|  |
| --- |
| **INTERNATIONAL ORGANIZATION FOR STANDARDIZATION ORGANISATION INTERNATIONALE DE NORMALISATION ISO/IEC JTC 1/SC 29/WG 5 MPEG JOINT VIDEO CODING TEAM(S) WITH ITU-T SG 16** |
| **ISO/IEC JTC 1 / SC 29 / WG 5 N 250** |
| **Hannover, DE – 13–20 October 2023** |
| |  |  | | --- | --- | | **Source:** | **Convenor (Jens-Rainer Ohm)** | | **Title:** | **Algorithm description of enhanced compression model 11 (ECM 11)** | | **Type:** | **General** | | **Subtype:** | **Other** | | **Status:** | **Approved** | | **Date:** | **2023-12-28** | | **Expected Action:** | **Info** | | **Action due date:** | **N/A** | | **Pages:** | **85** (not including this cover page) | | **Email of convenor:** | **ohm @ ient . rwth-aachen . de** | | **Committee URL:** | **https://sd.iso.org/documents/ui/#!/browse/iso/iso-iec-jtc-1/iso-iec-jtc-1-sc-29/iso-iec-jtc-1-sc-29-wg-5** | |

|  |  |
| --- | --- |
| **Joint Video Experts Team (JVET)**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29**  32nd Meeting, Hannover, DE, 13–20 October 2023 | Document: JVET-AF2025 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Algorithm description of Enhanced Compression Model 11 (ECM 11)** | | |
| *Status:* | Output document of JVET | | |
| *Purpose:* | Algorithm description of Enhanced Compression Model 11 | | |
| *Author(s) or Contact(s):* | Muhammed Coban Ru-Ling Liao Karam Naser Jacob Ström Li Zhang | Email: | [mcoban@qti.qualcomm.com](mailto:mcoban@qti.qualcomm.com) [ruling.lrl@alibaba.com](mailto:ruling.lrl@alibaba.com) karam.naser@interdigital.com [jacob.strom@ericsson.com](mailto:jacob.strom@ericsson.com) [lizhang.idm@bytedance.com](mailto:lizhang.idm@bytedance.com) |
| *Source:* | Editors | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

This document is Enhanced Compression Model 11 (ECM 11) software algorithm description. It includes the coding features and encoding methods implemented in ECM-11.0 software that are under coordinated exploration study by the Joint Video Exploration Team (JVET) of ITU-T VCEG and ISO/IEC MPEG as potential enhanced video coding technology beyond the capabilities of VVC.

ITU-T VCEG (Q6/16) and ISO/IEC MPEG (JTC 1/SC 29/WG 5) are studying the potential need for standardization of future video coding technology with a compression capability that significantly exceeds that of the current VVC standard. Such future standardization action could either take the form of additional extension(s) of VVC or an entirely new standard. The groups are working together on this exploration activity in a joint collaboration effort known as the Joint Video Exploration Team (JVET) to evaluate compression technology designs proposed by their experts in this area. The first Exploration Experiments (EE) were established at the JVET meeting of 6–15 January 2021.

Contents

[Abstract 1](#_Toc154584123)

[1 Introduction 4](#_Toc154584124)

[2 Scope 4](#_Toc154584125)

[3 Algorithm description of the Enhanced Compression Model Software 5](#_Toc154584126)

[3.1 Intra prediction 5](#_Toc154584127)

[3.1.1 Multi-model LM (MMLM) 5](#_Toc154584128)

[3.1.2 Gradient PDPC 6](#_Toc154584129)

[3.1.3 Primary and Secondary MPM 6](#_Toc154584130)

[3.1.4 Reference sample interpolation and smoothing for intra-prediction 7](#_Toc154584131)

[3.1.5 Decoder side intra mode derivation (DIMD) 7](#_Toc154584132)

[3.1.6 Fusion of chroma intra prediction modes 8](#_Toc154584133)

[3.1.7 Intra template matching 9](#_Toc154584134)

[3.1.8 Fusion for template-based intra mode derivation (TIMD) 11](#_Toc154584135)

[3.1.9 Intra prediction fusion 11](#_Toc154584136)

[3.1.10 Combination of CIIP with TIMD and TM merge 12](#_Toc154584137)

[3.1.11 Extended multiple reference line (MRL) list 12](#_Toc154584138)

[3.1.12 Template-based multiple reference line intra prediction 13](#_Toc154584139)

[3.1.13 Convolutional cross-component intra prediction model 14](#_Toc154584140)

[3.1.14 Local-Boosting Cross-Component Prediction (LB-CCP) 21](#_Toc154584141)

[3.1.15 Cross-Component Prediction (CCP) merge (a.k.a., non-local CCP) mode 21](#_Toc154584142)

[3.1.16 Spatial Geometric partitioning mode (SGPM) 22](#_Toc154584143)

[3.1.17 Directional planar mode 23](#_Toc154584144)

[3.1.18 Direct block vector (DBV) for chroma block 24](#_Toc154584145)

[3.2 Inter prediction 24](#_Toc154584146)

[3.2.1 Local illumination compensation (LIC) 24](#_Toc154584147)

[3.2.2 Non-adjacent spatial candidate 25](#_Toc154584148)

[3.2.3 Temporal motion information derivation 26](#_Toc154584149)

[3.2.4 Template matching (TM) 27](#_Toc154584150)

[3.2.5 Multi-pass decoder-side motion vector refinement 28](#_Toc154584151)

[3.2.6 Adaptive decoder-side motion vector refinement 30](#_Toc154584152)

[3.2.7 OBMC 31](#_Toc154584153)

[3.2.8 Template matching based OBMC 31](#_Toc154584154)

[3.2.9 History-parameter-based affine model inheritance and non-adjacent affine mode 33](#_Toc154584155)

[3.2.10 Sample-based BDOF 36](#_Toc154584156)

[3.2.11 Interpolation 37](#_Toc154584157)

[3.2.12 Multi-hypothesis prediction (MHP) 38](#_Toc154584158)

[3.2.13 Pixel based affine motion compensation 39](#_Toc154584159)

[3.2.14 Affine subblock BDOF refinement 39](#_Toc154584160)

[3.2.15 Adaptive reordering of merge candidates with template matching (ARMC-TM) 39](#_Toc154584161)

[3.2.16 MV candidate type based ARMC 42](#_Toc154584162)

[3.2.17 TM based reordering for MMVD and affine MMVD 42](#_Toc154584163)

[3.2.18 Regression based affine candidate derivation 43](#_Toc154584164)

[3.2.19 Geometric partitioning mode (GPM) with merge motion vector differences (MMVD) 44](#_Toc154584165)

[3.2.20 Geometric partitioning mode (GPM) with adaptive blending 44](#_Toc154584166)

[3.2.21 Geometric partitioning mode (GPM) with template matching (TM) 44](#_Toc154584167)

[3.2.22 GPM with inter and intra prediction 45](#_Toc154584168)

[3.2.23 Template matching based reordering for GPM split modes 46](#_Toc154584169)

[3.2.24 Bi-predictive GPM 47](#_Toc154584170)

[3.2.25 Bilateral matching AMVP-merge mode 47](#_Toc154584171)

[3.2.26 IBC merge/AMVP list construction 48](#_Toc154584172)

[3.2.27 IBC with Template Matching 50](#_Toc154584173)

[3.2.28 IBC reference area 50](#_Toc154584174)

[3.2.29 Fractional pel IBC 51](#_Toc154584175)

[3.2.30 Filtered IBC prediction 51](#_Toc154584176)

[3.2.31 MVD prediction 52](#_Toc154584177)

[3.2.32 BVD prediction 52](#_Toc154584178)

[3.2.33 Enhanced bi-directional motion compensation 52](#_Toc154584179)

[3.2.34 Motion compensated picture boundary padding 53](#_Toc154584180)

[3.2.35 Block level reference picture list reordering 54](#_Toc154584181)

[3.2.36 Reference picture resampling (RPR) 54](#_Toc154584182)

[3.2.37 Reconstruction-Reordered IBC (RR-IBC) 55](#_Toc154584183)

[3.2.38 Combination of IBC with other coding tools 56](#_Toc154584184)

[3.2.39 Template matching based BCW index derivation for merge mode 58](#_Toc154584185)

[3.2.40 DMVR for affine merge coded blocks 58](#_Toc154584186)

[3.2.41 InterCCCM 59](#_Toc154584187)

[3.2.42 CCP merge for chroma inter blocks 60](#_Toc154584188)

[3.3 Transform and coefficient coding 61](#_Toc154584190)

[3.3.1 Shifting the quantization centers 61](#_Toc154584191)

[3.3.2 Dependent quantization with 8-states 62](#_Toc154584192)

[3.3.3 Maximum Transform Size and Zeroing-out of Transform Coefficients 63](#_Toc154584193)

[3.3.4 Enhanced MTS for intra coding 63](#_Toc154584194)

[3.3.5 Inter MTS optimization 63](#_Toc154584195)

[3.3.6 Secondary Transformation: LFNST extension with large kernel 63](#_Toc154584196)

[3.3.7 Non-separable primary transform (NSPT) for intra coding 65](#_Toc154584197)

[3.3.8 NSPT/LFNST Context modeling for transform coefficient coding 66](#_Toc154584198)

[3.3.9 Sign prediction 67](#_Toc154584199)

[3.4 Adaptive loop filter 68](#_Toc154584200)

[3.4.1 ALF simplification removal 68](#_Toc154584201)

[3.4.2 ALF with fixed filters 68](#_Toc154584202)

[3.4.3 Filtering 68](#_Toc154584203)

[3.4.4 Classification 68](#_Toc154584204)

[3.4.5 Alternative 2x2 ALF classifier 69](#_Toc154584205)

[3.4.6 Residual based classifier 69](#_Toc154584206)

[3.4.7 CCALF with long tap filter 70](#_Toc154584207)

[3.4.8 Adaptive filter shape switch and using samples before deblocking filter for adaptive loop filter 70](#_Toc154584208)

[3.4.9 Extended Fixed-Filter-Output based Taps for ALF 72](#_Toc154584209)

[3.4.10 ALF with residual samples 72](#_Toc154584210)

[3.4.11 Additional fixed filter for ALF 72](#_Toc154584211)

[3.4.12 Improved fixed filters for ALF 73](#_Toc154584212)

[3.5 Bilateral filter 73](#_Toc154584213)

[3.6 Bilateral inloop filter on chroma 77](#_Toc154584214)

[3.7 Cross-Component Sample Adaptive Offset (CCSAO) 77](#_Toc154584215)

[3.8 Entropy coding 80](#_Toc154584216)

[3.8.1 Extended precision 80](#_Toc154584217)

[3.8.2 Slice-type-based window size 80](#_Toc154584218)

[3.8.3 Improved probability estimation for CABAC 81](#_Toc154584219)

[3.9 Gradual decoding refresh (GDR) 81](#_Toc154584220)

[3.10 Simplified linear model solver 83](#_Toc154584221)

[3.10.1 Implementation 84](#_Toc154584222)

[References 85](#_Toc154584223)

# Introduction

This document provides algorithm description and encoding methods of the coding tools implemented in Enhanced Compression Model 11 (ECM 11) software. Tools consist of extensions of the tools in the existing VVC design and methods that were proposed but not included in the VVC standard [1], and also new coding tools beyond that.

# Scope

The ECM-11.0 reference software is provided to demonstrate a reference implementation of encoding techniques and the decoding process for JVET Enhanced compression beyond VVC capability exploration work. The reference software can be accessed via

https://vcgit.hhi.fraunhofer.de/ecm/ECM.git.

This document provides an algorithm description as well as an encoder-side description of the ECM-11.0, which serves as a tutorial for the algorithm and encoding model implemented in the ECM-11.0 software. The purpose of this document is to share a common understanding of the coding features and the reference encoding methods supported in the ECM-11.0 software, in order to facilitate the assessment of the technical impact of new technologies during the exploration work.

# Algorithm description of the Enhanced Compression Model Software

## Intra prediction

In intra prediction the smallest chroma intra prediction unit (SCIPU) constraint in VVC is removed. In addition, the VPDU constraint for reducing CCLM prediction latency is also removed.

### Multi-model LM (MMLM)

CCLM included in VVC is extended by adding three Multi-model LM (MMLM) modes (JVET-D0110). In each MMLM mode, the reconstructed neighboring samples are classified into two classes using a threshold which is the average of the luma reconstructed neighboring samples. The linear model of each class is derived using the Least-Mean-Square (LMS) method. For the CCLM mode, the LMS method is also used to derive the linear model. A slope adjustment to is applied to cross-component linear model (CCLM) and to Multi-model LM prediction. The adjustment is tilting the linear function which maps luma values to chroma values with respect to a center point determined by the average luma value of the reference samples.

#### Slope adjustment of CCLM

CCLM uses a model with 2 parameters to map luma values to chroma values. The slope parameter “a” and the bias parameter “b” define the mapping as follows:

chromaVal = a \* lumaVal + b

An adjustment “u” to the slope parameter is signaled to update the model to the following form:

chromaVal = a’ \* lumaVal + b’

where

a’ = a + u

b’ = b - u \* yr.

With this selection the mapping function is tilted or rotated around the point with luminance value yr. The average of the reference luma samples used in the model creation as yr in order to provide a meaningful modification to the model. Picture below illustrates the process.



Figure 1. Illustration of the effect of the slope adjustment parameter “u”. Left: model created with the current CCLM. Right: model updated as proposed.

**Implementation**

Slope adjustment parameter is provided as an integer between -4 and 4, inclusive, and signaled in the bitstream. The unit of the slope adjustment parameter is 1/8th of a chroma sample value per one luma sample value (for 10-bit content).

Adjustment is available for the CCLM models that are using reference samples both above and left of the block (“LM\_CHROMA\_IDX” and “MMLM\_CHROMA\_IDX”), but not for the “single side” modes. This selection is based on coding efficiency vs. complexity trade-off considerations.

When slope adjustment is applied for a multimode CCLM model, both models can be adjusted and thus up to two slope updates are signaled for a single chroma block.

**Encoder approach**

The proposed encoder approach performs an SATD based search for the best value of the slope update for Cr and a similar SATD based search for Cb. If either one results as a non-zero slope adjustment parameter, the combined slope adjustment pair (SATD based update for Cr, SATD based update for Cb) is included in the list of RD checks for the TU.

### Gradient PDPC

In VVC, for a few scenarios, PDPC may not be applied due to the unavailability of the secondary reference samples. In these cases, a gradient based PDPC, extended from horizontal/vertical mode, is applied (JVET-Q0391). The PDPC weights (wT / wL) and nScale parameter for determining the decay in PDPC weights with respect to the distance from left/top boundary are set equal to corresponding parameters in horizontal/vertical mode, respectively. When the secondary reference sample is at a fractional sample position, bilinear interpolation is applied.

### Primary and Secondary MPM

Secondary MPM lists is introduced as described in JVET-D0114.The existing primary MPM (PMPM) list consists of 6 entries and the secondary MPM (SMPM) list includes 16 entries. A general MPM list with 22 entries is constructed first, and then the first 6 entries in this general MPM list are included into the PMPM list, and the rest of entries form the SMPM list. The first entry in the general MPM list is the Planar mode. The remaining entries are composed of the intra modes of the left (L), above (A), below-left (BL), above-right (AR), and above-left (AL) neighbouring blocks as shown in Figure 2, and DIMD modes which are sorted in ascending order of SAD cost. Up to 5 modes with the smallest SAD cost are added. The SAD cost is computed between the prediction and the reconstruction samples of the template. The sorted directional modes with added offset are added into the general MPM list, and then the default modes, until the general MPM list with 22 entries is constructed.

If a CU block is vertically oriented, the order of neighbouring blocks is A, L, BL, AR, AL; otherwise, it is L, A, BL, AR, AL.

Shape

Description automatically generated with low confidence

Figure 2. Neighbouring blocks (L, A, BL, AR, AL) used in the derivation of a general MPM list.

MPM list is equally divided into four groups and the group index is parsed first. Then, a mode index is further parsed to indicate which mode in the selected group is used.

### Reference sample interpolation and smoothing for intra-prediction

The 4-tap cubic interpolation is replaced with a 6-tap cubic interpolation filter, as described in JVET-D0119, for the derivation of predicted samples from the reference samples.

For reference sample filtering, a 6-tap gaussian filter is applied for larger blocks (W >= 32 and H >=32), existing VVC 4-tap gaussian interpolation filter is applied otherwise. The extended intra reference samples are derived using the 4-tap interpolation filter instead of the nearest neighbor rounding.

### Decoder side intra mode derivation (DIMD)

When DIMD is applied, up to five intra modes are derived from the reconstructed neighbor samples, and those five predictors are combined with the planar mode predictor with the weights derived from the histogram of gradients as described in JVET-O0449 . The division operations in weight derivation are performed utilizing the same lookup table (LUT) based integerization scheme used by the CCLM. For example, the division operation in the orientation calculation

is computed by the following LUT-based scheme:

x = Floor( Log2( Gx ) )

normDiff = ( ( Gx<< 4 ) >> x ) & 15

x +=( 3 + ( normDiff  !=  0 ) ? 1 : 0 )

Orient = (Gy\* ( DivSigTable[ normDiff ] | 8 ) + ( 1<<( x-1 ) )) >> x

where

DivSigTable[16] = { 0, 7, 6, 5 ,5, 4, 4, 3, 3, 2, 2, 1, 1, 1, 1, 0 }.

For a block of size , the weight for each of the five derived modes is modified if the one the above or left histogram magnitudes is twice larger than the other one. In this case, the weights are location dependent and computed as follows:

If the above histogram is twice the left, then:

.

If the left histogram is twice the above, then:

,

where is the unmodified uniform weight of the DIMD selected as in JVET-O0449, is pre-defined and set to 10.

Derived intra modes are included into the primary list of intra most probable modes (MPM), so the DIMD process is performed before the MPM list is constructed. The primary derived intra mode of a DIMD block is stored with a block and is used for MPM list construction of the neighboring blocks.

Finally, note the region of neighboring reconstructed samples used for computing the histogram of gradients is modified compared to JVET-O0449 method, depending on reconstructed samples availability. The region of decoded reference samples of current WxH luma CB is extended towards the above-right side if available, up to W additional columns. It is extended towards the bottom-left side if available, up to H additional rows.

#### DIMD chroma mode

The DIMD chroma mode uses the DIMD derivation method to derive the chroma intra prediction mode of the current block based on the neighboring reconstructed Y, Cb and Cr samples in the second neighboring row and column as shown in Figure 3. Specifically, a horizontal gradient and a vertical gradient are calculated for each collocated reconstructed luma sample of the current chroma block, as well as the reconstructed Cb and Cr samples, to build a HoG. Then the intra prediction mode with the largest histogram amplitude values is used for performing chroma intra prediction of the current chroma block.



Figure 3. Neighboring reconstructed samples used for DIMD chroma mode.

When the intra prediction mode derived from the DIMD chroma mode is the same as the intra prediction mode derived from the DM mode, the intra prediction mode with the second largest histogram amplitude value is used as the DIMD chroma mode. A CU level flag is signaled to indicate whether the proposed DIMD chroma mode is applied.

Finally, the luma region of reconstructed samples used for computing the histogram of gradients for chroma DIMD mode is modified compared to JVET-O0449. For a WxH pair of chroma CBs to predict, to build the histogram of gradients associated to the collocated luma CB, the pairs of a vertical gradient and a horizontal gradient are extracted from the second and third lines in this luma CB instead of being extracted from the regular set of DIMD decoded reference samples around this luma CB.

### Fusion of chroma intra prediction modes

In ECM, two chroma intra prediction signals can be fused together. One of the two chroma intra prediction signals is predicted using one of the DM mode, DIMD chroma mode and the four default modes (non-LM mode). The other chroma intra prediction signal is predicted using cross-component linear prediction modes (LM mode). Two different methods are supported.

In the first method, the LM mode can be either MM-CCLM or MM-CCCM, and the final predictor is derived as follows:

where is the predictor obtained by applying the non-LM mode, is the predictor obtained by applying the LM mode and is the final predictor of the current chroma block. The two weights, and are determined by the intra prediction mode of adjacent chroma blocks and is set equal to 2. Specifically, when the above and left adjacent blocks are both coded with LM modes, {}={1, 3}; when the above and left adjacent blocks are both coded with non-LM modes, {}={3, 1}; otherwise, {}={2, 2}. Two template costs are calculated by fusing the angular chroma prediction with MM-CCLM or MM-CCCM, respectively, and the one of the two CCPs which provides a smaller template cost is utilized to derive.

In the second method, the LM mode can be either MMLM or CCLM mode, and the final predictor is derived as follows:

where is the predictor obtained by applying the non-LM mode, is the set of downsampled reconstructed luma samples at co-located positions and is the final predictor of the current chroma block. is a fixed value and is set equal to 512 for 10-bit content. The three weights, , and are derived from the adjacent luma and chroma samples using the same LDL derivation method as in CCCM.

For the syntax design, one index is signaled to indicate whether fusion is applied and which method is used. It is noted that for I slices, the non-LM mode can be DM mode, DIMD chroma mode and the four default modes. For non-I slices, only DIMD chroma mode is allowed to be fused with LM modes.

|  |  |
| --- | --- |
| **Index value** | **Name** |
| 0 | No fusion |
| 1 | First method |
| 2 | Second method with CCLM |
| 3 | Second method with MMLM |

### Intra template matching

Intra template matching prediction (IntraTMP) is a special intra prediction mode that copies the best prediction block from the reconstructed part of the current frame, whose L-shaped template matches the current template. For a predefined search range, the encoder searches for the most similar template to the current template in a reconstructed part of the current frame and uses the corresponding block as a prediction block. The encoder then signals the usage of this mode, and the same prediction operation is performed at the decoder side.

The prediction signal is generated by matching the L-shaped, Top-only or Left-Only causal neighbor of the current block with another block in a predefined search area in Figure 4. There are 6 predefined search areas, i.e., R1 to R6 in Figure 4 which contain the reconstructed samples from the top and left CTUs as well as part of the reconstructed samples within the current CTU that are located above, left, bottom-left and top-right to the current block.

Sum of absolute differences (SAD) is used as a cost function.

A given search order of the 6 regions is utilized, i.e., R4, R5, R6, R1, R2, and R3. Within each region, the decoder constructs a candidate list of up to “19” template matching block vectors that are ranked in ascending order according to the template cost (SAD). The following modes are supported:

1. Single predictor: A single predictor is selected from the candidate list.
2. Fusion of multiple predictors: multiple predictors are blended multiple to derive the final prediction block. The blending weights are either computed from the template matching cost of each predictor, or with Wiener-filter based weight derivation method.
3. Sub-pel precision: When signle predictor is used, sub-pel precion can be used with 1/2-pel precision, 1/4-pel precision and 3/4-pel precision, each with 8 possible directions
4. linear filter model: A linear filter can be learned between the reference template and current template and be applied the linear model to reference block. This mode can be used for signle predictor when sub-pel precision is not used

The dimensions of all regions (SearchRange\_w, SearchRange\_h) are set proportional to the block dimension (BlkW, BlkH) to have a fixed number of SAD comparisons per pixel. That is:

SearchRange\_w = min(64,a \* BlkW)

SearchRange\_h = min(64,a \* BlkH)

Where ‘’ is a constant that controls the gain/complexity trade-off. In practice, ‘’is equal to 5.

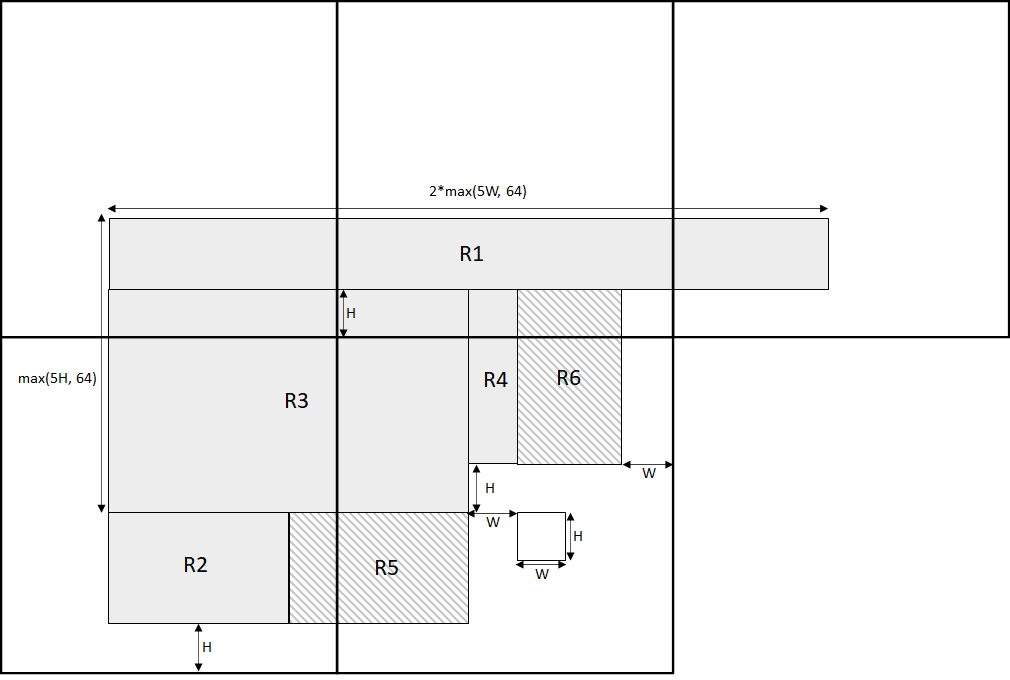


Figure 4. Intra template matching search area used.

To speed-up the template matching process, the search range of all search regions is subsampled by a factor of 3.. After finding the best match, a refinement process is performed. The refinement is done via a second template matching search around the best match with a reduced range.

The Intra template matching tool is enabled for CUs with size less than or equal to 64 in width and height. This maximum CU size for Intra template matching is configurable.

The Intra template matching prediction mode is signaled at CU level through a dedicated flag when DIMD is not used for current CU.

#### IntraTMP derived block vector candidates for IBC

In this method block vector (BV) derived from the intra template matching prediction (IntraTMP) is used for intra block copy (IBC). The stored IntraTMP BV of the neighbouring blocks along with IBC BV are used as spatial BV candidates in IBC candidate list construction.

IntraTMP block vector is stored in the IBC block vector buffer and, the current IBC block can use both IBC BV and IntraTMP BV of neighbouring blocks as BV candidate for IBC BV candidate list as shown in Figure 5.

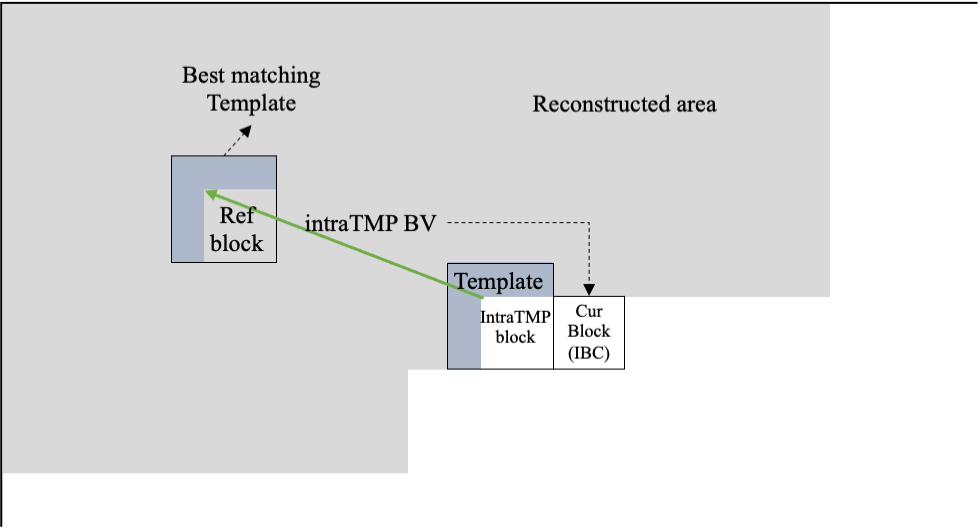


Figure 5. Use of IntraTMP block vector for IBC block.

IntraTMP block vectors are added to IBC block vector candidate list as spatial candidates. IntraTMP block vectors are stored in quarter-pel resolution for coding of IBC block vectors and HMVP.

### Fusion for template-based intra mode derivation (TIMD)

For each intra prediction mode in MPMs, as well as the wide-angle modes if the above-right and/or bottom-left reference samples are available, SATD between the prediction and reconstruction samples of the template is calculated. First two intra prediction modes with the minimum SATD are selected as the TIMD modes. These two TIMD modes are fused with the weights after applying PDPC process, and such weighted intra prediction is used to code the current CU. Position dependent intra prediction combination (PDPC) is included in the derivation of the TIMD modes.

The costs of the two selected modes are compared with a threshold, in the test the cost factor of 2 is applied as follows:

costMode2 < 2\*costMode1.

If this condition is true, the fusion is applied, otherwise the only mode1 is used.

Weights of the modes are computed from their SATD costs as follows:

weight1 = costMode2 / (costMode1+ costMode2)

weight2 = 1 - weight1

The division operations are conducted using the same lookup table (LUT) based integerization scheme used by the CCLM.

### Intra prediction fusion

This intra prediction method derives predicted samples as a weighted combination of multiple predictors generated from different reference lines. In this process multiple intra predictors are generated and then fused by weighted averaging. The process of deriving the predictors to be used in the fusion process is described as follows:

1. For angular intra prediction modes including the single mode case of TIMD and DIMD, the proposed method derives intra prediction by weighting intra predictions obtained from multiple reference lines represented as , where is the intra prediction from the default reference line and is the prediction from the line above the default reference line. The weights are set as and .
2. For TIMD mode with blending, is used for the first mode () and is used for the second mode ().
3. For DIMD mode with blending, the number of predictors selected for a weighted average is increased from 3 to 6.

Intra prediction fusion method is applied to luma blocks when angular intra mode has non-integer slope (required reference samples interpolation) and the block size is greater than 16, it is used with MRL and not applied for ISP coded blocks. In the method studied in the sub-test a, PDPC is applied for the intra prediction mode using the closest to the current block reference line

### Combination of CIIP with TIMD and TM merge

In CIIP mode, the prediction samples are generated by weighting an inter prediction signal predicted using CIIP-TM merge candidate and an intra prediction signal predicted using TIMD derived intra prediction mode. The method is only applied to coding blocks with an area less than or equal to 1024.

The TIMD derivation method is used to derive the intra prediction mode in CIIP. Specifically, the intra prediction mode with the smallest SATD values in the TIMD mode list is selected and mapped to one of the 67 regular intra prediction modes.

In addition, it is also proposed to modify the weights (wIntra, wInter) for the two tests if the derived intra prediction mode is an angular mode. For near-horizontal modes (2 <= angular mode index < 34), the current block is vertically divided as shown in Figure 6(a); for near-vertical modes (34 <= angular mode index <= 66), the current block is horizontally divided as shown in Figure 6(b).

The (wIntra, wInter) for different sub-blocks are shown in Table 1.



Figure 6. The division method for angular modes.

*Table 1. The modified weights used for angular modes.*

|  |  |
| --- | --- |
| **The sub-block index** | **(wIntra, wInter)** |
| 0 | (6, 2) |
| 1 | (5, 3) |
| 2 | (3, 5) |
| 3 | (2, 6) |

With CIIP-TM, a CIIP-TM merge candidate list is built for the CIIP-TM mode. The merge candidates are refined by template matching. The CIIP-TM merge candidates are also reordered by the ARMC method as regular merge candidates. The maximum number of CIIP-TM merge candidates is equal to two.

### Extended multiple reference line (MRL) list

MRL list in VVC is extended to include more reference lines for intra prediction. The extended reference line list consists of line indices {1, 3, 5, 7, 12} as shown Figure 7. For template-based intra mode derivation (TIMD), instead of the full MRL candidate list, only the first two reference line candidates, i.e., {1, 3}, are used.

Chart

Description automatically generated

Figure 7. Extended MRL candidate list.

### Template-based multiple reference line intra prediction

Template-based multiple reference line intra prediction (TMRL) mode combines reference line and prediction mode together and uses a template matching method to construct a list of candidate combinations. An index to the candidate combination list is coded to indicate which reference line and prediction mode is used in coding the current block. The regular multiple reference line (MRL) for the non-TIMD part is replaced by TMRL mode.

The TMRL mode extends reference line candidate list and the intra-prediction-mode candidate list. The extended reference line candidate list is {1, 3, 5, 7, 12}. The restriction on the top CTU row is unchanged. The size of the intra-prediction-mode candidate list is 10. The construction of the intra-prediction-mode candidate list is similar to MPM except the PLANAR mode is excluded from the intra-prediction-mode candidate list, DC mode is added after 5 neighboring PUs’ modes and DIMD modes if its not included and the angular modes with delta angles from to (compared the existing angular modes in the intra-prediction-mode candidate list) are added. The precision of angular prediction is extended from 65 to 129. Additionally non-adjacent positions are added as candidates in constructing the intra candidate list. If the neighbouring or non-adjacent blocks are coded with SGPM or GPM modes, the intra modes of the blocks are replaced by the partitioning angles.

The TMRL candidate is constructed as follows. There are 5x10=50 combinations of the extended reference line and the allowed intra-prediction modes for a block. Since the extended reference line starts from reference line 1, the area covered by reference line 0 is used for template matching. The SAD costs over the template area (see Figure 8) are calculated between the predictions (generated by 50 combinations) and the reconstructions. The 20 combinations with the least SAD cost are selected in an ascending order to form the TMRL candidate list.

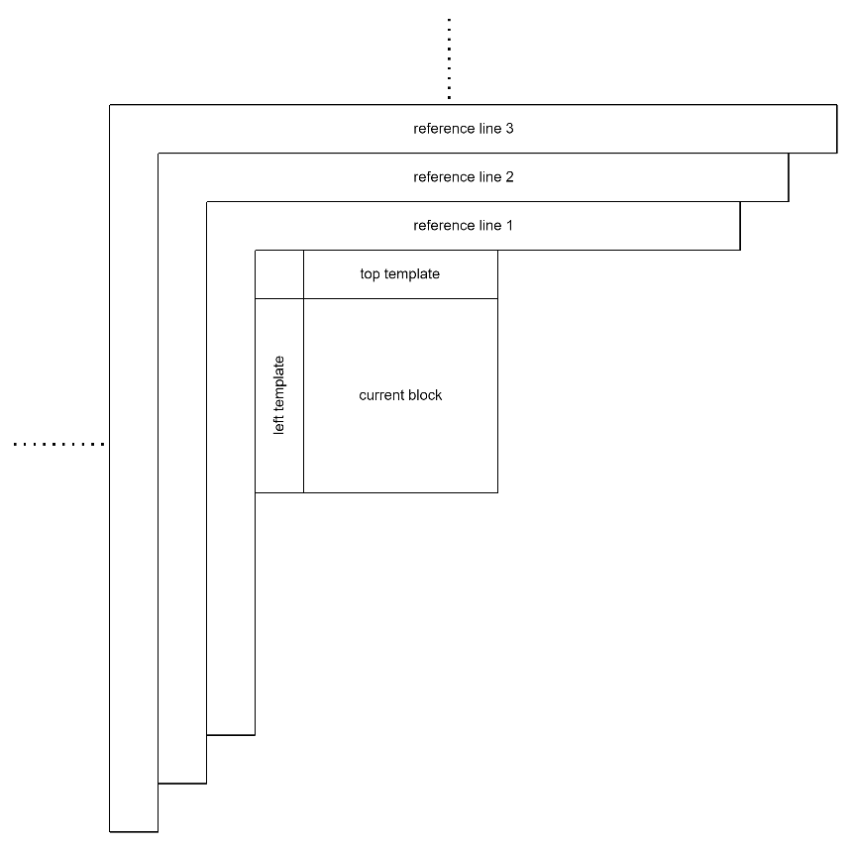


Figure 8. Illustration of the template area.

For TMR signalling instead of coding the reference line and the intra mode directly, an index to the TMRL candidate list is coded to indicate which combination of reference line and prediction mode is used for coding the current block.

### Convolutional cross-component intra prediction model

In this method convolutional cross-component model (CCCM) is applied to predict chroma samples from reconstructed luma samples in a similar spirit as done by the current CCLM modes. As with CCLM, the reconstructed luma samples are down-sampled to match the lower resolution chroma grid when chroma sub-sampling is used. Similar to CCLM top, left or top and left reference samples are used as templates for model derivation.

Also, similarly to CCLM, there is an option of using a single model or multi-model variant of CCCM. The multi-model variant uses two models, one model derived for samples above the average luma reference value and another model for the rest of the samples (following the spirit of the CCLM design). Multi-model CCCM mode can be selected for PUs which have at least 128 reference samples available.

#### Convolutional filter

The convolutional 7-tap filter consist of a 5-tap plus sign shape spatial component, a nonlinear term and a bias term. The input to the spatial 5-tap component of the filter consists of a center (C) luma sample which is collocated with the chroma sample to be predicted and its above/north (N), below/south (S), left/west (W) and right/east (E) neighbors as illustrated below.

A picture containing text, shoji, crossword puzzle

Description automatically generated

Figure 9. Spatial part of the convolutional filter.

The nonlinear term P is represented as power of two of the center luma sample C and scaled to the sample value range of the content:

P = ( C\*C + midVal ) >> bitDepth

That is, for 10-bit content it is calculated as:

P = ( C\*C + 512 ) >> 10

The bias term B represents a scalar offset between the input and output (similarly to the offset term in CCLM) and is set to middle chroma value (512 for 10-bit content).

Output of the filter is calculated as a convolution between the filter coefficients ci and the input values and clipped to the range of valid chroma samples:

predChromaVal = c0C + c1N + c2S + c3E + c4W + c5P + c6B

#### Calculation of filter coefficients

The filter coefficients ci are calculated by minimising MSE between predicted and reconstructed chroma samples in the reference area. Figure 10 illustrates the reference area which consists of 2 or 6 lines of chroma samples above and left of the PU. Whether to use 6 lines or 2 lines of neighbouring samples to derive the CCCM model parameters in the single model CCCM is determined by a template cost. Similarly, for the multi-model CCCM mode, the two candidates use 6 lines neighbouring luma samples or luma samples collocated to the current chroma block to derive mean values which separate samples into two groups. The cost is derived by applying the candidate CCP (either 2 or 6 lines) on a template, calculating the sum of absolute difference (SAD) between CCP predicted samples and reconstructed samples in the template.

Reference area extends one PU width to the right and one PU height below the PU boundaries. Area is adjusted to include only available samples. The extensions to the area shown in blue are needed to support the “side samples” of the plus shaped spatial filter and are padded when in unavailable areas.

Chart

Description automatically generated

Figure 10. Reference area (with its paddings) used to derive the filter coefficients.

The MSE minimization is performed by calculating autocorrelation matrix for the luma input and a cross-correlation vector between the luma input and chroma output. Autocorrelation matrix is LDL decomposed and the final filter coefficients are calculated using back-substitution. The process follows roughly the calculation of the ALF filter coefficients in ECM, however LDL decomposition was chosen instead of Cholesky decomposition to avoid using square root operations.

The autocorrelation matrix is calculated using the reconstructed values of luma and chroma samples. These samples are full range (e.g. between 0 and 1023 for 10-bit content) resulting in relatively large values in the autocorrelation matrix. This requires high bit depth operation during the model parameters calculation. It is proposed to remove fixed offsets from luma and chroma samples in each PU for each model. This is driving down the magnitudes of the values used in the model creation and allows reducing the precision needed for the fixed-point arithmetic. As a result, 16-bit decimal precision is proposed to be used instead of the 22-bit precision of the original CCCM implementation.

Reference sample values just outside of the top-left corner of the PU are used as the offsets (offsetLuma, offsetCb and offsetCr) for simplicity. The samples values used in both model creation and final prediction (i.e., luma and chroma in the reference area, and luma in the current PU) are reduced by these fixed values, as follows:

C' = C – offsetLuma

N' = N – offsetLuma

S' = S – offsetLuma

E' = E – offsetLuma

W' = W – offsetLuma

P' = nonLinear(C')

B = midValue = 1 << (bitDepth - 1)

and the chroma value is predicted using the following equation, where offsetChroma is equal to offsetCr and offsetCb for Cr and Cb components, respectively:

predChromaVal = c0C' + c1N' + c2S' + c3E' + c4W' + c5P' + c6B + offsetChroma

In order to avoid any additional sample level operations, the luma offset is removed during the luma reference sample interpolation. This can be done, for example, by substituting the rounding term used in the luma reference sample interpolation with an updated offset including both the rounding term and the offsetLuma. The chroma offset can be removed by deducting the chroma offset directly from the reference chroma samples. As an alternative way, impact of the chroma offset can be removed from the cross-component vector giving identical result. In order to add the chroma offset back to the output of the convolutional prediction operation the chroma offset is added to the bias term of the convolutional model.

The process of CCCM model parameter calculation requires division operations. Division operations are not always considered implementation friendly. The division operation are replaced with multiplication (with a scale factor) and shift operation, where scale factor and number of shifts are calculated based on denominator similar to the method used in calculation of CCLM parameters.

#### Gradient Linear Model

For YUV 4:2:0 color format, a gradient linear model (GLM) method can be used to predict the chroma samples from luma sample gradients. Two modes are supported: a two-parameter GLM mode and a three-parameter GLM mode.

Compared with the CCLM, instead of down-sampled luma values, the two-parameter GLM utilizes luma sample gradients to derive the linear model. Specifically, when the two-parameter GLM is applied, the input to the CCLM process, i.e., the down-sampled luma samples , are replaced by luma sample gradients . The other parts of the CCLM (e.g., parameter derivation, prediction sample linear transform) are kept unchanged.

In the three-parameter GLM, a chroma sample can be predicted based on both the luma sample gradients and down-sampled luma values with different parameters. The model parameters of the three-parameter GLM are derived from 6 rows and columns adjacent samples by the LDL decomposition based MSE minimization method as used in the CCCM.

For signaling, when the CCLM mode is enabled to the current CU, one flag is signaled to indicate whether GLM is enabled for both Cb and Cr components; if the GLM is enabled, another flag is signaled to indicate which of the two GLM modes is selected and one syntax element is further signaled to select one of 4 gradient filters for the gradient calculation.

* Four gradient filters are enabled for the GLM, as illustrated in Figure 11.

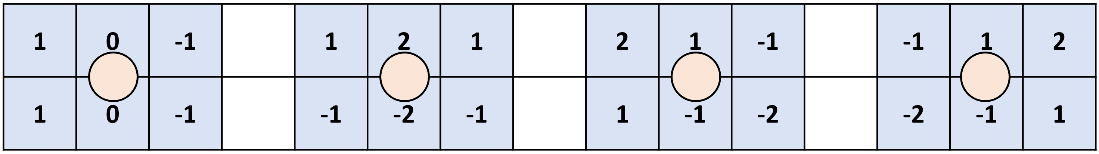


Figure 11. Four Sobel based gradient patterns for GLM.

#### CCCM signalling

Usage of the mode is signalled with a CABAC coded PU level flag. One new CABAC context was included to support this. When it comes to signalling, CCCM is considered a sub-mode of CCLM. That is, the CCCM flag is only signalled if intra prediction mode is LM\_CHROMA.

#### CCCM using non-downsampled luma samples

CCCM mode with 3x2 filter using non-downsampled luma samples is used, which consists of 6-tap spatial terms, four nonlinear terms and a bias term. The 6-tap spatial terms correspond to 6 neighboring luma samples (i.e., *L0, L1, …, L5*) around the chroma sample (i.e., C) to be predicted, the four non-linear terms are derived from the samples *L0, L1, L2,* and *L3* as shown in Figure 12.

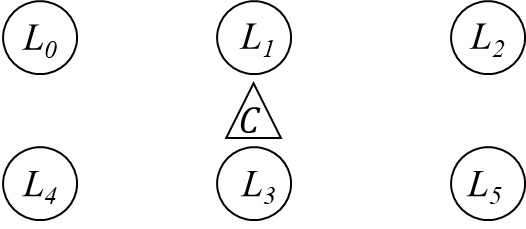


Figure 12. Non-downsampled luma samples.

where is the coefficient, is the offset. Same to the existing CCCM design, up to 6 lines/columns of chroma samples above and left to the current CU are applied to derive the filter coefficients. The filter coefficients are derived based on the same LDL decomposition method used in CCCM. The proposed method is signaled as an additional CCCM model besides the existing one, when the CCCM is selected, one single flag is signaled and used for both two chroma components to indicate whether the default CCCM model or the proposed CCCM model is applied. Additionally, SPS signaling is introduced to indicate whether the CCCM using non-downsampled luma samples is enabled.

#### Block-vector guided CCCM (BVG-CCCM)

When the co-located luma prediction is coded with IBC or IntraTMP in Intra slices, the BVG-CCCM mode can be used. In this mode, the block vectors of the co-located luma blocks, coded in IBC or intraTMP modes, are used to determine the reference area for calculating the CCCM parameters. The prediction is performed using uses the calculated model parameters and co-located luma samples. Figure 13 illustrates the reference area in BVG-CCCM method.

The BVG-CCCM mode uses an 11-tap filter for cross-component prediction as below:

predChromaVal = c0C + c1N + c2S + c3E + c4W + c5P(C) + c6P(N) + c7P(S) + c8P(W) + c9P(E)+ c10B

The input to the spatial 5-tap component of the filter consists of a center (C) luma sample which is collocated with the chroma sample to be predicted and its above/north (N), below/south (S), left/west (W) and right/east (E) neighbors as illustrated in Figure 1.

The nonlinear term P is represented as power of two of the corresponding luma sample and B is the bias term.

Diagram

Description automatically generated

Figure 13. Reference area for BVG-CCCM

Similar to Direct Block Vector (DBV) mode, five locations in the collocated luma block area are scanned and the associated block vectors are then used for determining the reference area for parameter calculation in BVG-CCCM method.

#### Gradient and Location based convolutional cross-component model (GL-CCCM)

This method maps luma values into chroma values using a filter with inputs consisting of one spatial luma sample, two gradient values, two location information, a nonlinear term, and a bias term. The GL-CCCM method uses gradient and location information instead of the 4 spatial neighbor samples used in the CCCM filter. The GL-CCCM filter used for the prediction is:

Where Gy and Gx are the vertical and horizontal gradients, respectively, and are calculated as Figure 14:

Moreover, the Y and X are the spatial coordinates of the center luma sample.

The rest of the parameters are the same as CCCM tool. The reference area for the parameter calculation is the same as CCCM method.

Table

Description automatically generated

Figure 14. Spatial samples used for GL-CCCM.

The usage of the mode is signalled with a CABAC coded PU level flag. When it comes to signalling, GL-CCCM is considered a sub-mode of CCCM. That is, the GL-CCCM flag is only signalled if original CCCM flag is true.

Similar to the CCCM, GL-CCCM tool has 6 modes for calculating the parameters:

* Single-model GL-CCCM from above and left templates
* Single-model GL-CCCM from above template
* Single-model GL-CCCM from left template
* Multi-model GL-CCCM from above and left templates
* Multi-model GL-CCCM from above template
* Multi-model GL-CCCM from left template

The encoder performs SATD search for the 6 GL-CCCM modes along with the existing CCCM modes to find the best candidates for full RD tests.

#### CCCM with Multiple Downsampling Filters

Multiple downsampling filters are applied to a group of reconstructed luma samples in a CCCM. The linear combination of these downsampled reconstructed samples is multiplied by derived filter coefficients to form the final chroma predictor. The horizontal or vertical location of the center luma sample are also considered in the tested model. The cross-component models shown below are tested as additional CCCM modes with a mode index signalled in the bitstream:

1. Model 1: predChroma = c0 \* H(C) + c1 \* G1(C) + c2 \* G2(C) + c3 \* G3(C) + c4 \* P(H(C)) + c5 \* P(G1(C)) + c6 \* P(G2(C)) + c7 \* X + c8 \* Y + c9 \* B
2. Model 2: predChroma = c0 \* H(C) + c1 \* H(W) + c2 \* H(E) + c3 \* G1(C) + c4 \* G1(W) + c5 \* G1(E) + c6 \* P(H(C)) + c7 \* P(H(W)) + c8 \* P(H(E)) + c9 \* X + c10 \* B
3. Model 3: predChroma = c0 \* H(C) + c1 \* H(NE) + c2 \* H(SW) + c3 \* G3(C) + c4 \* G3(NE) + c5 \* G3(SW) + c6 \* P(H(C)) + c7 \* P(H(NE)) + c8 \* P(H(SW)) + c9 \* Y + c10 \* B

where H(·), G1(·), G2(·), G3(·) are various downsampling filters as indicated in Figure 15, C denotes the current chroma sample position, and N, S, W, E, NE, SW are the positions around C, ci are filter coefficients, P and B are nonlinear term and bias term, and X and Y are the horizontal and vertical locations of the center luma sample with respect to the top-left coordinates of the block.

A picture containing text, pool ball, sport

Description automatically generated

Figure 15. Various downsampling filters used in cross-component models.

### Local-Boosting Cross-Component Prediction (LB-CCP)

Prediction samples of MM-CCLM/MM-CCCM can be filtered with neighbouring samples. As shown in Figure 16, a 3×3 low-pass filter is applied to filter prediction samples generated by MM-CCLM/MM-CCCM. For a sample at a top/left boundary, the filtering window may involve neighbouring reconstructed samples. For inner samples, the filtering window only involves prediction samples, which may be padded. A flag is signaled to indicate whether filtering is applied or not for a block coded with MM-CCLM/MM-CCCM.



Figure 16. Filter on samples of MM-CCLM/MM-CCCM.

### Cross-Component Prediction (CCP) merge (a.k.a., non-local CCP) mode

For chroma coding, a flag is signalled to indicate whether CCP mode (including the CCLM, CCCM, GLM and their variants) or non-CCP mode (conventional chroma intra prediction mode, fusion of chroma intra prediction mode) is used. If the CCP mode is selected, one more flag is signalled to indicate how to derive the CCP type and parameters, i.e., either from a CCP merge list or signalled/derived on-the-fly. a CCP merge candidate list is constructed from the spatial adjacent, temporal, spatial non-adjacent, history-based m or shifted temporal candidates. After including these candidates, default models are further included to fill the remaining empty positions in the merge list. In order to remove redundant CCP models in the list, pruning operation is applied. After constructing the list, the CCP models in the list are reordered depending on the SAD costs, which are obtained using the neighbouring template of the current block. More details are described below.

**Spatial adjacent and non-adjacent candidates**

The positions and inclusion order of the spatial adjacent and non-adjacent candidates are the same as those defined in ECM for regular inter merge prediction candidates.

**Temporal and shifted temporal candidates**

Temporal candidates are selected from the collocated picture. The position and inclusion order of the temporal candidates are the same as those defined in ECM for regular inter merge prediction candidates. The shifted temporal candidates are also selected from the collocated picture. The position of temporal candidates is shifted by a selected motion vector which is derived from motion vectors of neighboring blocks.

**History-based candidates**

A history-based table is maintained to include the recently used CCP models, and the table is reset at the beginning of each CTU row. If the current list is not full after including spatial adjacent and non-adjacent candidates, the CCP models in the history-based table are added into the list.

**Default candidates**

CCLM candidates with default scaling parameters are considered, only when the list is not full after including the spatial adjacent, spatial non-adjacent, or history-based candidates. If the current list has no candidates with the single model CCLM mode, the default scaling parameters are {0, 1/8, -1/8, 2/8, -2/8, 3/8, -3/8, 4/8, -4/8, 5/8, -5/8, 6/8}. Otherwise, the default scaling parameters are {0, the scaling parameter of the first CCLM candidate + {1/8, -1/8, 2/8, -2/8, 3/8, -3/8, 4/8, -4/8, 5/8, -5/8, 6/8}}.

A flag is signaled to indicate whether the CCP merge mode is applied or not. If CCP merge mode is applied, an index is signaled to indicate which candidate model is used by the current block. In addition, CCP merge mode is not allowed for the current chroma coding block when the current CU is coded by intra sub-partitions (ISP) with single tree, or the current chroma coding block size is less than or equal to 16.

### Spatial Geometric partitioning mode (SGPM)

SGPM is an intra mode that resembles the inter coding tool of GPM, where the two prediction parts are generated from intra predicted process. In this mode, a candidate list is built with each entry containing one partition split and two intra prediction modes as shown in Figure 17. 26 partition modes and 3 of intra prediction modes are used to form the combinations. the length of the candidate list is set equal to 16. The selected candidate index is signalled.

Figure 17. Spatial GPM candidates.

The list is reordered using template (Figure 18) where SAD between the prediction and reconstruction of the template is used for ordering. The template size is fixed to 1.



Figure 18. GPM template.

For each partition mode, an IPM list is derived for each part using the same intra-inter GPM list derivation (Sec. 3.2.22). The IPM list size is set to 3. In the list, TIMD derived mode is replaced by 2 derived modes with horizontal and vertical orientations.

The SGPM mode is applied with a restricted blocks size: 4<=width<=64, 4<=height<=64, width<height\*8, height<width\*8, width\*height>=32.

A PPS flag is coded to indicate whether no blending of two intra predictions is allowed. When this PPS flag is set to false, the following adaptive blending is also used for spatial GPM, where blending depth τ shown in Figure 19 is derived as follows:

* If min(width, height)==4, 1/2 τ is selected
* else if min(width, height)==8, τ is selected
* else if min(width, height)==16, 2 τ is selected
* else if min(width, height)==32, 4 τ is selected
* else, 8 τ is selected.

Otherwise (the PPS flag is set to true), 1/4 τ is always used for spatial GPM coded blocks to make sure no blending is used when SGPM block has partition angle completely horizontal or vertical, and much narrower blending width is used when SGPM block has other partition angles. It is noted that the flag is set to true in current Common Test Conditions (CTC) for the screen content videos.

Chart

Description automatically generated

Figure 19. GPM blending.

### Directional planar mode

Two additional planar modes where only the horizontal interpolation or only the vertical interpolation are used to obtain the predicted samples.

For planar horizontal mode, only the horizontal linear interpolation is performed based on the left reference sample and the top-right reference sample to predict the current sample as:

For planar vertical mode, only the vertical linear interpolation is performed based on the above reference sample and the bottom-left reference sample to predict the current sample as:

The transform kernel selection for planar horizontal and planar vertical mode is shown in Figure 20. If an intra prediction mode of a current block is the planar vertical mode, the horizontal intra prediction mode is used to derive a transform kernel in MTS set and LFNST set. Also, if an intra prediction mode of a current block is the planar horizontal mode, the vertical intra prediction mode is used to derive a transform kernel in MTS set and LFNST set.

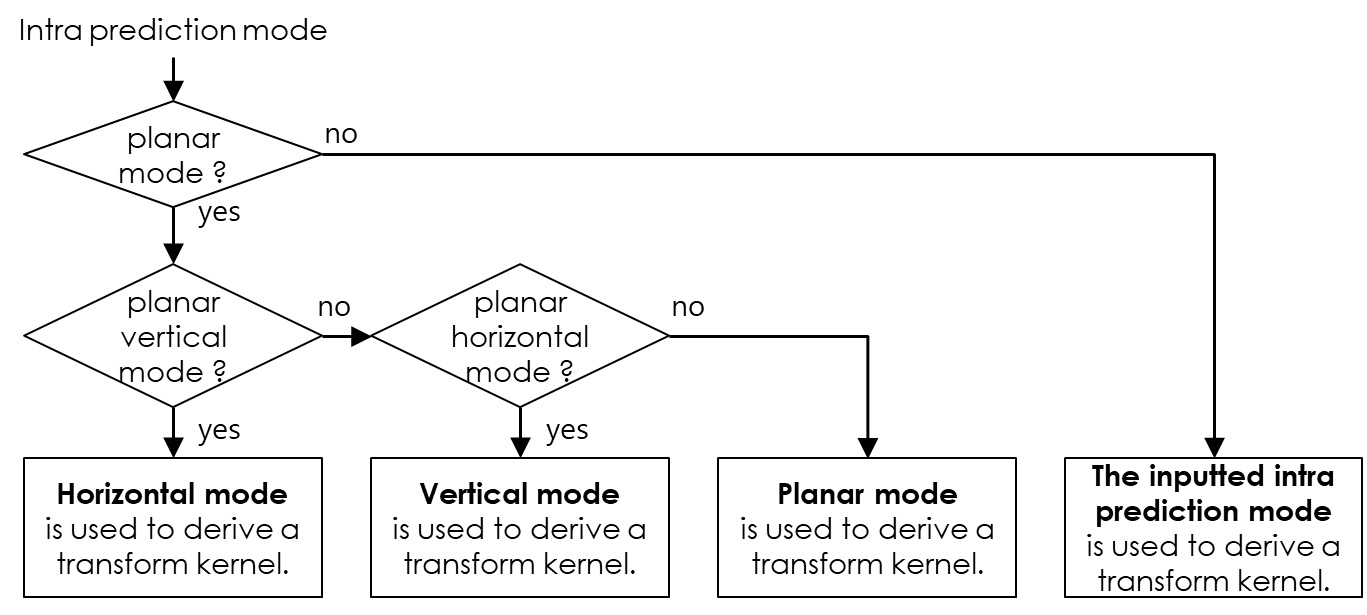


Figure 20. Transform selection process for directional planar modes.

### Direct block vector (DBV) for chroma block

The direct block vector is used for chroma blocks. A flag is signaled to indicate whether a chroma block is coded using IBC mode. If one of the luma blocks in five locations shown in Figure 21 is coded with IBC or intraTMP mode, its block vector is scaled and is used as block vector for the chroma block. Template matching is used to perform block vector scaling.

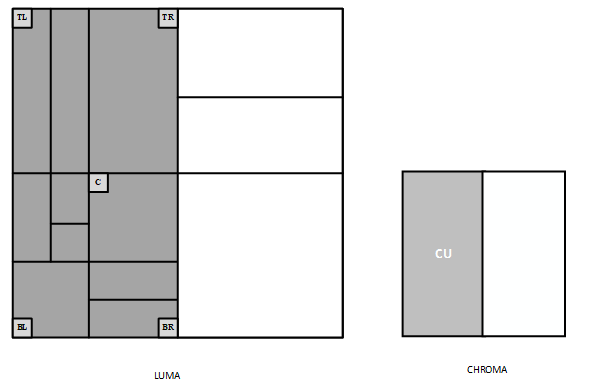


Figure 21. Luma blocks used to derive direct block vector.

## Inter prediction

### Local illumination compensation (LIC)

LIC is an inter prediction technique to model local illumination variation between current block and its prediction block as a function of that between current block template and reference block template. The parameters of the function can be denoted by a scale *α* and an offset *β*, which forms a linear equation, that is, *α*\*p[x]+*β* to compensate illumination changes, where p[x] is a reference sample pointed to by MV at a location x on reference picture. When wrap around motion compensation is enabled, the MV shall be clipped with wrap around offset taken into consideration. Since *α* and *β* can be derived based on current block template and reference block template, no signaling overhead is required for them, except that an LIC flag is signaled for AMVP mode to indicate the use of LIC. For the merge mode, the LIC flag is not inherited from a merge candidate, instead, it is derived on-the-fly. More specifically, of a merge candidate is derived by comparing two template costs: a SAD-based template cost, denoted as C0, and a Mean Removal SAD (MRSAD)-based template cost, denoted as C1. The LIC flag is set to be false, if C0 <= C1 and is set to be true, if C0 > C1. To favor the inherited LIC flag, C0 is multiplied by *α* if the inherited LIC flag is false while C1 is multiplied by *α* if the inherited LIC flag is true, where *α* < 1.

The local illumination compensation proposed in JVET-O0066 is used for inter CUs with the following modifications.

* Intra neighbor samples can be used in LIC parameter derivation;
* LIC is disabled for blocks with less than 32 luma samples;
* For both non-subblock and affine modes, LIC parameter derivation is performed based on the template block samples corresponding to the current CU, instead of partial template block samples corresponding to first top-left 16x16 unit;
* Samples of the reference block template are generated by using MC with the block MV without rounding it to integer-pel precision.

For the bi-predictive inter CUs, two sets of LIC parameters are separately derived for L0 and L1 prediction samples. An iterative manner to derive the L0 and L1 LIC parameters is applied. Specifically, L0 LIC parameters are firstly derived by minimizing difference between L0 template prediction T0 and the template T and the samples in T are updated by subtracting the corresponding samples in T0. Then, the L1 parameters are calculated that minimizes the difference between L1 template prediction T1 and the updated template. Finally, the L0 parameter is refined again in the same way.

### Non-adjacent spatial candidate

The non-adjacent spatial merge candidates as in JVET-L0399 are inserted after the temporal motion vector prediction (TMVP) in the regular merge candidate list. The pattern of spatial merge candidates is shown in Figure 22. The distances between non-adjacent spatial candidates and current coding block are based on the width and height of current coding block. The line buffer restriction is not applied.

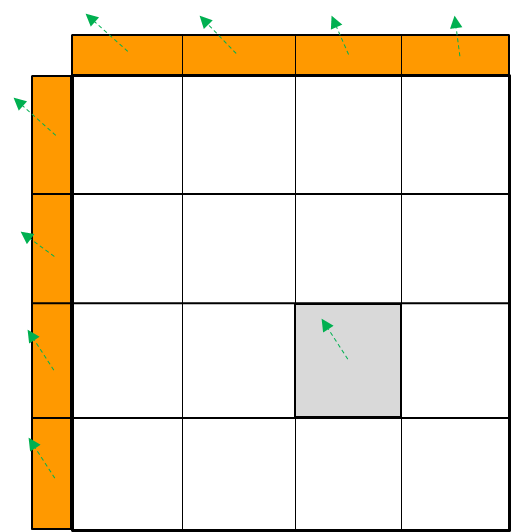


Figure 22. Spatial neighboring blocks used to derive the spatial merge candidates.

### Temporal motion information derivation

In VVC, the Temporal Motion Vector Prediction (TMVP) for the AMVP and merge mode is derived by fetching the motion information from the center or the bottom-right of the collocated block in a signaled collocated picture. Similarly, for the Subblock-based Temporal Motion Vector Prediction (SbTMVP) mode, the motion information from the left neighboring position is used as a motion shift, which is then employed to obtain TMVPs at sub-CU level.

In ECM, to further improve the coding efficiency of TMVP, two aspects are modified. Firstly, two collocated pictures are utilized which are the two reference frames with the least POC distance relative to the to-be-coded frame. Secondly, the motion shift to locate TMVP is adaptively determined from multiple locations according to template costs. More specifically, two motion shift candidate lists are constructed respectively for the two collocated frames. The motion shifts with the minimum template matching cost are used to derive SbTMVP or TMVP candidates. At most 4 SbTMVP candidates are included in the sub-block-based merge list. The SbTMVP candidate with the least template matching cost derived from the first collocated frame is placed in the first entry without reordering, while other SbTMVP candidates are sorted together with affine candidates. In addition, the prediction direction of each subblock template is determined based on the center subblock. As illustrated in Figure 23, if the center subblock is uni-predicted, then all the subblock templates are uni-predicted, and vice versa. If the motion vector of corresponding adjacent subblock at the determined reference list is not available for a subblock template, zero MV is used for that subblock template.



**Figure 23. Subblock templates generation of SbTMVP**

### Template matching (TM)

Template matching (TM) is a decoder-side MV derivation method to refine the motion information of the current CU by finding the closest match between a template (i.e., top and/or left neighbouring blocks of the current CU) in the current picture and a block (i.e., same size to the template) in a reference picture. As illustrated in Figure 24, a better MV is searched around the initial motion of the current CU within a [– 8, +8]-pel search range. The template matching method in JVET-J0021 is used with the following modifications: search step size is determined based on AMVR mode and TM can be cascaded with bilateral matching process in merge modes.

Diagram

Description automatically generated with medium confidence

Figure 24. Template matching performs on a search area around initial MV.

In AMVP mode, an MVP candidate is determined based on template matching error to select the one which reaches the minimum difference between the current block template and the reference block template, and then TM is performed only for this particular MVP candidate for MV refinement. TM refines this MVP candidate, starting from full-pel MVD precision (or 4-pel for 4-pel AMVR mode) within a [–8, +8]-pel search range by using iterative 16-point diamond search. The AMVP candidate may be further refined by using cross search with full-pel MVD precision (or 4-pel for 4-pel AMVR mode), followed sequentially by half-pel and quarter-pel ones depending on AMVR mode as specified in Table 2. This search process ensures that the MVP candidate still keeps the same MV precision as indicated by the AMVR mode after TM process. In the search process, if the difference between the previous minimum cost and the current minimum cost in the iteration is less than a threshold that is equal to the area of the block, the search process terminates.

Table 2. Search patterns of AMVR and merge mode with AMVR.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Search pattern** | **AMVR mode** | | | | **Merge mode** | |
| **4-pel** | **Full-pel** | **Half-pel** | **Quarter-pel** | **AltIF=0** | **AltIF=1** |
| 4-pel diamond | v |  |  |  |  |  |
| 4-pel cross | v |  |  |  |  |  |
| Full-pel diamond |  | v | v | v | v | v |
| Full-pel cross |  | v | v | v | v | v |
| Half-pel cross |  |  | v | v | v | v |
| Quarter-pel cross |  |  |  | v | v |  |
| 1/8-pel cross |  |  |  |  | v |  |

In merge mode, similar search method is applied to the merge candidate indicated by the merge index. As Table 2 shows, TM may perform all the way down to 1/8-pel MVD precision or skipping those beyond half-pel MVD precision, depending on whether the alternative interpolation filter (that is used when AMVR is of half-pel mode) is used according to merged motion information. Besides, when TM mode is enabled, template matching may work as an independent process or an extra MV refinement process between block-based and subblock-based bilateral matching (BM) methods, depending on whether BM can be enabled or not according to its enabling condition check.

When TM is applied to bi-predictive blocks, an iterative process is used. Specifically, the initial motion vectors of L0 and L1 are firstly refined and TM costs Cost0 and Cost1 are calculated for L0 and L1, respectively. When Cost0 is larger than Cost1, the refined motion vector of L1 () is used to derive a further refined motion vector of L0 (). Then, the is further refined using . Similarly, when Cost0 is not larger than Cost1, the refined motion vector of L0 () is used to derive a further refined motion vector of L1 (), and the is further refined using . Besides, TM for bi-prediction is enabled when DMVR condition is satisfied.

#### TM-based subblock motion refinement

In JVET-AF0168 test3.4a, it is proposed to apply the template matching to subblock based motion tools, including the affine and SbTMVP mode. More specifically, the control point motion vectors (CPMVs) of uni-predicted affine merge candidates and the motion shift of SbTMVP candidates are refined using TM. For a uni-predicted affine merge candidate, a same MV offset is assigned to all the CPMVs, and the TM cost of the affine candidate is calculated accordingly. The optimal CPMV offset with the minimum TM cost can be used to refine the corresponding affine candidate. For a SbTMVP candidate, the initial motion shift can be refined with TM, and then the refined motion shift will be utilized to derive subblock temporal motion information.

### Multi-pass decoder-side motion vector refinement

A multi-pass decoder-side motion vector refinement is applied. In the first pass, bilateral matching (BM) is applied to the coding block. In the second pass, BM is applied to each 16x16 subblock within the coding block. In the third pass, MV in each 8x8 subblock is refined by applying bi-directional optical flow (BDOF). The refined MVs are stored for both spatial and temporal motion vector prediction.

#### First pass – Block based bilateral matching MV refinement

In the first pass, a refined MV is derived by applying BM to a coding block. Similar to decoder-side motion vector refinement (DMVR), in bi-prediction operation, a refined MV is searched around the two initial MVs (MV0 and MV1) in the reference picture lists L0 and L1. The refined MVs (MV0\_pass1 and MV1\_pass1) are derived around the initiate MVs based on the minimum bilateral matching cost between the two reference blocks in L0 and L1.

BM performs local search to derive integer sample precision intDeltaMV. The local search applies a 3×3 square search pattern to loop through the search range [–sHor, sHor] in horizontal direction and [–sVer, sVer] in vertical direction, wherein, the values of sHor and sVer are determined by the block dimension, and the maximum value of sHor and sVer is 8.

The bilateral matching cost is calculated as: bilCost = mvDistanceCost + sadCost. When the block size cbW \* cbH is greater than 64, mean-removal SAD (MRSAD) cost function is applied to remove the DC effect of distortion between reference blocks. When the bilCost at the center point of the 3×3 search pattern has the minimum cost, the intDeltaMV local search is terminated. Otherwise, the current minimum cost search point becomes the new center point of the 3×3 search pattern and continue to search for the minimum cost, until it reaches the end of the search range.

The existing fractional sample refinement is further applied to derive the final deltaMV. The refined MVs after the first pass is then derived as:

* MV0\_pass1 = MV0 + deltaMV
* MV1\_pass1 = MV1 – deltaMV

#### Second pass – Subblock based bilateral matching MV refinement

In the second pass, a refined MV is derived by applying BM to a 16×16 grid subblock. For each subblock, a refined MV is searched around the two MVs (MV0\_pass1 and MV1\_pass1), obtained on the first pass, in the reference picture list L0 and L1. The refined MVs (MV0\_pass2(sbIdx2) and MV1\_pass2(sbIdx2)) are derived based on the minimum bilateral matching cost between the two reference subblocks in L0 and L1.

For each subblock, BM performs full search to derive integer sample precision intDeltaMV. The full search has a search range [–sHor, sHor] in horizontal direction and [– sVer, sVer] in vertical direction, wherein, the values of sHor and sVer are determined by the block dimension, and the maximum value of sHor and sVer is 8.

The bilateral matching cost is calculated by applying a cost factor to the SATD cost between two reference subblocks, as: bilCost = satdCost \* costFactor. The search area (2\*sHor + 1) \* (2\*sVer + 1) is divided up to 5 diamond shape search regions shown on Figure 25. Each search region is assigned a costFactor, which is determined by the distance (intDeltaMV) between each search point and the starting MV, and each diamond region is processed in the order starting from the center of the search area. In each region, the search points are processed in the raster scan order starting from the top left going to the bottom right corner of the region. When the minimum bilCost within the current search region is less than a threshold equal to sbW \* sbH, the int-pel full search is terminated, otherwise, the int-pel full search continues to the next search region until all search points are examined. Additionally, if the difference between the previous minimum cost and the current minimum cost in the iteration is less than a threshold that is equal to the area of the block, the search process terminates.



Figure 25. Diamond regions in the search area

The existing VVC DMVR fractional sample refinement is further applied to derive the final deltaMV(sbIdx2). The refined MVs at second pass is then derived as:

* MV0\_pass2(sbIdx2) = MV0\_pass1 + deltaMV(sbIdx2)
* MV1\_pass2(sbIdx2) = MV1\_pass1 – deltaMV(sbIdx2)

#### Third pass – Subblock based bi-directional optical flow MV refinement

In the third pass, a refined MV is derived by applying BDOF to an 8×8 grid subblock. For each 8×8 subblock, BDOF refinement is applied to derive scaled Vx and Vy without clipping starting from the refined MV of the parent subblock of the second pass. The derived bioMv(Vx, Vy) is rounded to 1/16 sample precision and clipped between -32 and 32.

The refined MVs (MV0\_pass3(sbIdx3) and MV1\_pass3(sbIdx3)) at third pass are derived as:

* MV0\_pass3(sbIdx3) = MV0\_pass2(sbIdx2) + bioMv
* MV1\_pass3(sbIdx3) = MV0\_pass2(sbIdx2) – bioMv

#### Fourth pass – Adaptive subblock based bi-directional optical flow MV refinement

In the fourth pass, a refined MV is derived by applying BDOF to an 4×4 or 8×8 grid subblock. When a block is smaller than 1024 pixels, the 4×4 grid subblock is used. Otherwise, 8×8 grid subblock is used. The MV of each subblock is refined in the same way as that used in third pass.

In all aforementioned sub- clauses, when wrap around motion compensation is enabled, the motion vectors shall be clipped with wrap around offset taken into consideration.

### Adaptive decoder-side motion vector refinement

Adaptive decoder side motion vector refinement method is an extension of multi-pass DMVR which consists of the two new merge modes to refine MV only in one direction, either L0 or L1, of the bi-prediction for the merge candidates that meet the DMVR conditions. The multi-pass DMVR process is applied for the selected merge candidate to refine the motion vectors, however either MVD0 or MVD1 is set to zero in the 1st pass (i.e., PU level) DMVR.

The merge candidates for the new merge mode are derived from spatial neighboring coded blocks, TMVPs, non-adjacent blocks, HMVPs, pair-wise candidate, similar as in the regular merge mode. The difference is that only those meet DMVR conditions are added into the candidate list. The same merge candidate list is used by the two new merge modes. If the list of BM candidates contains the inherited BCW weights and DMVR process is unchanged except the computation of the distortion is made using MRSAD or MRSATD if the weights are non-equal and the bi-prediction is weighted with BCW weights. Merge index is coded as in regular merge mode.

### OBMC

When OBMC is applied, top and left boundary pixels of a CU are refined using neighboring block’s motion information with a weighted prediction as described in JVET-L0101.

Conditions of not applying OBMC are as follows:

* When OBMC is disabled at SPS level
* When current block has intra mode or IBC mode
* When current luma block area is smaller or equal to 32

Additionally OBMC is adaptively controlled on a block level as follows:

* OBMC flag is inherited from a neighboring affine block for affine merge mode.
* OBMC is not applied to a block if there is a neighbor block coded with IBC, palette, or BDPCM modes.
* When applying OBMC to a block, block boundary check whether OBMC is applied to the boundary is further made based on the reference samples of the current block. If any absolute difference between the prediction sample and non-interpolated (integer pel) reference sample is greater than a threshold, the OBMC is not applied to that boundary.

A subblock-boundary OBMC is performed by applying the same blending to the top, left, bottom, and right subblock boundary pixels using neighboring subblocks’ motion information. It is enabled for the subblock based coding tools:

* Affine AMVP modes;
* Affine merge modes and subblock-based temporal motion vector prediction (SbTMVP);
* Subblock-based bilateral matching.

When OBMC mode is used in CIIP mode with LMCS, inter blending is performed prior to LMCS mapping of inter samples. LMCS is applied to blended inter samples which are combined with LMCS applied intra samples in CIIP mode,

, where represents the samples predicted by the motion of current block in the original domain, represents the samples predicted in the mapped domain, represents the samples predicted by the motion of neighboring blocks in the original domain, and and are the weights.

When OBMC mode is used in a LIC coded block, the LIC parameters are applied to generate the corresponding prediction samples for the OBMC of the LIC coded block. Besides, to reduce the complexity, the OBMC is only applied to the top and left CU boundaries while being always disabled for the boundaries of the internal sub-blocks of the LIC coded block.

### Template matching based OBMC

In template matching based OBMC scheme, instead of directly using the weighted prediction, the prediction value of CU boundary samples derivation approach is decided according to the template matching costs, including using current block’s motion information only, or using neighboring block’s motion information as well with one of the blending modes.

In this scheme for each block with a size of 4×4 at the top CU boundary, the above template size equals to 4×1. If *N* adjacent blocks have the same motion information, then the above template size is enlarged to 4*N*×1 since the MC operation can be processed at one time. For each left block with a size of 4×4 at the left CU boundary, the left template size equals to 1×4 or 1×4*N* (Figure 26).



Figure 26. Template.

For each 4×4 top block (or *N* 4×4 blocks group), the prediction value of boundary samples is derived following the below steps.

Take block *A* as the current block and its above neighboring block *AboveNeighbor\_A* for example. The operation for left blocks is conducted in the same manner.

First, three template matching costs (, , ) are measured by SAD between the reconstructed samples of a template and its corresponding reference samples derived by MC process according to the following three types of motion information:

is calculated according to *A*’s motion information.

is calculated according to *AboveNeighbor\_A*’s motion information.

is calculated according to weighted prediction of *A*’s *and AboveNeighbor\_A*’s motion information with weighting factors as and respectively.

Second, choose one approach to calculate the final prediction results of boundary samples by comparing *Cost1*, *Cost2* and *Cost 3*.

The original MC result using current block’s motion information is denoted as , and the MC result using neighboring block’s motion information is denoted as . The final prediction result is denoted as .

* If *Cost1* is minimum, then .
* If (*Cost2* + (*Cost2* >> 2) + (*Cost2* >> 3)) <= *Cost1*, then blending mode 1 is used.

For luma blocks, the number of blending pixel rows is 4.

For chroma blocks, the number of blending pixel rows is 1.

* If *Cost1* <= *Cost2*, then blending mode 2 is used.

For luma blocks, the number of blending pixel rows is 2.

For chroma blocks, the number of blending pixel rows/columns is 1.

* Otherwise, blending mode 3 is used.

For luma blocks, the number of blending pixel rows is 4.

For chroma blocks, the number of blending pixel rows is 1.

### History-parameter-based affine model inheritance and non-adjacent affine mode

History-parameter-based affine model inheritance (HAMI) allows the affine model to be inherited from a previously affine-coded block which may not be neighboring to the current block. Similar to the enhanced regular merge mode, non-adjacent affine mode (NA-AFF) is introduced.

A first history-parameter table (HPT) is established. An entry of the first HPT stores a set of affine parameters: *a*, *b*, *c* and *d*, each of which is represented by a 16-bit signed integer. Entries in HPT is categorized by reference list and reference index. Five reference indices are supported for each reference list in HPT. In a formular way, the category of HPT (denoted as HPTCat) is calculated as

HPTCat (RefList, RefIdx) = 5×RefList + min (RefIdx, 4),

wherein RefList and RefIdx represents a reference picture list (0 or 1) and a reference index, respectively. For each category, at most seven entries can be stored, resulting in 70 entries totally in HPT. At the beginning of each CTU row, the number of entries for each category is initialized as zero. After decoding an affine-coded CU with reference list RefListcur and RefIdxcur, the affine parameters are utilized to update entries in the category HPTCat(RefListcur, RefIdxcur) in a way similar to HMVP table updating.

A history-affine-parameter-based candidate (HAPC) is derived from one of the seven neighbouring 4×4 blocks denoted as A0, A1, A2, B0, B1, B2 or B3 in Fig. 1 and a set of affine parameters stored in a corresponding entry in the first HPT. The MV of a neighbouring 4×4 block served as the base MV. In a formulating way, the MV of the current block at position (*x*, *y*) is calculated as:

,

where (*mvhbase*, *mvvbase*) represents the MV of the neighbouring 4×4 block, (*xbase*, *ybase*) represents the center position of the neighbouring 4×4 block. (*x*, *y*) can be the top-left, top-right and bottom-left corner of the current block to obtain the corner-position MVs (CPMVs) for the current block, or it can be the center of the current block to obtain a regular MV for the current block.

A second history-parameter table (HPT) with base MV information is also appended. There are nine entries in the second HPT, wherein an entry comprises a base MV, a reference index and four affine parameters for each reference list, and a base position. An additional merge HAPC can be generated from the second HPT with the base MV information the corresponding affine models stored in an entry. The difference between the first HPT and the second HPT is illustrated in Figure 27.

Moreover, pair-wised affine merge candidates are generated by two affine merge candidates which are history-derived or not history-derived. A pair-wised affine merge candidates is generated by averaging the CPMVs of existing affine merge candidates in the list.

As a response to new HAPCs being introduced, the size of sub-block-based merge candidate list is increased from five to fifteen, which are all involved in the ARMC process.



Figure 27. First HPT and the second HPT.

In NA-AFF, the pattern of obtaining non-adjacent spatial neighbors is shown in Fig. 3. Same as the existing non-adjacent regular merge candidates [8], the distances between non-adjacent spatial neighbors and current coding block in the NA-AFF are also defined based on the width and height of current CU.

The motion information of the non-adjacent spatial neighbors in Fig. 3 is utilized to generate additional inherited and constructed affine merge/AMVP candidates. Specifically, for inherited candidates, the same derivation process of the inherited affine merge/AMVP candidates in the VVC is kept unchanged except that the CPMVs are inherited from non-adjacent spatial neighbors. The non-adjacent spatial neighbors are checked based on their distances to the current block, i.e., from near to far. At a specific distance, only the first available neighbor (that is coded with the affine mode) from each side (e.g., the left and above) of the current block is included for inherited candidate derivation. As indicated by the red dash arrows in Figure 28(a), the checking orders of the neighbors on the left and above sides are bottom-to-up and right-to-left, respectively.

For the first type of constructed candidates, as shown in the Figure 28(b), the positions of one left and above non-adjacent spatial neighbors are firstly determined independently; After that, the location of the top-left neighbor can be determined accordingly which can enclose a rectangular virtual block together with the left and above non-adjacent neighbors. Then, as shown in the Figure 29, the motion information of the three non-adjacent neighbors is used to form the CPMVs at the top-left (A), top-right (B) and bottom-left (C) of the virtual block, which is finally projected to the current CU to generate the corresponding constructed candidates.

The NA-AFF candidates are inserted into the existing affine merge candidate list and affine AMVP candidate list according to the following orders:

***Affine merge mode:***

1. SbTMVP candidate, if available
2. Inherited from adjacent neighbors
3. Inherited from non-adjacent neighbors
4. Constructed from adjacent neighbors
5. The first type of constructed affine candidates from non-adjacent neighbors
6. Zero MVs

***Affine AMVP mode:***

1. Inherited from adjacent neighbors
2. Constructed from adjacent neighbors
3. Translational MVs from adjacent neighbors
4. Translational MVs from temporal neighbors
5. Inherited from non-adjacent neighbors
6. The first type of constructed affine candidates from non-adjacent neighbors
7. Zero MVs

Due to the inclusion of the additional candidates generated by NA-AFF, the size of the affine merge candidate list is increased from 5 to 15. The subgroup size of ARMC for the affine merge mode is increased from 3 to 15.



Figure 28. Spatial neighbors for deriving affine merge/AMVP candidates: (a) for deriving inherited candidates (b) for deriving the first type of constructed candidates.



Figure 29. From non-adjacent neighbors to the first type of constructed affine merge/AMVP candidates.

In NA-AFF:

1. The area from where the non-adjacent neighbors come is restricted to be within the current CTU (i.e., no additional storage requirements for line buffer).
2. The storage granularity for affine motion information, including CPMVs and reference indexes, is reduced from 8x8 to 16x16 (i.e., only the affine motion from the top-left 8x8 block is saved). Additionally, the saved CPMVs are projected to each 16x16 block before storage, such that the position and size information are not needed.
3. Only the top-left and top-right CPMVs are stored (i.e., always using 4-parameter affine model for NA-AFF).

### Sample-based BDOF

In the sample-based BDOF, instead of deriving motion refinement (Vx, Vy) on a block basis, it is performed per sample.

The coding block is divided into 8×8 subblocks. For each subblock, whether to apply BDOF or not is determined by checking the SAD between the two reference subblocks against a threshold. If decided to apply BDOF to a subblock, for every sample in the subblock, a sliding 5×5 window is used and the existing BDOF process is applied for every sliding window to derive Vx and Vy. The derived motion refinement (Vx, Vy) is applied to adjust the bi-predicted sample value for the center sample of the window.

### Interpolation

The 8-tap interpolation filter used in VVC is replaced with a 12-tap filter. The interpolation filter is derived from the sinc function of which the frequency response is cut off at Nyquist frequency and cropped by a cosine window function. Table 3 gives the filter coefficients of all 16 phases. Figure 30 compares the frequency responses of the interpolation filters with the VVC interpolation filter, all at half-pel phase.

Table 3. Filter coefficients of the 12-tap interpolation filter

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1/16 | -1 | 2 | -3 | 6 | -14 | 254 | 16 | -7 | 4 | -2 | 1 | 0 |
| 2/16 | -1 | 3 | -7 | 12 | -26 | 249 | 35 | -15 | 8 | -4 | 2 | 0 |
| 3/16 | -2 | 5 | -9 | 17 | -36 | 241 | 54 | -22 | 12 | -6 | 3 | -1 |
| 4/16 | -2 | 5 | -11 | 21 | -43 | 230 | 75 | -29 | 15 | -8 | 4 | -1 |
| 5/16 | -2 | 6 | -13 | 24 | -48 | 216 | 97 | -36 | 19 | -10 | 4 | -1 |
| 6/16 | -2 | 7 | -14 | 25 | -51 | 200 | 119 | -42 | 22 | -12 | 5 | -1 |
| 7/16 | -2 | 7 | -14 | 26 | -51 | 181 | 140 | -46 | 24 | -13 | 6 | -2 |
| 8/16 | -2 | 6 | -13 | 25 | -50 | 162 | 162 | -50 | 25 | -13 | 6 | -2 |
| 9/16 | -2 | 6 | -13 | 24 | -46 | 140 | 181 | -51 | 26 | -14 | 7 | -2 |
| 10/16 | -1 | 5 | -12 | 22 | -42 | 119 | 200 | -51 | 25 | -14 | 7 | -2 |
| 11/16 | -1 | 4 | -10 | 19 | -36 | 97 | 216 | -48 | 24 | -13 | 6 | -2 |
| 12/16 | -1 | 4 | -8 | 15 | -29 | 75 | 230 | -43 | 21 | -11 | 5 | -2 |
| 13/16 | -1 | 3 | -6 | 12 | -22 | 54 | 241 | -36 | 17 | -9 | 5 | -2 |
| 14/16 | 0 | 2 | -4 | 8 | -15 | 35 | 249 | -26 | 12 | -7 | 3 | -1 |
| 15/16 | 0 | 1 | -2 | 4 | -7 | 16 | 254 | -14 | 6 | -3 | 2 | -1 |



Figure 30. Frequency responses of the interpolation filter and the VVC interpolation filter at half-pel phase

For chroma interpolation additional longer 6-tap filters are used. The coefficients of filters are tabulated in Table 4.

Table 4. The coefficients of the 6-tap interpolation filter for chroma components.

|  |  |
| --- | --- |
| **Fractional position** | **Coefficients (6 taps)** |
| 1/32 | {0, 0, 256, 0, 0, 0}, |
| 2/32 | {1, -6, 256, 7, -2, 0}, |
| 3/32 | {2, -11, 253, 15, -4, 1}, |
| 4/32 | {3, -16, 251, 23, -6, 1}, |
| 5/32 | {4, -21, 248, 33, -10, 2}, |
| 6/32 | {5, -25, 244, 42, -12, 2}, |
| 7/32 | {7, -30, 239, 53, -17, 4}, |
| 8/32 | {7, -32, 234, 62, -19, 4}, |
| 6/32 | {8, -35, 227, 73, -22, 5}, |
| 7/32 | {9, -38, 220, 84, -26, 7}, |
| 8/32 | {10, -40, 213, 95, -29, 7}, |
| 9/32 | {10, -41, 204, 106, -31, 8}, |
| 10/32 | {10, -42, 196, 117, -34, 9}, |
| 11/32 | {10, -41, 187, 127, -35, 8}, |
| 12/32 | {11, -42, 177, 138, -38, 10}, |
| 13/32 | {10, -41, 168, 148, -39, 10}, |
| 14/32 | {10, -40, 158, 158, -40, 10}, |
| 15/32 | {10, -39, 148, 168, -41, 10}, |
| 16/32 | {10, -38, 138, 177, -42, 11}, |
| 17/32 | {8, -35, 127, 187, -41, 10}, |
| 18/32 | {9, -34, 117, 196, -42, 10}, |
| 19/32 | {8, -31, 106, 204, -41, 10}, |
| 20/32 | {7, -29, 95, 213, -40, 10}, |
| 21/32 | {7, -26, 84, 220, -38, 9}, |
| 22/32 | {5, -22, 73, 227, -35, 8}, |
| 23/32 | {4, -19, 62, 234, -32, 7}, |
| 24/32 | {4, -17, 53, 239, -30, 7}, |
| 25/32 | {2, -12, 42, 244, -25, 5}, |
| 26/32 | {2, -10, 33, 248, -21, 4}, |
| 27/32 | {1, -6, 23, 251, -16, 3}, |
| 28/32 | {1, -4, 15, 253, -11, 2}, |
| 31/32 | {0, -2, 7, 256, -6, 1}, |

### Multi-hypothesis prediction (MHP)

In the multi-hypothesis inter prediction mode (JVET-M0425), one or more additional motion-compensated prediction signals are signaled, in addition to the conventional bi-prediction signal. The resulting overall prediction signal is obtained by sample-wise weighted superposition. With the bi-prediction signal and the first additional inter prediction signal/hypothesis , the resulting prediction signal is obtained as follows:

The weighting factor is specified by the new syntax element **add\_hyp\_weight\_idx**, according to the following mapping:

|  |  |
| --- | --- |
| **add\_hyp\_weight\_idx** |  |
| 0 | 1/4 |
| 1 | -1/8 |

Analogously to above, more than one additional prediction signal can be used. The resulting overall prediction signal is accumulated iteratively with each additional prediction signal.

The resulting overall prediction signal is obtained as the last (i.e., the having the largest index ). Within this EE, up to two additional prediction signals can be used (i.e., is limited to 2).

The motion parameters of each additional prediction hypothesis can be signaled either explicitly by specifying the reference index, the motion vector predictor index, and the motion vector difference, or implicitly by specifying a merge index. A separate multi-hypothesis merge flag distinguishes between these two signalling modes.

For inter AMVP mode, MHP is only applied if non-equal weight in BCW is selected in bi-prediction mode.

Combination of MHP and BDOF is possible, however the BDOF is only applied to the bi-prediction signal part of the prediction signal (i.e., the ordinary first two hypotheses).

### Pixel based affine motion compensation

The minimum affine subblock size is changed from 4x4 to 1x1 for both luma and chroma components, 1x1 subblock size allows pixel based affine MC. When affine subblock width or height is smaller than 4, PROF is disabled.

### Affine subblock BDOF refinement

BDOF subblock MV refinement and sample adjustment is applied to an affine or SbTMVP coded block with subblock MC when BDOF condition is satisfied.

An affine coded block, e.g. affine regular merge mode, affine BM merge mode, affine AMVP mode, derives MVs for each 4×4 subblock from the affine model. The BDOF process starts with the 4×4 subblocks grouping with identical MVs. The first iteration of BDOF MV refinement is processed in 8x8 subblock grid as in ECM-10.0. When the grouped subblock size is less than 256, the second iteration of BDOF MV refinement is processed in 4×4 subblock grid, and otherwise in 8×8 subblock grid. When the grouped subblock size is 4xN or Nx4, the first iteration of BDOF MV refinement is bypassed.

### Adaptive reordering of merge candidates with template matching (ARMC-TM)

The merge candidates are adaptively reordered with template matching (TM). The reordering method is applied to regular merge mode, TM merge mode, and affine merge mode (excluding the SbTMVP candidate). For the TM merge mode, merge candidates are reordered before the refinement process.

An initial merge candidate list is firstly constructed according to given checking order, such as spatial, TMVPs, non-adjacent, HMVPs, pairwise, virtual merge candidates. Then the candidates in the initial list are divided into several subgroups. For the template matching (TM) merge mode, adaptive DMVR mode, each merge candidate in the initial list is firstly refined by using TM/multi-pass DMVR. Merge candidates in each subgroup are reordered to generate a reordered merge candidate list and the reordering is according to cost values based on template matching. The index of selected merge candidate in the reordered merge candidate list is signalled to the decoder. For simplification, merge candidates in the last but not the first subgroup are not reordered. All the zero candidates from the ARMC reordering process are excluded during the construction of Merge motion vector candidates list. The subgroup size is set to 5 for regular merge mode and TM merge mode. The subgroup size is set to 3 for affine merge mode.

* Cost calculation

The template matching cost of a merge candidate during the reordering process is measured by the SAD between samples of a template of the current block and their corresponding reference samples. The template comprises a set of reconstructed samples neighboring to the current block. Reference samples of the template are located by the motion information of the merge candidate. When a merge candidate utilizes bi-directional prediction, the reference samples of the template of the merge candidate are also generated by bi-prediction as shown in Figure 31.

* Refinement of the initial merge candidate list

When multi-pass DMVR is used to derive the refined motion to the initial merge candidate list only the first pass (i.e., PU level) of multi-pass DMVR is applied in reordering. When template matching is used to derive the refined motion, the template size is set equal to 1. Only the above or left template is used during the motion refinement of TM when the block is flat with block width greater than 2 times of height or narrow with height greater than 2 times of width. TM is extended to perform 1/16-pel MVD precision. The first four merge candidates are reordered with the refined motion in TM merge mode.

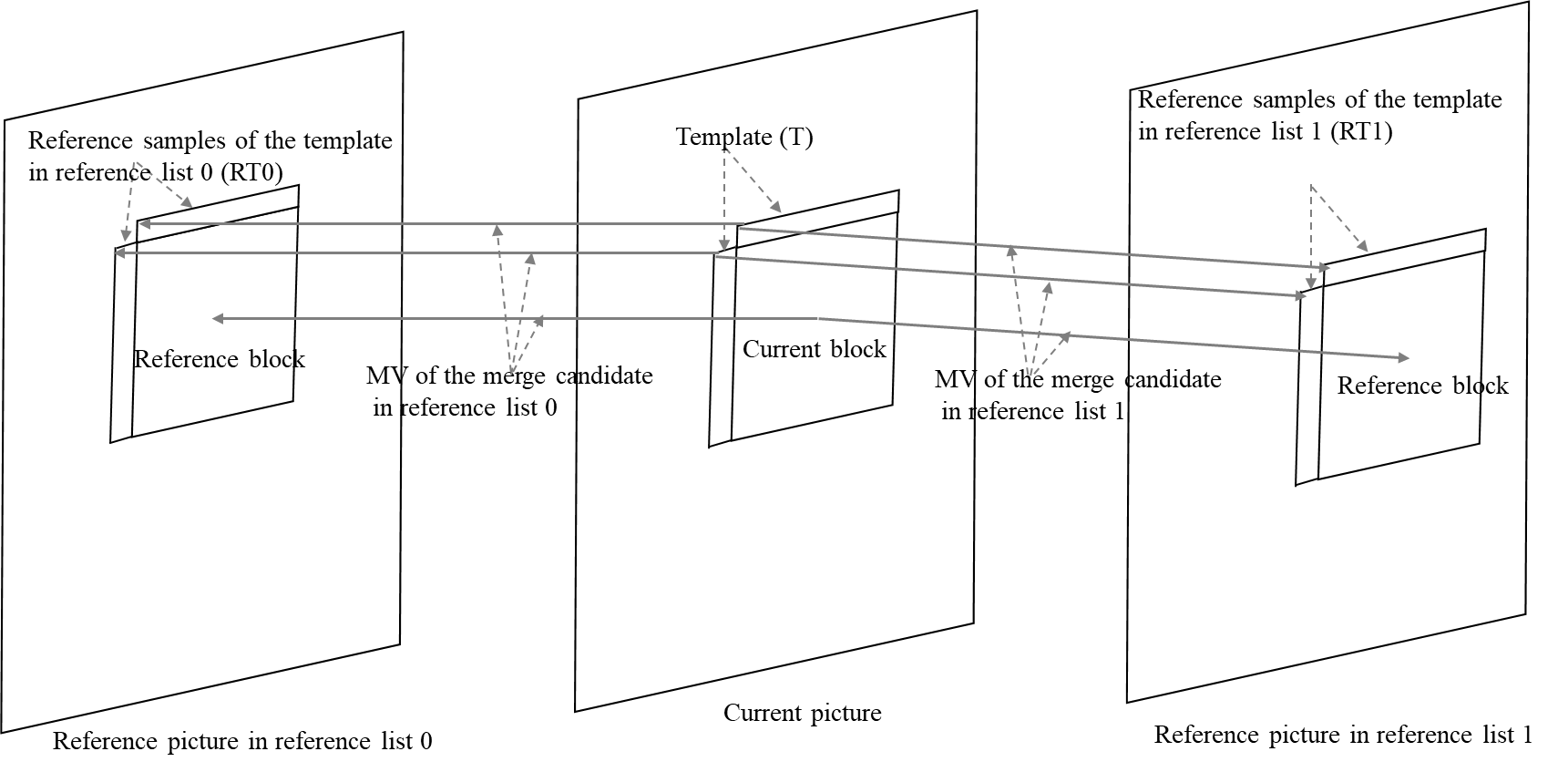
**

Figure 31. Template and reference samples of the template in reference pictures

For subblock-based merge candidates with subblock size equal to Wsub × Hsub, the above template comprises several sub-templates with the size of Wsub × 1, and the left template comprises several sub-templates with the size of 1 × Hsub. As shown in Figure 32, the motion information of the subblocks in the first row and the first column of current block is used to derive the reference samples of each sub-template.

* Reordering criteria

In the reordering process, a candidate is considered as redundant if the cost difference between a candidate and its predecessor is inferior to a lambda value e.g. |D1-D2| < λ, where D1 and D2 are the costs obtained during the first ARMC ordering and λ is the Lagrangian parameter used in the RD criterion at encoder side.

The proposed algorithm is defined as the following:

* Determine the minimum cost difference between a candidate and its predecessor among all candidates in the list
  + If the minimum cost difference is superior or equal to λ, the list is considered diverse enough and the reordering stops.
  + If this minimum cost difference is inferior to λ, the candidate is considered as redundant, and it is moved at a further position in the list. This further position is the first position where the candidate is diverse enough compared to its predecessor.
* The algorithm stops after a finite number of iterations (if the minimum cost difference is not inferior to λ).

This algorithm is applied to the Regular, TM, BM and Affine merge modes. A similar algorithm is applied to the Merge MMVD and sign MVD prediction methods which also use ARMC for the reordering.

The value of λ is set equal to the λ of the rate distortion criterion used to select the best merge candidate at the encoder side for low delay configuration and to the value λ corresponding to a another QP for Random Access configuration. A set of λ values corresponding to each signaled QP offset is provided in the SPS or in the Slice Header for the QP offsets which are not present in the SPS.

* Extension to AMVP modes

The ARMC design is also applicable to the AMVP mode wherein the AMVP candidates are reordered according to the TM cost. For the template matching for advanced motion vector prediction (TM-AMVP) mode, an initial AMVP candidate list is constructed, followed by a refinement from TM to construct a refined AMVP candidate list. In addition, an MVP candidate with a TM cost larger than a threshold, which is equal to five times of the cost of the first MVP candidate, is skipped.

Note, when wrap around motion compensation is enabled, the MV candidate shall be clipped with wrap around offset taken into consideration.

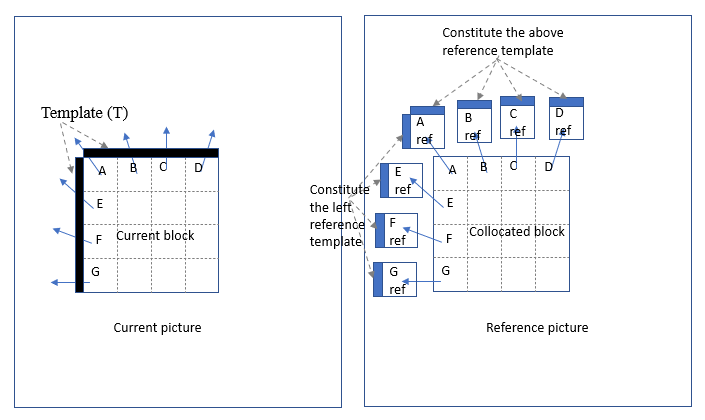


Figure 32. Template and reference samples of the template for block with sub-block motion using the motion information of the subblocks of the current block

### MV candidate type based ARMC

Merge candidates of one single candidate type, e.g., TMVP or non-adjacent MVP (NA-MVP), are reordered based on the ARMC TM cost values. The reordered candidates are then added into the merge candidate list. The TMVP candidate type adds more TMVP candidates with more temporal positions and different inter prediction directions to perform the reordering and the selection. Moreover, NA-MVP candidate type is further extended with more spatially non-adjacent positions. The target reference picture of the TMVP candidate can be selected from any one of reference picture in the list according to scaling factor. The selected reference picture is the one whose scaling factor is the closest to 1.

### TM based reordering for MMVD and affine MMVD

The MMVD offsets are extended for MMVD and affine MMVD modes. Additional refinement positions along k×π/8 diagonal angles are added shown in Figure 33, thus increasing the number of directions from 4 to 16. Second, based on the SAD cost between the template (one row above and one column left to the current block) and its reference for each refinement position, all the possible MMVD refinement positions (16×6) for each base candidate are reordered. Finally, the top 1/8 refinement positions with the smallest template SAD costs are kept as available positions, consequently for MMVD index coding. The MMVD index is binarized by the rice code with the parameter equal to 2. The affine MMVD reordering is extended, in which additional refinement positions along k×π/4 diagonal angles are added. After reordering top 1/2 refinement positions with the smallest template SAD costs are kept.

The first N motion candidates in the candidate list before being reordered are utilized as the base candidates for MMVD and affine MMVD. N is equal to 3 for MMVD, and [1, 3] depending on the neighboring block affine flags for affine MMVD. Two ways of adding MMVD offsets are allowed, including the ‘two-side’ and ‘one-side’, depending on whether the offset of the other reference picture list is mirrored or directly set to zero. Which way is applied to one block is dependent on the TM cost.



Figure 33. Additional directions along k×π/8 diagonal angles (red positions are used in the anchor).

### Regression based affine candidate derivation

The Regression based Motion Vector Field (RMVF) derivation method provides a new variety of subblock-based merge candidate. The motion vectors and center positions from the neighboring subblocks of the current CU, as illustrated in Figure 34, are used as the input to the linear regression process to derive a set of linear model parameters.

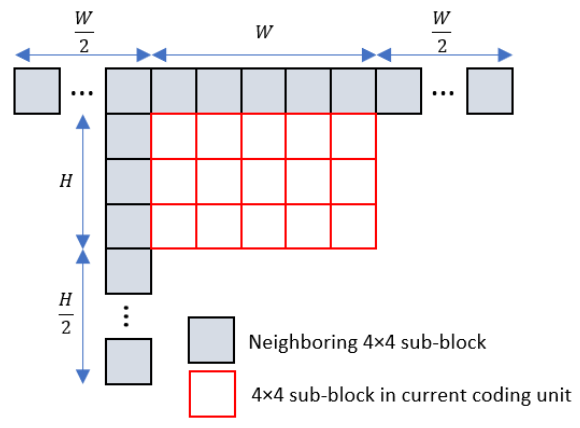


Figure 34: Illustration of the neighboring 4 x 4 subblocks that are used for RMVF parameter derivation. W and H are the width and height of the current CU.

The subblock motion field from a previous coded affine CU and the motion vectors from the adjacent subblocks of current CU are used as the input for the regression process. The predicted CPMVs for current block are derived as output.

The regression based affine merge candidates are derived and added to the affine merge list. Subblock motion field from a previously coded affine CU and motion information from adjacent subblocks of a current CU are used as the input to the regression process to derive proposed affine candidates.

The previously coded affine CU can be identified from scanning through non-adjacent positions and the affine HMVP table.

Adjacent subblock information of current CU is fetched from 4x4 sub-blocks represented by the grey zone as depicted in Figure 34. For each sub-block, given a reference list, the corresponding motion vector and center coordinate of the sub-block may be used.

For each affine CU, up to 2 affine candidates can be derived. One with adjacent subblock information and one without. All the linear-regression-generated candidates are pruned and collected into one candidate sub-group, TM cost based ARMC process is applied when ARMC is enabled. Afterwards, up to N linear-regression-generated candidates are added to the affine merge list when N affine CUs are found. The number of affine candidates for ARMC is 30, the output list size is 15.

### Geometric partitioning mode (GPM) with merge motion vector differences (MMVD)

GPM in VVC is extended by applying motion vector refinement on top of the existing GPM uni-directional MVs. A flag is first signalled for a GPM CU, to specify whether this mode is used. If the mode is used, each geometric partition of a GPM CU can further decide whether to signal MVD or not. If MVD is signalled for a geometric partition, after a GPM merge candidate is selected, the motion of the partition is further refined by the signalled MVDs information. All other procedures are kept the same as in GPM.

The MVD is signaled as a pair of distance and direction, similar as in MMVD. There are nine candidate distances (¼-pel, ½-pel, 1-pel, 2-pel, 3-pel, 4-pel, 6-pel, 8-pel, 16-pel), and eight candidate directions (four horizontal/vertical directions and four diagonal directions) involved in GPM with MMVD (GPM-MMVD). In addition, when pic\_fpel\_mmvd\_enabled\_flag is equal to 1, the MVD is left shifted by 2 as in MMVD.

### Geometric partitioning mode (GPM) with adaptive blending

In VVC, the final prediction samples are generated with by blending the prediction of the two prediction signals using weighted average. Two integer blending matrices (*W0* and *W1*) are used. The weights in the GPM blending matrices are derived from the ramp function based on the displacement from a predicted sample position to the GPM partitioning boundary. The blending area size is fixed to two (2 samples on each side of the GPM partition split boundary).

The blending process in ECM is improved by adding four extra blending area sizes (quarter, half, double, and quadrupole of the existing area size) as shown in Figure 35. A CU level flag is coded to signal the selected blending area size is signalled. Furthermore, the extended weighting precision is utilized, in which the maximum value of the weighs is changed from 8 (in VVC) to 32 to accommodate the extended blending area sizes.

A picture containing text, laser

Description automatically generated

Figure 35. The ramp function for the weights for GPM blending based on the displacement (d) from a predicted sample position to the GPM partitioning boundary and the blending area size (τ).

### Geometric partitioning mode (GPM) with template matching (TM)

Template matching is applied to GPM. When GPM mode is enabled for a CU, a CU-level flag is signaled to indicate whether TM is applied to both geometric partitions. Motion information for each geometric partition is refined using TM. When TM is chosen, a template is constructed using left, above or left and above neighboring samples according to partition angle, as shown in Table 5. The motion is then refined by minimizing the difference between the current template and the template in the reference picture using the same search pattern of merge mode with half-pel interpolation filter disabled.

Table 5. Template for the 1st and 2nd geometric partitions, where A represents using above samples, L represents using left samples, and L+A represents using both left and above samples.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Partition angle | 0 | 2 | 3 | 4 | 5 | 8 | 11 | 12 | 13 | 14 |
| 1st partition | A | A | A | A | L+A | L+A | L+A | L+A | A | A |
| 2nd partition | L+A | L+A | L+A | L | L | L | L | L+A | L+A | L+A |
| Partition angle | 16 | 18 | 19 | 20 | 21 | 24 | 27 | 28 | 29 | 30 |
| 1st partition | A | A | A | A | L+A | L+A | L+A | L+A | A | A |
| 2nd partition | L+A | L+A | L+A | L | L | L | L | L+A | L+A | L+A |

A GPM candidate list is constructed as follows:

1. Interleaved List-0 MV candidates and List-1 MV candidates are derived directly from the regular merge candidate list, where List-0 MV candidates are higher priority than List-1 MV candidates. A pruning method with an adaptive threshold based on the current CU size is applied to remove redundant MV candidates.
2. Interleaved List-1 MV candidates and List-0 MV candidates are further derived directly from the regular merge candidate list, where List-1 MV candidates are higher priority than List-0 MV candidates. The same pruning method with the adaptive threshold is also applied to remove redundant MV candidates.
3. Zero MV candidates are padded until the GPM candidate list is full.

The GPM-MMVD and GPM-TM are exclusively enabled to one GPM CU. This is done by firstly signaling the GPM-MMVD syntax. When both two GPM-MMVD control flags are equal to false (i.e., the GPM-MMVD are disabled for two GPM partitions), the GPM-TM flag is signaled to indicate whether the template matching is applied to the two GPM partitions. Otherwise (at least one GPM-MMVD flag is equal to true), the value of the GPM-TM flag is inferred to be false.

### GPM with inter and intra prediction

In GPM with inter and intra prediction, the final prediction samples are generated by weighting inter predicted samples and intra predicted samples for each GPM-separated region. The inter predicted samples are derived by inter GPM whereas the intra predicted samples are derived by an intra prediction mode (IPM) candidate list and an index signaled from the encoder. The IPM candidate list size is pre-defined as 3. The available IPM candidates are the parallel angular mode against the GPM block boundary (Parallel mode), the perpendicular angular mode against the GPM block boundary (Perpendicular mode), and the Planar mode as shown Figure 36 (a) ~ (c), respectively. Furthermore, GPM with intra and intra prediction as shown Figure 36d is restricted to reduce the signalling overhead for IPMs and avoid an increase in the size of the intra prediction circuit on the hardware decoder. In addition, a direct motion vector and IPM storage on the GPM-blending area is introduced to further improve the coding performance.

グラフ, ダイアグラム

中程度の精度で自動的に生成された説明

Figure 36. GPM with inter and intra prediction. Available IPM candidates (a) ~ (c). (d) Example of GPM with intra and intra prediction.

In DIMD and neighboring mode based IPM derivation Parallel mode is registered first. Therefore, max two IPM candidates derived from the decoder-side intra mode derivation (DIMD) method and/or the neighboring blocks can be registered if there is not the same IPM candidate in the list. As for the neighboring mode derivation, there are five positions for available neighboring blocks at most, but they are restricted by the angle of GPM block boundary as shown in Table 6, which are already used for GPM with template matching (GPM-TM).

Table 6. The position of available neighboring blocks for IPM candidate derivation based on the angle of GPM block boundary. A and L denotes the above and left side of the prediction block.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Angle of GPM | 0 | 2 | 3 | 4 | 5 | 8 | 11 | 12 | 13 | 14 |
| 1st partition | A | A | A | A | L+A | L+A | L+A | L+A | A | A |
| 2nd partition | L+A | L+A | L+A | L | L | L | L | L+A | L+A | L+A |
| Partition angle | 16 | 18 | 19 | 20 | 21 | 24 | 27 | 28 | 29 | 30 |
| 1st partition | A | A | A | A | L+A | L+A | L+A | L+A | A | A |
| 2nd partition | L+A | L+A | L+A | L | L | L | L | L+A | L+A | L+A |

GPM-intra can be combined with GPM with merge with motion vector difference (GPM-MMVD). TIMD is used for on IPM candidates of GPM-intra to further improve the coding performance. The Parallel mode can be registered first, then IPM candidates of TIMD, DIMD, and neighboring blocks.

### Template matching based reordering for GPM split modes

In template matching based reordering for GPM split modes, given the motion information of the current GPM block, the respective TM cost values of GPM split modes are computed. Then, all GPM split modes are reordered in ascending ordering based on the TM cost values. Instead of sending GPM split mode, an index using Golomb-Rice code to indicate where the exact GPM split mode located in the reordering list is signaled.

The reordering method for GPM split modes is a two-step process performed after the respective reference templates of the two GPM partitions in a coding unit are generated, as follows:

* extending GPM partition edge into the reference templates of the two GPM partitions, resulting in 64 reference templates and computing the respective TM cost for each of the 64 reference templates;
* reordering GPM split modes based on their TM cost values in ascending order and marking the best 32 split modes as available split modes.

The edge on the template is extended from that of the current CU, as Figure 37 illustrates, but GPM blending process is not used in the template area across the edge.

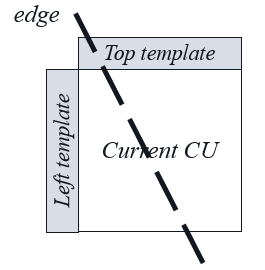


Figure 37. The edge on templates.

After ascending reordering using TM cost, an index is signaled.

### Bi-predictive GPM

The GPM design in VVC relies on uni-predictive motion vectors to generate motion compensated prediction samples for each inter GPM partition. In ECM, such a design has been extended to allow usage of bi-predictive motion vectors.

When constructing a GPM candidate list, the extraction process that extracts uni-predictive motion vectors from the initial merge list is invoked only for small blocks 8x8, 16x8 and 8x16. For larger blocks, the extraction process is bypassed, so the initial merge list (which may contain merged Bi-MVs) is directly used as the final GPM merge list. The generation of the initial merge list is the same as before (i.e., the normal merge list generation without any candidate reordering) except that when generating the initial merge list for larger blocks (i.e., blocks with the extraction process bypassed), the motion vector difference threshold for controlling whether a candidate can be added into the list is increased to be one full sample distance.

BDOF based motion vector refinement as in the multi-pass DMVR is used when generating motion compensated prediction samples.

When GPM-MMVD is used for a GPM partition and its base motion vector is bi-predictive, for low-delay pictures, the signalled MVD is applied on top of the L0 and L1 motion vector as in the existing merge MMVD design. For non-low-delay pictures, the bi-predictive motion vector is converted into a uni-predictive motion vector first and then the MVD is applied on top.

### Bilateral matching AMVP-merge mode

The bi-directional predictor is composed of an AMVP predictor in one direction and a merge predictor in the other direction. The mode can be enabled to a coding block when the selected merge predictor and the AMVP predictor satisfy DMVR condition, where there is at least one reference picture from the past and one reference picture from the future relatively to the current picture and the distances from two reference pictures to the current picture are the same, the bilateral matching MV refinement is applied for the merge MV candidate and AMVP MVP as a starting point. Otherwise, if template matching functionality is enabled, template matching MV refinement is applied to the merge predictor or the AMVP predictor which has a higher template matching cost.

AMVP part of the mode is signaled as a regular uni-directional AMVP, i.e. reference index and MVD are signaled, and it has a derived MVP index if template matching is used or MVP index is signaled when template matching is disabled.

For AMVP direction LX, X can be 0 or 1, the merge part in the other direction (1 – LX) is implicitly derived by minimizing the bilateral matching cost between the AMVP predictor and a merge predictor, i.e., for a pair of the AMVP and a merge motion vectors. For every merge candidate in the merge candidate list which has that other direction (1 – LX) motion vector, the bilateral matching cost is calculated using the merge candidate MV and the AMVP MV. The merge candidate with the smallest cost is selected. The bilateral matching refinement is applied to the coding block with the selected merge candidate MV and the AMVP MV as a starting point.

The third pass of multi pass DMVR which is sub-PU BDOF refinement of the multi-pass DMVR is enabled to AMVP-merge mode coded block. Sub-PU size of BDOF is adaptively selected depending on the width×height. For blocks smaller than 256, subblock size of 4×4, and otherwise 8×8 is used. In addition, the following high-precision equations to derive the BDOF MV refinement parameters are utilized:

∑Gx.Gx \* vx + ∑Gx.Gy \* vy = ∑dI . Gx 🡺 s1 \* vx + s2 \* vy = s3

∑Gx.Gy \* vx + ∑Gy.Gy \* vy = ∑dI . Gy 🡺 s2 \* vx + s5 \* vy = s6

where Gx/Gy are the summation of the 2 horizontal/vertical gradients derived for each reference block.

Summations (Σ) are weighted sums, where weights depend on the position in the target region Ω. The weights can also be applied to derive vx/vy in other cases.

The mode is indicated by a flag, if the mode is enabled AMVP direction LX is further indicated by a flag.

When bilateral matching (BM) AMVP-merge mode is used for the current block and template matching is enabled, MVD is not signalled. An additional pair of AMVP-merge MVPs is introduced. The merge candidate list is sorted based on the BM cost in increase order. An index (0 or 1) is signaled to indicate which merge candidate in the sorted merge candidate list to use. When there is only one candidate in merge candidate list, the pair of AMVP MVP and merge MVP without bilateral matching MV refinement is padded.

### IBC merge/AMVP list construction

The IBC merge/AMVP list construction compared to VVC is modified as follows:

* Only if an IBC merge/AMVP candidate is valid, it can be inserted into the IBC merge/AMVP candidate list.
* Above-right, bottom-left, and above-left spatial candidates (belonging to the adjacent spatial candidate category) and one pairwise average candidate can be added into the IBC merge/AMVP candidate list.
* Template based adaptive reordering (ARMC-TM) is applied to IBC merge list.
* Candidates from non-adjacent spatial neighboring blocks (a.k.a., non-adjacent candidates) can be added to the candidate lists of IBC merge modes and IBC AMVP. These non-adjacent candidates are inserted between the adjacent spatial candidates and the HBVP candidates for both IBC merge and IBC AMVP. The same reference area of non-adjacent merge in regular inter mode is reused for the IBC.
* Restriction that adjacent spatial candidates cannot be used for IBC merge of a 4x4 CU is removed.

The HMVP table size for IBC is increased to 25. After up to 20 IBC merge candidates are derived with full pruning, they are reordered together. After reordering, the first 6 candidates with the lowest template matching costs are selected as the final candidates in the IBC merge list.

The zero vectors’ candidates to pad the IBC Merge/AMVP list are replaced with a set of BVP candidates located in the IBC reference region. A zero vector is invalid as a block vector in IBC merge mode, and consequently, it is discarded as BVP in the IBC candidate list.

Three candidates are located on the nearest corners of the reference region, and three additional candidates are determined in the middle of the three sub-regions (A, B, and C), whose coordinates are determined by the width, and height of the current block and the ΔX and ΔY parameters, as is depicted in Figure 38.



Figure 38. Padding candidates for the replacement of the zero-vector in the IBC list.

During the IBC AMVP list construction, a clustering of the BVP candidates may be applied when both BV candidate components are non-zero. The clustering as shown in Figure 39 with L2 distance is applied if there are more than 2 valid BV candidates and up to 6 candidates are clustered, the clustering radius is defined as

The clustering method is applied in the candidate list order, and the candidates assigned to a group are removed from the list for the subsequent clusters. In each group, the BVP with a lowest TM cost is selected as the representative candidate of that group. Finally, the representative candidates of the two first groups are chosen as the candidates for the IBC AMVP list.



Figure 39. IBC candidate clustering based on the L2 distance and the TM cost.

Furthermore, if one of BV candidate components is zero or block is coded in RRIBC, a flag is signalled to indicate this case with a directional flag indicating horizontal or vertical component is non-zero. Instead of usual IBC AMVP list, two new BVP candidates are derived, and the sign of the non-zero BV component is derived at decoder side. The AMVP BVP0 is set to the nearest valid location to the current block (-cbWidth or -cbHeight), so the non-zero BVD is always negative, pointing to the left for a BV with a zero vertical component or to the above for a BV with a zero horizontal component. Likewise, the AMVP BVP1 is set to the farthest position from the current block in the valid reference region, that is the left boundary or the top boundary of the IBC search region. Consequently, if the BVP1 is selected, the BVD is always positive, pointing to the right for BV with a zero vertical component or to the bottom for BV with a zero horizontal component.

The optimal IBC AMVP index is signalled, which allows deriving the sign of the non-zero BVD component at the decoder side. The absolute magnitude of non-zero BVD component is further signalled. In RRIBC, the direction of the flipping mode is derived from the signalled directional flag.

### IBC with Template Matching

Template Matching is used in IBC for both IBC merge mode and IBC AMVP mode.

The IBC-TM merge list is modified compared to the one used by regular IBC merge mode such that the candidates are selected according to a pruning method with a motion distance between the candidates as in the regular TM merge mode. The ending zero motion fulfillment is replaced by motion vectors to the left (-W, 0), top (0, -H) and top-left (-W, -H), where W is the width and H the height of the current CU.

In the IBC-TM merge mode, the selected candidates are refined with the Template Matching method prior to the RDO or decoding process. The IBC-TM merge mode has been put in competition with the regular IBC merge mode and a TM-merge flag is signaled.

In the IBC-TM AMVP mode, up to 3 candidates are selected from the IBC-TM merge list. Each of those 3 selected candidates are refined using the Template Matching method and sorted according to their resulting Template Matching cost. Only the 2 first ones are then considered in the motion estimation process as usual.

The Template Matching refinement for both IBC-TM merge and AMVP modes is quite simple since IBC motion vectors are constrained (i) to be integer and (ii) within a reference region as shown in Figure 40. So, in IBC-TM merge mode, all refinements are performed at integer precision, and in IBC-TM AMVP mode, they are performed either at integer or 4-pel precision depending on the AMVR value. Such a refinement accesses only to samples without interpolation. In both cases, the refined motion vectors and the used template in each refinement step must respect the constraint of the reference region.

X

Curr

X

X

Curr

X

X

X

X

Curr

X

X

Curr

X

Figure 40: IBC reference region depending on current CU position.

### IBC reference area

The reference area for IBC is extended to two CTU rows above. Figure 41 illustrates the reference area for coding CTU (m, n). Specifically, for CTU (m, n) to be coded, the reference area includes CTUs with index (m–2, n–2)…(W, n–2),(0, n–1)…(W, n–1),(0, n)…(m, n), where W denotes the maximum horizontal index within the current tile, slice or picture. When CTU size is 256, the reference area is limited to one CTU row above. This setting ensures that for CTU size being 128 or 256, IBC does not require extra memory in the current ETM platform. The per-sample block vector search (or called local search) range is limited to [–(C << 1), C >> 2] horizontally and [–C, C >> 2] vertically to adapt to the reference area extension, where C denotes the CTU size.



Figure 41. Reference area for IBC when CTU (m, n) is coded. The blue block denotes the current CTU; green blocks denote the reference area; and the white blocks denote invalid reference area.

### Fractional pel IBC

The option of block vector resolutions is extended to include quarter-pel resolution in additional to full-pel and 4-pel. Like inter AMVR syntax, the first bin is signalled to indicate whether BV is in quarter-pel resolution, and the second bin is signalled to switch between full-pel and 4-pel resolutions. The interpolation filters applied to the luma (8-tap) and chroma (6-tap existed inter interpolation) components of an IBC block. For template-based IBC tools, a 2-tap bilinear interpolation filter is applied to generate template prediction blocks. Reference sample padding is performed when some of them are located outside IBC reference area. When needed, it performs in horizontal direction first and then vertical direction.

### Filtered IBC prediction

Additional filtered IBC mode is introduced, where a filter is applied to IBC predictor, which is derived by minimizing MSE between current and reference template.

Output of the filter is calculated as follows:

predLumaVal = c0C + c1N + c2S + c3E + c4W + c5P + c6B

The nonlinear term P is represented as power of two of the center sample C and scaled to the sample value range of the content:

P = ( C\*C + midVal ) >> bitDepth

The bias term B represents a scalar offset between the input and output and is set to middle luma value (512 for 10-bit content).

This filtered mode is used as an additional mode for non-merge IBC blocks, and it is not used together with IBC-LIC, IBC-CIIP or RR-IBC. For IBC merge modes, this filtering mode is inherited when merge mode list is constructed. The mode flag is signalled before the IBC-LIC flag.

### MVD prediction

In this method, possible MVD sign combinations and possible combinations of the first 6 most signification suffix bins of MVD magnitudes are sorted according to the template matching cost and index corresponding to the true MVD sign and MVD magnitudes is derived and context coded. At decoder side, the MVD are derived as following:

1. Parse the magnitude of MVD components
2. Parse context coded MVD prediction index
3. Build MV candidates by creating combination between possible signs and possible MVD magnitudes and add it to the MV predictor
4. Derive MVD prediction cost for each derived MV based on template matching cost and sort
5. Use the signaled index to pick the true MVD

MVD prediction is applied to inter AMVP, affine AMVP, MMVD and affine MMVD modes. Note, when wrap around motion compensation is enabled, the MV candidate shall be clipped with wrap around offset taken into consideration.

### BVD prediction

Similar to MVD prediction, possible BVD sign combinations of IBC mode are sorted according to the template matching cost. Moreover, the first 4 most signification suffix bins of exponential Golomb code used to represent BVD magnitudes is also sorted according to the TM cost. An example is shown in Figure 42. Template matching operation is used to determine a BVD candidate with the best cost, and indicate in the bitstream whether the best candidate is predicted correctly or not.



Figure 42. Prediction of BVD.

### Enhanced bi-directional motion compensation

In bi-directional motion compensation the out of boundary (OOB) prediction samples are discarded and only the non-OOB predictors, when available, are used to generate the final predictor. Specifically, let and denote the position of one prediction sample in one current block, and (x = 0,1) denote the MV of the current block; *, ,* andare the positions of four boundaries of the picture. One prediction sample is regarded as OOB when at least one of the following conditions is satisfied:

()>(+*half\_pixel*),

()<(- *half\_pixel*),

()>(+ *half\_pixel*),

()<(- *half\_pixel*)

where *half\_pixel* is equal to 8 that represents the half-pel sample distance in the 1/16-pel sample precision.

After examining the OOB condition for each sample, the final prediction samples of one bi-directional block are generated as follows:

If is OOB and is non-OOB

else if is non-OOB and is OOB

else

OOB checking process is also applicable when BCW is enabled.

Finally, note this sample-adaptive bi-prediction process only applies to prediction units for which at least a reference bock is first detected as partially or entirely out-of-bounds. Thus, a block-level OOB criteria is first checked. If both prediction blocks are non-OOB, then the usual bi-prediction takes place.

### Motion compensated picture boundary padding

The samples outside of the picture boundary are derived by motion compensation instead of using only repetitive padding. In the implementation, the total padded area size is increased by 16 compared to repetitive padding. This is to keep MV clipping, which implements repetitive padding Figure 43.



Figure 43. Motion compensated boundary padding method.

For motion compensation padding, MV of a 4×4 boundary block is utilized to derive a M×4 or 4×M padding block. The value M is derived as the distance of the reference block to the picture boundary as shown on Figure 44. Moreover, M is set at least equal to 4 as soon as the motion vector points to a position internal to the reference picture bounds. If boundary block is intra coded, then MV is not available, and M is set equal to 0. If M is less than 16, the rest of the padded area is filled with the repetitive padded samples.



Figure 44. An example of deriving a M×4 padding block with a left padding direction.

In case of bi-directional inter prediction, only one prediction direction, which has a motion vector pointing to the pixel position farther away from the picture boundary in the reference picture in terms of the padding direction, is used in MC boundary padding.

The pixels in MC padding block are corrected with an offset, which is equal to the difference between the DC values of the reconstructed boundary block and its corresponding reference block.

### Block level reference picture list reordering

A block level reference picture reordering method based on template matching is used. For the uni-prediction AMVP mode, the reference pictures in List 0 and List 1 are interweaved to generate a joint list. For each hypothesis of the reference picture in the joint list template matching is performed to calculate the cost. The joint list is reordered based on ascending order of the template matching cost. The index of the selected reference picture in the reordered joint list is signaled in the bitstream. For the bi-prediction AMVP mode, a list of pairs of reference pictures from List 0 and List 1 is generated and similarly reordered based on the template matching cost. The index of the selected pair is signaled.

### Reference picture resampling (RPR)

Reference picture resampling is inherited from VVC. Compared to the filter lengths in VVC, e.g., 8, 6 and 4 taps for luma affine coded blocks, luma non-affine coded blocks and chroma respectively, the corresponding RPR filters in ECM are increased to 12, 10 and 6 taps.

The LIC and template-based inter reordering tools, including ARMC, MMVD and affine MMVD reordering, template-based BCW derivation, block level reference picture list reordering and MVD prediction, are enabled when any of reference pictures is in different resolution to the current picture.

### Reconstruction-Reordered IBC (RR-IBC)

A Reconstruction-Reordered IBC (RR-IBC) mode is allowed for IBC coded blocks. When RR-IBC is applied, the samples in a reconstruction block are flipped according to a flip type of the current block. At the encoder side, the original block is flipped before motion search and residual calculation, while the prediction block is derived without flipping. At the decoder side, the reconstruction block is flipped back to restore the original block.

Two flip methods, horizontal flip and vertical flip, are supported for RR-IBC coded blocks. A syntax flag is firstly signalled for an IBC AMVP coded block, indicating whether the reconstruction is flipped, and if it is flipped, another flag is further signaled specifying the flip type. For IBC merge, the flip type is inherited from neighbouring blocks, without syntax signalling. Considering the horizontal or vertical symmetry, the current block and the reference block are normally aligned horizontally or vertically. Therefore, when a horizontal flip is applied, the vertical component of the BV is not signaled and inferred to be equal to 0. Similarly, the horizontal component of the BV is not signaled and inferred to be equal to 0 when a vertical flip is applied.

To better utilize the symmetry property, a flip-aware BV adjustment approach is applied to refine the block vector candidate. For example, as shown in Figure 45, (*xnbr*, *ynbr*) and (*xcur*, *ycur*) represent the coordinates of the center sample of the neighbouring block and the current block, respectively, *BVnbr* and *BVcur* denotes the BV of the neighbouring block and the current block, respectively. Instead of directly inheriting the BV from a neighbouring block, the horizontal component of *BVcur* is calculated by adding a motion shift to the horizontal component of *BVnbr* (denoted as *BVnbrh*) in case that the neighbouring block is coded with a horizontal flip, i.e., *BVcurh* =2(*xnbr* -*xcur*) + *BVnbrh* . Similarly, the vertical component of *BVcur* is calculated by adding a motion shift to the vertical component of *BVnbr* (denoted as *BVnbrv*) in case that the neighbouring block is coded with a vertical flip, i.e., *BVcurv* =2(*ynbr* -*ycur*) + *BVnbrv* .



(a)



(b)

Figure 45. Illustration of BV adjustment for (a) horizontal flip, and (b) vertical flip, respectively.

### Combination of IBC with other coding tools

#### IBC merge mode with block vector differences (IBC-MBVD)

Affine-MMVD and GPM-MMVD have been adopted to ECM as an extension of regular MMVD mode. It is natural to extend the MMVD mode to the IBC merge mode.

In IBC-MBVD, the distance set is {1-pel, 2-pel, 4-pel, 8-pel, 12-pel, 16-pel, 24-pel, 32-pel, 40-pel, 48-pel, 56-pel, 64-pel, 72-pel, 80-pel, 88-pel, 96-pel, 104-pel, 112-pel, 120-pel, 128-pel}, and the BVD directions are two horizontal and two vertical directions.

The base candidates are selected from the first five candidates in the reordered IBC merge list. And based on the SAD cost between the template (one row above and one column left to the current block) and its reference for each refinement position, all the possible MBVD refinement positions (20×4) for each base candidate are reordered. Finally, the top 8 refinement positions with the lowest template SAD costs are kept as available positions, consequently for MBVD index coding. The MBVD index is binarized by the rice code with the parameter equal to 1.

In IBC-MBVD list derivation, adaptive BVD offsets along MVBD directions are enabled for IBC MBVD mode. The MBVD candidates search is a two-step process, which starts with checking template SAD costs of offsets added to BVP along each direction with the interval of 1-pel. The second step of the search checks template SAD costs with 1/4-pel interval for the candidates around the selected candidates from the first step. For the integer MBVD (when existed in ECM ph\_fpel\_mbvd\_enabled\_flag is 0), those intervals are multiplied by 4. The candidates with the lowest TM cost are included into the final MBVD list.

An IBC-MBVD coded block does not inherit flip type from a RR-IBC coded neighbor block.

#### Combined intra block copy and intra prediction

Combined intra block copy and intra prediction (IBC-CIIP) is a coding tool for a CU which uses IBC and intra prediction to obtain two prediction signals, and the two prediction signals are weighted summed to generate the final prediction as follows:

wherein and denote the IBC prediction signal and intra prediction signal. are set equal to (13, 4) and (1, 1) for IBC merge mode and IBC AMVP mode.

An intra prediction mode (IPM) candidate list is used to generate the intra prediction signal, and the IPM candidate list size is pre-defined as 2. An IPM index is signalled to indicate which IPM is used.

#### IBC with Geometry Partitioning

Intra block copy with geometry partitioning mode (IBC-GPM) is a coding tool which divides a CU into two sub-partitions geometrically. The prediction signals of the two sub-partitions are generated using IBC and intra prediction. IBC-GPM can be applied to regular IBC merge mode or IBC TM merge mode. An intra prediction mode (IPM) candidate list is constructed using the same method as GPM with inter and intra prediction for intra prediction, and the IPM candidate list size is pre-defined as 3. There are 48 geometry partitioning modes in total, which are divided into two geometry partitioning mode sets as follows:

Table 7: Geometry partitioning modes in the first geometry partitioning mode set

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ibc\_gpm\_partition\_idx** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **angleIdx** | 0 | 0 | 8 | 8 | 16 | 16 | 24 | 24 |
| **distanceIdx** | 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 |

Table 8: Geometry partitioning modes in the second geometry partitioning mode set

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ibc\_gpm\_partition\_idx** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **angleIdx** | 2 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 4 | 5 |
| **distanceIdx** | 0 | 1 | 3 | 0 | 1 | 3 | 0 | 1 | 3 | 0 |
| **ibc\_gpm\_partition\_idx** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** |
| **angleIdx** | 5 | 5 | 11 | 11 | 11 | 12 | 12 | 12 | 13 | 13 |
| **distanceIdx** | 1 | 3 | 0 | 1 | 3 | 0 | 1 | 3 | 0 | 1 |
| **ibc\_gpm\_partition\_idx** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** |
| **angleIdx** | 13 | 14 | 14 | 14 | 18 | 18 | 19 | 19 | 20 | 20 |
| **distanceIdx** | 3 | 0 | 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 |
| **ibc\_gpm\_partition\_idx** | **30** | **31** | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** |
| **angleIdx** | 21 | 21 | 27 | 27 | 28 | 28 | 29 | 29 | 30 | 30 |
| **distanceIdx** | 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 |

When IBC-GPM is used, an IBC-GPM geometry partitioning mode set flag is signalled to indicate whether the first or the second geometry partitioning mode set is selected, followed by the geometry partitioning mode index. An IBC-GPM intra flag is signalled to indicate whether intra prediction is used for the first sub-partition. When intra prediction is used for a sub-partition, an intra prediction mode index is signalled. When IBC is used for a sub-partition, a merge index is signalled.

In bi-predictive IBC GPM, two flags are signalled to indicate the prediction modes of two partitions, the first flag indicates whether the first partition is intra predicted, and if not then the second flag is signalled to indicate whether intra prediction is used for the second partition. This method is applied to SCC only.

#### IBC BVP-merge and bi-predictive IBC merge

IBC-BVP-merge is similar to AMVP-merge, derives one BV from IBC block vector prediction (BVP) and the second BV from IBC merge to form bi-prediction for IBC. Two different indices for the IBC BVP and the IBC merge candidates are signalled.

Bi-predictive IBC merge is enabled together with MBVD and uni-merge. In bi-predictive IBC merge, two BVs from the existing IBC merge candidate list are derived, utilizing two different indices, which are signalled. Bi-predictive IBC merge is applied to IBC regular merge and IBC MBVD. Bi-predictive IBC merge, IBC MBVD, and IBC uni-merge are enabled for non-SCC classes.

#### IBC MBVD list derivation

In the test 2.4a, adaptive BVD offsets along MVBD directions and enabled for IBC MBVD mode. The MBVD candidates search is a two-step process, which starts with checking template SAD costs of offsets added to BVP along each direction with the interval of 1-pel. The second step of the search checks template SAD costs with 1/4-pel interval for the candidates around the selected candidates from the first step. For the integer MBVD (when existed in ECM ph\_fpel\_mbvd\_enabled\_flag is 0), those intervals are multiplied by 4. The candidates with the lowest TM cost are included into the final MBVD list

#### IBC with Local Illumination Compensation

Intra block copy with local illumination compensation (IBC-LIC) is a coding tool which compensates the local illumination variation within a picture between the CU coded with IBC and its prediction block with a linear equation. The parameters of the linear equation are derived same as LIC for inter prediction except that the reference template is generated using block vector in IBC-LIC. IBC-LIC can be applied to IBC AMVP mode and IBC merge mode. For IBC AMVP mode, an IBC-LIC flag is signalled to indicate the use of IBC-LIC. Top-only, left-only, or L-shape templates are allowed for deriving the single model parameters. MMLM is extended to IBC-LIC, which allows IBC-LIC to have two linear models in one CU. And only L-shape template is used in IBC-LIC MMLM. A mode index is signalled. For IBC merge mode, the IBC-LIC flag is inferred from the merge candidate. The IBC-LIC flag is inherited from an IBC HMVP candidate to harmonize IBC HMVP and IBC-LIC similar to the inter LIC case.

### Template matching based BCW index derivation for merge mode

The BCW index for merge coded CUs is derived based on template matching cost instead of being derived from neighboring blocks. Given a selected merge candidate, the TM cost values are calculated with different bi-prediction weights, and then, the bi-prediction weight with minimum TM cost value is used to predict the merge CU.

When calculating TM cost for bi-predicted weights, the following rules are applied:

* Since the inherited bi-predicted weight is likely to have higher accuracy than others, only the inherited bi-prediction weight and its two neighboring weights (i.e. ±1) are considered. For example, if the inherited bi-predicted weight is 4, then only three weights {3, 4, 5} are involved in TM cost calculation.
* The TM cost of the inherited BCW index is multiplied with 0.90625, that is, the cost is reduced by 3/32.
* The TM cost of the equal weight is multiplied with 0.90625 since bi-predicted samples are beneficial for BDOF and BDOF is only applied to CU with equal weights.

The template matching based BCW index derivation is applied to CUs coded in regular merge, template matching, adaptive decoder-side motion vector refinement and MMVD modes.

In addition, the bi-prediction weights for merge mode are extended from {-2, 3, 4, 5, 10} to {1, 2, 3, 4, 5, 6, 7}. Furthermore, the negative bi-predicted weights for non-merge mode {-2, 10} are replaced with positive weights {1, 7}.

### DMVR for affine merge coded blocks

DMVR is applied to affine merge coded blocks and affine MMVD coded blocks when DMVR condition is satisfied. It is also extended to adaptive BM merge mode.

An affine motion field is modelized as follows (6-parameters affine case):

wherein is the motion vector at location and is the base MV representing the translation motion of the affine model. Parameters , , and represent the non-translation parameters (rotation, scaling).

Motion vectors and are called the control point motion vectors (CPMVs) of the considered affine coding unit.In the DMVR process applied to affine, the bilateral matching cost is calculated per subblock. Then, the subblock bilateral matching costs and refined subblock MVs are used to determine the overall best refined CPMVs for the affine block. More specific, the CPMVs are refined according to the following steps:

1. Perform integer-pel bilateral matching for subblocks. Accumulate the subblock bilateral matching cost to determine the best integer-pel MV offset.
2. Perform half-pel bilateral matching search using the best integer MV offset as initial offset and output the best MV offset that minimizes the bilateral matching cost for the same set of the subblocks of step 1.
3. Perform linear regression using the refined subblock MVs from step 1 as input and output a set of control-point motion vectors.
4. Compare the bilateral matching cost of the output of the steps 2 and 3 to select the one with the smallest cost.

In addition, the non-translation parameters of affine model are refined after the base MV are determined. Each of CPMVs is fixed as base MV in turn, and an offset is added to the non-translation parameter of affine model by minimizing the bilateral matching cost, and then the other two CPMVs are calculated according to based MV and refined non-translation parameters.

For affine merge and affine MMVD modes, both CPMVs and non-translation parameters refinements are applied. When applying to affine MMVD mode, the MMVD offset is added to the affine DMVR refined affine merge base candidate if the base candidate meets the affine DMVR refinement condition. For adaptive BM merge mode, an affine merge list that only contains affine merge candidates that meet the affine DMVR conditions are constructed and then CPMVs refinement and non-translation parameters refinment are applied.

### InterCCCM

InterCCCM applies the CCCM method for predicting chroma samples from reconstructed luma samples when the CU uses inter prediction or intra block copy (IBC). Figure 46 illustrates the decoder side of the method. The cross-component filters are derived using the prediction blocks of luma and chroma. The derived filters are applied to the reconstructed luma block and blended with the prediction blocks of chroma to produce the final chroma prediction blocks. In the blending process the filtered reconstructed luma blocks use blending weight of 0.75 and chroma prediction blocks use blending weight of 0.25.

A diagram of a flowchart

Description automatically generated

Figure 46. The InterCCCM method on the decoder.

The 8-tap filter consist of 6 spatial luma samples, a nonlinear term, and a bias term. The spatial luma samples (L0,…,L5) are obtained from the luma grid selecting the 6 luma samples closest to the chroma position C without down sampling as shown in Figure 47. The predicted chroma value is obtained as,

*predChromaVal = c0 L0+ c1L1 + c2L2 + c3L3 + c4L4 + c5L5 + c6 nonlinear((L0+L3+1) >> 1) + c7 B,*

where *nonlinear* is CCCM’s nonlinear operator and B is bias. The filter coefficients are derived using ECM’s division-free Gaussian elimination method and the necessary offsets are applied to samples prior to filter derivation. The offsets for division-free Gaussian elimination method are obtained using a four-point average of the luma and chroma prediction blocks, where the four points correspond to the top-left, top-right, bottom-left and bottom-right corners of the blocks. For filter coefficient derivation at most 256 chroma samples are used.

A white grid with black letters and numbers

Description automatically generated

Figure 47. Luma samples L0,..,L5 in relation to the chroma sample C.

Usage of the mode is signalled with a CABAC coded TU level flag. One new CABAC context was included to support this. The InterCCCM flag is only signalled if the TU’s luma Cbf is non-zero and the CU’s predMode is either MODE\_INTER or MODE\_IBC.

The encoder performs an RD decision in the transform selection loop for the chroma components when luma Cbf is non-zero and the CU’s predMode is either MODE\_INTER or MODE\_IBC.

### CCP merge for chroma inter blocks

The cross-component prediction merge mode described in section 3.1.15 is extended to chroma inter coding. The CCP models including CCLM, MMLM, CCCM, GLM, chroma fusion, CCP merge modes, and inter CCCM are stored and inherited for the following coding chroma intra and inter blocks. Similar to the CCP merge for chroma intra blocks, a flag is signaled to indicate whether a chroma inter block is coded using this mode. If the CCP merge mode is used, a CCP merge list is constructed in a similar way as that for chroma intra blocks except that additional shifted temporal candidate and on-the-fly derived candidates are included in the CCP merge list. The additional shifted temporal candidates are derived from the collocated picture. And, the position of these candidates are the same as those defined in ECM for regular inter merge prediction candidates with a shift obtained from the motion vector of the current block. The on-the-fly derived candidates are only used for low delay pictures and are obtained using the neighboring reconstructed samples of the current block. At most 1 on-the-fly derived candidates including single/multi-model CCCM and single/multi-model CCLM are added to the CCP merge list. After the CCP merge list is constructed, the candidate with the lowest template cost is selected for the chroma inter block. The chroma inter block is then predicted in the same way as that of inter CCCM. That is, the motion compensation predicted samples are blended with the cross-component predicted samples to form the final prediction.

## Transform and coefficient coding

### Shifting the quantization centers

The dequantized transform coefficients (or quantization centers) are adaptively shifted in an amount proportional to the gradient of the rate. It is assumed that rate increases by logarithm of the absolute value of the quantization centers. Thus, a simple proxy of rate prediction is assumed where the quantization indices are independent, and rate increases by logarithm of the absolute value of the quantization index:

Here, is the quantization index of i-th transform coefficient and are constants. Using this assumption, the gradient of the rate can be written as:

In implementation, a look up table is used, where each element is the shifting amount for each possible unique absolute quantization indices. For 10-bit representation, the shifting of dequantized coefficients is performed as:

(3)

Here,is length of the lookup table, is the quantization index of i-th transform coefficient, is the dequantized value of the i-th transform coefficient, is the shifted dequantized coefficient, is the auxiliary quantization index that can be calculated as:

(4)

The value of is fixed to 63. The corresponding lookup table is :

If the absolute quantization index is 0 or above 63, this method is not applied and the default dequantization is used.

### Dependent quantization with 8-states

The coding efficiency of trellis-coded quantization in VVC increased by increasing the number of quantization states (at the cost of a higher encoder complexity). Dependent quantization with 8 quantization states in addition to the current variant of dependent quantization with 4 quantization state is supported (JVET-Q0243).

For supporting both variants of dependent quantization (4 and 8 states) in a unified framework, the decoding process for the VVC variant of dependent quantization is re-written.

The state transition table (sec. 7.4.12.11 in VVC) is modified from

QStateTransTable[ ][ ] = { { 0, 2 }, { 2, 0 }, { 1, 3 }, { 3, 1 } }

to

QStateTransTable[ ][ ] = { { 0, 1 }, { 2, 3 }, { 1, 0 }, { 3, 2 } }

There are three aspects that depend on the quantization state QState: (a) the mapping of transmitted transform coefficient levels to intermediate quantization indexes (part of the dequantization specified in the syntax); (b) the context selection for the sig\_coeff\_flag; (c) the derivation of the mapping parameter ZeroPos[ ] for transform coefficient levels coded in bypass mode. All three aspects are re-written in order to reflect the swapping of quantization states:

1. The mapping of transmitted transform coefficient levels to intermediate quantization indexes (see syntax structure residual\_coding() in VVC) is modified from

TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =  
 ( 2 \* AbsLevel[ xC ][ yC ] − ( QState > 1 ? 1 : 0 ) ) \* ( 1 − 2 \* coeff\_sign\_flag[ n ] )

to

TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =  
 ( 2 \* AbsLevel[ xC ][ yC ] − ( QState & 1 ) ) \* ( 1 − 2 \* coeff\_sign\_flag[ n ] )

1. The context selection of the sig\_coeff\_flag (see sec. 9.3.4.2.8 in VVC) depends on a parameter (context set id) that is derived based on the quantization state. In VVC, this parameter is given by

Max( 0, QState – 1 )

With the relabelling of the quantization states, this parameter can be derived according to

ctxSetId[ QState & 3 ] with ctxSetId[ ] = { 0, 1, 0, 2 }

It should be noted that for the 4-state version, the result of (QState & 3) is equal to QState. The masking is only required for the 8-state version of dependent quantization.

1. The derivation of the mapping parameter ZeroPos[ ] for transform coefficient levels coded in bypass mode is modified from

ZeroPos[ n ] = ( QState < 2 ? 1 : 2 )  <<  cRiceParam

to

ZeroPos[ n ] = ( 1 + ( QState & 1 ) )  <<  cRiceParam

### Maximum Transform Size and Zeroing-out of Transform Coefficients

Both CTU size and maximum transform size (i.e., all MTS transform kernels) are extended to 256, where the maximum intra coded block can have a size of 128x128. The maximum CTU size is set to 256 for UHD sequences and it is set to 128, otherwise. In the primary transformation process, there is no normative zeroing out operation applied on transform coefficients. However, if LFNST is applied, the primary transform coefficients outside the LFNST region are normatively zeroed-out.

### Enhanced MTS for intra coding

In the current VVC design [1], for MTS, only DST7 and DCT8 transform kernels are utilized which are used for intra and inter coding.

Additional primary transforms including DCT5, DST4, DST1, and identity transform (IDT) are employed. Also MTS set is made dependent on the TU size and intra mode information. For blocks predicted via IntraTMP, DIMD process is used on the predition block to derive an intra mode that is used for transform selection. Specifically, a horizontal gradient and a vertical gradient are calculated for each predicted sample to build a HoG. Then the intra prediction mode with the largest histogram amplitude values is used to the MTS transform set.

Overall, 16 different TU sizes are considered, and for each TU size 5 different classes are considered depending on intra-mode information. For each class, 1, 4 or 6 different transform pairs are considered. Number of intra MTS candidates are adaptively selected (between 1, 4 and 6 MTS candidates) depending on the sum of absolute value of transform coefficients. The sum is compared against the two fixed thresholds to determine the total number of allowed MTS candidates:

1 candidate: sum <= th0

4 candidates: th0 < sum <= th1

6 candidates: sum > th1

Note, although a total of 80 different classes are considered, some of those different classes often share exactly same transform set. So there are 58 (less than 80) unique entries in the resultant LUT.

For angular modes, a joint symmetry over TU shape and intra prediction is considered. So, a mode i (i > 34) with TU shape A×B will be mapped to the same class corresponding to the mode j = (68 – i) with TU shape B×A. However, for each transform pair the order of the horizontal and vertical transform kernel is swapped. For example, for a 16x4 block with mode 18 (horizontal prediction) and a 4x16 block with mode 50 (vertical prediction) are mapped to the same class. However, the vertical and horizontal transform kernels are swapped. For the wide-angle modes the nearest conventional angular mode is used for the transform set determination. For example, mode 2 is used for all the modes between -2 and -14. Similarly, mode 66 is used for mode 67 to mode 80.

### Inter MTS optimization

For the MTS of inter-coded CUs, four candidates: {(DST7, DST7), (DST7, DCT8), (DCT8, DST7), (DCT8, DCT8)} are used for every CU. For the larger resolution sequences (width > 1080) maximum CU size for Inter-MTS usage is set to 32 (i.e., Inter-MTS is used for CU with width <=32 and height <=32), and for the remaining sequences (smaller resolution) it is set to 16. For 4-pt, 8-pt and 16-pt transforms, the current AMT transform cores, i.e., DST-7 and DCT-8, is replaced with separable KLTs, as proposed in JVET-J0021.

### Secondary Transformation: LFNST extension with large kernel

The LFNST design in VVC is extended as follows:

* The number of LFNST sets (*S*) and candidates (*C*) are extended to *S*=35 and *C*=3, and the LFNST set (lfnstTrSetIdx) for a given intra mode (predModeIntra) is derived according to the following formula:
  + For predModeIntra < 2, lfnstTrSetIdx is equal to 2
  + lfnstTrSetIdx = predModeIntra, for predModeIntra in [0,34]
  + lfnstTrSetIdx = 68 – predModeIntra, for predModeIntra in [35,66]
* Three different kernels, LFNST4, LFNST8, and LFNST16, are defined to indicate LFNST kernel sets, which are applied to 4xN/Nx4 (N4), 8xN/Nx8 (N8), and MxN (M, N16), respectively.

The kernel dimensions are specified by:

(LFSNT4, LFNST8\*, LFNST16\*) = (16x16, 32x64, 32x96)

The forward LFNST is applied to top-left low frequency region, which is called Region-Of-Interest (ROI). When LFNST is applied, primary-transformed coefficients that exist in the region other than ROI are zeroed out, which is not changed from the VVC standard.

The ROI for LFNST16 is depicted in Figure 48. It consists of six 4x4 sub-blocks, which are consecutive in scan order. Since the number of input samples is 96, transform matrix for forward LFNST16 can be Rx96. R is chosen to be 32 in this contribution, 32 coefficients (two 4x4 sub-blocks) are generated from forward LFNST16 accordingly, which are placed following coefficient scan order.



Figure 48. The ROI for LFNST16

The ROI for LFNST8 is shown in Figure 49. The forward LFNST8 matrix can be Rx64 and R is chosen to be 32. The generated coefficients are located in the same manner as with LFNST16.

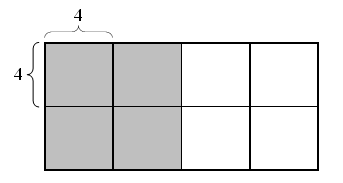


Figure 49. The ROI for LFNST8

The mapping from intra prediction modes to these sets is shown in Table 1,

Table 9. Mapping of intra prediction modes to LFNST set index



For blocks using MIP or IntraTMP prediction, the LFNST set index is derived as follows. DIMD is used to derive the intra prediction mode of the current block based on the MIP or IntraTMP predicted samples. For MIP, this is done before upsampling. Specifically, a horizontal gradient and a vertical gradient are calculated for each predicted sample to build a HoG, as shown in Figure 50. Then the intra prediction mode with the largest histogram amplitude values is used to determine the LFNST transform set and LFNST Transpose flag.



Figure 50. MIP prediction samples to build HOG.

### Non-separable primary transform (NSPT) for intra coding

The separable DCT-II plus LFNST transform combinations are replaced with NSPT for the block shapes 4x4, 4x8, 8x4 and 8x8, 4x16, 16x4, 8x16 and 16x8.

The affected block sizes are summarized in Figure 51.

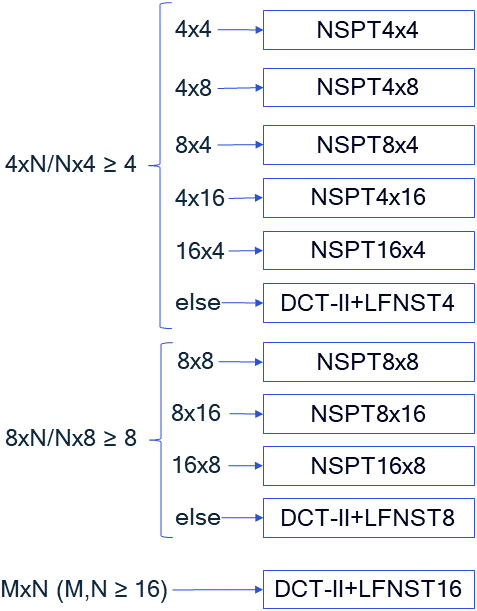


Figure 51: Overview of proposed NSPTs among existing LFNSTs.

All NSPTs consist of 35 sets and 3 candidates (similar to the current LFNST). The kernels of NSPTs have the following shapes:

* NSPT4x4: 16x16
* NSPT4x8/NSPT8x4: 32x20
* NSPT8x8: 64x32
* NSPT4x16/NSPT16x4: 64x24
* NSPT8x16/NSPT16x8: 128x40
* NSPT4x32/NSPT32x4: 128x20
* NSPT8x32/NSPT32x8: 256x24

Therefore, 12, 32, 40 and 88 coefficients are zeroed-out using NSPT4x8/NSPT8x4, NSPT8x8, NSPT4x16/NSPT16x4 and NSPT8x16/NSPT16x8 respectively. For NSPT4x32/NSPT32x4 and NSPT8x32/NSPT32x8, remaining 108 and 232 positions in each transform block are zeroed-out, respectively.

### NSPT/LFNST Context modeling for transform coefficient coding

For coding of LFNST/NSPT coefficients, modified context model using the previous 5 coefficients in the coding order are used for context derivation instead of the 2D coefficient neighborhood.

A number on a black background

Description automatically generated

Figure 52. Context modeling for LFNST/NSPT coefficients.

The lfnstIdx is signalled after all last\_sig\_coeff\_pos syntax elements in a CU since lfsntIdx is required for parsing the transform coefficients.

### Sign prediction

The basic idea of the coefficient sign prediction method (JVET-D0031 and JVET-J0021) is to calculate reconstructed residual for both negative and positive sign combinations for applicable transform coefficients and select the hypothesis that minimizes a cost function.

To derive the best sign, the cost function is defined as discontinuity measure across block boundary shown on Figure 53. It is measured for all hypotheses, and the one with the smallest cost is selected as a predictor for coefficient signs.

A picture containing text

Description automatically generated

Figure 53. Discontinuity measure.

The cost function is defined as a sum of absolute second derivatives in the residual domain for the above row and left column as follows:

where *R* is reconstructed neighbors, *P* is prediction of the current block, and *r* is the residual hypothesis. The term can be calculated only once per block and only residual hypothesis is subtracted.

The transform coefficients with the largest K qIdx value of the top-left 4x4 area are selected. qIdx value is the transform coefficient level after compensating the impact from the multiple quantizers in DQ. A larger qIdx value will produce a larger de-quantized transform coefficient level. qIdx is derived as follows

qIdx = (abs(level) << 1) − (state & 1);

where level is the transform coefficient level parsed from the bitstream and state is a variable maintained by the encoder and decoder in DQ.

The sign prediction area was extended to maximum 32x32. Signs of top-left MxN block are predicted. The value of M and N is computed as follows:

where, *w* and *h* are the width and height of the transform block. The maximum area for sign prediction is not always set to 32x32. Encoder sets the maximum area (maxW, maxH) based on configuration, sequence class and QP, and signaled the area in SPS

The maximum number of predicted signs is kept unchanged. The sign prediction is also applied to LFNST blocks. And for LFNST block, a maximum of 4 coefficients in the top-left 4x4 area are allowed to be sign predicted.

## Adaptive loop filter

### ALF simplification removal

ALF gradient subsampling and ALF virtual boundary processing are removed. Block size for classification is reduced from 4x4 to 2x2. Filter size for both luma and chroma, for which ALF coefficients are signalled, is increased to 9x9.

### ALF with fixed filters

To filter a luma sample, three different classifiers (C0, C1 and C2) and three different sets of filters (F0, F1 and F2) are used. Sets F0 and F1 contain fixed filters, with coefficients trained for classifiers C0 and C1. Coefficients of filters in F2 are signalled. Which filter from a set Fi is used for a given sample is decided by a class assigned to this sample using classifier Ci

### Filtering

At first, two 13x13 diamond shape fixed filters F0 and F1 are applied to derive two intermediate samples and . After that, F2 is applied to , , and neighboring samples to derive a filtered sample as

where is the clipped difference between a neighboring sample and current sample and is the clipped difference between and current sample. The filter coefficients are signalled.

### Classification

Based on directionality and activity , a class is assigned to each 2x2 block:

where represents the total number of directionalities .

As in VVC, values of the horizontal, vertical, and two diagonal gradients are calculated for each sample using 1-D Laplacian. The sum of the sample gradients within a 4×4 window that covers the target 2×2 block is used for classifier C0 and the sum of sample gradients within a 12×12 window is used for classifiers C1 and C2. The sums of horizontal, vertical and two diagonal gradients are denoted, respectively, as , , and . The directionality is determined by comparing

with a set of thresholds. The directionality is derived as in VVC using thresholds 2 and 4.5. For and , horizontal/vertical edge strength and diagonal edge strength are calculated first. Thresholds are used. Edge strength is 0 if ; otherwise, is the maximum integer such that Edge strength is 0 if ; otherwise, is the maximum integer such that . When , i.e., horizontal/vertical edges are dominant, the is derived by using Table 10 (a); otherwise, diagonal edges are dominant, the is derived by using Table 10 (b).

Table 10. Mapping of and to

(a) (b)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 28 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 2 | 0 | 0 | 0 | 0 | 0 |  | 1 | 29 | 30 | 0 | 0 | 0 | 0 | 0 |
| 2 | 3 | 4 | 5 | 0 | 0 | 0 | 0 |  | 2 | 31 | 32 | 33 | 0 | 0 | 0 | 0 |
| 3 | 6 | 7 | 8 | 9 | 0 | 0 | 0 |  | 3 | 34 | 35 | 36 | 37 | 0 | 0 | 0 |
| 4 | 10 | 11 | 12 | 13 | 14 | 0 | 0 |  | 4 | 38 | 39 | 40 | 41 | 42 | 0 | 0 |
| 5 | 15 | 16 | 17 | 18 | 19 | 20 | 0 |  | 5 | 43 | 44 | 45 | 46 | 47 | 48 | 0 |
| 6 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |  | 6 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |

To obtain , the sum of vertical and horizontal gradients is mapped to the range of 0 to , where is equal to 4 for and 15 for and .

In an ALF\_APS, up to 4 luma filter sets are signalled, each set may have up to 25 filters.

### Alternative 2x2 ALF classifier

Classification in ALF is extended with an additional alternative classifier. For a signalled luma filter set, a flag is signalled to indicate whether the alternative classifier is applied. Geometrical transformation is not applied to the alternative band classifier. When the band-based classifier is applied, the sum of sample values of a 2x2 luma block is calculated at first. Then the class index is calculated as below,

class\_index = (sum \* 25) >> (sample bit depth + 2).

### Residual based classifier

A third classifier based on luma residual sample values. For each 2x2 luma block, the sum of absolute values of the residual samples in a neighbouring 8x8 window is calculated, and the class index is derived as:

classIdx = sum >> (sample bit depth – 4).

The value of classIdx is in the range of 0 to 24, same as in ECM-8.0. The classifier usage is signalled for each luma filter set in APS.

### CCALF with long tap filter

Different from VVC wherein only luma samples are involved in CCALF, in ECM, the CCALF process uses a linear filter to filter luma sample values, luma residual samples and generate a residual correction for the chroma samples. In addition, the CCALF filter shape is constructed by 23 luma spatial taps and 5 luma residual taps, which is illustrated in Figure 54. For a given slice, the encoder can collect the statistics of the slice, analyze them and can signal up to 16 filters through APS.

**图示

描述已自动生成**

Figure 54. 25-tap long filter.

### Adaptive filter shape switch and using samples before deblocking filter for adaptive loop filter

Two candidate filter shapes: a diamond shape as shown in Figure 55 and a new cross shape as shown in Figure 56, can be adaptively selected by the luma filters in ALF. The number of coefficients of a luma filter is 22 for both the filter shapes. Please note that these 22 taps are constituted with 20 spatial taps and 2 fixed filters based taps in both shapes.

In each adaptation parameter set (APS), a shape index for the derived luma filters is signaled to decoder. Each APS contains the luma filters that are associated with the filter shape index.

For each CTB, an APS index is signaled to indicate which luma filter shape is used to filter the current CTB. When filtering a luma sample, the coefficients and clip indices are also rearranged according to the corresponding filter shape.

The diamond shape luma ALF is replaced by the longer filter shown in Figure 56.

图表

描述已自动生成

Figure 55: The diamond shape of ALF in ECM-5.0.

图示

低可信度描述已自动生成

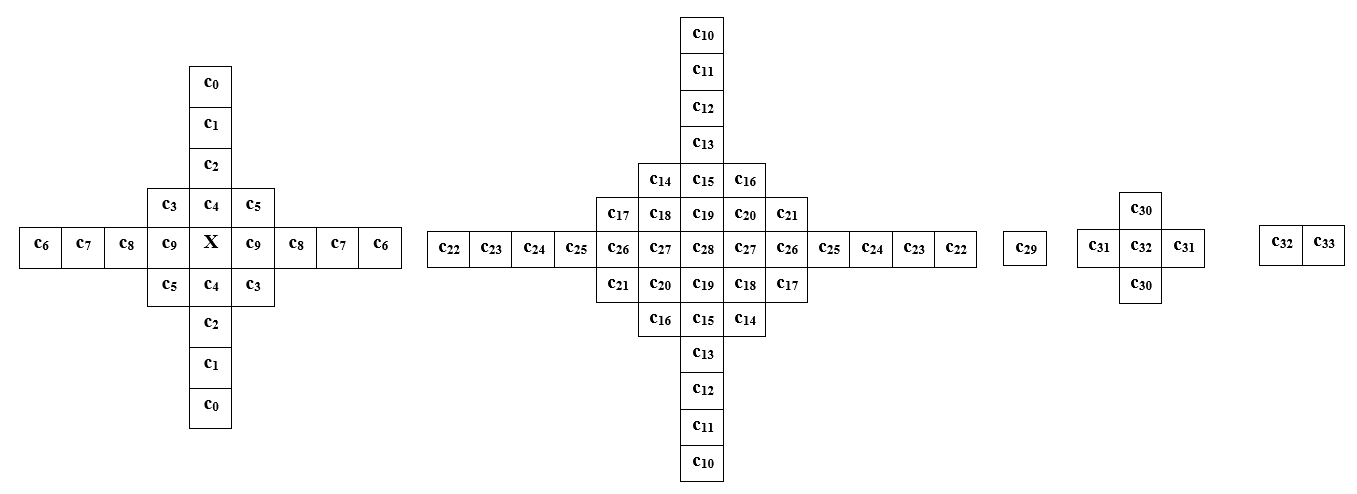
Figure 56. The new filter shape for ALF.

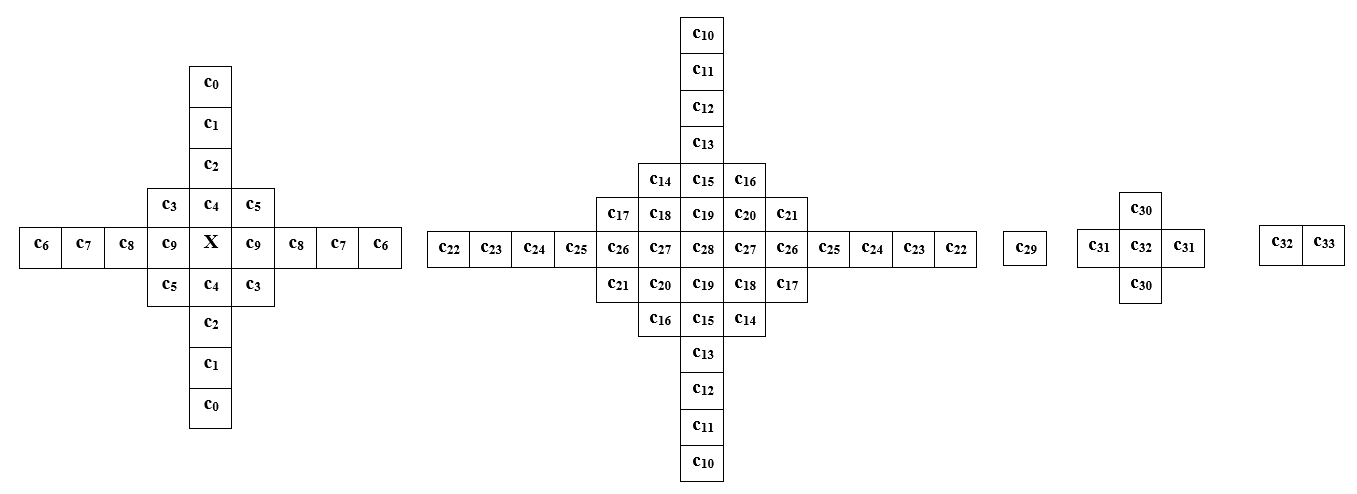
The samples before deblocking filters are used as additional inputs for ALF. A final ALF sample is derived by weighting the regular ALF and the filter applied to the samples before the deblocking filter. Specifically, a filtered sample is derived as

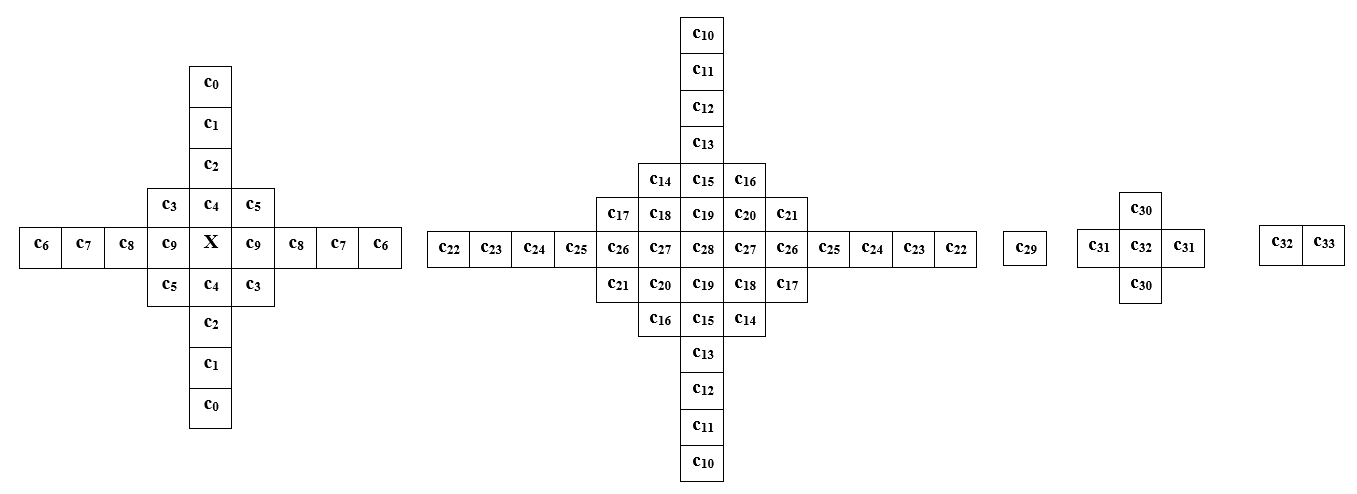
where is the clipped difference between a neighboring sample and current sample , is the clipped difference between an intermediate sample and current sample and is the clipped difference between a neighboring sample before DBF and current sample . The filter coefficients are signalled. In this test, 3x3 diamond shape is applied to samples before deblocking filter. In an APS, a flag is signalled to indicate whether samples before DBF are used for ALF which is always set as true at encoder.

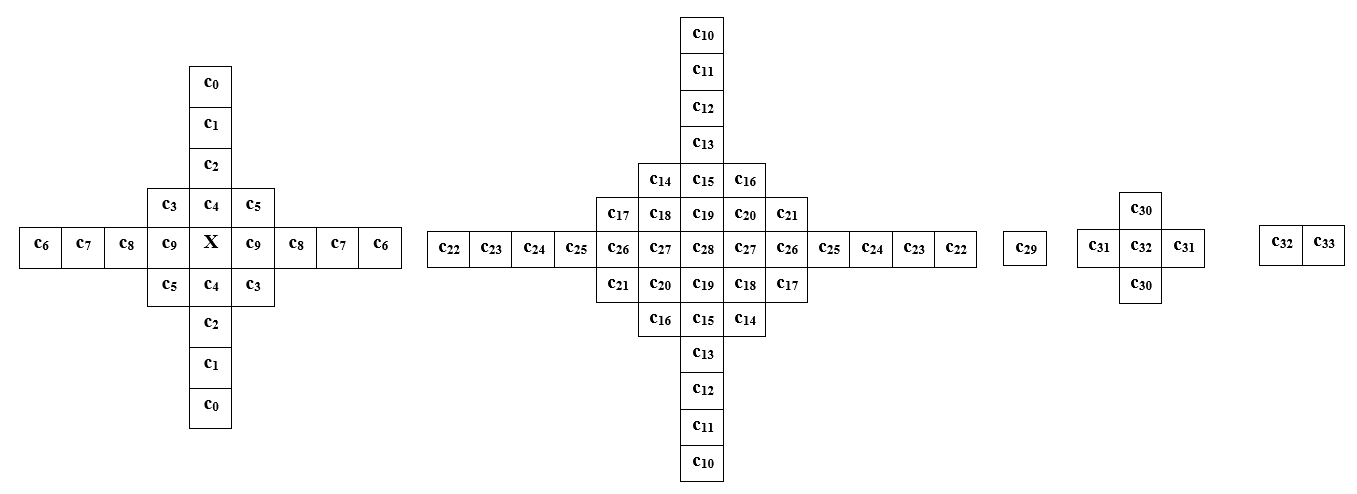
### Extended Fixed-Filter-Output based Taps for ALF

In ALF online-trained filters consist of 4 kinds of filter taps: spatial taps, reconstruction-before-DBF based taps, residual based taps and fixed-filter-output based taps as shown in **Error! Reference source not found.**.



Residual based taps

Spatial taps

RecBeforeDB taps

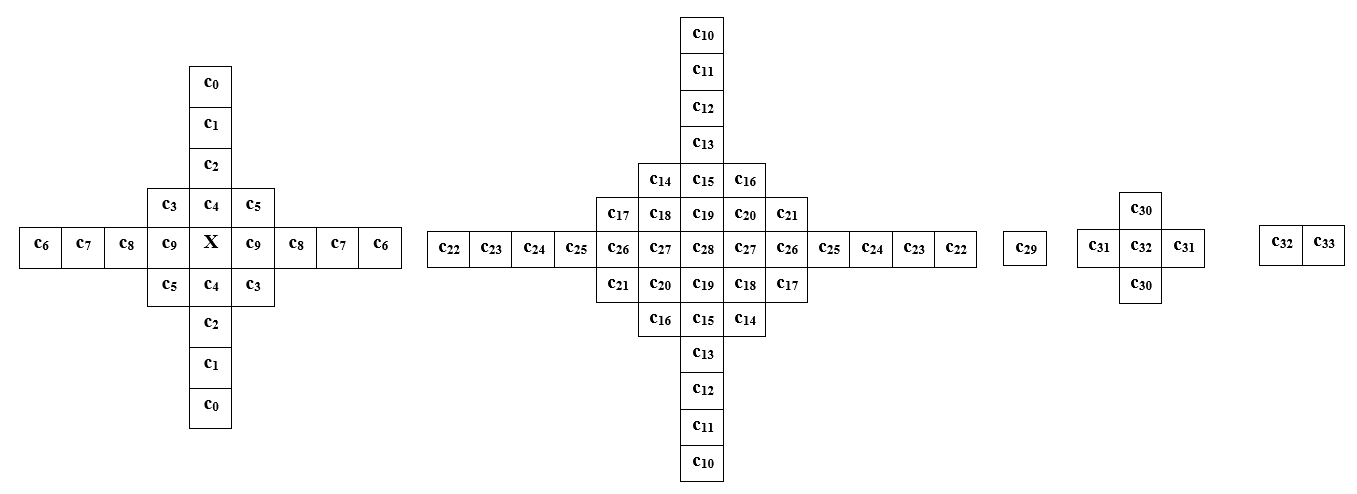
Fixed filter output based taps

Figure 57. ALF filter shapes.

### ALF with residual samples

The residual samples are used as additional inputs to the ALF. A filtered sample is derived as

where is the clipped neighboring residual sample value and is the clipped residual sample filtered by the fixed-filter. For residual samples, the fixed filter reuses the offline fixed filter trained for reconstruction after SAO.

### Additional fixed filter for ALF

Additional fixed filter with a shape of diamond 7x7 is introduced, the filter parameters are stored at both encoder and decoder. There is no classification for the newly added fixed filter.

An online filter of the proposed method is shown in Figure 58, where spatial taps (i.e., tap #0 ~ #19), reconstruction-before-DBF-based taps (i.e., tap #26, #27, #36), residual-based taps (i.e., #37 ~ #38) and fixed-filter-output-based taps (i.e., tap #20 ~ #25, #34, #35) are kept the same as the ECM-8.0, and several extended taps (i.e., tap #28 ~ #33, #39) are introduced into luma online-trained filters. The reconstruction before DBF is fed into the additional fixed filter to produce the filter outputs, then these filter outputs are used as input for newly extended taps.

This filter is always enabled without any filter shape switching.

图示, 示意图

描述已自动生成

Figure 58. Additional fixed filter.

### Improved fixed filters for ALF

Two Laplacian-based classifiers (one for each fixed filter) are applied to a 2x2 block. In each classifier, activity and directionality values are derived based on vertical, horizontal, and diagonal gradients using a window surrounding each 2x2 block. For each 2x2 block, the mean value of a surrounding window is calculated. Then, for each sample of this window, the difference between the sample value and the mean value is calculated. A scaling factor is determined based on the activity value derived from a Laplacian classifier. The square root of the sum of the squared differences is further quantized to by a scaling factor. The value of is an integer between 0 and 7, inclusively. With *i*=0, 1, let denote the classifier from the classifier of i-th fixed filter in ECM-9.0. Then the proposed class index is derived as

.

The total number of the fixed filters is not changed.

Then a class index is determined based on the activity and directionality values. Two diamond shaped fixed filters are selected from the two filter sets by using the derived two class indices. Both fixed filters are applied to samples before DBF and ALF input, where additional diamond 9x9 filter is used for the samples before DBF. The shape of the first fixed filter applied to the ALF input samples is reduced from 13x13 to 9x9, and the shape of the second fixed filter, which is 13x13, applied to ALF input is unchanged as shown in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | ECM-9.0 | Improved fixed filtering | |
|  | ALF input | Samples before DBF | ALF input |
| Fixed filter | 13x13 | 9x9 | 9x9 |
| Fixed filter | 13x13 | 9x9 | 13x13 |

Fixed filter is applied to outputs of (instead of ALF input) and samples before DBF.

Finally, a signalled filter is applied to the ALF input samples, samples before the deblocking filter (DBF), outputs of the two fixed filters, output of a gaussian filter and the residual data.

## Bilateral filter

The filter is carried out in the sample adaptive offset (SAO) loop-filter stage, as shown in Figure 59. Both the bilateral filter (BIF) and SAO are using samples from deblocking as input. Each filter creates an offset per sample, and these are added to the input sample and then clipped, before proceeding to ALF.

Diagram

Description automatically generated

Figure 59. Both BIF and SAO use samples from the deblocking stage as input. Both create an offset, and these are added to the input sample and clipped.

In detail, the output sample is obtained as

where is the input sample from deblocking, is the offset from the bilateral filter and is the offset from SAO.

The implementation provides the possibility for the encoder to enable or disable filtering at the CTU and slice level. The encoder takes a decision by evaluating the RDO cost.

For CTUs that are filtered, the filtering process proceeds as follows.

At the picture border, where samples are unavailable, the bilateral filter uses extension (sample repetition) to fill in unavailable samples. For virtual boundaries, the behavior is the same as for SAO, i.e., no filtering occurs. When crossing horizontal CTU borders, the bilateral filter can access the same samples as SAO is accessing. As an example, if the center sample (see Figure 60) is located on the top line of a CTU, , and are read from the CTU above, just like SAO does, but is padded, so no extra line buffer is needed compared to JVET-P0073.

The samples surrounding the center sample are denoted according to Figure 60, where A, B, L and R stands for above, below, left and right and where NW, NE, SW, SE stands for north-west etc. Likewise, AA stands for above-above, BB for below-below etc. This diamond shape is different from JVET-P0073 which used a square filter support, not using , , , or .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Figure 60. Naming convention for samples surrounding the center sample, I\_C.

Each surrounding sample , etc will contribute with a corresponding modifier value , , etc. These are calculated the following way: Starting with the contribution from the sample to the right, , we calculate the difference

where denotes absolute value. For data that is not 10-bit, we instead use , where n = 8 for 8-bit data etc. The resulting value is now clipped so that it is smaller than 16:

The modifier value is now calculated as

where is an array of 16 values determined by the value of qpb = clip(0, 25, QP + bilateral\_filter\_qp\_offset-17):

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }, if qpb = 0  
{ 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }, if qpb = 1  
{ 0, 2, 2, 2, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, }, if qpb = 2  
{ 0, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, -1, }, if qpb = 3  
{ 0, 3, 3, 3, 2, 2, 1, 2, 1, 1, 1, 1, 0, 1, 1, -1, }, if qpb = 4  
{ 0, 4, 4, 4, 3, 2, 1, 2, 1, 1, 1, 1, 0, 1, 1, -1, }, if qpb = 5  
{ 0, 5, 5, 5, 4, 3, 2, 2, 2, 2, 2, 1, 0, 1, 1, -1, }, if qpb = 6  
{ 0, 6, 7, 7, 5, 3, 3, 3, 3, 2, 2, 1, 1, 1, 1, -1, }, if qpb = 7  
{ 0, 6, 8, 8, 5, 4, 3, 3, 3, 3, 3, 2, 1, 2, 2, -2, }, if qpb = 8  
{ 0, 7, 10, 10, 6, 4, 4, 4, 4, 3, 3, 2, 2, 2, 2, -2, }, if qpb = 9  
{ 0, 8, 11, 11, 7, 5, 5, 4, 5, 4, 4, 2, 2, 2, 2, -2, }, if qpb = 10  
{ 0, 8, 12, 13, 10, 8, 8, 6, 6, 6, 5, 3, 3, 3, 3, -2, }, if qpb = 11  
{ 0, 8, 13, 14, 13, 12, 11, 8, 8, 7, 7, 5, 5, 4, 4, -2, }, if qpb = 12  
{ 0, 9, 14, 16, 16, 15, 14, 11, 9, 9, 8, 6, 6, 5, 6, -3, }, if qpb = 13  
{ 0, 9, 15, 17, 19, 19, 17, 13, 11, 10, 10, 8, 8, 6, 7, -3, }, if qpb = 14  
{ 0, 9, 16, 19, 22, 22, 20, 15, 12, 12, 11, 9, 9, 7, 8, -3, }, if qpb = 15  
{ 0, 10, 17, 21, 24, 25, 24, 20, 18, 17, 15, 12, 11, 9, 9, -3, }, if qpb = 16  
{ 0, 10, 18, 23, 26, 28, 28, 25, 23, 22, 18, 14, 13, 11, 11, -3, }, if qpb = 17  
{ 0, 11, 19, 24, 29, 30, 32, 30, 29, 26, 22, 17, 15, 13, 12, -3, }, if qpb = 18  
{ 0, 11, 20, 26, 31, 33, 36, 35, 34, 31, 25, 19, 17, 15, 14, -3, }, if qpb = 19  
{ 0, 12, 21, 28, 33, 36, 40, 40, 40, 36, 29, 22, 19, 17, 15, -3, }, if qpb = 20  
{ 0, 13, 21, 29, 34, 37, 41, 41, 41, 38, 32, 23, 20, 17, 15, -3, }, if qpb = 21  
{ 0, 14, 22, 30, 35, 38, 42, 42, 42, 39, 34, 24, 20, 17, 15, -3, }, if qpb = 22  
{ 0, 15, 22, 31, 35, 39, 42, 42, 43, 41, 37, 25, 21, 17, 15, -3, }, if qpb = 23  
{ 0, 16, 23, 32, 36, 40, 43, 43, 44, 42, 39, 26, 21, 17, 15, -3, }, if qpb = 24  
{ 0, 17, 23, 33, 37, 41, 44, 44, 45, 44, 42, 27, 22, 17, 15, -3, }, if qpb = 25

This is different from JVET-P0073 where 5 such tables were used, and the same table was reused for several qp-values.

As described in JVET-N0493 section 3.1.3, these values can be stored using six bits per entry resulting in 26\*16\*6/8=312 bytes or 300 bytes if excluding the first row which is all zeros.

The modifier values for , and are calculated from , and in the same way. For diagonal samples , , ,, and the samples two steps away , , and , the calculation also follows Equations 2 and 3, but uses a value shifted by 1. Using the diagonal sample as an example, we get

and the other diagonal samples and two-steps-away samples are calculated likewise. The modifier values are summed together

Note that equals for the previous sample. Likewise, equals for the sample above, and similar symmetries can be found also for the diagonal- and two-steps-away modifier values. This means that in a hardware implementation, it is sufficient to calculate the six values , , , , and and the remaining six values can be obtained from previously calculated values.

The value is now multiplied either by or , which can be done using a single adder and logical AND gates in the following way:

where denotes logical and and is the most significant bit of the multiplier and is the least significant bit. The value to multiply with is obtained using the minimum block dimension as shown in Table 11:

Table 11. Obtaining the c parameter from the minimum size D = min(width, height) of the block.

|  |  |  |  |
| --- | --- | --- | --- |
| **Block type** |  |  |  |
| Intra | 3 | 2 | 1 |
| Inter | 2 | 2 | 1 |

Finally, the bilateral filter offset is calculated. For full strength filtering, we use

whereas for half-strength filtering, we instead use

A general formula for n-bit data is to use

where bilateral\_filter\_strength can be 0 or 1 and is signalled in the pps.

## Bilateral inloop filter on chroma

Same as BIF-luma, proposed BIF-chroma is also performed in parallel with the SAO and CCSAO process as shown in Figure 61. BIF-chroma, CCSAO and SAO use the same chroma samples produced by the deblocking filter as input and generate three offsets per chroma sample in parallel. Then these three offsets are added to the input chroma sample to obtain a sum, which is then clipped to form the final output chroma sample value. The proposed BIF-chroma provides an on/off control mechanism on CTU level and slice level.

Diagram

Description automatically generated

Figure 61. Filtering stage of BIF-Chroma.

The filtering process of BIF-chroma is similar to that of BIF-luma. For a chroma sample, a 5×5 diamond shape filter is used for generating the filtering offset. The difference between the central sample and each surrounding sample is calculated first. The coefficient for each reference sample is extracted from a pre-defined look-up-table based on the calculated difference directly. The coefficients used for chroma components are retrained, different from those from BIF-luma. In the BIF-luma design, the block-level filtering strength parameter is determined based on luma TU size and CU mode. While in the BIF-chroma design, the parameter for chroma components is determined based the chroma TU size and mode when dual-tree partitioning is enabled for the current slice and based on the corresponding luma TU size and mode when dual-tree partitioning is disabled.

## Cross-Component Sample Adaptive Offset (CCSAO)

Cross-component Sample Adaptive Offset (CCSAO) is used to refine reconstructed chroma samples. Similarly to SAO, the CCSAO classifies the reconstructed samples into different categories, derives one offset for each category and adds the offset to the reconstructed samples in that category. However, different from SAO which only uses one single luma/chroma component of current sample as input, the CCSAO utilizes all three components to classify the current sample into different categories. To facilitate the parallel processing, the output samples from the de-blocking filter are used as the input of the CCSAO. Figure 62 shows the diagram of the decoding workflow when the CCSAO is applied.

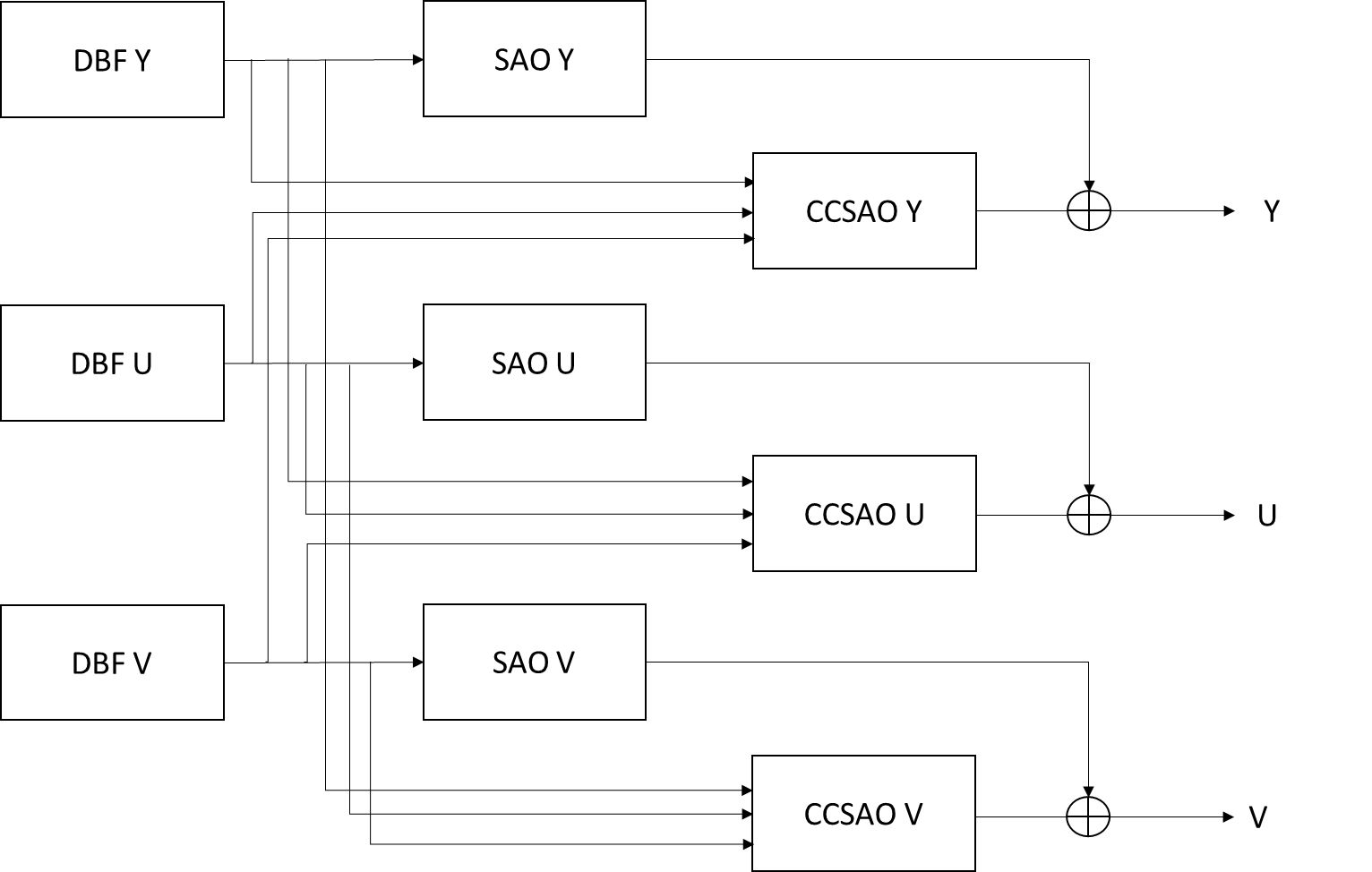


Figure 62: Modified SAO process when the proposed CCSAO is applied.

In CCSAO, only BO is used to enhance the quality of the reconstructed samples. For a given luma/chroma sample, three candidate samples are selected to classify the given sample into different categories: one collocated Y sample, one collocated U sample, and one collocated V sample. The sample values of these three selected samples are then classified into three different bands {}, and a joint index represents the category of the given sample. One offset is signaled and added to the reconstructed samples that fall into that category, which can be formulated as

where {*, ,* } are the three selected collocated samples used to classify current sample; {} are the numbers of equally divided bands applied to {*, ,* } full range respectively; is the internal coding bit-depth; and are the reconstructed samples before and after the CCSAO is applied; is the value of the CCSAO offset applied to *i*-th BO category. The collocated luma sample can be chosen from 9 candidate positions, while the collocated chroma sample positions are fixed, as depicted in Figure 63.

Similar to SAO, different classifiers can be applied to different local region to further enhance the whole picture quality. The parameters for each classifier (i.e., the position of , , , , and offsets) are signaled at picture level, and the classifier to be used is explicitly signaled and switched at CTB level. For each classifier, the maximum of {} is set to {16, 4, 4}, and offsets are constrained to be within the range [-15, 15]. At most 4 classifiers are used per frame.

Une image contenant texte, piscine à balles

Description générée automatiquement

Figure 63: Illustration of the candidate positions used for the CCSAO classifier.

SAO, Bilateral filter (BIF) and CCSAO offset are computed in parallel, added to the reconstructed chroma samples and jointly clipped, as shown in Figure 64.

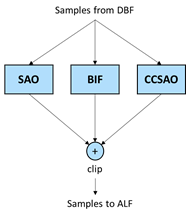


Figure 64. Joint clipping after adding SAO/BIF/CCSAO offsets to the input sample.

Similar to the edge classifier of SAO in VVC the edge-based classifier of CCSAO also uses the four 1-D directional patterns for sample classification: horizontal, vertical, 135° diagonal and 45° diagonal, as shown in Figure 65.



Figure 65. Four 1-D directional patterns for CCSAO EO sample classification: horizontal (EO class = 0), vertical (EO class =1), 135° diagonal and 45° diagonal

For every 1-D pattern, each sample is classified based on the sample difference between the luma sample value labeled as “c” and its two neighbor luma samples labeled as “a” and “b” along the selected 1-D pattern.

Similar to SAO, the encoder may decide the best 1-D directional pattern using the rate-distortion optimization (RDO) and signal this additional information in each classifier/set. Both the sample differences “a-c” and “b-c” are compared against a pre-defined threshold value (Th) to derive the final “class\_idx” information.

The encoder selects the best “Th” value from an array of pre-defined threshold values based on RDO and the index into the “Th” array is signalled.

Furthermore, an additional difference between CCSAO edge-based classifier and the SAO edge classifier in VVC is that, in the former, Chroma samples use the co-located Luma samples for deriving the edge information (samples “a”, “c” and “b” are the co-located luma samples) whereas, in the later Chroma samples use its own neighboring samples for deriving the edge information.

Two edge-based classifiers are supported in ECM, and the edge-based classifier is decided by RDO and is signaled in slice header.

The first edge-based classifier process is formulated as follows:

Ea=(a-c<0)? (a-c<(-Th)? 0:1) : (a-c<(Th)? 2:3) – (1)

Eb=(b-c<0)? (b-c<(-Th)? 0:1) : (b-c<(Th)? 2:3) – (2)

= \* 16 + Ea \* 4 + Eb – (3)

– (4)

variable “” in equation (3) is derived as follows.

(or) (or) , – (5)

wherein, sample “cur” is the current sample being processed, are the co-located samples. When Luma samples are processed, are the co-located and samples respectively. When Chroma( samples are processed, are the co-locatedandsamples respectively. Similarly When Chroma(samples are processed, are the co-locatedandsamples respectively.

Based on RDO, encoder signals one among the samples “cur”, “, used in deriving the band information.

The second edge-based classifier is a subset of the first edge-based classifier with less edge range divisions, and is formulated as follows:

Ea=(a-c<(-Th)? 0:1) – (6)

Eb= (b-c<(-Th)? 0:1) – (7)

= \* 4 + Ea \* 2 + Eb – (8)

– (9)

To reduce the signaling overhead, the CCASO offsets and classifier parameters can be inherited from previous coded pictures. A FIFO buffer is used to store the CCASO parameters. An index is signaled in slice header to indicate which candidate in the FIFO buffer is selected for the current slice.

## Entropy coding

### Extended precision

The intermediate precision used in the arithmetic coding engine is increased, including three elements. First, the precisions for two probability states are both increased to 15 bits, in comparison to 10 bits and 14 bits in VVC. Second, the LPS range update process is modified as below,

if q >= 16384

q = 215 – 1 – q

RLPS = ((range \* (q>>6)) >>9) + 1,

where range is a 9-bit variable representing the width of the current interval, q is a 15-bit variable representing the probability state of the current context model, and RLPS is the updated range for LPS. This operation can also be realized by looking up a 512×256-entry in 9-bit look-up table. Third, at the encoder side, the 256-entry look-up table used for bits estimation in VTM is extended to 512 entries.

### Slice-type-based window size

Since statistics are different with different slice types, it is beneficial to have a context’s probability state updated at a rate that is optimal under the given slice type. Therefore, for each context model, three window sizes are pre-defined for I-, B-, and P-slices, respectively, like the initialization parameters.

The context initialization parameters and window sizes are retrained.

### Improved probability estimation for CABAC

##### Multi-hypothesis probability estimation with adaptive weight

The multi-hypothesis-based probability is estimated based on adaptive weights (MHP-AW). Specifically, two separate probability estimates and are maintained for each context and updated according to their own adaptation rates. However, instead of using simple average, multiple weights are introduced to derive the resulting probability used for the binary arithmetic coding, as illustrated as follows:

where and are the weights selected from a pre-defined set ; is the bitwise right-shift value, which is equal to 5 when and 6 otherwise. Three different sets of weights are pre-determined for each context model at I-, B- and P-slice types. The weights of I-slice type are only allowed for intra slices while the weights of B- and P-slice types are allowed to be switched for inter slices at slice level.

##### CABAC initialization from previous inter slice and windows adjustment

Context initialization stored at previously coded picture after coding the last CTU can be used to initialize an inter slice having the same slice type, QP, and temporal ID. The buffer size for storing previous initializations is set equal to 5 for each slice type, when the buffer is full, the entry with the smallest QP and temporal ID is removed first before storing the initialization.

The CABAC employs two probability states that are updated with a short and a long window size, respectively. The window sizes, predefined for each context model, are not optimal for varying statistics in different regions, hence window sizes are adjusted according to the previously coded bin of each context.

The short and long window sizes used in CABAC update are adjusted by two delta parameters stored in a look-up table per context and retrieved by a previous coded bin used as an index. The previous coded bin is used as an index to get the adjustment parameters from a look-up table: delta0 for the short window and delta1 for the long window. Denote the original short and long window sizes stored in the existed initialization tables and defined for the context model as shift0 and shift1, respectively. The actual window sizes used to code the current bin after adjustment is (shift0+delta0) and (shift1+delta1), where shift0 and shift1 are existed predefined windows sizes stored in the context initialization tables.

## Gradual decoding refresh (GDR)

A coded video sequence consists of intra coded pictures (e.g. I picture) and inter coded pictures (e.g. P and B pictures). Intra coded pictures usually use many more bits than inter coded pictures. Transmission time of such big intra coded pictures increases the encoder to decoder delay. For (ultra) low delay applications, it is desirable that all the coded pictures have similar number of bits so that the encoder to decoder delay can be reduced to around 1 picture interval. Hence, intra coded picture seems not fit for (ultra) low delay applications. However, on the other hand, an intra coded picture is indeed needed at random access point.

Gradual Decoding Refresh (GDR) (Gradual Random Access (GRA) or Progressive Intra Refresh (PIR)) approaches alleviate the delay issue with intra coded pictures. Instead of coding an intra picture at a random access point, GDR progressively refreshes pictures by spreading intra coded areas over several pictures.

Figure 66 illustrates the basic concept of (vertical) GDR, where a GDR period starts with picture of POC (n) and ends at picture of POC(n+N-1). The first picture of POC(n) within the GDR period is called GDR picture. Forced intra coded areas (green) gradually spread over the N pictures of the GDR period from the left to the right. The picture of POC(n+N-1) is called recovery point picture. The pictures between GDR picture of POC(n) and recovery point picture of POC(n+N-1) are called recovering pictures of GDR picture of POC(n).



Figure 66. A GDR period starts with a GDR picture of POC(n) and ends at recovery point picture of POC(n+N-1). A picture within GDR period consists of a refreshed area and a non-refreshed area separated by a virtual boundary.

One of the requirements for GDR (such as in VVC) is “exact match” at recovery points. With exact match, the reconstructed pictures at recovery points of encoder and decoder should be identical (or matched). To achieve exact match, coding units (CUs) in refreshed areas should not use any coding information (e.g., reconstructed pixels, code mode, motion vector (MV), reference picture index (refIdx), reference picture list (refList), etc.) from non-refreshed areas, because the coding information in non-refreshed areas may not be decoded correctly at decoder. The incorrectly decoded information from non-refreshed areas may contaminate the refreshed areas, which will result in mismatch at recovery points (or leaks). Many coding tools, however, may involve in using the coding information from non-refreshed areas for CUs in refreshed areas. To support GDR functionality, the followings are included.

1. All types of partitions within a current CTU are allowed, but no CU should span both the refreshed and non-refreshed areas,
2. For CUs in refreshed area of a current picture, the reconstructed pixels in non-refreshed areas of the current picture and reference pictures are considered as “not available”, and if needed, they are horizontally padded with samples from refereshed areas.
3. For CUs in refreshed area of a current picture, coding information in non-refreshed areas of the current picture and reference pictures are considered “not available”, which will prevent CUs in refreshed area of the current picture from using coding information in non-refreshed area of the current picture and reference pictures.
4. CUs in non-refreshed area of the current picture are allowed to use the reconstructed pixels and the coding information of both refreshed and non-refreshed areas of the current picture and reference pictures.
5. In-loop filters are performed in both refreshed and non-refreshed areas, but not crossing virtual boundaries of refreshed-areas and non-refreshed areas.

Figure 67 shows an example, where with the reconstructed pixels and the coding information of non-refreshed areas of the current picture and reference picture(s) treated as “not available”, all intra and inter coding modes can be applied to CUs (green) in refreshed area of a current (recovering point) picture without restrictions.



Figure 67. For a CU in refreshed area of a current picture, the reconstructed pixels, and the coding information of non-refreshed areas of the current picture and the reference pictures are considered as “not available”.

Asymmetric deblocking may be enabled for better subjective quality around virtual boundaries of GDR/recovering pictures. Specifically, deblocking is still disabled for refreshed-area pixels () of a virtual boundary if their deblocking requires use of the coding information of non-refreshed area, but deblocking will be performed for non-refreshed-area pixels () of the virtual boundary even if their deblocking requires use of the coding information of refreshed area, as shown in Figure 68. In addition, refreshed-area pixels of a virtual boundary are used to compensate the output of deblocking of corresponding non-refreshed-area pixels. The final output of deblocking of non-refreshed area pixel is therefore equal to

where,

* is the final output of deblocking of ,
* is the output of deblocking of , and
* is the output of deblocking of when assuming use of non-refreshed area were allowed.

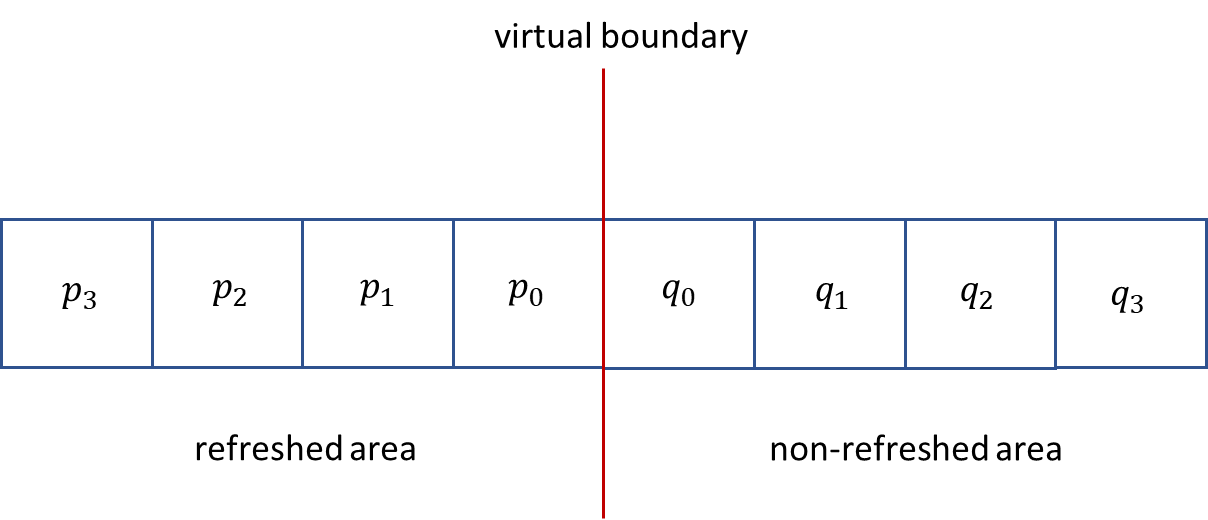


Figure 68. Deblocking is performed for pixels in non-refreshed area, but not for pixels in refreshed area.

## Simplified linear model solver

Some ECM tools, such as CCCM (Convolutional Cross-Component Model) and GLM (Gradient Linear Model), are solving sets of linear equations when calculating filter coefficient using Gaussian elimination based approach.

### Implementation

Elimination and back-substitution can be implemented for example as given below. The same function can be used to solve coefficients for one or both chrominance channels by setting numFilters parameter to 1 or 2, respectively. The xGetDivScaleRoundShift function returns scale, round and shift parameters approximating a division operation with the input diag parameter. Calculation of those parameters is implemented in ECM “xDivide” function.

**for**( **int** i = 0; i < numEq; i++ )

{

TCccmCoeff \*src = C[i];

TCccmCoeff diag = src[i] < 1 ? 1 : src[i];

xGetDivScaleRoundShift(diag, dscale, dround, dshift);

**for**( **int** j = i+1; j < numEq+numFilters; j++ )

{

src[j] = (src[j] \* dscale + dround) >> dshift;

}

**for**( **int** j = i + 1; j < numEq; j++ )

{

TCccmCoeff \*dst = C[j];

TCccmCoeff scale = dst[i];

// On row j all elements with k < i+1 are now zero

**for**( **int** k = i + 1; k < numEq+numFilters; k++ )

{

dst[k] -= FIXED\_MULT(scale, src[k]);

}

}

}

// Solve with backsubstitution

**if** ( numFilters == 2 )

{

backsubstitution(C, x0, numEq, colChr0);

backsubstitution(C, x1, numEq, colChr1);

}

**else**

{

backsubstitution(C, x0, numEq, colChr0);

}

Where:

**void** backsubstitution(TE2 C, Ty x, **int** numEq, **int** col)

{

x[numEq-1] = C[numEq-1][col];

**for**( **int** i = numEq-2; i >= 0; i-- )

{

x[i] = C[i][col];

**for**( **int** j = i+1; j < numEq; j++ )

{

x[i] -= FIXED\_MULT(C[i][j], x[j]);

}

}

}

# References

1. B. Bross, J. Chen, S. Liu, and Y.-K. Wang "Versatile Video Coding (Draft 10)," document JVET-2001, 19th JVET meeting: by teleconference, 22 June – 1 July 2020.