 ISO/IEC JTC 1/SC 29/WG 04 N0127

**ISO/IEC JTC 1/SC 29/WG 04  
MPEG Video Coding   
Convenorship: CN**

**Document type:** Output Document

**Title:** Test Model Description of Essential Video Coding

**Status:** Approved

**Date of document:** 2021-07-16

**Source:** ISO/IEC JTC 1/SC 29/WG 04

# Expected action: None

# Action due date: None

**No. of pages:** 23 (without cover page)

**Email of Convenor:** yul@zju.edu.cn

**Committee URL:** <https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg4>

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 04 MPEG VIDEO CODING**

**ISO/IEC JTC 1/SC 29/WG 04 N** **0127**

**July 2021, Online**

|  |  |
| --- | --- |
| **Title** | **Test Model Description of Essential Video Coding** |
| **Source** | **WG 04, MPEG Video Coding** |
| **Status** | **Approved** |
| **Serial Number** | **20741** |

**Abstract**

This document provides description of the Test Model of Essential Video Coding version 7 (ETM 7.3).

1. **Introduction**

This document provides a description of the Test Model of Essential Video Coding version 7 (ETM 7.3). Brief description of design principles, utilized algorithms and software organization are provided in the document as well as tutorial information on software configuration and usage.

1. **Supported configurations of ETM**

To address the requirements of the [1], ETM 7.3 supports four major configurations: Baseline, Main, Baseline still picture, and Main still picture configurations, each of which specifies certain set of tools.

1. **Baseline and Baseline still picture configurations**

In this section, coding tools on Baseline configuration are described according to each category.

* 1. **Coding structure**

The coding structure employs a quad-tree based coding structure, which can use blocks up to 64x64 samples.

* 1. **Intra prediction**

For the Baseline configuration 5 directional prediction modes are employed; DC, horizontal (H), vertical (V), diagonal left (DL), diagonal right (DR). A codeword for prediction mode of the current block is adaptively assigned by using a mapping table between symbol and prediction mode which is selected based on the prediction modes of neighbouring upper and left blocks.

* 1. **Inter prediction**

The inter prediction for Baseline configuration exploits three neighbouring motion vectors and a motion vector of temporally co-located blocks. After choosing one of the candidate motion vectors as a predictor, the index of the predictor is coded. Then the difference between the motion vector for the current block and the predictor may be coded based on encoder side decision. If the difference between the current block and the predictor is relatively small, the motion vector difference and a block residue are not coded, which is called the skip mode. Otherwise, the motion vector difference and the block residue are coded and signalled in the bitstream. The bi-directional prediction is a linear combination of two motion compensated blocks that involve two motion vectors, a forward and backward motion vector.

* 1. **Transform and quantization**

DCT2 transforms are applied to a residual block between an original block and the corresponding prediction block, as a conventional hybrid video codec does. Since transforms are applied to coding blocks, the transform size is equal to the coding block size, i.e. from 2x2 to 64x64. After the transform is conducted, scalar quantization is applied to the transformed coefficients. The range of the quantization parameter (QP) is from 0 to 51 and a scaling factor (SF) corresponding to each QP is defined by a look-up table.

* 1. **Coefficients coding**

Transform coefficients of coded block after quantization are scanned in a predefined scan pattern and entropy coded. In the baseline configuration of the ETM, a run-length based coefficient coding method is used. Visualization of propose method is given in Figure 1. Table 1 below gives an example of coded symbols for a chunk of transform coefficients.

3

0

5

0

-2

0

0

0

-1

0

0

0

1

0

0

0

0

0

0

0

1

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

-1

0

0

0

0

**Figure 1. Visualization of the coefficients scan method and last position**

**Table 1. Coded symbols for the coefficients of a chunk of transform coefficients**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Position | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Coeffs stream | 3 | 0 | 5 | -1 | 0 | -2 | 0 | 0 | 0 | 0 |
| Output stream | 0,2,0,0 |  | 1,4,0,0 | 0,0,1,0 |  | 1,1,1,0 |  |  |  |  |
| Position | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| Coeffs stream | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | -1 |
| Output stream |  |  | 6,0,0,0 |  |  | 2,0,0,0 |  |  |  | 3,0,1,1 |

* 1. **Deblocking Loop filter**

Most block-based video coding schemes introduce noticeable blocking artifacts. A loop filter based on H.263 Annex J [2] was employed to increase objective and subjective image quality for Baseline configuration.

* 1. **Entropy Coding**

The binary arithmetic coding scheme in JPEG Annex D [3] is applied as the entropy coding engine of the proposed codec. After a binarization process of the given symbol, an arithmetic coding engine encodes each binary value with the corresponding context that stores the occurrence probability of a given value. After each binary value is encoded, the probability is updated by using a look-up table and the binary value of symbol is stored in the corresponding context. To code the transformed and quantized coefficient values, run/level symbols are generated after scanning with a zig-zag pattern. Each run or level symbol is binarized by unary coding and the binary value is coded with the corresponding run or level contexts. The sign value and the last symbol indicate whether the level is the last one in the block should be followed to each level. The sign value is coded with fixed length coding and the last symbol is coded with the arithmetic engine.

Some entropy coding technical details are summarized below:

* 9 bits are used for storing current probability of the LPS;
* 14 bits are used for representing probability range;
* 10 bits are used for representing context model initialization values.
  1. **Delta QP**

In order to support a delta QP method, a simple delta QP method is used in baseline profile. It assumes sending delta QP only at CU level when cbf exists at least one of Y, Cb, and Cr components. Delta QP parsing at CU block is performed according to the following condition:

* If (cbf\_luma || cbf\_cb || cbf\_cr) parse delta QP at the CU.

1. **Main and Main still picture configurations**

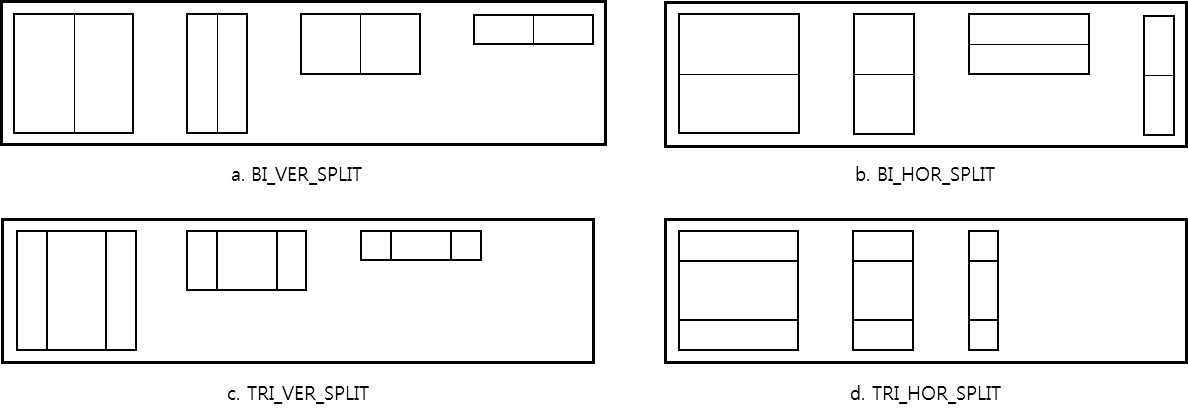
In this section, coding tools on Main configuration are described according to each category.

* 1. **Coding structure**

A coding tree unit (CTU) is the basic unit of the proposed coding scheme. Maximal CTU size is 128x128.

A CTU can be further partitioned recursively by binary and triple tree (BTT) structure. For instance, if width and height of a block are the same it can be represented as a 1:1 ratio CU or a square CU, and if the width is equal to 64 and the height is equal to 16 it can be represented as a 1:4 ratio CU. CU partitioning is conducted based on the allowed CU shapes and their allowed maximum and minimum sizes. In order to support 64x64 pipeline the ternary tree split and 1:4 or 4:1 ratio CU are disallowed when a CU size is greater than 64x64.

The proposed structure has two CU split modes i.e. the binary split and the triple split mode, and each split mode has two directions, vertical and horizontal. Thus, a CU can be partitioned by four different split modes: BI\_VER\_SPLIT, BI\_HOR\_SPLIT, TRI\_VER\_SPLIT, and TRI\_HOR\_SPLIT. These split modes are shown in Figure 2.

****

**Figure 2. Bi/ Triple split modes**

In order to improve memory bandwidth of small intra chroma blocks processing, i.e. blocks with sizes 4x2/2x4 and 2x2, it is proposed to introduce local dual tree at the very last step of partitioning. More specifically, local dual tree applies if size of the smallest child is less than 16 chroma pixels.

In such cases prediction mode flag is signalled in a bitstream and all further leafs inherit the prediction mode. If the prediction mode is Intra, luma and chroma trees are separated from each other’s. The luma tree further splits in a normal way and the chroma tree is forced to be a leaf node. Otherwise, if prediction mode is Inter, luma and chroma trees split together as single tree.

* 1. **Split Unit Coding Order**

The units in the partitioning process, which are further split into two, three, or four units are named split units (SUs). Usually the coding order of a split unit is from left to right and from above to bottom because of z-scanning order of quad tree structure and raster scan of CTUs in a picture. However, normal left to right coding order is more beneficial to left inclined features than right inclined features. This is the case not just in intra prediction. Even in inter prediction, blocks with right inclined features cannot find similar motion information from the left and above neighbourhood.



**Figure 3. Allowed coding order in QT / TT / BT split**

The Split unit coding order (SUCO) enables a more flexible coding order, such as left to right (L2R) and right to left (R2L), to allow intra prediction from right reference pixels and inter prediction with right motion vector predictors. If an SU is partitioned horizontally (horizontal splitting), a flag is signalled to indicate L2R or R2L coding order of partitioned units as shown in Figure 3. If a SU is partitioned by a quad tree structure, a flag is shared for the two above units and two bottom units as shown in Figure 3. If no flag is signalled for coding order of an SU, the coding order follows its parent’s SU coding order.

* 1. **Intra prediction**
     1. **Normal Intra prediction**

To exploit spatial correlation efficiently based on flexible coding structure, a total of 33 intra prediction modes for luma component and 5 modes for chroma component are applied. DC, Bi-linear, Plane, DM modes, and 30 angular modes are applied, with straightforward extension for flexible block size.

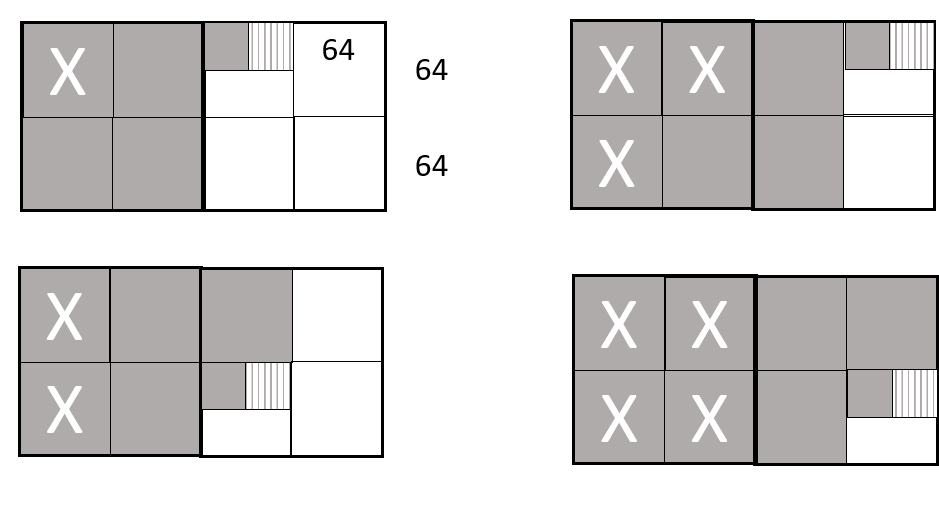
* + 1. **Intra Block Copy**

Intra Block Copy (IBC) mode is used in order to address requirements for Screen Content material.

Technical aspects of the proposed IBC implementation are summarized follows.

* IBC mode (IBC flag) is signaled in a CU level. The IBC mode is considered as a prediction mode other than intra and inter prediction modes. There is no need to include current picture as one of the reference pictures in the reference picture list 0. The motion vector of IBC is derived in integer pixel.
* The maximum block size for IBC coded block is signaled in SPS and up to 64 luma samples for either size.
* The coding of block vector (BV) is straightforward without using prediction. The coding engine reuses the one used in mvd coding. The BV considered is in integer resolution.
* The whole reference block for IBC mode should be already reconstructed. In addition, some search range constraints are imposed such that the allowed search range for IBC mode is the reconstructed part of the current CTU plus some areas of the immediate left CTU (given the CTU size is 128x128). The allowed reference areas in the left CTU exactly match the not-yet-coded areas in the current CTU. In this way, all the reference samples for IBC can be stored in a CTU size memory. According to the location of the current block in the current CTU, the available reference areas change in a way that when a 64x64 region starts encoding/decoding, the whole region is considered as already coded. Therefore, the collocated 64x64 region in the left CTU can no longer be used for IBC reference.

In Figure 4 examples of the reference areas for IBC mode are given, when the current block (in vertical strips) is located in different 64x64 regions of the current CTU. The grey areas already reconstructed samples. The white areas are not-yet-coded regions. The 64x64 regions with “X” marks are not allowed to be used for reference.



**Figure 4. Reference area for IBC mode (with a vertical BT split at 128x128 level)**

* 1. **Inter prediction**

Motion model of the video content for each currently coded block is described conventionally through two parameters: reference index (refIdx) indicating the picture stored in decoded picture buffer (DPB) utilized as a reference for the current block, and the motion vector (MV) – amount of displacement in *x* and *y* directions between current block and the spatial position of the reference block in reference picture indicated by refIdx. Each current block can be predicted either from a single reference (uni-direction) or from two references in so called bi-directional prediction mode.

MV can be signaled either in merge or AMVP mode. Both signalling mechanism utilizes *motion vector prediction* (MVP) list (of different size) constructed from motion information available from spatial or temporal neighboring of the currently coded blocks. In the merge mode, an index that specify a certain entry in the MVP list is signaled and fully describe motion information for the current block. For AMVP mode, in addition to the index in the MVP list, a reference index and the difference between an actual MV and referenced MVP (from the list) is signaled.

* + 1. **Merge/Skip modes**

For the skip mode and the merge mode, a motion vector competition scheme is used to select a motion candidate from a given candidate set that includes spatial and temporal motion candidates. Only merge indices are transmitted for the block coded in merge or skip mode, and the current block inherits the inter prediction indicator, reference indices, and motion vectors from a neighboring block referred by the coded merge index. In the case of a skip mode, the residual signal is omitted. The inherited motion information might be further refined by different decoder-side MV derivation (DMVD) methods and thus different merge modes are introduced in this proposal according to different DMVD approaches.

A merge candidate list is first constructed by inserting (in the case of availability) the motion vectors and reference indices of the spatial neighboring and temporal neighboring blocks in the order specified in the Figure 5. Position of spatial neighbors providing motion information are shown in solid-lined blocks, position of collocated neighbors are shown as dash-lined blocks. These spatial-temporal MV candidate positions are inverted along the axis to reflect selected SUCO coding order. The merge candidate list for the merge/skip mode is constructed by inserting spatial candidates, temporal candidates, affine candidates, History MVP candidate and the conventional combined candidates and Zero candidates according to a pre-defined insertion order:

Current block

7

6

4

0

3

1

2

5

**Figure 5. Spatial-temporal pattern and visiting order utilized for merge list construction**

1. Spatial candidates 0-4.
2. TMVP candidates 5-7
3. History-based MVP candidate.
4. Combined candidates
5. Zero candidates

To reduce the implementation complexity, the size of MVP list size is reduced for small block sizes (number of pixels less than 32) and bi-directional prediction is constrained for small block sizes.

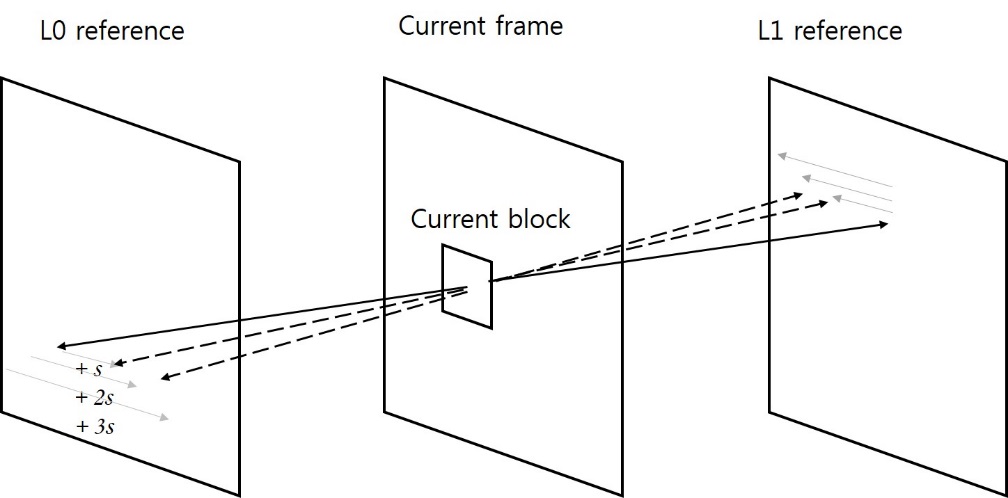
* + 1. **Adaptive motion vector resolution**

Adaptive motion vector resolution (AMVR) supports multiple motion vector resolutions. Supported motion vector resolutions in proposed method range from 1/4–pel to 4–pel (1/4–pel, 1/2–pel, 1–pel, 2–pel and 4–pel). Information about the motion vector resolution is signalled at the CU level when MVD information is signalled. Depending on the resolution of CU, both motion vector (MV) and motion vector predictor (MVP) of the CU are adjusted.

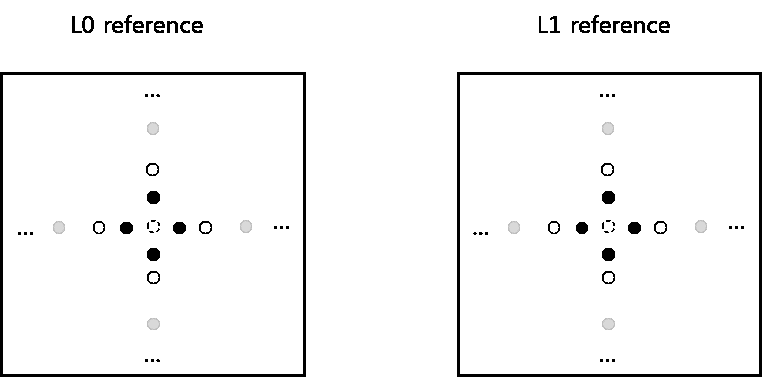
* + 1. **Merge with motion vector difference**

Merge with MVD (MMVD) can be either skip or merge modes, which use the proposed motion vector expression method with neighbouring motion information. As with the skip and the merge modes in HEVC, MMVD also makes a candidate list from neighbouring motion information. Among those candidates in the list, a MV candidate is selected, and is further expanded by a new motion vector expression method.

MMVD provides a new motion vector expression with simplified signalling. The expression method includes starting point, motion magnitude, and motion direction.



**Figure 6. MMDV Search Process**



**Figure 7. MMVD Search Point**

Base candidate index defines the starting point. Base candidate index indicates the best candidate among candidates in the merge candidate list as follows:

**Table 2. Base candidate IDX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Base candidate IDX | 0 | 1 | 2 | 3 |
| Nth MVP | 1st MVP | 2nd MVP | 3rd MVP | 4th MVP |

Distance index defines motion magnitude information. Distance index indicates the pre-defined distance from the starting point. Pre-defined distance is as follows:

**Table 3. Distance IDX**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Distance IDX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Pixel distance | 1/4-pel | 1/2-pel | 1-pel | 2-pel | 4-pel | 8-pel | 16-pel | 32-pel |

Direction index represents the direction of the MVD relative to the starting point. The direction index can represent of the four directions as shown below.

**Table 4. Direction IDX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Direction IDX | 00 | 01 | 10 | 11 |
| x-axis | + | – | N/A | N/A |
| y-axis | N/A | N/A | + | – |

* + 1. **Affine Prediction**

There are three affine prediction motion modes: AF\_4\_INTER mode, AF\_6\_INTER and AF\_MERGE mode.

When merge/skip flag is true, and both width and height for the CU are larger than or equal to 8 samples, an affine flag in CU level is signalled in the bitstream to indicate whether affine merge mode is used. And when the CU is coded as AF\_MERGE, the merge candidate index with maximum value 4 is signalled for specifying which motion information candidate in the affine merge candidate list is used for the CU.

The affine merge candidate list is constructed as following steps:

1. Insert model based affine candidates

Model based candidate is derived from the affine motion model of its valid spatial neighboring affine coded block. The scan order for the candidate positions is identical to the merge list order in Figure 5 and includes positions from 0 to 5.

1. Insert control point based affine candidates

If limit on the affine merge list size is not met, control point based affine candidates are inserted. Control point based affine candidate means the candidate is constructed by combining the neighboring motion information of each control point to form an affine merge candidate.

Total number of 4 CP (CP1-CP4) are used with coordinates (0, 0), (W, 0), (H, 0) and (W, H), respectively, where W and H are the width and height of current block.

To simplify the complexity of affine merge list construction process, no scaling is performed when deriving the control point based affine merge candidate. If the control point motion vectors are pointing to different reference indices or the reference index is invalid, this candidate will be considered as unavailable.

When merge/skip flag is false, and both width and height for the CU are larger than or equal to 16 samples, an affine flag in CU level is signalled in the bitstream to indicate whether affine inter mode is used. And when the CU is coded as affine inter mode, a model flag is signalled for specifying whether 4-parameter or 6-parameter affine model is used for this CU. If the model flag is true, AF\_6\_INTER mode (6-parameter affine model) is applied and 3 MVDs will be parsed; otherwise, AF\_4\_INTER mode (4-parameter affine model) is applied and 2 MVDs will be parsed.

The affine AMVP candidate list is constructed as following steps:

1. Insert model based affine candidates;
2. Insert control point based affine candidates;
3. Insert translational based affine AMVP candidate;
4. Padding with zero motion vectors.

If the number of candidates in affine merge candidate list is less than 2, zero motion vectors with zero reference indices are insert until the list is full. To reduce the complexity of the list construction, no pruning is applied.

* + 1. **Decoder-side motion vector refinement**

Decoder-side motion vector refinement (DMVR) method operates with the two motion vectors of the bi-prediction are further refined by a bilateral template matching process. DMVR applies for blocks 8x8 or large. The bilateral template matching applied in the decoder to perform a distortion-based search between a bilateral template and the reconstruction samples in the reference pictures in order to obtain a refined MV without transmission of additional motion information.

Bilateral template is generated as the weighted combination (i.e. average) of the two prediction blocks, from the initial MV0 of list0 and MV1 of list1, respectively. The template matching operation consists of calculating cost measures between the generated template and the sample region (around the initial prediction block) in the reference picture. For each of the two reference pictures, the MV that yields the minimum template cost is considered as the updated MV of that list to replace the original one.

* + 1. **History-based Motion Vector Prediction**

History-based Motion Vector Prediction (HMVP) method is a inter coding tool which can be applied to both merge candidate list and Advanced motion vector prediction (AMVP) candidate list construction process. A short summary is provided as follows:

In HMVP, a table of HMVP candidates is maintained and updated on-the-fly. Whenever a non-affine inter coded block is decoded, the decoded motion information is used to update the HMVP table in the last position following a First-In-First-Out (FIFO) rule to remove and add entry.



**Figure 8. Decoding flow char with the proposed HMVP method**

The HMVP table size is set to be 23. For each of the HMVP entry, a single MV and refIdx for uni-prediction or two MVs and two refIdx for bi-prediction are stored. No pruning on HMVP table is applied.

For merge list construction, the HMVP candidate is added to the merge candidate list after TMVP merge candidates and before the combined bi-prediction candidates. In order to reduce the complexity of adding HMVP candidate to the merge list, instead of checking each of the HMVP table entry, a subsampling with rate 1 over 4 is applied. To further reduce the redundancy, the latest 2 entries in the HMVP table are skipped when adding HMVP candidate to the merge list. Pruning process is applied toward the merge list after HMVP candidate is added.

For AMVP list construction, HMVP candidate is added after the spatial neighboring candidates.

For fetching candidate from HMVP, the reference index of the last 4 entries in HMVP list are checked, and first candidate with matching reference index is added to the AMVP list.

If no candidate among 4 checked found, the first available candidate of the 4 entries will be scaled to the current reference index and added to the AMVP list. No pruning process is required for AMVP list construction process.

* 1. **Transform and quantization**

DCT2 transforms are applied to a residual block between an original block and the corresponding prediction block, as a conventional hybrid video codec does. In order to support 64x64 pipeline, the maximum allowed transform size is set to 64. If length of a side of CU is greater than the maximum transform size, the side is automatically split into two partitions.

* + 1. **Adaptive transform selection (ATS)**

In addition to normal DCT2 transform an Adaptive Transform Selection (ATS) method can be used for intra and inter cases. Table 5 shows basis functions of for kernel design of adaptive transform selection.

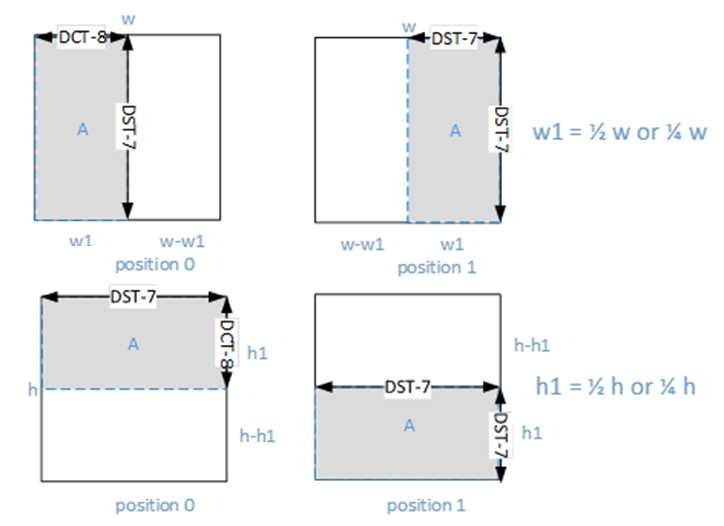
**Table 5. Transform basis functions of DCT- VIII and DST-VII for N-point input**

|  |  |
| --- | --- |
| **Transform Type** | **Basis function *Ti*(*j*), *i*, *j* = 0, 1,…, *N*−1** |
| DCT-VIII |  |
| DST-VII |  |

ATS is applied to the block that is smaller than 32 block size with both width and height. If width or height is over 32 pixels in length, the block is not considered to be applied.

For intra coded block, a flag is used to signal to the decoder whether ATS applied or not. If encoder selects using ATS in a CU as core transform, two more flags are signaled to decoder to indicate which type is used, respectively for the horizontal and vertical directions. The value 0 indicate DST-VII is used and value 1 indicate DCT-VIII is used**.**

For an inter-predicted CU that have residuals, it is signaled to indicate whether the whole residual block or a sub-part of the residual block should be decoded. When only a sub-part of the residual block coded, the part of the residual block is coded with inferred transform type and the other part of the residual block is zeroed out. The sub-part position information and corresponding transform type are illustrated in Figure 9. As shown in the figure, the sub-part that contains residual information can be half or one quarter size of the current CU. ATS is allowed for CU with width and height both no larger than 64. Figure 9 also shows that the transform type is derived based on the position of the sub-block, instead of signalling the transform type as done for intra coded CU. For example, the horizontal and vertical transforms position 0 sub-part is DCT-8 and DST-7, respectively. When at least one side of the residual TU is greater than 32, transform is set as DCT-2.



**Figure 9. Illustration of ATS for inter coded block**

After the transform is conducted, scalar quantization is applied to the transformed coefficients. The range of the quantization parameter (QP) is from 0 to 51 and a scaling factor (SF) corresponding to each QP is defined by a look-up table.

* 1. **Coefficients coding**

Transform coefficients of coded block after quantization are scanned in a predefined scan pattern and entropy coded. To further employ statistical properties of transform coefficients a bit-plane based coefficient coding method, so called Advanced Coefficient Coding (ADCC), is utilized in the Main Profile of EVC instead of the run-length coding method currently used.

ADCC method utilizes the following design elements:

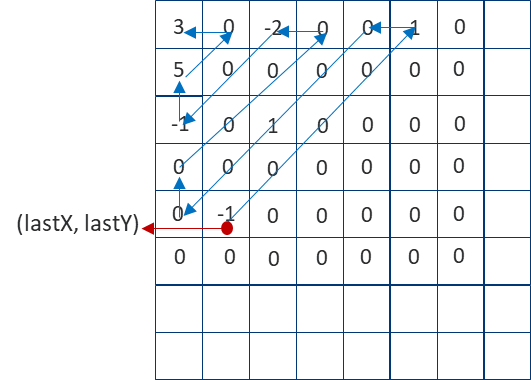
1. Fixed zig-zag scan pattern.
2. Signalling coordinates of the last-non zero transform coefficient in the scan order.
3. Parsing transform coefficients in chunks of 16.
4. Signalling coefficients within each processing chunk as a sequence of significance and levels flags, sign flag and remaining levels.

Among these symbols, the bins of sigMapFlag, flagLevelA and flagLevelB are encoded with adaptive context models; signFlag and binarized bins of levelRem are encoded through by-pass mode.

To decrease number of context-coded bins by adaptively switching the explicit flagLevelA and flagLevelB into levelRem coding, which is binalized with Golomb code and encoded with bypass mode with equal probability.

To improve the throughput, number of explicitly coded symbols flagLevelA and flagLevelB within coded chunk is limited. Only first N flagLevelA and first M flagLevelB symbols are coded. Explicit coding of these symbols is omitted when the specified threshold is met.

Visualization of propose method is given in Figure 10. Table 6 below gives an example of coded symbols for a chunk of 16 transform coefficients, with N=1. Symbols with omitted signalling are marked with X.



**Figure 10. Visualization of the coefficients scan method and last position**

**Table 6. Coded symbols for the coefficients of a chunk of 16 coefficients**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Position | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Coefficients | 0 | 0 | 0 | 0 | 1 | -1 | 0 | 2 | 0 | 3 | 2 | -1 | 0 | 5 | -7 | 10 |
| *sigMapFlag* | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| *flagLevelA* |  |  |  |  | 0 | 0 |  | 1 |  | 1 | X | X |  | X | X | X |
| *flagLevelB* |  |  |  |  |  |  |  | 0 |  | 1 | X |  |  | X | X | X |
| *signFlag* |  |  |  |  | 0 | 1 |  | 0 |  | 0 | 0 | 1 |  | 0 | 1 | 0 |
| *levelRem* |  |  |  |  |  |  |  |  |  | 0 | 1 | 0 |  | 4 | 6 | 9 |

* 1. **Loop filter**

Following in-loop filters are used: Deblocking Filter (DBF), Adaptive Loop Filter (ALF) and Hadamard Transform Domain Filter (HTDF), which is applied directly following the block pixel reconstruction stage and thus used to improve intra-prediction as well.

* + 1. **Deblocking filter**

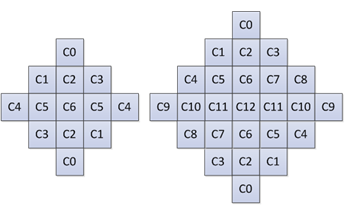
A loop filter based on ISO/IEC 14496-10 [4] was employed to increase objective and subjective image quality for Main profile configuration.

* + 1. **Adaptive Loop Filter**

Following the deblocking Adaptive Loop filter (ALF) with block-based filter adaption is applied. For the luma component, one among 25 filters is selected through the classification process for each 4×4 block, based on local statistics estimates, such as gradient and directionality. To benefit from symmetrical properties of filters, utilized ALF employs filter coefficient transformation process. More details on the ALF design is provide below.

* + - 1. **Filter shape**

Two diamond filter shapes (as shown in Figure 11) are used. The 7×7 and 5×5 diamond is used to filter luma samples and the 5×5 diamond shape is used for chroma samples.



**Figure 11. Filter support area” 5x5 diamond for and 7×7 diamond**

* + - 1. **Block classification**

For luma component, each block is categorized into one out of 25 classes. The classification index *C* is derived based on its directionality and a quantized value of activity , as follows:

To calculate and , gradients of the horizontal, vertical and two diagonal direction are first calculated using 1-D Laplacian:

Where indices and refer to the coordinates of the upper left sample within the block and indicates a reconstructed sample at coordinate .

To reduce the complexity of block classification, the subsampled 1-D Laplacian calculation is applied. As shown in Figure 12, the same subsampled positions are used for gradient calculation of all directions.

|  |  |
| --- | --- |
|  |  |
| (a) Subsampled positions for vertical gradient | (b) Subsampled positions for horizontal gradient |
|  |  |
| (c) Subsampled positions for diagonal gradient | (d) Subsampled positions for diagonal gradient |

**Figure 12. Subsampled Laplacian calculation**

Then maximum and minimum values of the gradients of horizontal and vertical directions are set as:

,

The maximum and minimum values of the gradient of two diagonal directions are set as:

,

To derive the value of the directionality , these values are compared against each other and with two thresholds and :

**Step 1**. If both and are true, is set to .

**Step 2**. If , continue from Step 3; otherwise continue from Step 4.

**Step 3.** If , is set to ; otherwise is set to .

**Step 4**. If , is set to ; otherwise is set to .

The activity value is calculated as:

value is further quantized to the range of 0 to 4, inclusively, and the quantized value is denoted as .

For chroma components in a picture, no classification method is applied, i.e. a single set of ALF coefficients is applied for each chroma component.

* + - 1. **Geometric transformations of filter coefficients**

Before filtering each 4×4 luma block, geometric transformations such as rotation or diagonal and vertical flipping are applied to the filter coefficients depending on gradient values calculated for that block. This is equivalent to applying these transformations to the samples in the filter support region. The idea is to make different blocks to which ALF is applied more similar by aligning their directionality.

Three geometric transformations, including diagonal, vertical flip and rotation are introduced:

* diagonal:
* vertical flip:
* rotation:

where is the size of the filter and are coefficients coordinates, such that location is at the upper left corner and location is at the lower right corner. The transformations are applied to the filter coefficients f (k, l) depending on gradient values calculated for that block. The relationship between the transformation and the four gradients of the four directions are summarized in the following table.

**Table 7. Mapping of the gradient calculated for one block and the transformations**

|  |  |
| --- | --- |
| Gradient values | Transformation |
| gd2 < gd1 and gh < gv | No transformation |
| gd2 < gd1 and gv < gh | Diagonal |
| gd1 < gd2 and gh < gv | Vertical flip |
| gd1 < gd2 and gv < gh | Rotation |

* + - 1. **Filter parameters signalling**

ALF filter parameters are signalled in the adaptation parameter set (APS) NAL unit. Each APS is identified by the unique adaptation\_parameter\_set\_id which is used for referencing of the current APS information from other syntax elements. APSs can be shared across pictures and can be different in different tile groups within a picture. When tile\_group\_alf\_enabled\_flag is equal to 1, an APS is referenced by a tile group header, and the ALF parameters carried in the APS is used by the tile group. A key advantage of using APS for carriage of ALF parameters is that APSs can be sent out-of-band, i.e., provided by external means.

Filter applicability can be controlled at CTB level. A flag is always signalled to indicate whether ALF is applied to a luma CTB. For chroma CTB, a flag might be signalled to indicate whether ALF is applied to a chroma CTB depends on the value of alf\_chroma\_ctb\_present\_flag.

* + - 1. **Filtering process**

At decoder side, when ALF is enabled for a CTB, each sample within the CU is filtered, resulting in sample value as shown below, where *L* denotes filter length, represents filter coefficient, and denotes the decoded filter coefficients.

* + - 1. **Fixed filters**

Proposed ALF design is being initialized with a set of fixed filters provided to decoder as a side information. There are totally 64 7x7 filters (each filter contains 13 coefficients). For each class of classification, a mapping is applied to define which 16 fixed filters from the 64 filters can be used for the current class. The choice index (0-15) of each class is signaled as fixed filter index. When an adaptively derived filter is used, the difference between fixed filter coefficients and adaptive filter coefficients is signalled.

* + - 1. **Temporal filters**

To further benefit from temporal correlation, the ALF design utilizes reusage of the ALF coefficients signalled earlier in APS NAL units. Each APS is identified by the unique adaptation\_parameter\_set\_id which is used for referencing of the current APS information from other syntax elements, e.g. from the tiles group header. All signalled APS with unique set id value are stored in APS buffer size up to 32 entries. To enable the RA coding configuration, encoder’s choice of the APS adaptation\_parameter\_set\_id usage is constrained. For example, to keep the temporal scalability, only temporal filters from the same or lower temporal layers can be used.

* + 1. **Hadamard Transfrom Domain Filter**

Hadamard transform domain filter (HTDF) is applied after reconstruction process to luma reconstructed blocks if quantization parameter (QP) is larger than 17. The filter parameters are explicitly derived from the coded information.

For each pixel from reconstructed block the processing comprises the following steps:

Figure 13. HTDF filtering process

Scan

Hadamard transform

Filtering with 16 entries look-up table

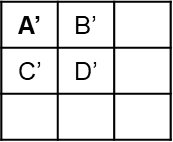
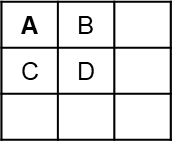
Inverse Hadamard transform

To accumulation buffer



A – current pixel

BCD – neighboring pixels



* Scan for 4 neighboring pixels around processing pixel including current one according to scan pattern
* 4 point Hadamard transform of read pixels
* Spectrum filtering based on the following formula:

wherein *(i)* is index of spectrum component in Hadamard spectrum, *R(i)* is spectrum component of reconstructed pixels corresponding to index, *m=4* is normalization constant equal to number of spectrum components, σ is filtering parameter deriving from codec quantization parameter QP using following equation:

The first spectrum component corresponding to DC value is bypassed without filtering.

The filtering is implemented using look-up-table allowing to exclude multiplications and division from filtering process by following equation:

,

where .

* Inverse 4 point Hadamard transform of filtered spectrum.
* After filtering step the filtered pixels are placed to its original positions into accumulation buffer.

The abovementioned filtering process is schematically illustrated in Figure 13.

After completing filtering of all pixels in the block the accumulated values are normalized by number of processing groups as illustrated in Figure 14.



**Figure 14. HTDF accumulating procedure**

* 1. **Entropy coding**

The same entropy coding engine in used as described for baseline profile in section 3.7.

In addition pre-trained context initialization values are used in order to increase coding efficiency.

* 1. **Delta QP**

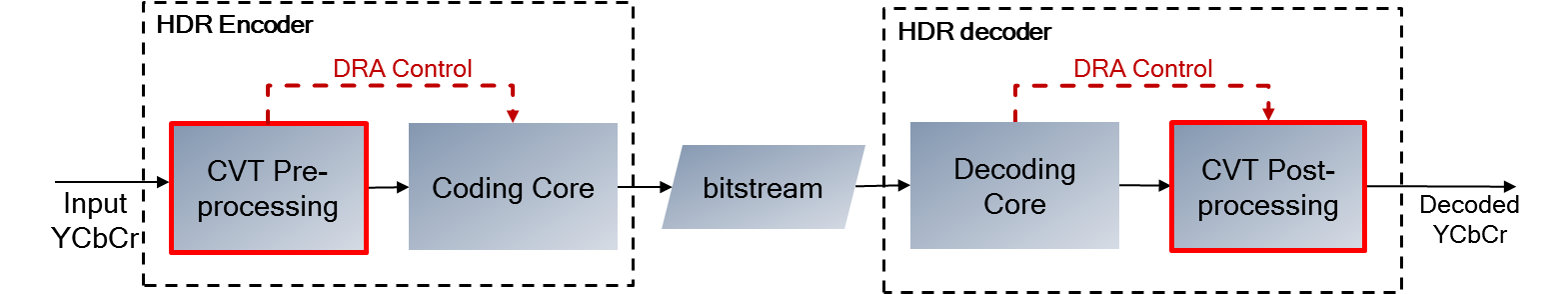
For Main profile, the delta QP is signalled at CU level when an area of block meets a condition using a certain block area. To maintain the spatial granularity level of previous standard such as AVC (i.e., 16x16 blocks), a delta QP signalling for Main profile is designed in the consideration of block area (e.g., 256 residuals). Delta QP parsing at CU block is performed according to the following condition:

* If (log2CbWidth + log2CbHeight >= cuQpDeltaArea && (cbf\_luma || cbf\_cb || cbf\_cr)), then parse a delta QP at the CU.
* If ((log2CbWidth + log2CbHeight + 1 = = cuQpDeltaArea) && (next block partition == tri-partition)), then parse a delta QP at the first split CU.
  1. **HDR support**

To allow more efficient coding of various types of video content, e.g. video in HDR/WCG formats, EVC allows to utilize a user-defined Dynamical Range Adjustment (DRA) tables in order to apply samples remapping at the picture output stage of the decoding process.

The DRA is performed on the input video samples prior to coding and the inverse process is performed on the samples after decoding. The DRA modifies luma and chroma samples in the picture based on the value of the luma or chroma samples and on the value of other component samples located at the same or closest relative location.

Block diagram of the proposed DRA is shown in figure below. At the encoder, the DRA is applied to the input Y’CbCr 4:2:0 picture (e.g. 10 bits per sample) which results in an intermediate picture which is being encoded. Parameters of the inverse DRA are encapsulated in the coded bitstream and used at the decoder side to configure the inverse DRA which is applied to the decoded picture at the picture output process.



**Figure 15: Block diagram of the proposed DRA**

1. **Software Information**
   1. **General**

The Test Model is implemented in C99 and supports windows build using Microsoft Visual Studio IDE (msvc) and Linux build using GNU Makefile.

Visual Studio solutions include five main projects:

1. Encoder Application Project

The application project, which serves file I/O operations, performs encoder configuring and calls encoder API.

1. Decoder Application Project

The application project, which serves file I/O operations, performs decoder configuring and calls decoder API

1. Encoder Library project

The library project, which encapsulates encoder routines and provides encoder API.

1. Decoder Library project

The library project, which encapsulates decoder routines and provides decoder API.

1. Common Library project

The library project, which encapsulates common routines for both encoder and decoder.

Besides five main projects, Test Model also includes bitstream merge service project which allows to merge several independently encoder bitstreams into the one joint valid bitstream.

Figure 16 demonstrates dependencies between the projects.



**Figure 16. A diagram of projects dependencies**

* 1. **Encoder and decoder usage**

All executable files being called without parameters output their detailed usage. Some examples of basic encoder and decoder usage are given below.

Example of encoder command line:

evca\_encoder.exe -i d:/orig/BasketballDrill\_832x480\_50.yuv -o str.bin -w 832 -h 480 -q 37 -z 50 -p 48 -v 1 -f 17 -d 8 --output\_bit\_depth 10 -r rec.yuv --config "..\..\cfg\encoder\_randomaccess.cfg"

Where

* -i, --input [STRING]: file name of input video
* -o, --output [STRING] (optional): file name of output bitstream
* -w, --width [INTEGER]: pixel width of input video
* -h, --height [INTEGER]: pixel height of input video
* -q, --op\_qp [INTEGER] (optional): QP value (0~51)
* -z, --hz [INTEGER]: frame rate (Hz)
* -p, --iperiod [INTEGER] (optional): I-picture period
* -v, --verbose [INTEGER] (optional): verbose level
  + 0: no message
  + 1: frame-level messages (default)
  + 2: all messages
* -f, --frames [INTEGER] (optional): maximum number of frames to be encoded
* -d, --input\_bit\_depth [INTEGER] (optional): input bitdepth (8(default), 10)
* --output\_bit\_depth [INTEGER] (optional): output bitdepth (8, 10)(default: same as input bitdpeth)
* -r, --recon [STRING] (optional): file name of reconstructed video
* --config [STRING] (optional): file name of configuration

Example of decoder command line:

evca\_decoder.exe -i str.bin -o rec\_dec.yuv –v 1 --output\_bit\_depth 10

Where

* -i, --input [STRING]: file name of input bitstream
* -o, --output [STRING] (optional): file name of decoded output
* -v, --verbose [INTEGER] (optional): verbose level
  + 0: no message
  + 1: frame-level messages (default)
  + 2: all messages
* --output\_bit\_depth [INTEGER] (optional): output bitdepth (8(default), 10)
  1. **Basic coding style**

The Test Model coding style is based on GNU coding standards, and assumes usage of following styling concepts.

* + 1. **Indentation**

Four character spaces are used for code indentation. Tabs usage is not allowed.

* + 1. **Long lines and strings**

There is no strict rule about maximal lines or string length, however breaking long lines into sensible chunks is recommended to consider for lines longer than 80 characters.

* + 1. **Placing Braces and Spaces**

The opening brace is placed on a new line on the same indentation level as the defining keyword. The included code block starts at the following line and is indented. The closing brace is placed on the same indentation level as the opening brace.

Example:

|  |
| --- |
| void do\_something() {  if (<expression>)  {  <code here>  } } |

Code following conditionals (e.g. if, else, do, while) is always enclosed in braces, even if it is only a single statement.

* + 1. **Naming**

Filenames are all in lower case with underscore to separate words in a name.

Example:

evc\_mc.c

evca\_decoder.c

Type names (macros, structs, enum elements, constants, etc.) are all in upper case with underscore to separate words in a name.

Example:

#define AFFINE

EVC\_PIC\* pic\_ref[MAX\_NUM\_REF\_PICS];

#define MAX\_NUM\_REF\_PICS 17

Variable and function names are all in lower case with underscore to separate words in a name.

Example:

int pps\_pic\_parameter\_set\_id;

void evc\_get\_mpm(…)

Types, functions and variables shall give useful information about the meaning of them. In case of local variables, names can be shorter assuming they are usually used only within a context.

* + 1. **Functions**

Functions should be short and do only one thing. It is recommended to keep function size inside one screen.

In source files, functions are separated from each other with one blank line.

* + 1. **Commenting**

Comments are used to explain the intent (or implementation detail) of code sections that are not obvious. Note that obviousness may have a different meaning for users who are reading this section for the first time.

In general, it is always recommended to prefer clearly written self-documenting code rather than explicitly commenting certain unobvious places.

* + 1. **Line endings**

Unix style line endings (LF) is used. It is recommended to consider auto converting to LF on checking in stage of version control system in case of non-Unix system development.

* + 1. **Compiler warnings**

Contributed code must not produce compiler warnings with following compiler settings:

|  |  |
| --- | --- |
| gcc | -Wall |
| MS Visual C++ | level 3 warnings |

For other compilers similar warnings level shall be used.

All warnings are treated as errors.

In rare especial cases, certain warnings can be disabled on makefile/project level or explicitly suppressed in place.

* 1. **Software repository**

Test Model of Essential Video Coding (ETM) is available at Gitlab repository in the following web address:

* http://mpegx.int-evry.fr/software/MPEG/Video/EVC/ETM/

The SW can be accessed by using MPEG id and password or getting personal credentials from SW coordinators of EVC project.

* 1. **Issues tracking**

Test Model of Essential Video Coding (ETM) issues tracker (<http://mpegx.int-evry.fr/software/MPEG/Video/EVC/ETM/-/issues>) is used as a regular bug tracker system. It is highly encouraged to submit a ticket in case of identifying any SW issue.

1. **References**
2. “Call for Proposals for a new Video Coding Standard”, MPEG output document w18075.
3. Video Coding for Low bit rate Communication, ITU-T Recommendation H.263, version 1, Nov. 1995; version 2, Jan. 1998; version 3, Nov. 2000.
4. Information technology-digital compression and coding of continuous-tone still images-requirements and guidelines. International Telecommunication Union. CCITT recommendation, 81, p.09.
5. ITU-T and ISO/IEC JTC 1, “Advanced Video Coding for Generic Audiovisual Services,” ITU-T Rec. H.264 & ISO/IEC 14496-10, Version 13, July 2019.