, highlighted in Figure 7(b). Consequently, when QTL is not used, the CTUs corresponding to that depth area are split, as seen in Figure 7(d). When QTL is used, the depth quadtree is limited to the texture quadtree: the depth CTUs are not split, hence smoothing out the wrong edge, as shown in Figure 7(e).

These wrong edges are still present in reference coding, and when comparing to an uncompressed synthesized view, rendered with an original depth that contains these wrong edges, the reference coding appears to be better than QTL+PC. This is visible in the "Synth." column in Table VII which reports losses, due to a PSNR drop on synthesized views from smoothing out wrong edges. While there is a significant depth bitrate reduction brought by QTL+PC, evaluated at 23.1% on the depth component as shown in Table VII, this reduction is not able to compensate this PSNR drop.

However, smoothing out the wrong edges in depth is actually an improvement since the quality of the synthesized views will be increased. Indeed, the subjective results in Table IX confirm that there is no actual degradation in the quality of the synthesized views. In fact, on average, our tool was found to be better than the reference, especially on the 2D test where the potential artefacts resulting from the use of our tool, if any, would be more noticeable anyway.



(a) Texture area (b) Depth area (c) Texture quadtree

(d) Depth quadtree with reference (e) Depth quadtree with QTL+PC

Fig. 7. Depth quadtree with reference coding and with QTL. Gray CUs are coded in SKIP mode, green CUs in Inter and red CUs in Intra (best viewed in color)

In the coded+synthesized column, the PSNR considered is the average between the 6 PSNRs of the synthesized views and the 3 PSNRs of the coded texture views, which do not change using our method since QTL does not affect the texture component. Consequently, the PSNR drop on the synthesized views is attenuated in this column, and can therefore be compensated by the bitrate reduction on the depth component, leading to the shown overall BD-Rate gains.

Furthermore, since neither the PSNR nor the bitrate of the coded texture videos change in QTL+PC, the depth bitrate reductions are directly visible in the "Video total" column, and although attenuated by the added texture bitrate, they are still significant.

Compared to QTL+PC, ECU applied only on depth achieves lower runtime reduction than QTL+PC. ECU however gives a more significant bitrate reduction on the depth component alone, which is visible in the "Video total" column in Table X (-2.2% gain). ECU, which was presented initially as a 2D encoder shortcut in HM, is based on the assumption that if a CU is best coded in SKIP mode, splitting it further would not allow to find a better configuration, R-D wise. Altough the assumption is valid for texture, it is not appropriate for depth since any failures in this assumption would cause smoothing out an edge in depth, hence heavily impacting the synthesis. And unlike in QTL+PC, these are not necessarily wrong edges. Consequently, a significant PSNR drop on the synthesized views is observed, causing a 1.1% loss. These losses are not present in QTL+PC. EDCU conditions the application of ECU using an additional constraint: the co-located texture CU must be coded in SKIP as well. Consequently, EDCU is less applied than ECU, hence giving lower runtime reductions (18% instead of 24%), but also lower losses on synthesized views, and an R-D performance only slightly worse than QTL+PC. As for CBF applied only on depth, it gives a worse R-D performance than QTL+PC and a lower runtime reduction.

Finally, note that at the time of writing this paper, the most recent HTM software version was HTM-4.0. Since then, many

tools have been added or removed from the software. In the more recent HTM-7.0 version, released in June 2013, the performance of QTL+PC was only slightly reduced: it still achieves a significant runtime reduction of 26% with a 1.0% loss on synthesized views (but as said earlier, these are not real losses) and -1.2% gain in the "Video total" column.

VII. Conclusion

In this work, we have presented the texture quadtree initialization (QTI) and the depth quadtree limitation (QTL) coding tools and their associated predictive coding (PC) part. QTI+PC and QTL+PC are able to reduce encoder runtime and achieve coding gains by exploiting texture-depth correlations.

QTI+PC brings 6% encoder runtime reduction. The method is promising, although the assumption failures, essentially due to wrong edges in depth, cause coding losses. In a more flexible scheme, the runtime reductions are maintained, and the coding losses are almost entirely compensated by the bitrate reductions the predictive coding part brings. In the future, a more flexible scheme that achieves a better compromise between level of texture quadtree alteration and bitrate reduction potential needs to be found.

QTL+PC, on the other hand, reduces encoder runtime significantly by 31% while achieving -0.3% coding gain on average for coded and synthesized views, -23.1% on depths, and -1.8% on coded videos when the depth bitrate is considered. It can also smooth out wrong edges in depth, which eventually leads to a better synthesis quality. QTL+PC was presented in the 2nd JCT-3V meeting and was adopted in both the 3D-HEVC working draft and software.

References

[1] C. Fehn, E. Cooke, O. Schreer, and P. Kauff, "3D analysis and image-based rendering for immersive TV applications," *Signal Processing: Image Communication*, vol. 17, no. 9, pp. 705 – 715, 2002.

[2] J. Carranza, C. Theobalt, M. A. Magnor, and H.-P. Seidel, "Free-viewpoint video of human actors," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 569–577, July 2003.

[3] *Advanced Video Coding for Generic Audiovisual Services*, ITU-T Recommendation H.264 and ISO/IEC 14496-10 AVC Std., Rev. version 3, 2005.

[4] B. Bross, W.-J. Han, J.-R. Ohm, G. Sullivan, Y.-K. Wang, and T. Wiegand, "High Efficiency Video Coding (HEVC) text specification draft 10 (for FDIS & Last Call)," ITU-T SG16 WP3 & ISO/IEC JTC1/SC29/WG11 JCTVC-L1003, January 2013.

[5] Y. Chen, Y.-K. Wang, K. Ugur, M. Hannuksela, J. Lainema, and M. Gabbouj, "The emerging MVC standard for 3D video services," *EURASIP Journal on Advances in Signal Processing*, 2009.

[6] "Call for proposals on 3D video coding technology," ISO/IEC JTC1/SC29/WG11 N12036, March 2011.

[7] W.-J. Han, J. Min, I.-K. Kim, E. Alshina, A. Alshin, T. Lee, J. Chen, V. Seregin, S. Lee, Y. M. Hong, M.-S. Cheon, N. Shlyakhov, K. McCann, T. Davies, and J.-H. Park, "Improved video compression efficiency through flexible unit representation and corresponding extension of coding tools," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 12, pp. 1709 –1720, December 2010.

[8] P. Helle, S. Oudin, B. Bross, D. Marpe, M. Bici, K. Ugur, J. Jung, G. Clare, and T. Wiegand, "Block merging for quadtree-based partitioning in HEVC," *IEEE Transactions on Circuits and Systems for Video Technology*, no. 99, 2012.

[9] J. Jung and E. Mora, "3D-CE3.h: Depth quadtree prediction for 3DHTM 4.1," ITU-T SG16 WP3 & ISO/IEC JTC1/SC29/WG11 JCT3V-B0068, October 2012.

[10] H. Schwarz, C. Bartnik, and S. Bosse, "Description of 3D video technology proposal by Fraunhofer HHI," ISO/IEC JTC1/SC29/WG11 MPEG2011/M22571, November 2011.

[11] J. Jung and J.-L. Lin, "AHG 10 report on inter-component dependencies," ITU-T SG16 WP3 & ISO/IEC JTC1/SC29/WG11 JCT3V-B0010, October 2012.

[12] M. Domanski, T. Grajek, D. Karwowski, K. Klimaszewski, J. Konieczny, M. Kurc, A. Luczak, R. Ratajczak, J. Siast, O. Stankiewicz, J. Stankowski, and K. Wegner, "Technical description of Poznan University of Technology proposal for Call on 3D Video Coding Technology," ISO/IEC JTC1/SC29/WG11 MPEG2011/ M22697, November 2011.

[13] F. Jager, "3D-CE1.h results on view synthesis prediction," ITU-T SG16 WP3 & ISO/IEC JTC1/SC29/WG11 JCT3V-B0034, October 2012.

[14] J. Y. Lee, H.-C. Wey, and D.-S. Park, "A fast and efficient multi-view depth image coding method based on temporal and inter-view correlations of texture images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 12, pp. 1859 –1868, December 2011.

[15] W.-S. Kim, A. Ortega, P. Lai, D. Tian, and C. Gomila, "Depth map distortion analysis for view rendering and depth coding," in *16th IEEE International Conference on Image Processing (ICIP)*, November 2009, pp. 721 –724.

[16] I. Daribo, C. Tillier, and B. Pesquet-Popescu, "Motion vector sharing and bitrate allocation for 3D video-plus-depth coding," *EURASIP Journal on Advances in Signal Processing*, vol. 2009, p. 13, 2009.

[17] G. Bang, S. Yoo, and J. Nam, "Description of 3D video coding technology proposal by ETRI and Kwangwoon University," ISO/IEC JTC1/SC29/WG11 MPEG2011/M22625, November 2011.

[18] E. Mora, J. Jung, B. Pesquet-Popescu, and M. Cagnazzo, "Codage de vidéos de profondeur basé sur l'héritage des modes Intra de texture," in *COmpression et REprésentation des Signaux Audiovisuels (CORESA)*, April 2012.

[19] I. Daribo, C. Tillier, and B. Pesquet-Popescu, "Adaptive wavelet coding of the depth map for stereoscopic view synthesis," in *IEEE 10th Workshop on Multimedia Signal Processing (MMSP)*, October 2008, pp. 413 –417.

[20] R. H. Gweon, Y.-L. Lee, and J. Lim, "Early termination of CU encoding to reduce HEVC complexity," ITU-T SG16 WP3 & ISO/IEC JTC1/SC29/WG11, JCTVC-F045, July 2011.

[21] K. Choi and E. S. Jang, "Fast coding unit decision method based on coding tree pruning for high efficiency video coding," *Optical Engineering*, vol. 51, no. 3, pp. 030 502–1–030 502–3, 2012.

[22] L. Shen, Z. Liu, P. An, R. Ma, and Z. Zhang, "Low-complexity mode decision for MVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 6, pp. 837–843, 2011.

[23] L. Shen, Z. Zhang, and Z. Liu, "Inter mode selection for depth map coding in 3D video," *IEEE Transactions on Consumer Electronics*, vol. 58, no. 3, pp. 926–931, 2012.

[24] S. Khattak, R. Hamzaoui, S. Ahmad, and P. Frossard, "Fast encoding techniques for Multiview Video Coding," *Signal Processing: Image Communication*, 2012.

[25] J. Y. Lee and C. Kim, "3D-CE3.h related: Removal of depth quadtree prediction and simple early depth CU decision for fast encoding," ITU-T SG 16 WP 3 & ISO/IEC JTC1/SC29/WG11 JCT3V-D0121, April 2013.

[26] L. Shen, Z. Liu, X. Zhang, W. Zhao, and Z. Zhang, "An effective CU size decision method for HEVC encoders," *IEEE Transactions on Multimedia*, no. 99, 2012.

[27] H. L. Tan, F. Liu, Y. H. Tan, and C. Yeo, "On fast coding tree block and mode decision for high-efficiency video coding (HEVC)," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2012, pp. 825 –828.

[28] W.-S. Kim, A. Ortega, P. Lai, D. Tian, and C. Gomila, "Depth map coding with distortion estimation of rendered view," *Proceedings of the SPIE Visual Information Processing Community*, pp. 75 430B–75 430B–10, 2010.

[29] Y. Morvan, P. H. N. de With, and D. Farin, "Platelet-based coding of depth maps for the transmission of multiview images," in *Proceedings of SPIE, Stereoscopic Displays and Applications*, 2006, pp. 93–100.

[30] M. Maitre and M. Do, "Joint encoding of the depth image based representation using shape-adaptive wavelets," in *15th IEEE International Conference on Image Processing (ICIP)*, 2008, pp. 1768–1771.

[31] A. Sanchez, G. Shen, and A. Ortega, "Edge-preserving depth-map coding using graph-based wavelets," in *Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers*, 2009, pp. 578–582.

[32] E. G. Mora, G. Valensize, J. Jung, B. Pesquet-Popescu, M. Cagnazzo, and F. Dufaux, "Depth Video Coding Technologies," in *Emerging Technologies for 3D Video*. Wiley, 2013, ch. 7, pp. 121–137.

[33] G. Tech. 3DV-HTM-4.0 software. [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_3DVCSoftware/tags/HTM-4.0

[34] ——. 3DV-HTM-4.0 software including FCO. [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_3DVCSoftware/branches/HTM-4.0-Nokia
[35] D. Rusanovsky, K. Muller, and A. Vetro, "Common Test Conditions of 3DV Core Experiments," ITU-T SG16 WP3 & ISO/IEC JTC1/SC29/WG11 JCT3V-A1100, July 2012.
[36] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," in *VCEG Meeting*, Austin, USA, April 2001.
[37] J. Jung and E. Mora, "Subjective test results on quadtree limitation and predictive coding," ITU-T SG16 WP3 & ISO/IEC JTC1/SC29/WG11 JCT3V-B0222, October 2012.